中山大学移动信息工程本科生实验报告

课程名称:*Artificial Intelligence*

年级	1501	专业（方向）	移动互联网
学号	15352015	姓名	曹广杰
电话	13727022190	Email	1553118845@qq.com

一、实验题目

神经网络算法

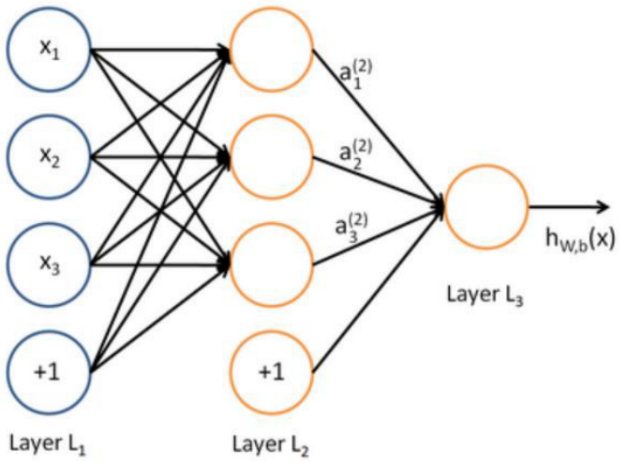
Neural Network

二、实现内容

1. 必须实现三层神经网络（输入层，隐藏层，输出层）
2. 必须在给出的优化建议中任意选择一项实现
3. 自己划分验证集（报告里说明是怎么分的）调整参数

算法原理

先用张图描述一下神经网络算法的模型。



输入层

自变量x表示输入，在本次实验中输入信息可能会有多个选项，可能是连续值或者离散值。算法将会根据这些特征值实现对于输出信息的预测。

隐藏层

中间层称为隐藏层，是对输入层输出数据的处理层，隐藏层与上一层的连线表示每一个隐藏层节点都有针对输入层输出的权值，这些权值将会变动。随着对误差信息的分析而逐渐调节。

隐藏层是神经网络模型计算数据的主要部分，任务是计算。

输出层

输出层的作用是将隐藏层（可能有多层）的数据进一步整合，输出数据并将输出与实际数据进行对比获得误差值，并返回误差值，实现隐藏层的更新。

对比新生儿学习的过程

在开始输入的时候，由于权值是初始化的，此时预测的结果可能与预期的结果不一致。这时候就像是新生儿不会走路，输出的结果与预期的状态相差甚远。

在输出数据与实际数据比较之后，就可以知道输出的偏差，此时隐藏层的权值也就对应地更新，可以类比新生儿逐渐学习的过程。但是走路并不容易，婴幼儿走路也许要摔倒很多次才可以建造完整的模型。同样的，迭代一次的学习程度也许并不足够，此时再次根据之前的输入得到的输出还会与实际值有差别，于是继续学习——多次迭代的过程其实就是重复练习的过程。

隐藏层的多个结点模拟的正是神经系统中的诸多神经元，权值信息也就对应着神经元之间的突触连接，神经元之间需要默契配合才可以完成复杂的工作，同样的，合适的权值信息才可以完美地构造一个数学模型，并将该数学模型用于预测某一个问题的答案。

伪代码

```
do
  for x_k in hiden layer
    for I_k of x_k
        O_k = w_k * I_k;
  error = abs(T - O);
  for x_k in hiden layer
    for w_k of x_k
        w_k = w_k + \delta{w_k};
while error
```

关键代码截图

先介绍使用的数据结构，即整体框架。

神经元的数据结构：

```
class Neure{
public:
    Neure();
    /*其他接口*/

    Neure *next;     // 后一层的神经元信息
    vector<double> wgt; // 权值信息
    double bias;         // 偏置项
    double error;        // 误差项
    double foretell;     // 预测值
    double(*function)(const double& v); // 激活函数
};
```

神经元结构主要用于维护管理权值序列的信息。

神经系统的数据结构：

```
class NeuNet{
public:
    NeuNet();
    int Forepass(const vector<double> &attrs);// 前向传递
    bool Backpass(const double &);  // 后向传递

    vector<Neure> neulayer; // 隐藏层神经元
    vector<Neure*> nxtlayer;// 输出层神经元（可能有多个）
};
```

神经系统的主要接口为前向传递与后向传递。

前向传递：

```
int NeuNet::Forepass(const vector<double> &attrs){
    vector<double> hidop; // 隐藏层的输出集合
  // 隐藏层的计算
    for(int npin=0; npin<neulayer.size(); npin++){
        Neure ne = neulayer[npin];
        double hidot = ne.Sprawl(attrs, sigmoid);
        hidop.push_back(hidot);
        neulayer[npin] = ne;
    }

    double predict;
    if(nxtlayer.size() == 0){
        Neure out;// 输出层
      // 输出层处理隐藏层的数据
        predict = out.Sprawl(hidop, plain);
        Propagate(&out);
    }else{
        predict = nxtlayer[0]->Sprawl(hidop, plain);
    }
    // 返回预测值
    return (int)predict;
}
```

前向传递用于使用并测试隐藏层与输出层，实现神经系统的实践预测过程。

后向传递：

```cpp
bool NeuNet::Backpass(const double &real){
  // 更新输出层权值嘻嘻
    for(int xpin=0; xpin<nxtlayer.size(); xpin++){
        Neure* crt = nxtlayer[xpin];
        crt->Errsum(real);
        crt->Learn();
        nxtlayer[xpin] = crt;
    }
// 更新隐藏层权值信息
    for(int npin=0; npin<neulayer.size(); npin++){
        Neure crt = neulayer[npin];
        crt.Errsum(real);
        crt.Learn();
        neulayer[npin] = crt;
    }
    // 更新完成，收工~!
    return true;
}
```

前向传递用于更新并维护隐藏层与输出层，实现神经系统的归纳学习过程。

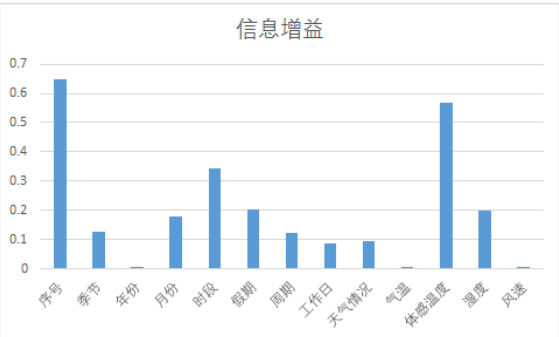
创新点与优化

从文件中截取训练数据

本次实验中文件给出的数据比较原始，很难直接用于处理，因而为了处理的方便，在实现编程之前需要对数据进行基本的处理。

首先是对于数据的选择，本次实验中一共有14组特征值，但是却未必每一个特征值都有意义，特征值“序号”和“日期”的意义显然不是很大。这也就意味着，有一些数据是对预测是有帮助的，而有一些数据，反而会对分类操作造成干扰。因此，笔者这里对每一个特征值进行评估。

笔者决定使用信息增益率作为评估的指标，检查是否对预测的结果有影响，得到信息增益率与特征值之间的关系如下：



尽管序号的信息增益率很高（没想到没想到……），但是笔者并不打算使用序号。index在本次评估中的表现是将条件熵无限地缩小了，以此掩盖了自身熵值较高的缺点，如果采用了这一属性，就会使得对序号与训练集相同的数据的预测趋向于训练数据的值，可能需要很多次迭代才可以消除该属性带来的负面影响。

所以根据属性值对标签信息的影响程度，笔者最终选择出的特征值有9个，分别为季节、月份、时段、假期、周几、工作日、天气、湿度以及体感温度。

划分训练集与验证集

由于整体模型是连续的数学模型，笔者决定使用间隔取样，每3条数据选取一条作为验证集的数据。这样，训练集训练之后得到的数学模型在预测验证集的时候就属于根据离散的点还原原有数学模型曲线的操作了。

sigmoid函数的修正

本次实验中使用的激活函数大多为sigmoid函数，该函数可以将输入的处于负无穷到正无穷之间的数值映射到0到1之间，由是计算得到输出值。显然，如果激活函数是sigmoid函数，那么对应的输出值就是0到1之间，这也就为本次实验埋下了隐患。

我们知道，在权值更新的时候，梯度中存在一个激活函数的导数——这并不难理解，计算斜率（或者梯度）存在一个导数非常正常。然而经过笔者的计算，本次实验中输入sigmoid的值90%以上都是大于0.5的，这就意味着，在该数值下，计算**梯度迭代项中**激活函数的导数的时候，该值会非常小，大多小于0.001，这就大大减缓了收敛的速度，使得收敛过程复杂而且缓慢。因此，笔者对于激活函数进行了修正。

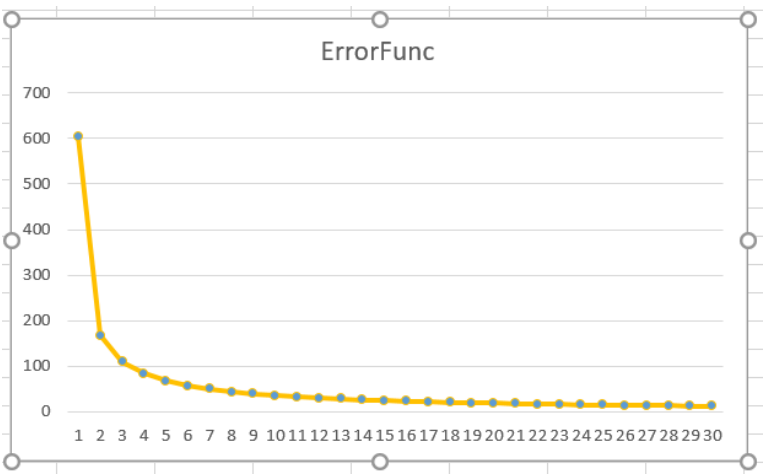
三、实验结果与分析

实验结果示例展示

小数据集测试与结果展示，笔者使用的小数据集如下：

x轴坐标	y轴坐标	距离原点
1	0	1
1	1	2
0	0	0

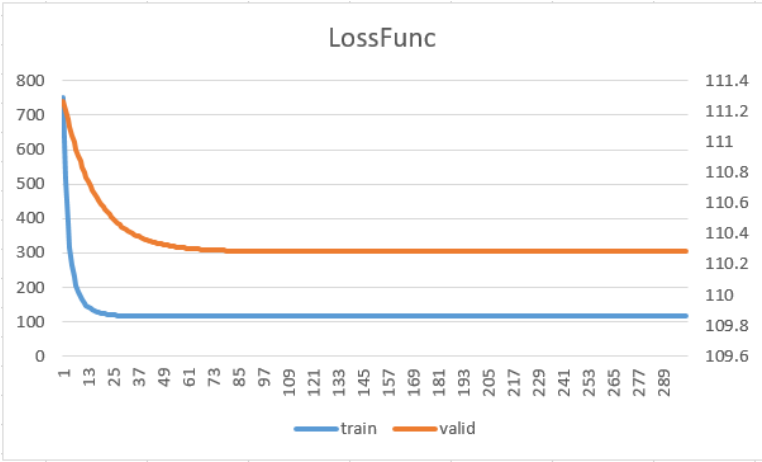
损失函数的变化趋势如下：



表示变化从605到10的过程。

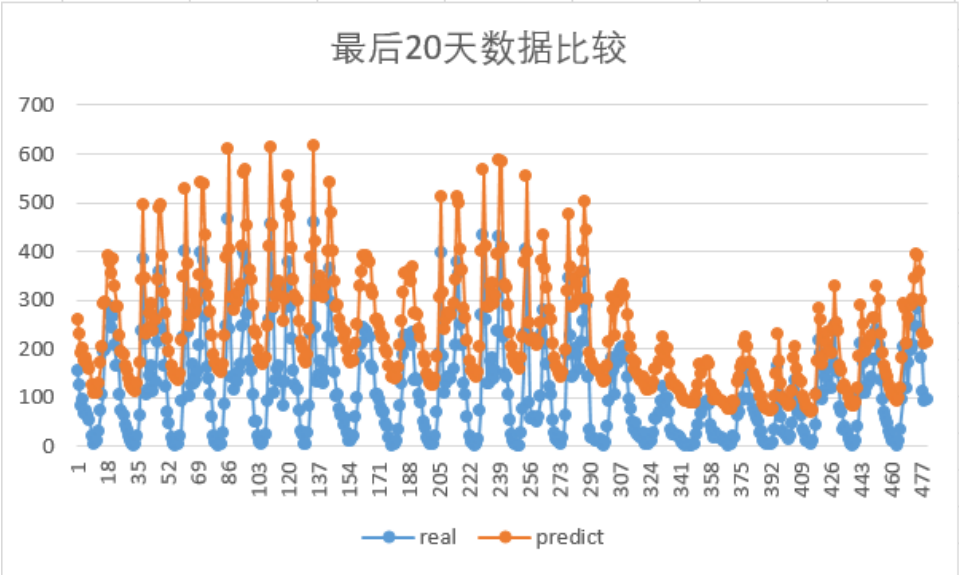
评测指标展示及分析

针对损失函数变化趋势的要求，笔者针对实现划分的训练集与验证集的损失函数获得如下记录：



由于两份数据的极值相差太大，橙色的线（验证集损失函数变化）对应右侧的损失函数值，而训练集损失函数变化则对应左侧的损失函数变化趋势。可以看到，在经过训练集训练之后的数学模型预测验证集数据时，更容易预测正确——因为损失函数的值更小。

这里添加一幅图，表示预测给定的所有8000条数据中最后2000天的预测结果与原有结果的比较：



该模型的极高相似度是因为在笔者划分的训练集中，包含了整体数据集2/3的比重，因此，在迭代的过程中就有很多的数据可以用来修正当前的数据模型，以至于非常相似。

四、思考

- 尝试说明下其他激活函数的优缺点

  Tanh函数，该函数与sigmoid函数类似，这是sigmoid函数的放缩。从迭代下降的速度上，Tanh函数要更快一些，因为在靠近原点的位置，其斜率更大。但是Tanh函数正是由于继承了sigmoid函数的性质，也同时具备了sigmoid函数中数据区分不明显、处理大的参数时分辨能力差的缺点。

  Relu函数是一个分段函数，会非常明显地区分激活和不激活两种状态。这种分明的激活方式，使得计算更加轻快灵活，收敛速度也更快。Leaky类的Relu函数，可以在保留快速收敛性质的基础上，保存部分负半轴的信息。但是因为Relu的简单结构，导致该激活函数的神经元权值更新过于直接，如果是直接将权值信息设置为0，则今后的来自该突触的数据始终为0，如此往复，该神经元将失去更新的能力。

- 有什么方法可以实现传递过程中不激活所有节点？

  1. 利用梯度消失，控制某一个神经节点的激活函数，可以实现几次迭代后其权值为0 的情况，于是其不参与运算，可以视为不激活。

2. 为每一个节点的更新迭代设置惩罚值，如果在某一次迭代中，该神经节点贡献的数据引起了输出节点的巨大误差，则触发神经节点的自我销毁机制——强制与其相关的所有的权值为0，并还原之前的权值状态。

- 梯度消失和梯度爆炸是什么？可以怎么解决？

  梯度消失是指在递归的过程前后，输出层到隐藏的更新程度逐渐消减以趋于零的情况；

  梯度爆炸是指在传递的过程中，每一层的梯度都比前一层大，迅速攀升的情况；

  梯度消失的模型是使用的激活函数不合理，比如使用sigmoid函数将会使得输出仅仅限于[0-1]的区间，如果使用其他的函数就不会造成这种问题。

  梯度爆炸的问题与此类似，大多数情况是神经元的输出与其权值呈正相关，相关系数过大，导致传递过程中，权值变化指数式攀升的情况，可以使用sigmoid的变种函数，控制每一层输出的上限。