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	周五 5-6 节	专业（方向）	软件工程（移动信息工程）M2
学号	15352204	姓名	林丹

一、 实验题目

实验六——反向传播神经网络

二、 实验内容

1. 算法原理

1.1 算法背景：

单层感知机 ➔ 多层感知机 ➔ 反向传播神经网络

- 感知器学习算法 PLA 是一个监督学习算法，处理分类问题，缺点分界面是线性的，无法处理非线性问题，。回顾 PLA，其核心思想是梯度下降法，即以训练样本被错分的误分类率为目标函数，训练中每次出现错误时便使权重 W 朝着目标函数相对于权系数负梯度方向更新，一直到目标中没有被错分的样本，或者达到迭代次数上限为止。
- 而多层感知器 MLP 中，比 PLA 多了至少一个隐藏层（除了一个输入层和一个输出层以外）。单层感知器只能学习线性函数，而多层感知器通过多个分割面也可以学习非线性函数。
- MLP 其实是一种只有前向过程的人工神经网络模型，其将输入的多个数据集映射到单一的输出的数据集上。而采用了反向传播算法进行训练的 MLP，其实就是反向传播神经网络（Back Propagation Nerual Network）

1.2 算法总览

BPNN 算法分为正向传播和反向传播过程。

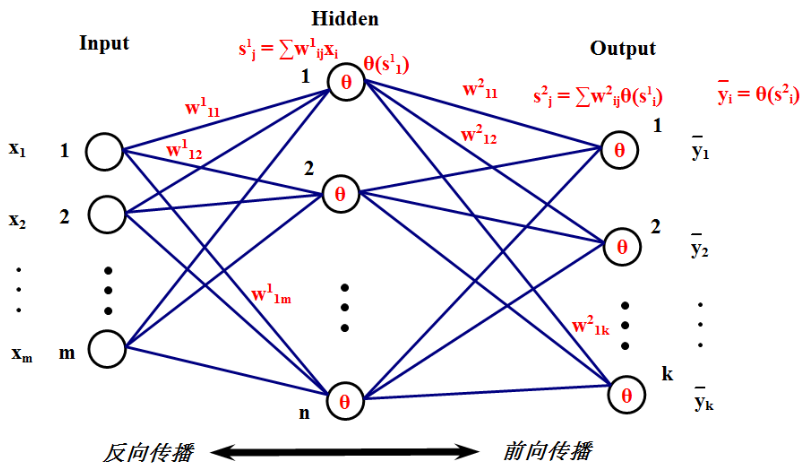
正向传播时，输入样本从输入层传入,经一个或多个隐藏层逐层加权求和后，通过传输到

函数到下一层作为输入再计算到输出层。若输出层的实际输出与期望的输出不符,则转入误差的反向传播阶段。

反向传播时，将输出值以某种学习算法（例如梯度下降法）通过隐藏层向输入层逐层反传,并将误差分摊给各层的所有单元,从而获得各层单元的误差信号,此误差信号即作为修正各单元权值的依据。

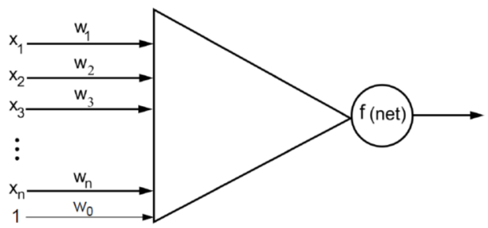
- 网络拓扑结构；

BPNN 网络拓扑结构和多层感知器类似：



- ➢ 输入节点：输入节点从外部世界提供信息，总称为「输入层」。在输入节点中，不进行任何的计算，仅向隐藏节点传递信息。
- ➢ 隐藏节点：隐藏节点和外部世界没有直接联系。这些节点进行计算，并将信息从输入节点传递到输出节点。隐藏节点总称为「隐藏层」，隐藏层可以有多层。
- ➢ 输出节点：输出节点总称为「输出层」，负责计算，并从网络向外部世界传递信息。

每一个隐藏/输出节点可以看做单个神经元，是 BPNN 的基本单位。一个神经元就是一个线性分类器。简单结构图如下：



单个神经元求得输入向量X与权向量W的内积，经激活函数$f(\cdot)$得到一个标量结果。

单个神经元的作用：把一个 n 维向量空间用一个超平面分区成两部分（称之为判断边界），给定一个输入向量，神经元可以判断出这个向量位于超平面的哪一边。

- 传递函数；

BPNN 采用非线性变换函数作为传递函数，这里叫做激活函数$f(\cdot)$：

激活函数的作用是将非线性引入神经元的输出。因为现实世界的大多数数据都是非线性的，我们希望神经元能够学习非线性的函数表示。

在实践中，可能会碰到几种激活函数：Sigmoid 函数、tanh 函数、ReLU 函数等等。具体分析见 *思考题第一题*。

- 学习算法（更新 W）。

学习算法的目标是确定一组权重，使得输出层节点的误差最小。

采用梯度下降法：对于单个样本，误差定义为 E（前向传播中 H 为隐藏层输出，T 为输出层真实值，O 为的输出值）：

$$E = \frac{1}{2} \times (T_k - O_k)^2 \qquad \text{Eq(1)}$$

$$O_k = f(\sum w_{jk} H_j), \qquad \text{Eq(2)}$$

其中输出层在回归问题时候，激活函数$f(a) = a,\ f' = 1$

定义输入层某个结点为i，隐藏层某个结点为j，输出层结点为k

输出层误差梯度：

$$\Delta w_{jk} = -\nabla E_k = -\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial O_k} \times \frac{\partial O_k}{\partial w_{jk}} = (T_k - O_k) H_j = -Err_k H_j \qquad \text{Eq(3)}$$

$$Err_k = (T_k - O_k) \qquad \text{Eq(4)}$$

隐藏层反向传播误差：

$$H_j = f(\sum w_{ij} x_i) \qquad \text{Eq(5)}$$

其中这里用激活函数 $f(a) = sigmoid = \frac{1}{1+e^{-a}}$,$f' = f(1-f)$

$$\Delta w_{ij} = -\nabla E_j = -\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial O_k} \times \frac{\partial O_k}{\partial H_j} \times \frac{\partial H_j}{\partial w_{ij}} = Err_k x_i \qquad \text{Eq(6)}$$

$$Err_j = H_j(1 - H_j) \sum Err_k w_{jk} \qquad \text{Eq(7)}$$

更新权重：

$$w_{jk} = w_{jk} + \Delta w_{jk} \quad \text{Eq(8)}$$

$$w_{ij} = w_{ij} + \Delta w_{ij} \quad \text{Eq(9)}$$

2. 伪代码

Function *[w_{ij}, w_{jk}] = BPNN(x_i, T ,H, limit, η)*

Input：*m-samples-dataset with n attributes* $X \in \mathbb{R}^{m\times n}$,

for each sample, $X_m = \{x_1, x_2, \dots, x_i, \dots, x_n\}$

corresponding label set $T \in \mathbb{R}^{m\times 1}$

H: number of nodes in HIDDEN layer

limit: maximum iteration

η: the learning rate

Output:

w_{ij}, *weight between Input Node I to Hidden Node j*

w_{jk} *, weight between Hidden Node j to Output Node k*

Initialize:

$X_m := augmented\ matrix = \{1, x_1, x_2, \dots, x_i, \dots, x_n\}$

X := standardize X

while iter < limit

$\Delta_H = 0,\ \Delta_O = 0$

For each sample

Forward propagation:

Compute O_k *according to Eq(2)*

Back propagation:

Compute $\Delta w_{jk}, \Delta w_{ij}$ *according to Eq(3) Eq(6)*

$\Delta_O = \Delta_O + \Delta w_{jk}$

$\Delta_H = \Delta_H + \Delta w_{ij}$

Update Weight:

$w_{jk} = w_{jk} + \eta \frac{\Delta_O}{m}$

$w_{ij} = w_{ij} + \eta \frac{\Delta_H}{m}$

end while

3. 关键代码截图（带注释）

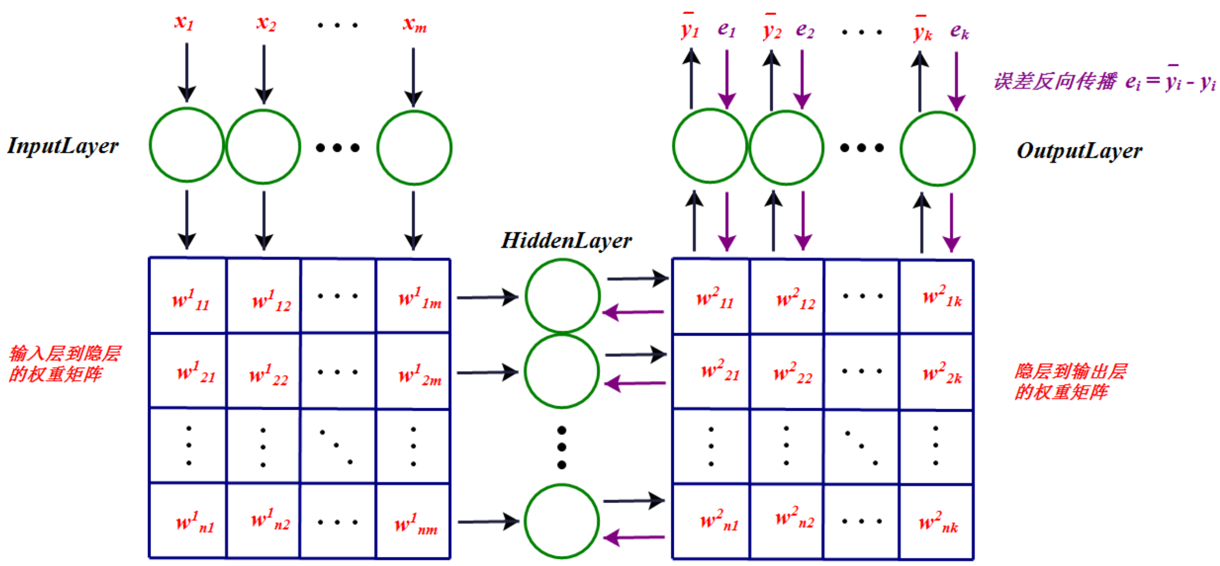
基于 **MATLAB** 的实现

```matlab

function output = BPNN(X, t, v_X, v_truth,H, max_iter,step)
% X -- 输入训练数据样本特征数据矩阵，N行,即N个样本，n列，即n个特征，
% t -- 输入训练数据的标签，N×1
% v_X -- 输入验证数据样本特征数据矩阵
% v_truth -- 输入验证数据的标签
% H -- 隐藏层的

[N,n] = size(X);
% N -- 训练样本的样本数
% n -- 训练样本的特征个数
v_N = size(v_X, 1) ;
% N -- 验证集样本的样本数
X = [ ones(N,1) , X ] ;
% 增广矩阵

if N~= length(t)
    error('inconsistent sample size');
end
%% 随机初始化
W = rand(H,n+1);         % W --  输入到隐藏层的权重（包括偏置）
wo = rand(1,H+1);        % wo -- 隐藏层到输出层的权重（包括偏置）

iter = 0;        % 迭代计数
cost_training = zeros(1,max_iter);  % training loss
cost_val = zeros(1,max_iter);       % validation loss
while iter < max_iter
    delta_W = zeros(H,n+1);      % 初始化Δ_W=0
    delta_wo = zeros(1,H+1);     % 初始化Δ_wo=0
    for i=1:N
        %% 前向传播
        % 隐藏层，激活函数sigmoid
        ho = sigmoid( X(i,:) * W' ) ;    % 隐藏层输出结果
        h_t = [1,ho ];                   % 隐藏层输出结果增广向量
        yo = h_t * wo';
        %% 后向传播
        % 计算输出层δ
        Err_wo = ( t(i)-yo );
        delta_wo = delta_wo + [1, ho ] * Err_wo;
        % 计算隐藏层层δ
        Err_W = ho .* ( 1- ho ) * Err_wo .* wo(2:H+1);
        delta_W = delta_W + repmat( Err_W',1,n+1) .* repmat(X(i,:),H,1) ;
    end
    %% 更新权重
    wo = wo + step*delta_wo;
    W = W + step.*delta_W ;
```

```
    cost = 0;
    %% 计算training loss (MSE)
    for i=1:N
        ho = sigmoid( X(i,:) * W' ) ;
        yo = [1, ho ] * wo';
        cost = cost + (yo - t(i) ) ^2;
    end
    iter = iter  + 1;
    cost_training(iter) = cost / N;
    %% 计算validation loss (MSE)
    predicts = BPNN_predicts( v_X, W, wo );
    e = predicts - v_truth ;
    cost_val(iter) = e' * e / v_N ;

end
output.W = W;
output.wo = wo;
output.cost = cost_training;
output.cost_val = cost_val;
% End function BPNN()
%% 根据权重，计算输出层预测结果
function predicts = BPNN_predicts( X, W, wo )
[N,~] = size(X);
X = [ ones(N,1) , X ] ;
predicts = zeros(N,1);
for i=1:N
        ho = sigmoid( X(i,:) * W' ) ;
        h_t = [1,ho ];
        predicts(i) =  ceil(h_t * wo');
        if predicts(i) < 0
            predicts(i) = 1 ;
        end
end
% End function BPNN_predicts()
```

4. 创新点&优化（如果有）

4.1 MATLAB 向量化运算，简化代码



① 将权重包含了偏置，即$w_0 = b$

② 加权求和利用 MATLAB 的矩阵乘法计算，避免用循环函数；

③在根据 Eq(7)计算$Err_j = H_j(1 - H_j)\sum Err_k w_{jk}$ 的时候，利用了 MATLAB 的复制和平铺矩阵函数 repmat，简化代码：

```
%% 后向传播
 % 计算输出层δ
Err_wo = ( t(i)-yo );
delta_wo = delta_wo + [1, ho ] * Err_wo;
% 计算隐藏层层δ
Err_W = ho .* ( 1- ho ) * Err_wo .* wo(2:H+1);
delta_W = delta_W + repmat( Err_W',1,n+1) .* repmat(X(i,:),H,1) ;
```

4.2 数据预处理

① 选择删除了数据集中的以下属性：

- instant: record index　　类似于记录编号，没有实际意义

- dteday : date　　　　　年月日没有太大意义，而且字符串类型 MATLAB 不支持读入。

剩余以下属性：

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	season	mnth	hr	holiday	weekday	workingda	weathersi	temp	atemp	hum	windspeec	cnt		
2	1	1	0	0	6	0	1	0.24	0.2879	0.81	0	16		
3	1	1	1	0	6	0	1	0.22	0.2727	0.8	0	40		
4	1	1	2	0	6	0	1	0.22	0.2727	0.8	0	32		
5	1	1	3	0	6	0	1	0.24	0.2879	0.75	0	13		
6	1	1	4	0	6	0	1	0.24	0.2879	0.75	0	1		
7	1	1	5	0	6	0	2	0.24	0.2576	0.75	0.0896	1		

② 利用 MATLAB 的 mapminmax 函数进行归一化。

mapminmax 将矩阵的每一行处理成[-1,1]区间；

数学公式为 y = (ymax-ymin)*(x-xmin)/(xmax-xmin) + ymin。

如果某行的数据全部相同，此时 xmax=xmin，除数为 0，则此时数据不变。

但是必须注意 MATLAB 的这个函数是对行进行归一化，（因为此时对于统计学来说，数据应该是每一列是一个样本，每一行是多个样本的同一维）

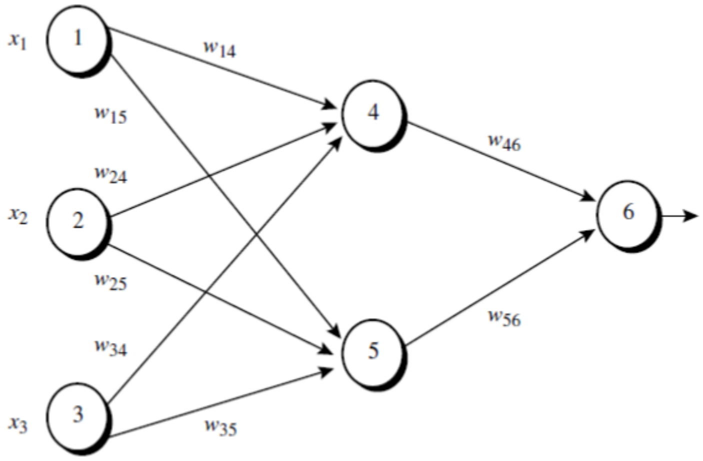
```
[trainVectors,PS] = mapminmax(trainVectors_raw(:,1:trainColumn-1)' ,xMIN,xMAX);
trainVectors = [trainVectors', trainVectors_raw(:,trainColumn) ] ;
```

③ 采用其他的激活函数，注意更新公示的求导。但是尝试了 tanh 函数和 ReLU，效果并不好，遗憾暂时未找到原因。

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】

采用课件例子：构建如下的 BPNN。



初始化数据：输入、权重以及偏置

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

- 前向传播：

隐藏层：H_j

Unit j	Net input, I_j	Output, O_j
4	$0.2+0-0.5-0.4=-0.7$	$1/(1+e^{0.7})=0.332$
5	$-0.3+0+0.2+0.2=0.1$	$1/(1+e^{-0.1})=0.525$

```
ho =

    0.3318    0.5250
```

输出层：

Unit 6 Net input, $I_k = -0.3 \times 0.332 - 0.2 \times 0.525 + 0.1 = -0.105$

Output, $O_k = I_k = -0.105$

```
yo =

   -0.1045
```

- 后向传播：

输出结点： $Err_6 = 1 - (-0.105) = 1.105$

```
Err_wo =

    1.1045
```

传递到隐藏层：

Unit	Err_j
4	$O_4(1-O_4)Err_6 \times w_{46} = (0.332)(1-0.332)(1.105)(-0.3) = -0.0735$
5	$O_5(1-O_5)Err_6 \times w_{56} = (0.525)(1-0.525)(1.105)(-0.2) = -0.0551$

```
Err_W =

   -0.0735   -0.0551
```

更新权重和偏置，学习率$\eta = 0.9$

$\theta_y = 0.1 + (0.9)(1.105)(1) = 1.0941$

$w_{46} = -0.3 + (0.9)(1.105)(0.332) = 0.0298$

$w_{56} = -0.3 + (0.9)(1.105)(0.525) = 0.3219$

```
wo =

    1.0941    0.0298    0.3219
```

$\theta_0 = -0.4 + (0.9)(-0.0735)(1) = -0.4661$

$w_{14} = 0.2 + (0.9)(-0.0735)(1) = 0.1339$

$w_{24} = 0.4 + (0.9)(-0.0735)(0) = 0.4$

$w_{34} = -0.5 + (0.9)(-0.0735)(1) = 0.3219$

$\theta_1 = 0.2 + (0.9)(-0.0551)(1) = 0.15041$

$w_{15} = -0.3 + (0.9)(-0.0551)(1) = -0.1339$

$w_{25} = 0.1 + (0.9)(-0.0551)(0) = 0.1$

$w_{35} = 0.2 + (0.9)(-0.0551)(1) = 0.1504$

```
W =

   -0.4661    0.1339    0.4000   -0.5661
    0.1504   -0.3496    0.1000    0.1504
```

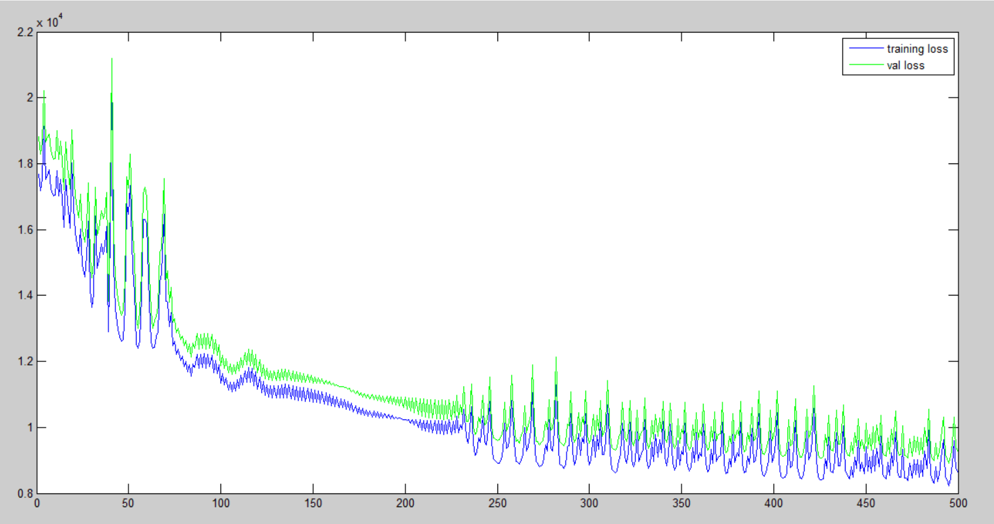
2. 评测指标展示即分析（如果实验题目有特殊要求，否则使用准确率）

1）对所有的数据打乱，取其中 80%的数据作为训练集，20%为验证集

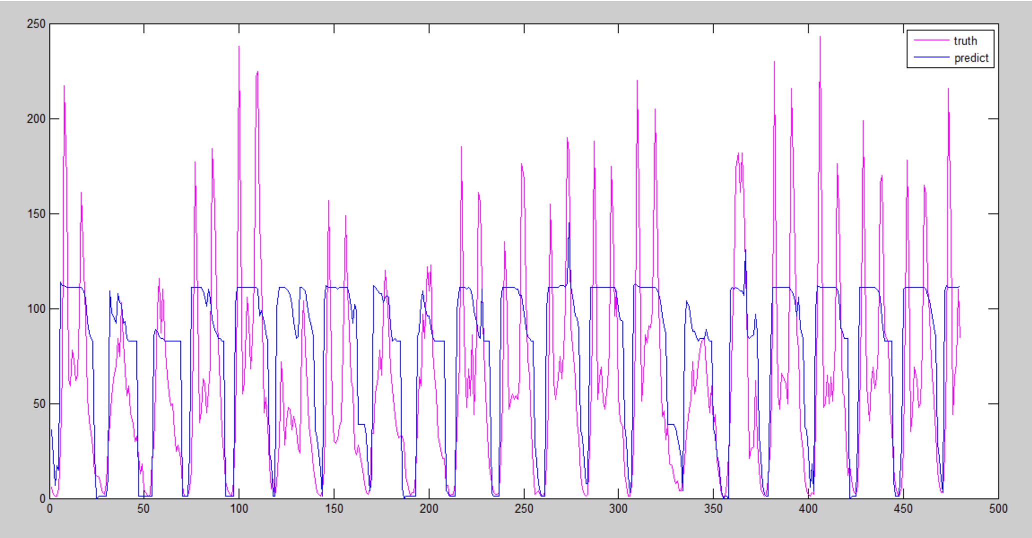
2）通过 MATLAB 的 rand 函数，初始化 W 为[0,1]在(0, 1)之间均匀分布的随机数

3）根据多次试验验证，取隐藏层结点个数为 H= log2(样本数)效果好

- 数据预处理归一化[0,1]，迭代 500 次，学习率 10^-5

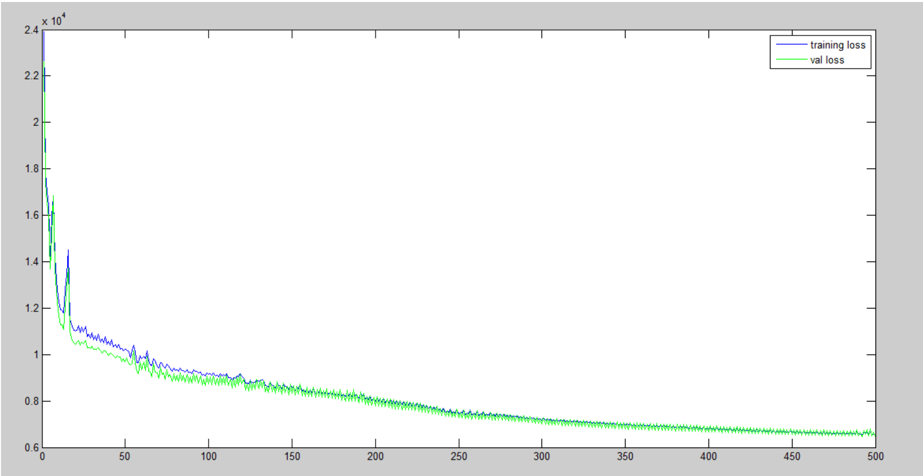


取二十天观察：

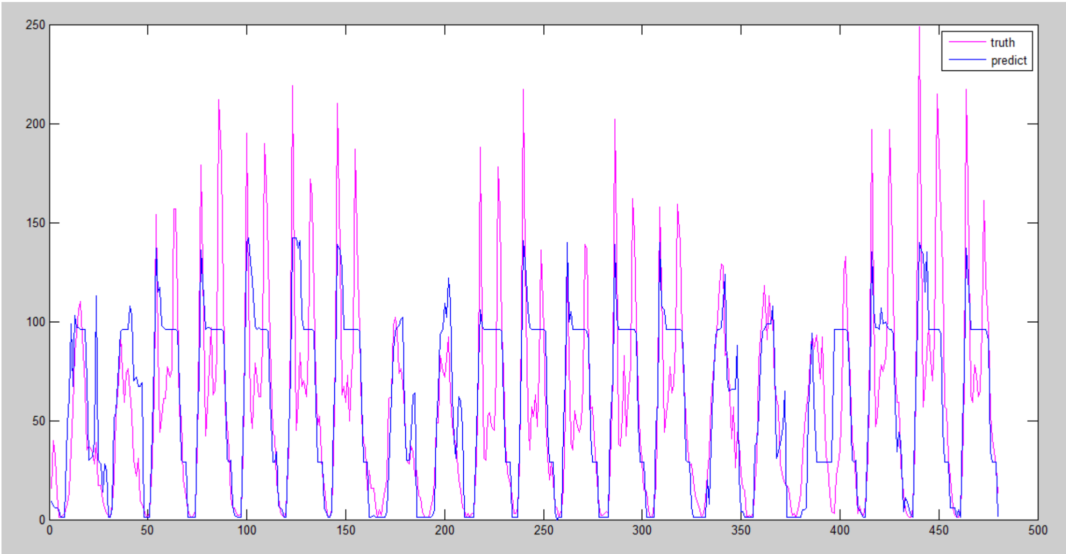


分析：在数值小的地方效果较好，无法输出交大的数值，不合适。考虑修改标准化范围

- 数据预处理标准化[-1,1]，迭代 500 次，学习率 10^-5，

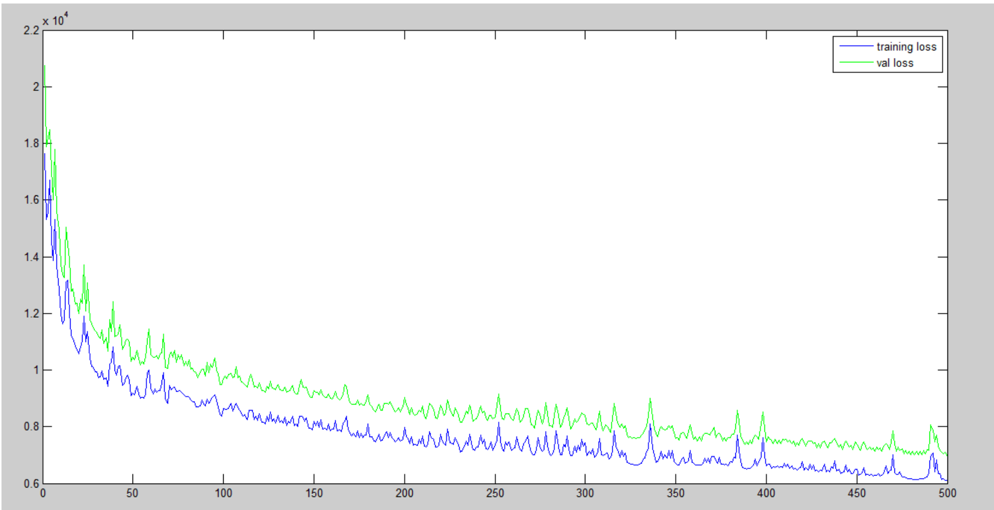


取二十天观察：

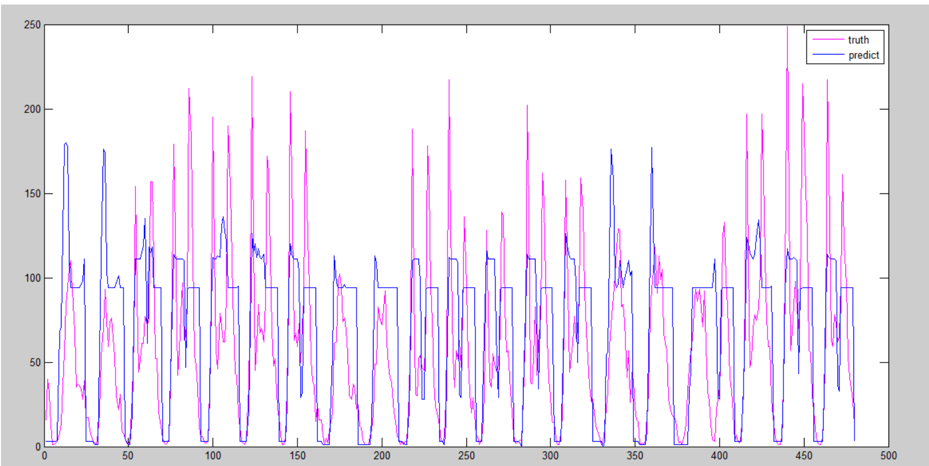


分析：标准化[-1,1]在数值大的地方标准化[0,1]效果好，但是依旧不够，考虑修改范围。

- 数据预处理归一化[-1,2]，迭代 500 次，学习率 10^-5，隐藏层节点数 H= log2(样本数)

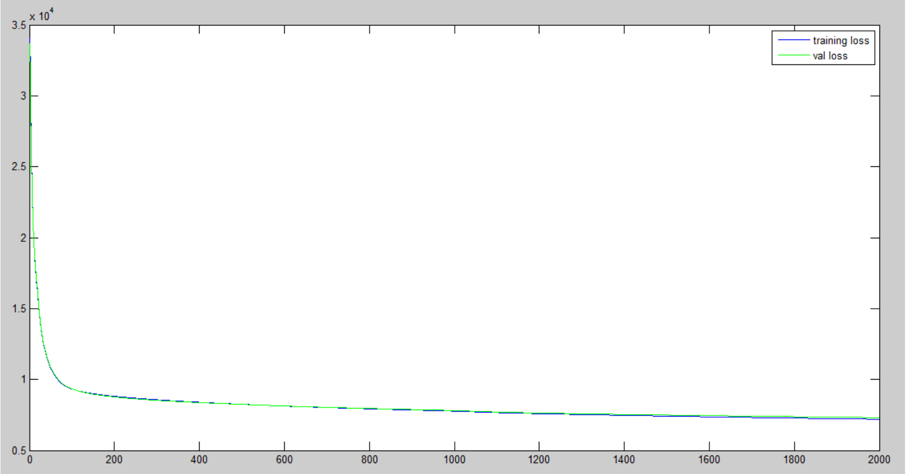


取二十天观察：

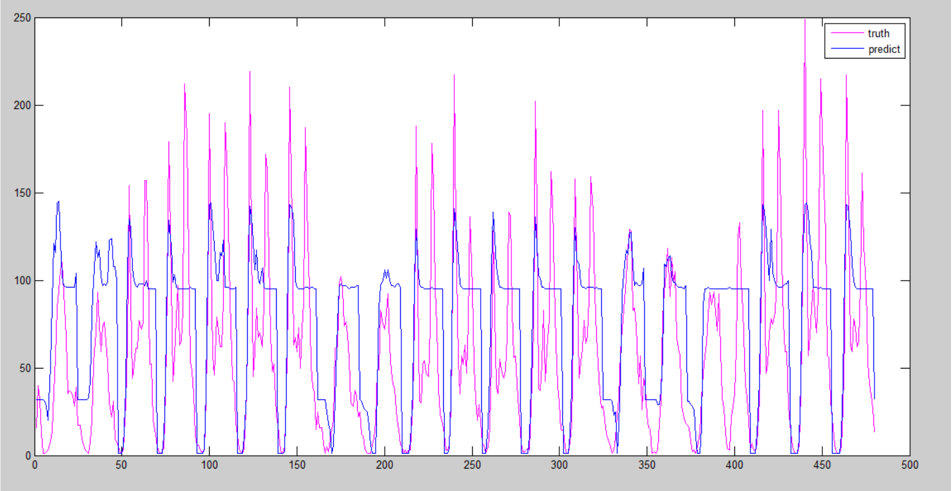


分析：标准化[-1,2]可以预测较大的数值，但是并不是在该增大的地方增大。增大迭代次数

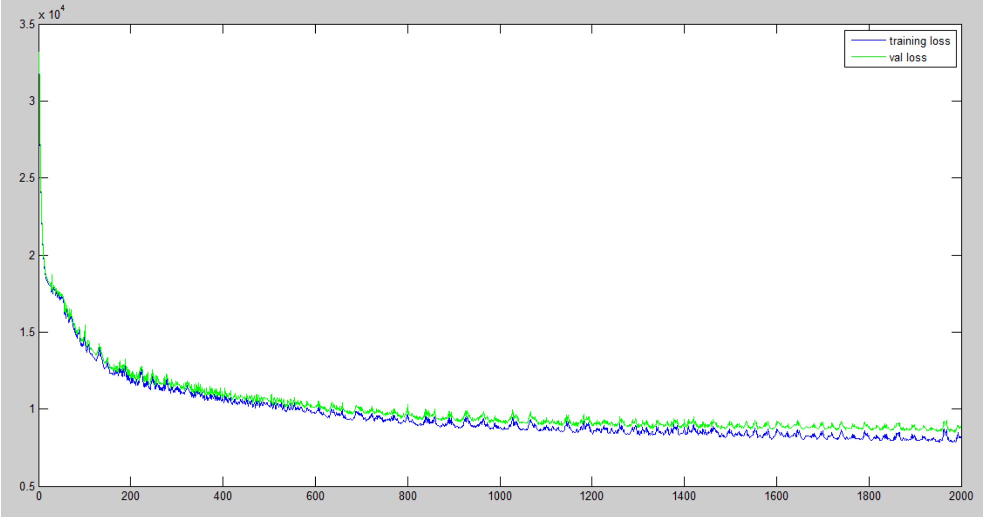
- 数据预处理归一化[-1,2]，迭代 2000 次，学习率 10^-6，



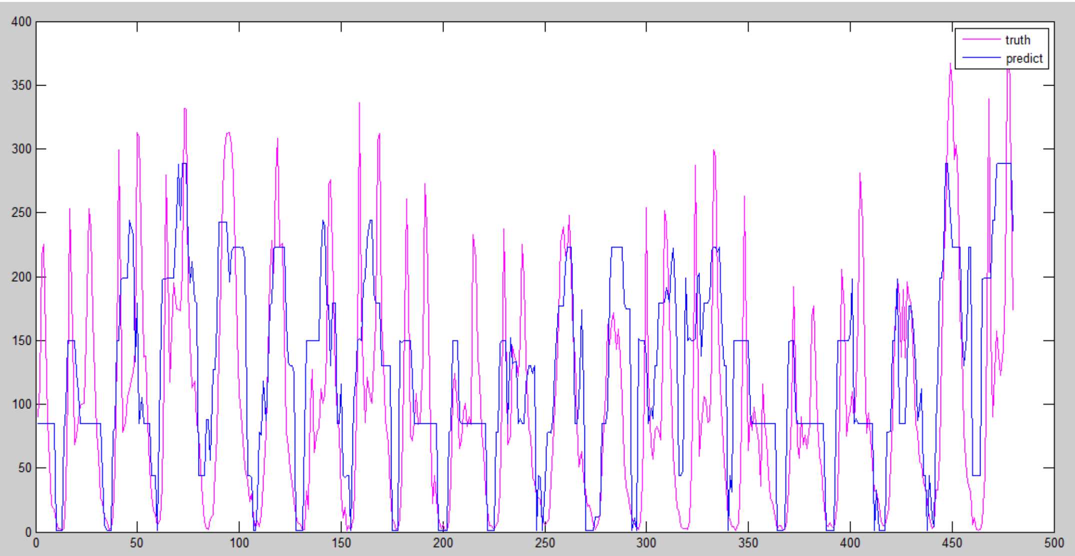
取二十天观察：



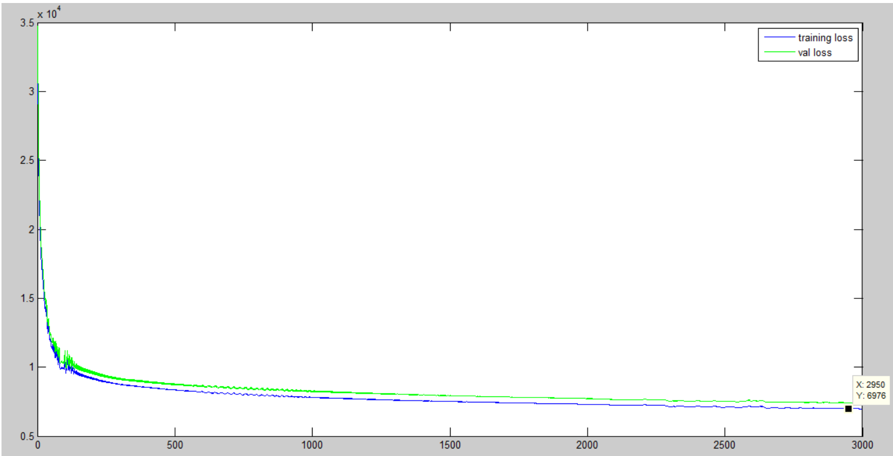
- 数据预处理归一化[0,10]，迭代 2000 次，学习率 10^-6，



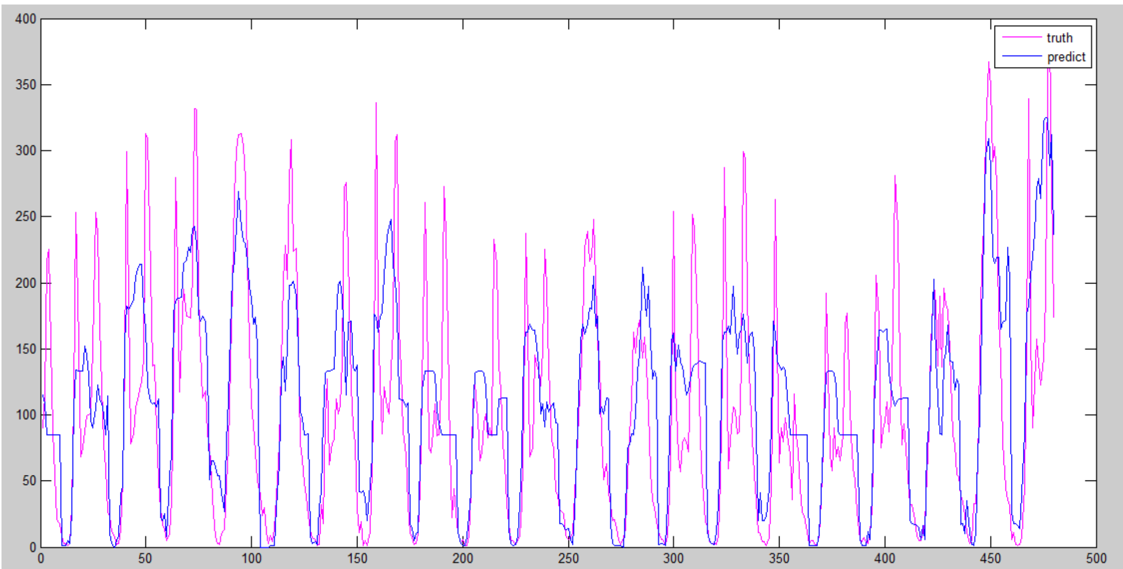
取二十天观察：



- 数据预处理归一化[-1,5]，迭代 3000 次，学习率 10^-6，



取二十天观察：

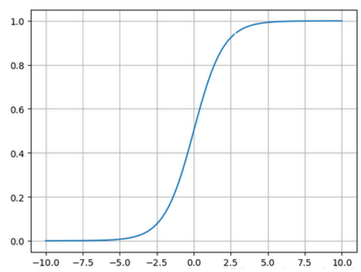


分析：由于最后的结果是一个正整数，初始化的 W 是[0,1]，所以归一化的时候大于零的部分大一些，这样子才能输出数值较大的的数。

四、 思考题

1、尝试说明下其他激活函数的优缺点。

1）Sigmoid 函数，从（$-\infty,\infty$）映射到（0,1）



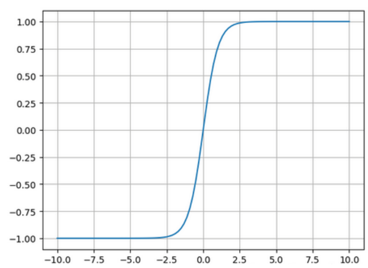
$$f(x) = \frac{1}{1+e^{-x}}$$

具备可求导的属性

$$f'(x) = f(x)(1-f(x))$$

- 优点：从数学上来看，Sigmoid 函数对中央区的信号增益较大，对两侧区的信号增益小，在信号的特征空间映射上，有很好的效果。从神经科学上来看，中央区酷似神经元的兴奋态，两侧区酷似神经元的抑制态，因而在神经网络学习方面，可以将重点特征推向中央区，将非重点特征推向两侧区。
- 缺点：① sigmoid 容易饱和，发生梯度消失。当输入非常大或者非常小的时候，神经元的梯度就接近于 0 了。这就使得我们在反向传播算法中反向传播接近于 0 的梯度，导致最终权重基本没什么更新，我们就无法递归地学习到输入数据了。② sigmoid 的输出不是零中心的，即均值不为 0，这会导致传递到后面的数据均值不为零也不为零，导致梯度下降时的晃动，因为如果数据到了神经元永远时正数时，反向传播时权值 w 就会全为正数或者负数。

2）tanh 函数，从（$-\infty,\infty$）映射到（-1,1）



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

具备可求导的属性

$$f'(x) = 1 - f(x)^2$$

- 优点：输出均值,为 0，解决了 sigmoid 函数输出均值不为 0 的问题。比 sigmoid 函数延迟了饱和期。

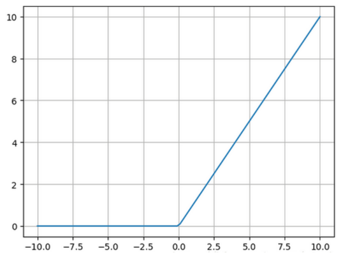
- 缺点：是 sigmoid 函数的放缩版，也存在饱和的问题，导致训练效率低。

3）修正线性单元 ReLU (Rectified linear unit)

$$f(x) = max(0, x)$$

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

$$f'(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 1 \end{cases}$$



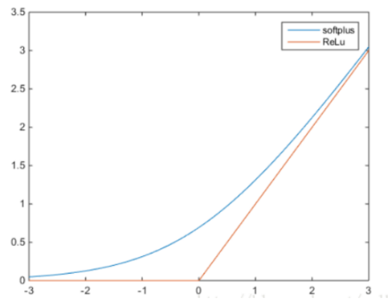
- 优点：① ReLU 函数非饱和；在梯度下降上有更快的收敛速度；②没有指数运算，只是简单的设置一个阈值，比 sigmoid/tanh 函数操作开销小；
- 缺点：Relu 会使一部分神经元的输出为 0，这样就造成了网络的稀疏性。当这发生时，经过此单元的梯度将永远为零。

4) softplus 函数

$$y = log(1 + e^x)$$

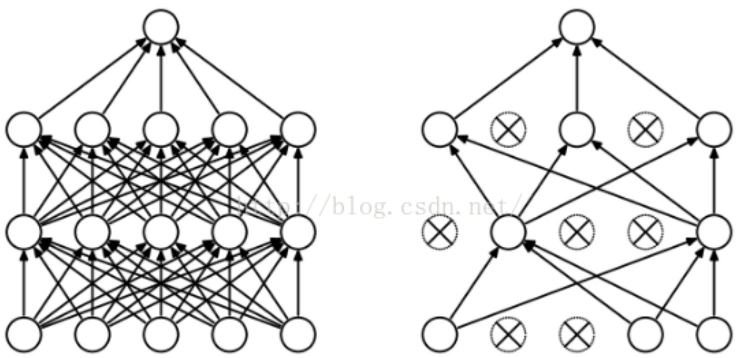
具备可求导的属性

Softplus‘(x)=Sigmoid(x)



2、有什么方法可以实现传递过程中不激活所有节点？

dropout 方法：在每次训练的时候，让一半的特征检测器停过工作，可以提高网络的泛化能力。在每次训练的时候，每个神经元有百分之 50 的几率被移除，这样可以让一个神经元的出现不应该依赖于另外一个神经元。在前向传导的时候，让某个神经元的激活值以一定的概率 p，让其停止工作，示意图如下：



(a) Standard Neural Net　　(b) After applying dropout.

怎么让某个神经元以一定的概率停止工作？

以前我们网络的计算公式是：

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

采用 dropout 后计算公式就变成了：

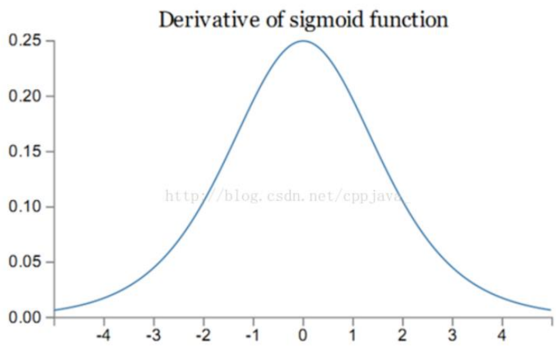
$$
\begin{aligned}
r_j^{(l)} &\sim \mathrm{Bernoulli}(p), \\
\widetilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\widetilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

上面公式中 Bernoulli 函数，是为了以概率 p，随机生成一个 0、1 的向量。

3、梯度消失和梯度爆炸是什么？可以怎么解决？

以 sigmoid 函数为例：

梯度下降：sigmoid 函数的导数曲线为：



因此导致前面的层比后面的层梯度变化更小，故变化更慢，从而引起了梯度消失问题。

梯度爆炸：当权值过大，前面层比后面层梯度变化更快，会引起梯度爆炸问题。

因为 sigmoid 导数最大为 1/4，故只有当 abs(w)>4 时才可能出现梯度爆炸，所以梯度小时比梯度爆炸更普遍

如何解决梯度消失和梯度爆炸？

使用 ReLU、softplus 等激活函数替代 sigmoid。