机器学习第二次作业

统计 81　宋程　2184312719

目录

1 正则化

正则化是给平面不可约代数曲线以某种形式的全纯参数表示，是对最小化经验误差函数上加约束，这样的约束可以解释为先验知识(正则化参数等价于对参数引入先验分布)。约束有引导作用，在优化误差函数的时候倾向于选择满足约束的梯度减少的方向，使最终的解倾向于符合先验知识。[1] 应用于机器学习中即对损失函数引入正则项，减少模型复杂度，提高模型的泛化能力。常见的正则化方法有 L1 正则、L2 正则、L0 正则等。

2 正则化的求解

2.1 正则化的求解问题可以转换成两个步骤进行求解：

$\min_w f(w) + \lambda P(w)$

$\Rightarrow$

$\min_w f(w) + \gamma \| z - w \|_2^2$ ①

$\min_z \gamma \| z - w \|_2^2 + \lambda P(z)$ ②

求解②式 $\Rightarrow$ 求解①式 $\Rightarrow$ 梯度下降

由于每个 w_i 仅与每个 z_i 有关 且求导(梯度下降)时为对某个单独的 w_i, z_i 有关

不妨仅对某个 w_i, z_i 分析 故可记为 w, z 不妨设 $\gamma = 1$

2.2 对 L2、L1、L0 范数正则分别求解

2.2.1 L2 范数正则求解

1> L_2 范数正则求解

求解①式 $\min_z \| z - w \|_2^2 + \lambda |z|_{L_2}$ 即 $\min_z F(z)$ $F(z) = (z - w)^2 + \lambda z^2$

$\frac{\partial F}{\partial z} = 2(z - w) + 2\lambda z = 0$

$\Rightarrow z = \frac{w}{1+\lambda}$ 代入①式

求解②式 $\min_w f(w) + \| z - w \|_2^2 \Rightarrow \min_w f(w) + \left(\frac{\lambda}{1+\lambda}\right)^2 \| w \|_2^2$

梯度下降

2.2.2 L1 范数正则求解

2) L_1 范数正则求解

次微分　左、右导数为端点的区间（此直线满足　经过该点，在一邻域内与函数重合或在函数下方　直线斜率集合）

次梯度　0点次梯度 $[w-z, w+z]$

左右导数

辅助域 $\min_z \frac{1}{2}\|z-w\|_2^2 + \lambda|z|_1$ 为 $\min_z F(z)$　$F(z) = \frac{1}{2}\|z-w\|_2^2 + \lambda|z|$

(1) $z<0$　$F'(z) = f_1(z) = z - w - \lambda$

(2) $z>0$　$F'(z) = f_2(z) = z - w + \lambda$

考虑 $F(z)$，$f_1(z)$，$f_2(z)$ 在0点处性质

F在零点处左、右导数不等　故可分为3种情况

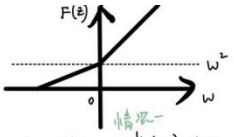
情况一　情况二　情况三

情况一：$f_1(z)>0$　$f_2(z)>0$　当 z 在0的某邻域内

$\forall w < -\lambda$

且 $f_1(z)$，$f_2(z)$ 均为线性　$\therefore$ 故 $f_1(z)=0$　$\exists z_0=0$　$\forall z=0$　$f_2(z)\neq 0$

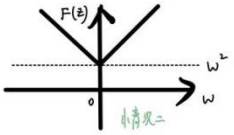
$\therefore z = w+\lambda$

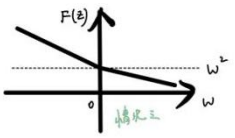


情况二，由图可知　零点为 $F(z)$ 极小点　且由 $f_1(z)$，$f_2(z)$ 形式 $\therefore$

$f_1(z)<0$　$f_2(z)>0$　0为 $F(z)$ 全局极小点

情况三同理　$z_{min} = \begin{cases} w-\lambda & w>\lambda \\ 0 & -\lambda \le w \le \lambda \\ w+\lambda & w<-\lambda \end{cases}$





辅助① $\min_w f(w) + \|z-w\|_2^2$

$\|z-w\|_2^2 = \begin{cases} \lambda^2 & |w|>\lambda \\ \|w\|^2 & -\lambda \le w \le \lambda \end{cases}$

$\min_w f(w) + \|w\|^2$　当 $-\lambda \le w \le \lambda$

$\min_w f(w)$　当 $|w|>\lambda$

2.2.3 L0 范数求解

3) L_0 范数正则化

辅助域 $\min_z \|w-z\| + \lambda|z|_0$

$z^* = \begin{cases} w & |w|>\lambda \\ 0 & |w|\le\lambda \end{cases}$

辅助1式 $\min_w \begin{cases} f(w) & |w|>\lambda \\ f(w)+\|w\|^2 & |w|\le\lambda \end{cases}$

2.3 求解分析

通过上述求解过程可以发现：

（1）L2 范数正则的求解过程也可以简化为直接使用 2.1 中①式，针对该式进行梯度下降，从而逼近极优解。通过与未加正则项的的损失函数进行比较，我们可以很直观的发现，L2 正则通过压缩神经网络或模型的权重系数的大小来减小模型的复杂度，获得更加的泛化性能。

（2）L1 范数正则与 L0 范数正则求解结果是类似的，其设置了一个阈值，当模型的权重系数的绝对值大于阈值时，对该稀疏的优化过程与未加正则项相同。当该系数绝对值小于阈值时，其通过调节另一个参数，来控制该系数的大小，从直观上来看，作用是对于较大的系数不进行处理，对较小的参数，令其尽可能趋向于 0，因此，这两种正则化起到稀疏的作用。

3 实验

由于当样本量较少时，机器学习模型易发生过拟合。我们采用一个含有一个隐藏层的神经网络模型，进行如下两个实验，通过对拟合图形和损失函数大小的分析来判断正则化的影响。我们数据通过对一个多项式函数生成，并随机引入噪声。除此之外，本文还对 MNIST 识别进行了正则化的分析。

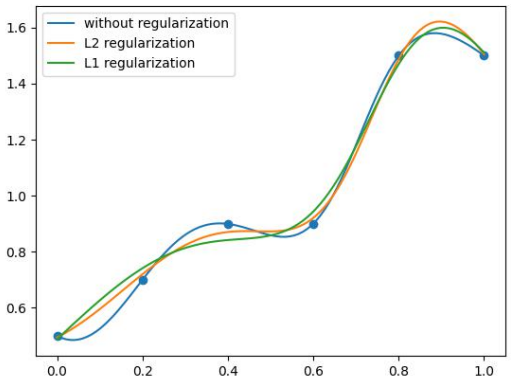
3.1 少量样本点拟合图形

我们通过将[0，1]区间等分隔得到六个 x 的数据，并通过 y=x+0.5 得到数据 y，然后在数据 y 上引入大小为[-0.2，0.2]的噪声，得到如下数据：

X=[0.　0.2　0.4　0.6　0.8　1.]

Y=[0.5　0.7　0.9　0.9　1.5　1.5]

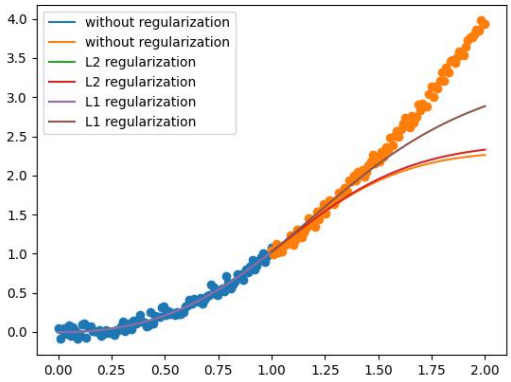
并通过设置不同大小的正则化参数得到如下结果：



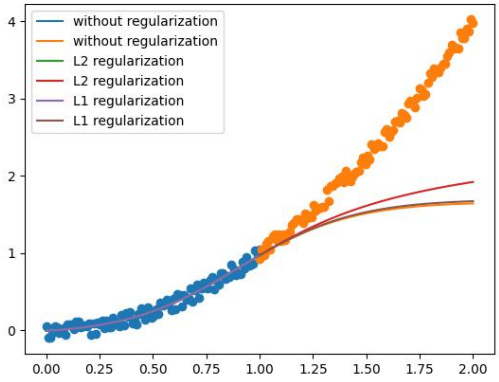
通过上图，我们发现，无论是 L2 还是 L1 正则都使得拟合曲线更加平滑，在一定程度上减少了过拟合。

3.2 分析误差

我们通过将[0，2]区间等分隔得到 200 个 x 的数据，并通过一个多项式得到数据 y，然后在数据 y 上引入大小为[-0.1，0.1]的噪声，得到训练、测试数据.



无正则 cost：[55.33674536]
L2 正则 cost：[50.58699689]
L1 正则 cost：[19.02673174]



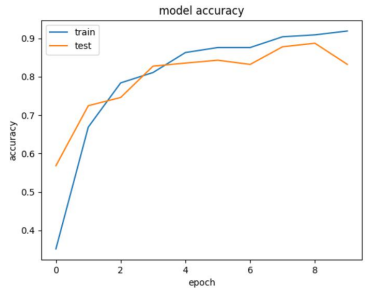
无正则 cost：[129.0885807]
L2 正则 cost：[98.89835335]
L1 正则 cost：[125.04846214]

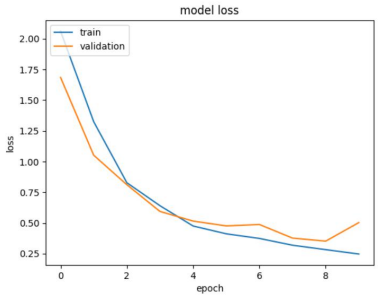
由于神经网络迭代次数较多，因此我们不采取均值加方差的形式，而仅仅以部分实验结果做出分析。

我们可以发现在实验中，两种正则方法都起到了较好的泛化效果。

3.3 MINIST 识别及正则化作用

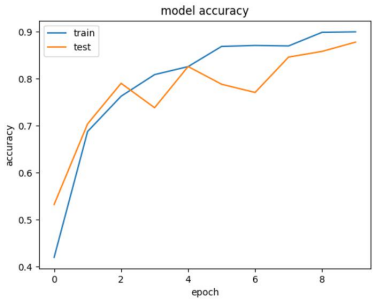
本文采取了 Le-Net5 的网络结构对 MNIST 进行分类，为了在未引入正则的情况达到过拟合的效果，仅采用了 1000 个训练图片进行训练，并在最后的全连接层进行无正则、L2 正则、L1 正则得到如下结果。

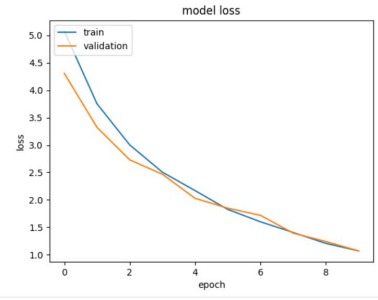




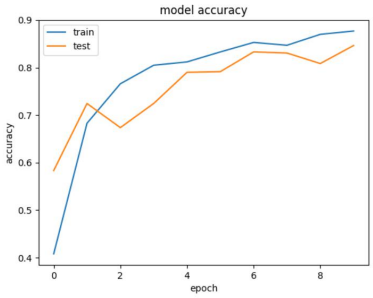
无正则化

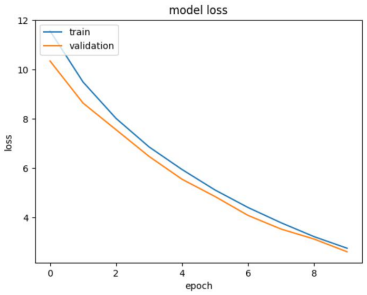
Test loss：0.5034097　　Test accuracy：0.8322





Test loss:1.0689405　　Test accuracy0.8782





L1 正则

Test loss:2.6031404　　Test accuracy：0.8465

通过上述实验我们可以发现，在后几轮的训练中，不加入正则化的训练精度出现了训练精度提升，而测试精度下降的情况，可以猜测发生了一定的过拟合，而引入正则项后有效的改善了过拟合的现象。

4 分析

通过上述分析以及实验，我们可以得到如下结论：

1. 正则化项确实能起到提升模型泛化性能的效果，尽管有时候这些方法是以放弃一定的训练精确度得到的，但加入正则项更适合于应用于大多数现实场景，因为我们现实中的数据往往应各种各样的原因，会引入部分无法预测的量，

即噪声。

2. L2 的正则化方法与 L1 正则有较大的区别，一个是通过压缩权重矩阵的元素来完成泛化能力的提升，一个是通过稀疏的方法来得到泛化能力的提升。两者各有优缺点：

（1）L1 正则化方法稀疏了权重矩阵，更有利于后续对模型的解释，更有利于提取有用的特征，放弃无用的特征，从而起到减少数据收集成本的作用，但其在训练过程中的计算量较大。

（2）L2 正则化方法虽然没有 L1 正则的稀疏作用，但是其求解及训练过程都更加简单，训练过程中的逻辑语句更少，因此能减少训练的计算。

3.稀疏化的作用不仅体现在通过减少权重大小来简单化复杂的模型，对于神经网络而言，当某一个权重为 0 时，就可视为没有连接，因此稀疏化也是在简单化模型的结构。通过求解过程，我们发现 L0 与 L1 都起到了稀疏的作用，我们可以很直观的感受到 L0 的稀疏作用，而通过求解，我们发现 L0 与 L1 正则的求解结果是类似的，所以 L1 正则的稀疏作用也就不难理解了。

5. 参考文献

[1] 正则化百度百科

[2] 周志华 机器学习[D] 北京：清华大学出版社 2016.

[3] 李航 统计学习方法[D] 北京： 清华大学出版社 2012.

[4] Evgeniou T， Pontil M， Poggio T . Regularization Networks and Support Vector Machines[J]. Advances in Computational Mathematics, 2000, 13(1):1-50.

[5] Girosi F， Jones M， Poggio T . Regularization Theory and Neural Networks Architectures[J]. Neural Comp, 1995, 7(2):219-269.

[6] https://www.cnblogs.com/jianxinzhou/p/4083921.html

6. 代码

3.1

```
import numpy as np

import copy

import matplotlib

from array import array

import random

import matplotlib.pyplot as plt



import pandas as pa

from pandas import read_csv

#使用 Pandas 导入 csv 数据


#w1=np.random.rand(2,4)

#print(w1)

def sigmoid(z):

    return 1.0/(1.0+np.exp(-z))
```

```
def sigmoid_prime(z):
    return sigmoid(z) * (1-sigmoid(z))



#print(len(layers_dim))调试
def initial_parameter(layers_dim):
    parameters = {}
    for i in range(1, len(layers_dim)):
        #print(i)#调试
        parameters["w"+str(i)] = np.random.rand(layers_dim[i], layers_dim[i-1])
        parameters["b"+str(i)] = np.random.rand(layers_dim[i], 1)
    return parameters
    #print(len(layer_dim))


def forward(datax,parameters):
    a=[]
    z=[]
    a.append(datax)
    z.append(datax)
    layers=len(parameters) // 2
    caches={}
    for i in range(1,layers):
        z_temp = parameters["w" + str(i)].dot(a[i-1]) + parameters["b" + str(i)]
        z.append(z_temp)
        a.append(sigmoid(z_temp))
        # 最后一层不用sigmoid
    z_temp = parameters["w" + str(layers)].dot(a[layers - 1]) + parameters["b" + str(layers)]
    z.append(z_temp)
    a.append(z_temp)

    caches["z"] = z
    caches["a"] = a

    return caches, a[layers]

def backpr(parameters,caches,output, datay):

    l=len(parameters) // 2
    l=l+1

    m = datay.shape[1]
    gradient={}
    error={}
    L1=copy.deepcopy(parameters)
```

```
    for j in range(0,len(parameters["w"+str(l-1)])):
        if  np.sum(parameters["w"+str(l-1)][j])>lamda1 or np.sum(parameters["w"+str(l-1)][j])<lamda1:
            L1["w"+str(l-1)][j]=0
        if np.sum(parameters["b" + str(l - 1)][j]) > lamda1 or np.sum(parameters["b" + str(l - 1)][j]) < lamda1:
            L1["b" + str(l - 1)][j] = 0
    error[str(l-1)]=(output-datay)*sigmoid_prime(caches["z"][l-1])
    gradient["dw"+str(l-1)]=error[str(l-1)].dot(caches["a"][l-2].T)/m+2*lamda*L1["w"+str(l-1)]/m

    gradient["db"+str(l-1)]=error[str(l-1)].sum(axis=1,keepdims = True)/m+2*lamda*L1["b"+str(l-1)]/m



    #gradient["dz"+str(l-1)]= output - datay
    #gradient["dw"+str(l-1)]=np.dot
    #(gradient["dz"+str(l-1)]*sigmoid_prime(caches["a"][l-1]), caches["a"][l-2].T)/m
    #gradient["db"+str(l-1)]=gradient["dz"+str(l-1)].dont(sigmoid_prime(output))
    for i in reversed(range(1,l-1)):
        for j in range(0, len(parameters["w" + str(i)])):
            if np.sum(parameters["w" + str(i)][j]) > lamda1 or np.sum(parameters["w" + str(i)][j]) < lamda1:
                L1["w" + str(i)][j] = 0
            if np.sum(parameters["b" + str(i)][j]) > lamda1 or np.sum(parameters["b" + str(i)][j]) < lamda1:
                L1["b" + str(i)][j] = 0
        error[str(i)]=parameters["w"+str(i+1)].T.dot(error[str(i+1)])*sigmoid_prime(caches["z"][i])
        gradient["dw"+str(i)]=error[str(i)].dot(caches["z"][i-1].T)/m+2*lamda*L1['w'+str(i)]/m
        gradient["db"+str(i)]=np.sum(error[str(i)],axis = 1,keepdims = True) /m++2*lamda*L1['b'+str(i)]/m







        #gradient["dz"+str(i)]=parameters["w"+str(i+1)].T.dot(gradient["dz"+str(i+1)])* sigmoid_prime(caches["z"][i])
        #gradient["dw"+str(i)]=np.dot( gradient["dz"+str(i)],caches["a"][i-1].T)
        #gradient["db"+str(i)] = gradient["dz"+str(i)]
    return gradient


def  update_gradient(parameters,gradient,learning_rate):
        layers = len(parameters) // 2
        for i in range(1,layers+1):
            parameters["w"+str(i)] -= learning_rate * gradient["dw"+str(i)]
            parameters["b"+str(i)] -= learning_rate * gradient["db"+str(i)]
        return parameters



def polynominal(x):
    return  (x-0.2)*(x-0.1)*(x-0.3)*(x-10)*(x+2)*(x-0.65)*(x-0.08)
```

```
def load_data():
    """
    加载数据集
    """
    '''
    datax = np.linspace(0, 1, 10)
    datay = ((datax**2-1)**3+0.5)*np.sin(datax*2)
    for i in range(len(datay)):
        z=np.random.randint(low=-10,high=10)/100
        datay[i]=datay[i]+z
    '''
    datax = np.linspace(0, 1, 6)
    datay=np.linspace(0, 1, 6)
    for i in range(len(datax)):

        datay[i]=datax[i]+0.5

    datay[3]=datay[3]-0.2
    datay[4]=datay[4]+0.2
    #datax = np.arange(0.0,1.0,0.01)
    #datay = polynominal(datax)
    # 数据可视化
    plt.scatter(datax,datay)
    return datax,datay
#进行测试
datax, datay = load_data()
n=6
datax = datax.reshape(1,n)
datay = datay.reshape(1,n)

layers_dim=np.array([1, 10,1])
print(datax)
print(datay)
#plt.scatter(datax,datay)
parameters = initial_parameter([1,10,1])
output = 0
lamda=0
lamda1=0

for i in range(100000):
    caches,output = forward(datax, parameters)
```

```
    gradient = backpr(parameters, caches,output, datay)
    parameters = update_gradient(parameters, gradient, learning_rate= 5)
'''
for i in range(50000):
    caches,output = forward(datax, parameters)

    gradient = backpr(parameters, caches,output, datay)
    parameters = update_gradient(parameters, gradient, learning_rate= 3)

for i in range(50000):
    caches,output = forward(datax, parameters)

    gradient = backpr(parameters, caches,output, datay)
    parameters = update_gradient(parameters, gradient, learning_rate= 1)
'''
for i in range(50000):
    caches, output = forward(datax, parameters)

    gradient = backpr(parameters, caches, output, datay)
    parameters = update_gradient(parameters, gradient, learning_rate=0.3)

'''
output=output.reshape(n)
datax=datax.reshape(n)
plt.plot(datax,output)
'''
#plt.scatter(datax,output)
plotx=np.linspace(0, 1, 100)
plotx=plotx.reshape(1,100)
caches,output=forward(plotx,parameters)


plotx=plotx.reshape(100)
output=output.reshape(100)

plt.plot(plotx,output,label="without regularization")


parameters1 = initial_parameter([1,10,1])
output1 = 0
lamda=1000
lamda1=10000

for i in range(100000):
```

```
    caches,output1 = forward(datax, parameters1)

    gradient = backpr(parameters1, caches,output1, datay)
    parameters1 = update_gradient(parameters1, gradient, learning_rate= 5)



for i in range(50000):
    caches, output1 = forward(datax, parameters1)

    gradient = backpr(parameters1, caches, output1, datay)
    parameters1 = update_gradient(parameters1, gradient, learning_rate=0.3)



#plt.scatter(datax,output)
plotx=np.linspace(0, 1, 100)
plotx=plotx.reshape(1,100)
caches,output1=forward(plotx,parameters1)



plotx=plotx.reshape(100)
output1=output1.reshape(100)

plt.plot(plotx,output1,label="L2 regularization")



parameters2 = initial_parameter([1,10,1])
output = 0
lamda=200
lamda1=1

for i in range(100000):
    caches,output2 = forward(datax, parameters2)

    gradient = backpr(parameters2, caches,output2, datay)
    parameters2 = update_gradient(parameters2, gradient, learning_rate= 5)

for i in range(50000):
    caches, output2 = forward(datax, parameters2)

    gradient = backpr(parameters2, caches, output2, datay)
    parameters2 = update_gradient(parameters2, gradient, learning_rate=0.3)
```

```
#plt.scatter(datax,output)
plotx=np.linspace(0, 1, 100)
plotx=plotx.reshape(1,100)
caches,output2=forward(plotx,parameters2)



plotx=plotx.reshape(100)
output2=output2.reshape(100)

plt.plot(plotx,output2,label="L1 regularization")

plt.legend()
plt.show()
```

3.2

```
import numpy as np
import copy
import matplotlib
from array import array
import random
import matplotlib.pyplot as plt



import pandas as pa
from pandas import read_csv
#使用 Pandas 导入 csv 数据

#w1=np.random.rand(2,4)
#print(w1)
def sigmoid(z):
    return 1.0/(1.0+np.exp(-z))

def sigmoid_prime(z):
    return sigmoid(z) * (1-sigmoid(z))



#print(len(layers_dim))调试
def initial_parameter(layers_dim):
    parameters = {}
    for i in range(1, len(layers_dim)):
        #print(i)#调试
        parameters["w"+str(i)] = np.random.rand(layers_dim[i], layers_dim[i-1])
        parameters["b"+str(i)] = np.random.rand(layers_dim[i], 1)
    return parameters
    #print(len(layer_dim))
```

```
def forward(datax,parameters):
    a=[]
    z=[]
    a.append(datax)
    z.append(datax)
    layers=len(parameters) // 2
    caches={}
    for i in range(1,layers):
        z_temp = parameters["w" + str(i)].dot(a[i-1]) + parameters["b" + str(i)]
        z.append(z_temp)
        a.append(sigmoid(z_temp))
        # 最后一层不用sigmoid
    z_temp = parameters["w" + str(layers)].dot(a[layers - 1]) + parameters["b" + str(layers)]
    z.append(z_temp)
    a.append(z_temp)

    caches["z"] = z
    caches["a"] = a

    return caches, a[layers]

def backpr(parameters,caches,output, datay):

    l=len(parameters) // 2
    l=l+1

    m = datay.shape[1]
    gradient={}
    error={}
    L1=copy.deepcopy(parameters)
    for j in range(0,len(parameters["w"+str(l-1)])):
        if  np.sum(parameters["w"+str(l-1)][j])>lamda1 or np.sum(parameters["w"+str(l-1)][j])<lamda1:
            L1["w"+str(l-1)][j]=0
        if np.sum(parameters["b" + str(l - 1)][j]) > lamda1 or np.sum(parameters["b" + str(l - 1)][j]) < lamda1:
            L1["b" + str(l - 1)][j] = 0
    error[str(l-1)]=(output-datay)*sigmoid_prime(caches["z"][l-1])
    gradient["dw"+str(l-1)]=error[str(l-1)].dot(caches["a"][l-2].T)/m+2*lamda*L1["w"+str(l-1)]/m

    gradient["db"+str(l-1)]=error[str(l-1)].sum(axis=1,keepdims = True)/m+2*lamda*L1["b"+str(l-1)]/m


    #gradient["dz"+str(l-1)]= output - datay
    #gradient["dw"+str(l-1)]=np.dot
```

```
    #(gradient["dz"+str(l-1)]*sigmoid_prime(caches["a"][l-1]), caches["a"][l-2].T)/m
    #gradient["db"+str(l-1)]=gradient["dz"+str(l-1)].dont(sigmoid_prime(output))
    for i in reversed(range(1,l-1)):
        for j in range(0, len(parameters["w" + str(i)])):
            if np.sum(parameters["w" + str(i)][j]) > lamda1 or np.sum(parameters["w" + str(i)][j]) < lamda1:
                L1["w" + str(i)][j] = 0
            if np.sum(parameters["b" + str(i)][j]) > lamda1 or np.sum(parameters["b" + str(i)][j]) < lamda1:
                L1["b" + str(i)][j] = 0
        error[str(i)]=parameters["w"+str(i+1)].T.dot(error[str(i+1)])*sigmoid_prime(caches["z"][i])
        gradient["dw"+str(i)]=error[str(i)].dot(caches["z"][i-1].T)/m+2*lamda*L1['w'+str(i)]/m
        gradient["db"+str(i)]=np.sum(error[str(i)],axis = 1,keepdims = True) /m++2*lamda*L1['b'+str(i)]/m




        #gradient["dz"+str(i)]=parameters["w"+str(i+1)].T.dot(gradient["dz"+str(i+1)])* sigmoid_prime(caches["z"][i])
        #gradient["dw"+str(i)]=np.dot( gradient["dz"+str(i)],caches["a"][i-1].T)
        #gradient["db"+str(i)] = gradient["dz"+str(i)]
    return gradient


def  update_gradient(parameters,gradient,learning_rate):
         layers = len(parameters) // 2
         for i in range(1,layers+1):
             parameters["w"+str(i)] -= learning_rate * gradient["dw"+str(i)]
             parameters["b"+str(i)] -= learning_rate * gradient["db"+str(i)]
         return parameters



def polynominal(x):
    return  (x-0.2)*(x-0.1)*(x-0.3)*(x-10)*(x+2)*(x-0.65)*(x-0.08)



def load_data():
    """
    加载数据集
    """

    datax = np.linspace(0, 1, 100)
    datay = datax*datax
    for i in range(len(datay)):
        z=np.random.randint(low=-10,high=10)/100
        datay[i]=datay[i]+z
```

```
    plt.scatter(datax,datay)
    return datax,datay

def sse(testy,output):
    return sum((testy-output).dot((testy-output).T))

def load_data1():
    """
    加载数据集
    """

    datax = np.linspace(1, 2, 100)
    datay = datax*datax
    for i in range(len(datay)):
        z = np.random.randint(low=-10, high=10) / 100
        datay[i] = datay[i] + z

    plt.scatter(datax, datay)
    return datax, datay
#进行测试
datax, datay = load_data()
testx,testy=load_data1()
n=100
datax = datax.reshape(1,n)
datay = datay.reshape(1,n)
testx=testx.reshape(1,n)
testy=testy.reshape(1,n)
layers_dim=np.array([1, 10,1])
#print(datax)
#print(datay)
#plt.scatter(datax,datay)
parameters = initial_parameter([1,10,1])
output = 0
lamda=0
lamda1=0

for i in range(100000):
    caches,output = forward(datax, parameters)

    gradient = backpr(parameters, caches,output, datay)
    parameters = update_gradient(parameters, gradient, learning_rate= 5)

'''
```

```
for i in range(50000):
    caches,output = forward(datax, parameters)

    gradient = backpr(parameters, caches,output, datay)
    parameters = update_gradient(parameters, gradient, learning_rate= 3)

for i in range(50000):
    caches,output = forward(datax, parameters)

    gradient = backpr(parameters, caches,output, datay)
    parameters = update_gradient(parameters, gradient, learning_rate= 1)
'''
for i in range(50000):
    caches, output = forward(datax, parameters)

    gradient = backpr(parameters, caches, output, datay)
    parameters = update_gradient(parameters, gradient, learning_rate=0.3)

'''
output=output.reshape(n)
datax=datax.reshape(n)
plt.plot(datax,output)
'''
#plt.scatter(datax,output)
plotx=np.linspace(0, 1, 100)
plotx=plotx.reshape(1,100)
caches,output=forward(plotx,parameters)


plotx=plotx.reshape(100)
output=output.reshape(100)

plt.plot(plotx,output,label="without regularization")
caches,output=forward(testx,parameters)
cost1=sse(testy,output)
testx1=testx.reshape(100)
output=output.reshape(100)
plt.plot(testx1,output,label="without regularization")
print("cost1")
print(cost1)

parameters1 = initial_parameter([1,10,1])
output1 = 0
lamda=1000
```

```
lamda1=100000


for i in range(100000):
    caches,output1 = forward(datax, parameters1)


    gradient = backpr(parameters1, caches,output1, datay)
    parameters1 = update_gradient(parameters1, gradient, learning_rate= 5)




for i in range(50000):
    caches, output1 = forward(datax, parameters1)


    gradient = backpr(parameters1, caches, output1, datay)
    parameters1 = update_gradient(parameters1, gradient, learning_rate=0.3)



#plt.scatter(datax,output)
plotx=np.linspace(0, 1, 100)
plotx=plotx.reshape(1,100)
caches,output1=forward(plotx,parameters1)



plotx=plotx.reshape(100)
output1=output1.reshape(100)

plt.plot(plotx,output1,label="L2 regularization")
caches,output1=forward(testx,parameters1)
cost2=sse(testy,output1)
print("cost2")
print(cost2)
output1=output1.reshape(100)
plt.plot(testx1,output1,label="L2 regularization")
parameters2 = initial_parameter([1,10,1])
output = 0
lamda=200
lamda1=1

for i in range(100000):
    caches,output2 = forward(datax, parameters2)


    gradient = backpr(parameters2, caches,output2, datay)
    parameters2 = update_gradient(parameters2, gradient, learning_rate= 5)
```

```
for i in range(50000):
    caches, output2 = forward(datax, parameters2)

    gradient = backpr(parameters2, caches, output2, datay)
    parameters2 = update_gradient(parameters2, gradient, learning_rate=0.3)



#plt.scatter(datax,output)
plotx=np.linspace(0, 1, 100)
plotx=plotx.reshape(1,100)
caches,output2=forward(plotx,parameters2)



plotx=plotx.reshape(100)
output2=output2.reshape(100)

plt.plot(plotx,output2,label="L1 regularization")
caches,output2=forward(testx,parameters2)
cost3=sse(testy,output2)
print("cost3")
print(cost3)



output2=output2.reshape(100)
plt.plot(testx1,output2,label="L1 regularization")

plt.legend()
plt.show()
```

3.3

```
import keras as keras
from keras import layers
from keras import models

from keras import regularizers
from keras.datasets import mnist
from keras.utils import to_categorical
import tensorflow as tf
import numpy
import matplotlib.pyplot as plt

'''model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))'''

model1 = models.Sequential()
model1.add(layers.Conv2D(6, (5, 5), activation='relu', input_shape=(28, 28, 1)))
model1.add(layers.MaxPooling2D((2, 2)))
model1.add(layers.Conv2D(16, (5, 5), activation='relu'))
model1.add(layers.MaxPooling2D((2, 2)))

model1.add(layers.Flatten())
model1.add(layers.Dense(84, activation='relu'))
model1.add(layers.Dense(10, activation='softmax',kernel_regularizer=regularizers.l2(0.2)))
#,kernel_regularizer=regularizers.l1(0.1)
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images=train_images[0:1000,:,:]
train_labels=train_labels[0:1000]



train_images = train_images.reshape((1000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
import time


class TimeHistory(keras.callbacks.Callback):
    def on_train_begin(self, logs={}):
        self.times = []
        self.totaltime = time.time()

    def on_train_end(self, logs={}):
        self.totaltime = time.time() - self.totaltime

    def on_epoch_begin(self, batch, logs={}):
        self.epoch_time_start = time.time()
```

```
    def on_epoch_end(self, batch, logs={}):
        self.times.append(time.time() - self.epoch_time_start)



time_callback = TimeHistory()
model1.compile(optimizer='rmsprop',
               loss="categorical_crossentropy",
               metrics=['accuracy'])
history1 =model1.fit(train_images, train_labels, epochs=10, batch_size=64,
validation_data=(test_images,test_labels),callbacks=[time_callback])
test_loss, test_acc =model1.test_on_batch(test_images, test_labels)
print(time_callback.times)
print(time_callback.totaltime)
print(test_loss)
print(test_acc)
model1.test_on_batch(test_images, test_labels)


print(history1.history.keys())
#  "Accuracy"
plt.plot(history1.history['acc'])
plt.plot(history1.history['val_acc'])


plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# "Loss"
plt.plot(history1.history['loss'])
plt.plot(history1.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```