

BP学习率对单隐层神经网络的训练影响分析

机器学习第八周作业

樊泽羲 2200010816

问题定义

考虑如下两类监督学习问题：

1. *Classification* (1)

Dataset : $D = \{x_i, y_i\}_{i=1}^N$, where $y_i \in \{1, 2, \dots, M\}, 1 \leq i \leq N$ (2)

Goal : find discrete - valued function $f(x_i) = y_i$ (3)

2. *Regression* (4)

Dataset : $D = \{x_i, y_i\}_{i=1}^N$, where $y_i \in I, 1 \leq i \leq N$ (5)

Goal : find continuous - valued function $f(x_i) = y_i$ (6)

单隐层神经网络通过仿射变换与非线性激活逼近目标函数，其基本模型如下：

$$\begin{aligned} h^{(0)} &= x = z^{(0)} \\ h^{(1)} &= f^{(1)}(x) = a^{(1)}(z^{(1)}) = a^{(1)}(W^{(1)T}x + b^{(1)}) \\ y &= h^{(2)} = f^{(2)}(x) = a^{(2)}(z^{(2)}) = a^{(2)}(W^{(2)T}h^{(1)} + b^{(2)}) \end{aligned} \quad (7)$$

其中 W 代表权重， b 代表偏置， a 代表激活函数。注意这里隐藏层和输出层的激活函数可能不同，视任务而定。

分类器和回归器分别采用交叉熵和MSE作为分类问题和回归问题的损失函数。设batch size为 n ，则损失函数变为：

$$\begin{aligned} \text{Classification loss} : L(f) &= - \sum_{i=1}^n [y_i \log f(x) + (1 - y_i) \log(1 - f(x))] \\ \text{Regression loss} : L(f) &= \frac{1}{2} \sum_{i=1}^n (y_i - f(x))^2 \end{aligned} \quad (8)$$

在训练过程中，采用BP算法更新参数，在单隐层神经网络中，各层误差计算如下：

$$\begin{aligned} \delta^{(2)} &= L(f) \\ \delta^{(1)} &= \frac{\partial a^{(2)}}{\partial z^{(1)}} \odot (W^{(2)T} \cdot \delta^{(2)}) \end{aligned} \quad (9)$$

批梯度更新如下：

$$\begin{aligned} \nabla_{W^{(t)}} L &= \delta^{(t)} \cdot h^{(t-1)T} \\ \nabla_{b^{(t)}} L &= \delta^{(t)} \\ W^{(t)} &\leftarrow W^{(t)} - \eta \nabla_{W^{(t)}} L \\ b^{(t)} &\leftarrow b^{(t)} - \eta \nabla_{b^{(t)}} L \end{aligned} \quad (10)$$

其中 $t \in \{1, 2\}$ ， η 为学习率

本报告希望进一步分析学习率 η 对单隐层前馈神经网络训练过程的影响。具体来说，我们寻求回答以下问题：

1. 在训练过程中，学习率如何影响收敛速度和稳定性？
2. 获得最佳表现的最佳学习率是多少？
3. 高学习率是否会导致训练数据的过拟合？

为此，我们用matlab实现了单隐层前馈神经网络，在UCI的两个数据集上分别进行了分类和回归实验。比较了两个问题设置下，不同学习率引起的分类器和回归器的收敛性态变化。这将有助于理解浅层神经网络训练对学习率的依赖关系，并为特定数据集上的超参数调优提供了经验证据

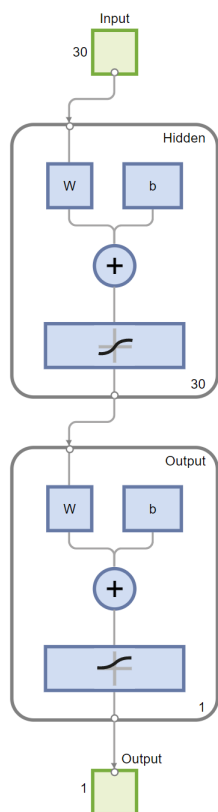
算法设计

采用matlab的Deep Learning Toolbox实现，神经网络结构和训练配置如下

神经网络结构

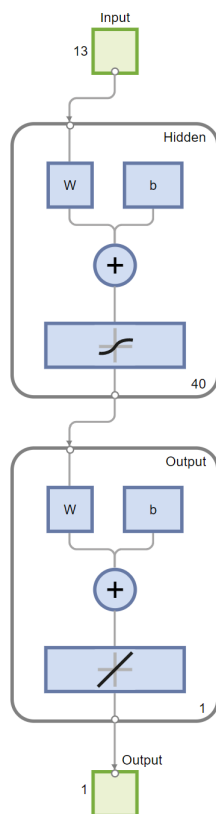
对于分类任务，网络包含一个输入层，一个隐层和一个输出层：

- 输入层神经元数目等于特征数,对Wisconsin Breast Cancer来说是30个输入神经元
- 隐层节点数目设置为30个
- 输出层节点数目为1，用0和1分别表征良性和恶性
- 隐层采用tanh作为激活函数，输出层采用sigmoid作为激活函数
- 训练该网络时采用二元交叉熵作为损失函数
- 采用早停法作为正则化策略



对于回归任务，同样包含一个输入层，一个隐层和一个输出层：

- 输入层神经元数目等于特征数,对Boston Housing来说是13个输入神经元
- 隐层节点数目设置为40个
- 输出层节点数目等于回归变量维数,对Boston Housing来说是1个输出节点(房价中位数)
- 隐层使用tanh作为激活函数，输出层采用identity map作为激活函数
- 训练该网络时采用MSE作为损失函数
- 采用早停法作为正则化策略



训练配置

训练神经网络采用批梯度下降法来最小化损失函数

对于分类器，我们测试以下学习率对训练过程的影响：

- 1
- $5e-3$
- $1e-7$

对于回归器，我们测试以下学习率对训练过程的影响：

- $5e-4$
- $8e-5$
- $1e-7$

其他超参数如batch size和迭代次数等保持不变

到此,我们设计了两个基本的单隐层前馈神经网络分别对不同数据集进行分类和回归任务，同时通过测试多个学习率来分析其对训练过程的影响程度

数据集处理

分类数据集: Wisconsin Breast Cancer¹

Wisconsin Breast Cancer 数据集被选中的原因如下：

- 样本数目为699，大小中等，这足以让神经网络学习有意义的模式，但又不会因为太大而导致算力溢出
- 对恶性肿瘤和良性肿瘤进行分类是一个常见的现实问题，也很容易理解。这使得结果更加可解释
- 拥有30个特征，比Iris等简单数据集维度更多，可以更清晰地区分不同学习率下网络的分类能力

以下列出了该数据集的一些关键特征：

Characteristic	Details
Number of samples	699
Number of features	30
Feature types	real-valued
Target variable type	binary (malignant, benign)

回归数据集: Boston Housing²

Boston Housing 数据集被选中的原因如下：

- 作为一个回归任务，它很好地补充了上一个分类任务的问题设置，学习率的影响应该在这两种情形下分别讨论，再取综合
- 有506个样本和13个特征，与Wisconsin Breast Cancer大小相似，具有可比性
- 根据房产特征预测价格同样也是一个具有可解释性的实际应用

以下列出了该数据集的一些关键特征：

Characteristic	Details
Number of samples	506
Number of features	13
Feature types	real-valued
Target variable type	continuous (median housing price)

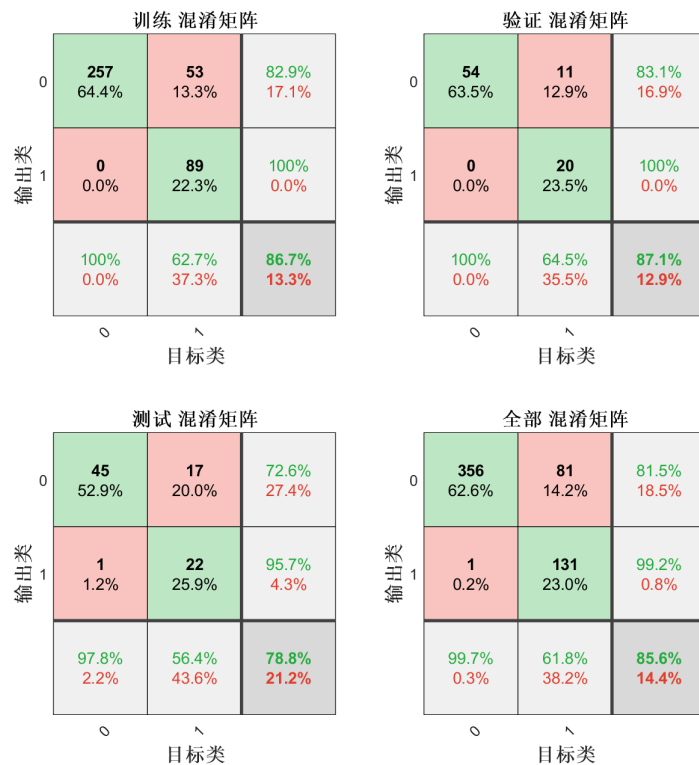
两个数据集均从UCI数据集存储库下载，没有使用其他外部数据。在输入神经网络之前，特征被标准化。每个数据集被随机分割，70%用于训练，15%用于验证，15%用于测试

实验结果

分类任务结果

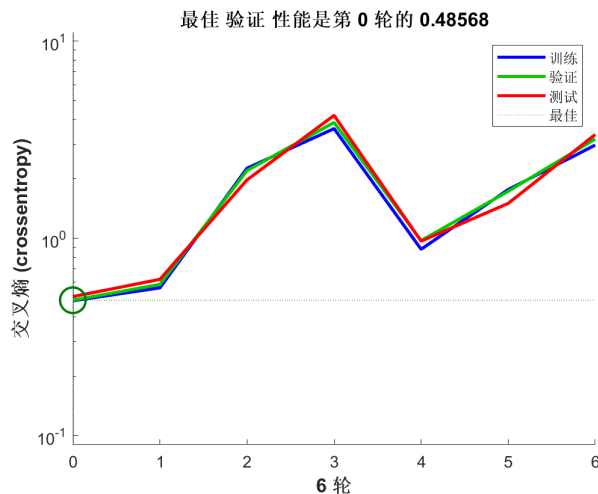
1. $\eta = 1$

混淆矩阵



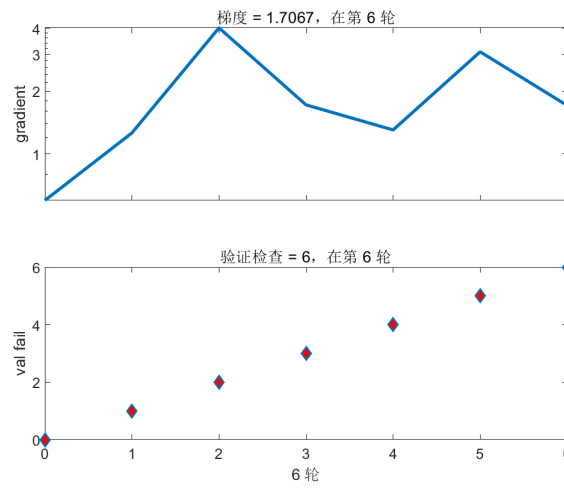
测试集的混淆矩阵表现出较差的分类性能，大部分样本被错误分类。这表明该模型在如此高的学习率下对新数据的泛化效果不好

损失曲线



再看损失曲线，我们可以看到训练性能在未训练时反而最优，梯度更新仅仅进行了6轮，随后由于误差不断上升而由早停法中止。说明学习率过大导致步长过大，算法在初始点附近振荡，从最开始就无法向最优接近。这也可以解释为严重的过拟合，仅对单个样本点改进预测结果，完全不考虑泛化

梯度和验证误差

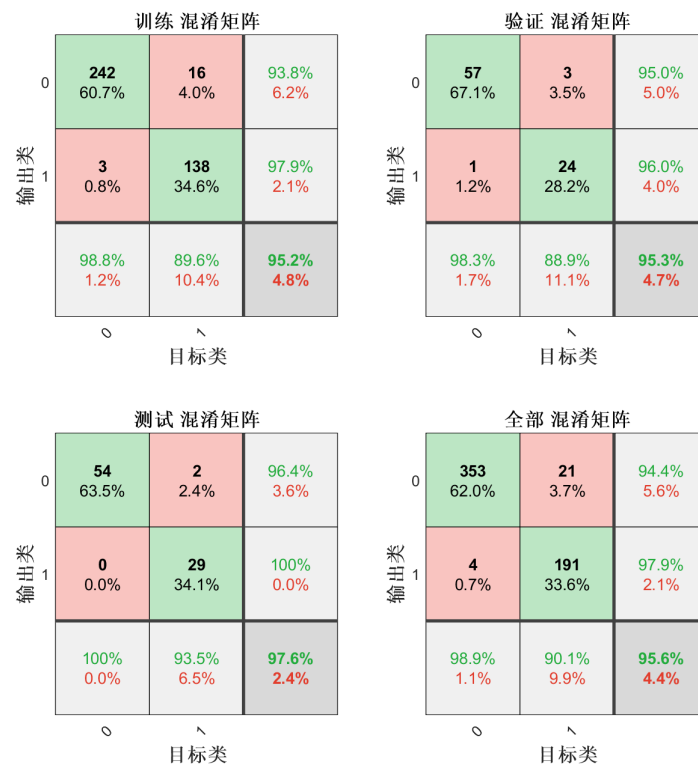


由于仅仅进行了6轮迭代，梯度没有明显下降，验证误差也无法得到改善

总而言之，就此分类任务而言，学习率 η 过高。从混淆矩阵可以看出，该模型缺乏泛化能力；从性能曲线上看，在初始点附近振荡，从最开始就无法改善预测结果；梯度大小表明算法在梯度足够小之前就被迫中止，不利于收敛

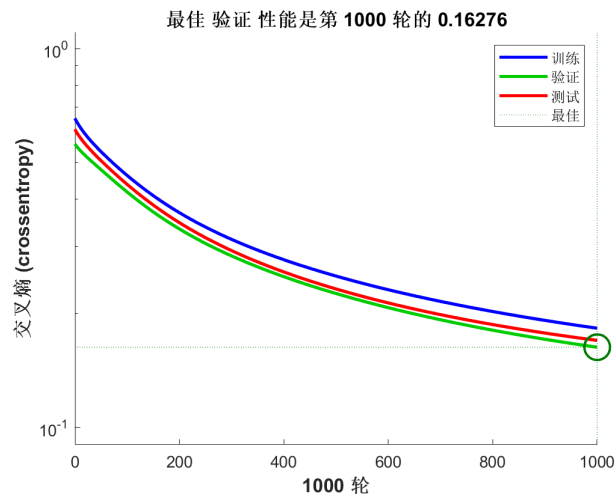
$$2. \eta = 5e - 3$$

混淆矩阵



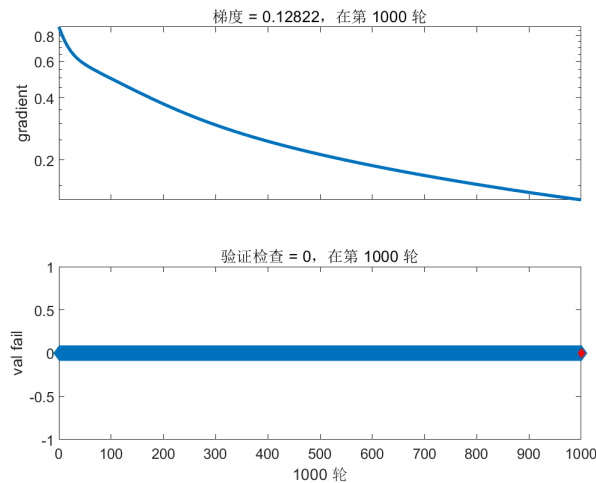
混淆矩阵显示在测试集上的分类效果很好，大多数样本分类正确。这表明在这个较小的学习率下，模型对新数据有较好的泛化能力

损失曲线



损失曲线显示,在1000轮迭代中, 损失函数稳定下降, 未被早停法中止。相较 $\eta = 1$ 的情况, 振荡现象明显改善, 函数稳步接近最优

梯度和验证误差

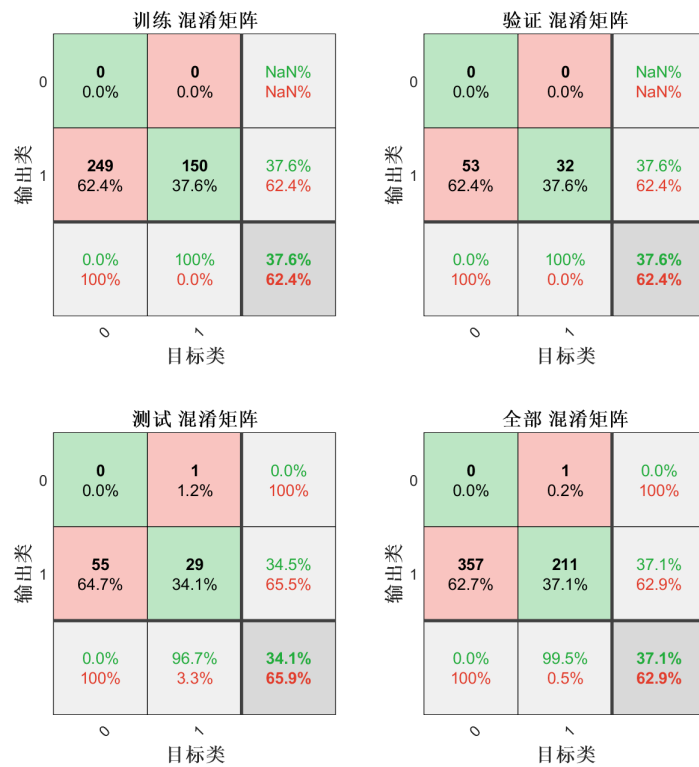


由上图可见, 模型的梯度在训练中波动幅度较小, 下降明显更为稳定, 利于逼近最优解。同时, 验证误差几乎为0, 说明模型获得了很好的泛化能力

总而言之, 学习率 $5e-3$ 对该分类任务来说较为理想。混淆矩阵和曲线表明剧烈振荡问题得到明显缓解, 而梯度显示优化过程也更为稳定。用这个学习率训练可以取得较好效果

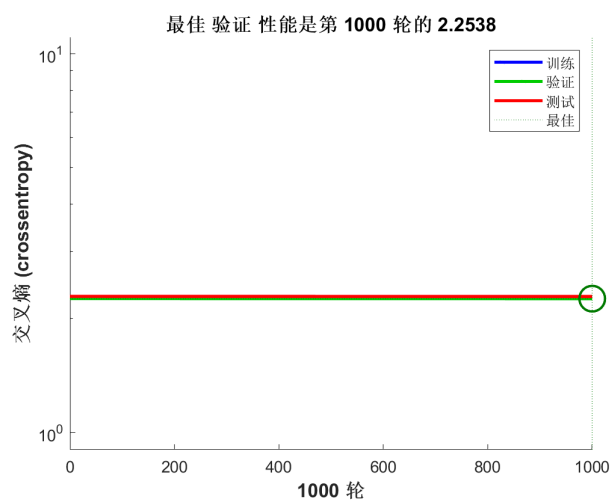
$$3. \eta = 1e - 7$$

混淆矩阵



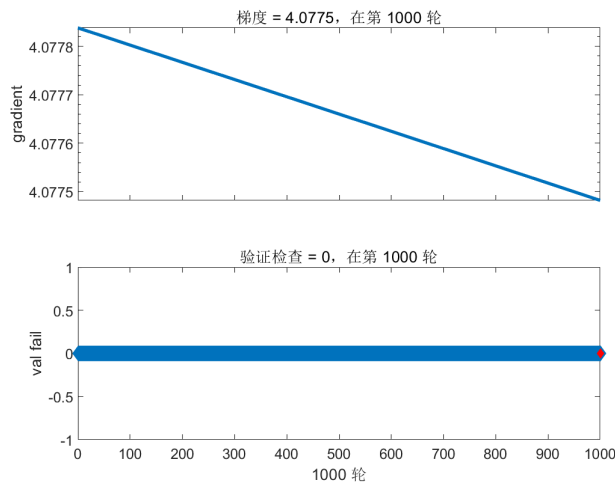
混淆矩阵显示该学习率下分类效果效果较差，总体准确率仅达到37.1%，可能是由欠拟合导致的

损失曲线



由于学习率过小，损失函数在三个数据集上均没有明显变化。虽然最优性能确实在1000轮达到和早停法未中止训练说明模型的损失确实在稳步下降，但是改善过小，最终对泛化能力没有明显贡献

梯度和验证误差



梯度和验证误差曲线显示,梯度下降训练过程非常平稳, 但是经过1000轮迭代仅仅下降了0.0003, 远小于 $\eta = 5e - 3$ 时的0.8, 模型远未达到收敛条件

总体来说, 学习率 $1e-7$ 可以使模型训练过程非常稳, 但代价是长时间收敛和欠拟合。相较 $5e-3$ 而言, 这个学习率过低, 模型在进入局部最优的邻域之前就达到最大迭代次数中止, 导致泛化误差较大

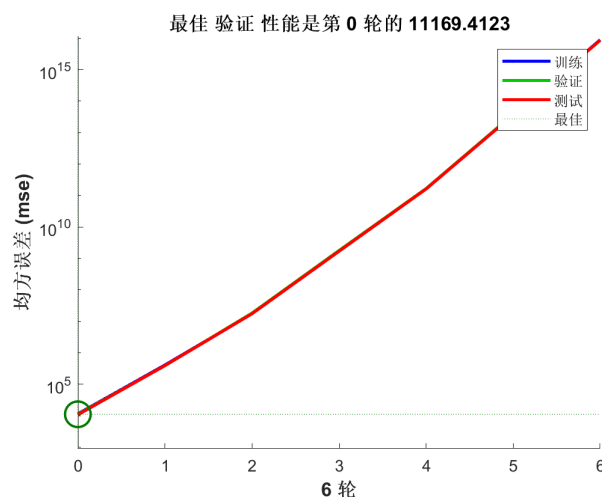
综合以上三组实验结果可以总结:

1. 学习率 $1e-7$ 时: 模型训练过程非常稳定, 几乎没有过拟合。但收敛速度极其缓慢,需要更长时间迭代才能达到更好效果, 模型发生欠拟合。此学习率可能过小
2. 学习率 $5e-3$ 时: 训练过程趋于稳定, 过拟合程度也得到很好控制。混淆矩阵显示较好的泛化性能。此学习率可能是三者中效果最佳
3. 学习率 $1e-3$ 时: 模型训练过程不稳定, 优化算法难以收敛, 导致被早停法中止。混淆矩阵和曲线表明存在严重过拟合问题。此学习率明显过大

回归任务结果

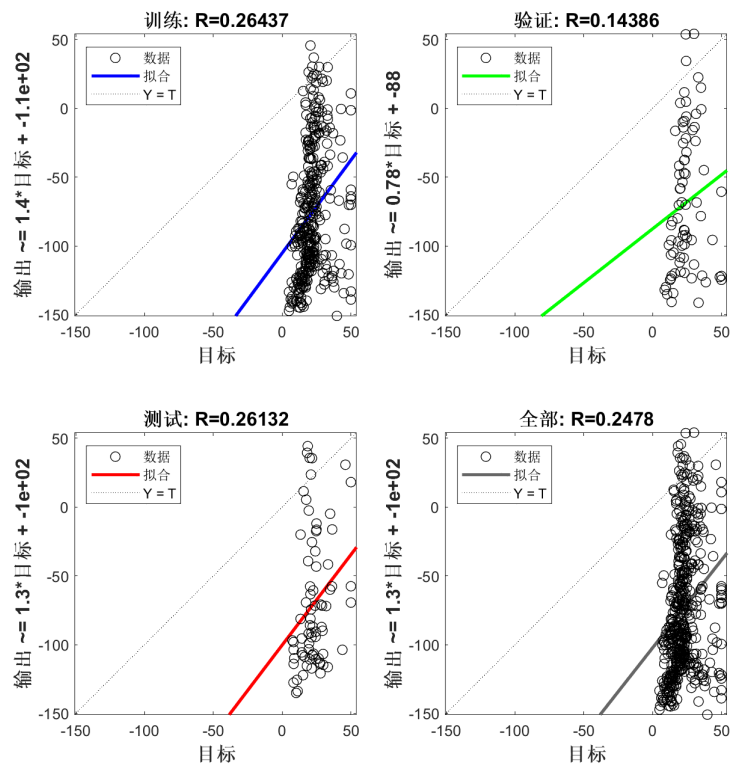
1. $\eta = 5e - 4$

损失曲线



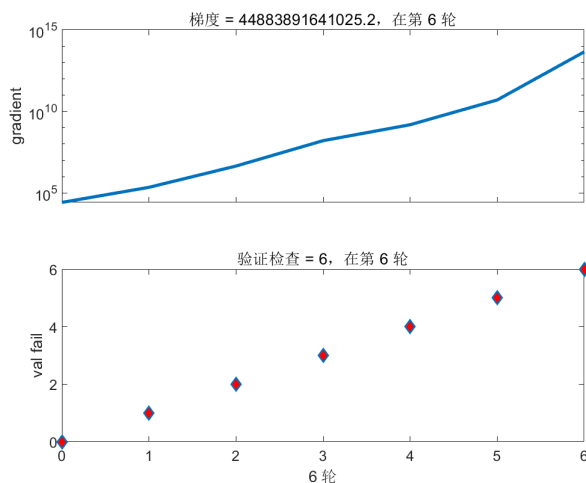
由损失曲线可见, 由于学习率设置过大, 模型在最开始时拟合误差反而最小。随后由于梯度爆炸导致误差快速上升, 被早停法中止。说明学习率过大在回归问题中不仅会导致无法收敛, 还会导致数值大小爆炸, 因此采取正则化策略更加必要

回归曲线



由于仅仅进行了6次迭代，拟合的函数离真实情况仍然十分遥远

梯度和验证误差

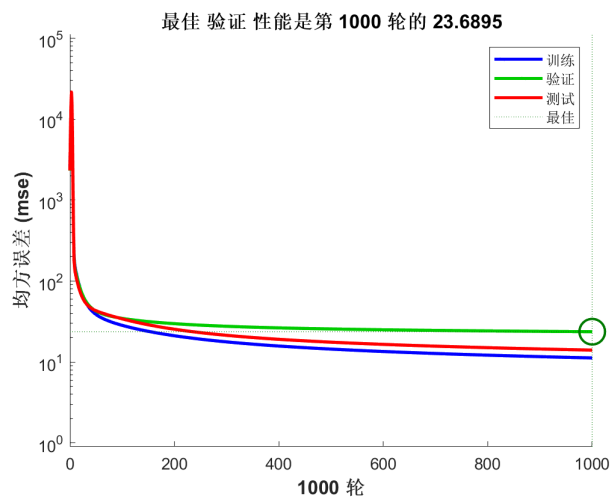


梯度大小呈指数级增长，达到 10^{15} 量级，印证了我们的梯度爆炸分析。同时验证误差在6轮中接近线性上升

总而言之，学习率 $5e-4$ 对该回归问题而言过大，底层的优化过程非常不稳定，发生了梯度爆炸现象，导致算法不能接近最优解。注意到这个学习率对之前的分类问题已经很小，说明MSE作为损失函数较交叉熵而言可能需要更小的学习率来防止数值大小失控

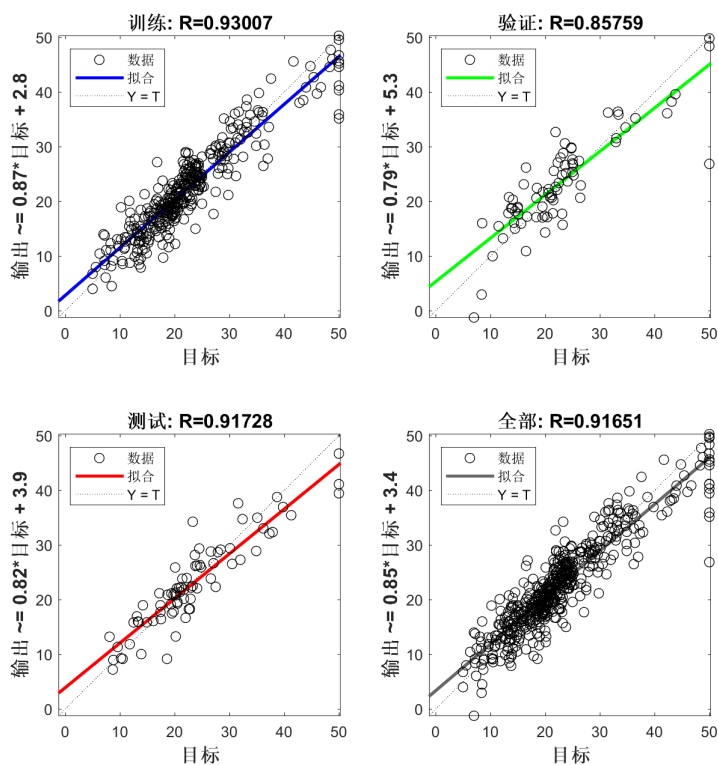
$$2.\eta = 8e - 5$$

损失曲线



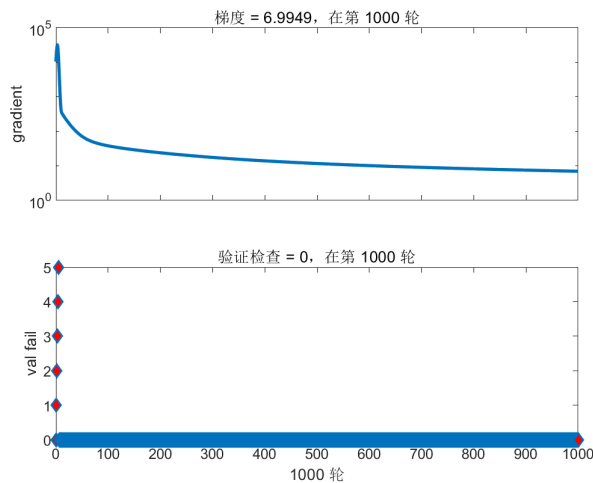
由损失曲线可见，该学习率下，三种数据的损失函数随迭代次数稳步下降，说明模型的拟合能力确实在稳步下降。而且三条曲线较为接近，说明模型具有较好的泛化能力，没有发生过拟合现象

回归曲线



分别在不同类型数据上画出拟合函数，发现与真实点吻合程度很好，逼近精确解 $Y=T$ ，进一步印证了模型的泛化能力

梯度和验证误差

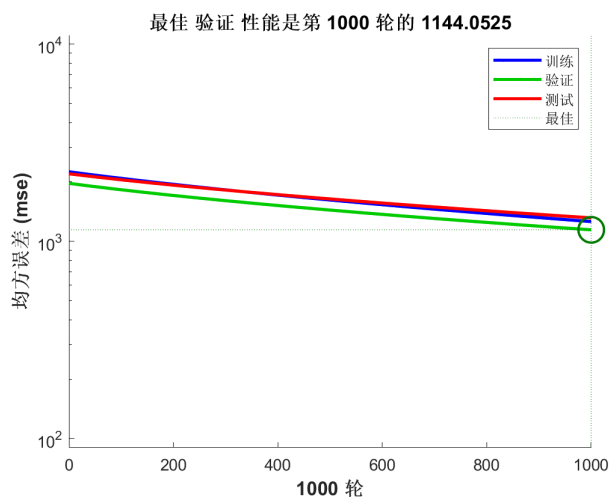


由梯度变化可见，模型梯度逐步下降，达到 10^0 数量级，相较 $5e-4$ 明显改善，没有发生爆炸现象。但是较原本数据标签 10^2 数量级仍然较大，还有进一步改善的空间，这可以通过增加迭代轮数实现

总而言之，学习率 $8e-5$ 使回归器在不同样本类型上的拟合效果均较好，且优化算法收敛性能也更强，没有发生爆炸现象。相较 $5e-4$ ， $8e-5$ 更加适用于该回归任务

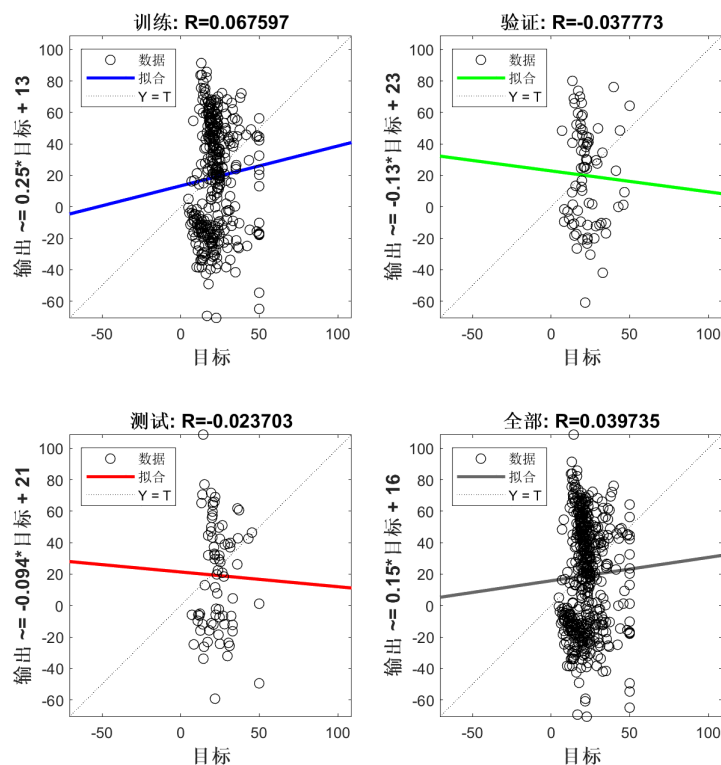
$$3.\eta = 1e - 7$$

损失曲线



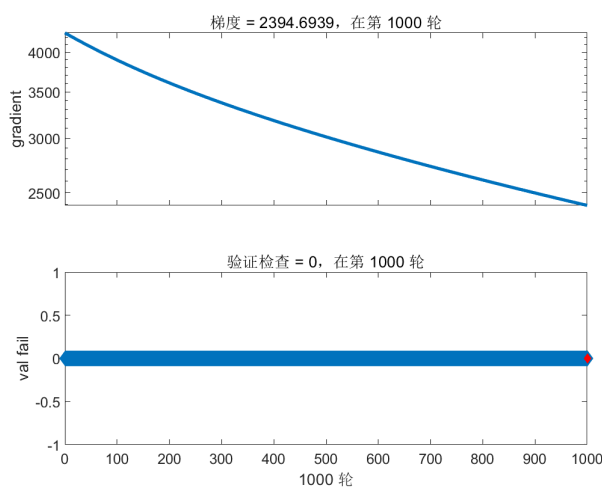
此学习率下误差下降缓慢且稳定，没有被早停法中止。但是均方误差仍然大于 10^3 ，几乎是 $8e-5$ 时的50倍，说明算法收敛过慢

回归曲线



拟合的函数明显偏离精确解 $Y=T$ ，且大多样本点均离拟合解较远，说明泛化能力不足，发生欠拟合

梯度和验证误差



相较 $5e-4$ ，梯度没有发生爆炸，并且稳步下降，说明确实有在逼近精确解。但是最终停留在2394.69，比真实数据还要大一个数量级，说明无法逼近最优，仍然处于振荡阶段

总而言之，与 $5e-4$ 相比， $1e-7$ 没有发生梯度爆炸，防止了数值溢出，但是相较 $8e-5$ 过于保守，模型收敛过慢，在1000轮内没有取得理想的泛化能力

综合以上三组实验结果可以总结：

1. 学习率 $5e-4$ 时：底层的优化过程不稳定，发生梯度爆炸现象，算法不能接近最优解。说明MSE作为损失函数较交叉熵而言可能需要更小的学习率来防止数值大小失控
2. 学习率 $8e-5$ 时：不同样本类型上的拟合效果均较好，且优化算法收敛性能也更强，没有发生爆炸现象。在三者中处于最优
3. 学习率 $1e-7$ 时：没有发生梯度爆炸，防止了数值溢出，但是相较 $8e-5$ 过于保守，模型收敛过慢，在1000轮内没有取得理想的泛化能力

结论

综合分类与回归问题实验结果分析可以总结以下结论:

1. 学习率选择对模型训练质量和效率影响显著: 学习率过大可能导致训练不收敛或过拟合, 学习率过小训练会过于缓慢, 导致算法短时间内无法收敛
2. 分类和回归任务对学习率的敏感度不同: 回归模型对学习率选择要求较严, 需要保证训练稳定 (不发生数值溢出) 且优化目标下降
3. 不同损失函数特性决定学习率范围上限: 回归任务中, MSE函数易导致梯度爆炸, 对应的学习率上限应低于交叉熵损失
4. 恰当学习率范围需针对具体任务和数据集进行调整: 各数据集特征、任务类别均能显著影响学习率的最优区间, 在我们的两个问题中, 二者的最优学习率相差大约在100倍左右
5. 系统学习率选择实验对理解其影响机制有很重要意义: 这对新任务的学习率初值选取至关重要
6. 未来工作可以尝试动态学习率和其他优化算法: 比如贝叶斯优化和网格搜索, 以提升训练效率和效果

总体来说, 本次实验深入探讨了学习率作为超参数在BP训练神经网络过程中的影响, 为类似问题提供了有价值的参考。在神经网络的训练中, 需要结合实例特点进行学习率的进一步调整

附录

1. Wisconsin Breast Cancer classifier

```
% This script assumes these variables are defined:
%
% wdbc_feature - input data.
% wdbc_bool - target data.

x = wdbc_feature';
t = wdbc_bool';

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'traingd'; % random gradient backpropagation.

% Create a Pattern Recognition Network
hiddenLayerSize = 30;
net = patternnet(hiddenLayerSize, trainFcn);

% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.input.processFcns = {'removeconstantrows', 'mapminmax'};

% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivision
net.divideFcn = 'dividerand'; % Divide data randomly
net.divideMode = 'sample'; % Divide up every sample
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

%here we set different learning rate
net.trainParam.lr=1;
```

```

% net.trainParam.lr=5e-3;
% net.trainParam.lr=1e-7;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'crossentropy'; % 交叉熵

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotconfusion', 'plotroc'};

% Train the Network
[net,tr] = train(net,x,t);

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)

% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotconfusion(t,y)
%figure, plotroc(t,y)

% Deployment
% Change the (false) values to (true) to enable the following code blocks.
% See the help for each generation function for more information.
if (false)
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
    genFunction(net,'myNeuralNetworkFunction');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a matrix-only MATLAB function for neural network code
    % generation with MATLAB Coder tools.
    genFunction(net,'myNeuralNetworkFunction','Matrixonly','yes');
    y = myNeuralNetworkFunction(x);
end

```

```

end
if (false)
    % Generate a Simulink diagram for simulation or deployment with.
    % Simulink Coder tools.
    gensim(net);
end

```

2. Boston Housing Regressor

```

% This script assumes these variables are defined:
%
%   BH_feature - input data.
%   BH_label - target data.

x = BH_feature';
t = BH_label';

% Choose a Training Function
% For a list of all training functions type: help nntrain
% 'trainlm' is usually fastest.
% 'trainbr' takes longer but may be better for challenging problems.
% 'trainscg' uses less memory. Suitable in low memory situations.
trainFcn = 'traingd'; % random gradient descent

% Create a Fitting Network
hiddenLayersize = 40;
net = fitnet(hiddenLayersize,trainFcn);

% Choose Input and Output Pre/Post-Processing Functions
% For a list of all processing functions type: help nnprocess
net.input.processFcns = {'removeconstantrows','mapminmax'};
net.output.processFcns = {'removeconstantrows','mapminmax'};

% Setup Division of Data for Training, Validation, Testing
% For a list of all data division functions type: help nndivision
net.divideFcn = 'dividerand'; % Divide data randomly
net.divideMode = 'sample'; % Divide up every sample
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

%here we set different learning rate
net.trainParam.lr=5e-4;
% net.trainParam.lr=8e-5;
% net.trainParam.lr=1e-7;

% Choose a Performance Function
% For a list of all performance functions type: help nnperformance
net.performFcn = 'mse'; % 均方误差

% Choose Plot Functions
% For a list of all plot functions type: help nnplot
net.plotFcns = {'plotperform','plottrainstate','ploterrhist', ...
    'plotregression', 'plotfit'};

% Train the Network
[net,tr] = train(net,x,t);

```



```

% Test the Network
y = net(x);
e = gsubtract(t,y);
performance = perform(net,t,y)

% Recalculate Training, Validation and Test Performance
trainTargets = t .* tr.trainMask{1};
valTargets = t .* tr.valMask{1};
testTargets = t .* tr.testMask{1};
trainPerformance = perform(net,trainTargets,y)
valPerformance = perform(net,valTargets,y)
testPerformance = perform(net,testTargets,y)


% View the Network
view(net)

% Plots
% Uncomment these lines to enable various plots.
%figure, plotperform(tr)
%figure, plottrainstate(tr)
%figure, ploterrhist(e)
%figure, plotregression(t,y)
%figure, plotfit(net,x,t)

% Deployment
% Change the (false) values to (true) to enable the following code blocks.
% See the help for each generation function for more information.
if (false)
    % Generate MATLAB function for neural network for application
    % deployment in MATLAB scripts or with MATLAB Compiler and Builder
    % tools, or simply to examine the calculations your trained neural
    % network performs.
    genFunction(net,'myNeuralNetworkFunction');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a matrix-only MATLAB function for neural network code
    % generation with MATLAB Coder tools.
    genFunction(net,'myNeuralNetworkFunction','Matrixonly','yes');
    y = myNeuralNetworkFunction(x);
end
if (false)
    % Generate a Simulink diagram for simulation or deployment with.
    % Simulink Coder tools.
    gensim(net);
end

```

参考文献

1. Wolberg,William, Mangasarian,Olvi, Street,Nick, and Street,W.. (1995). Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. <https://doi.org/10.24432/C5DW2B>. 
2. Harrison, D. and Rubinfeld, D.L. 'Hedonic prices and the demand for clean air', J. Environ. Economics & Management, vol.5, 81-102, 1978. 