哈尔滨工业大学计算机科学与技术学院

实验报告

课程名称： 机器学习

课程类型：选修

实验题目： 多项式拟合正弦函数

学号：1160300905

姓名：吴逸飞

# 实验目的

* 掌握最小二乘法求解（无惩罚项的损失函数）
* 掌握加惩罚项（2-范数）的损失函数优化
* 掌握梯度下降法优化
* 掌握共轭梯度法优化
* 理解过拟合、克服过拟合的方法（如加惩罚项、增加样本）

# 二、实验要求及实验环境

## 实验要求：

1. 生成数据，加入噪声。
2. 用高阶多项式函数拟合曲线。
3. 用解析解求解两种loss的最优解（无正则项和有正则项）。
4. 优化方法求解最优解（梯度下降、共轭梯度）。
5. 用得到的实验数据，解释过拟合。
6. 用不同数据量，不同超参数，不同的多项式阶数，比较实验结果。
7. 语言不限，可以用matlab、python。求解解析解时可以利用现成的矩阵求逆。梯度下降，共轭梯度要求自己求梯度，迭代优化自己写。不得使用现成的平台，例如pytorch，tensorflow的自动微分工具。

## 实验环境：

Ubuntu 18.04 x64，Python 3.6.5。

# 三、设计思想

## 1．算法原理

### 求解析解法[1]

设训练数据中所有自变量构成的向量对应的Vandermonde矩阵为，所有对应因变量构成的向量为，回归多项式各项系数构成向量，则其最小二乘回归的目标即最小化。

为了抑制过拟合，加入正则项，并为该项添加正则化系数，即正则项形如，当时表示不作正则化。

综上所述，损失函数。

为了消除训练数据量对的影响，将对训练数据做平均，即的最终形式为。

该函数是关于的二次函数，求导并令导数为可解得该函数的最小值点，即。

### 梯度下降法[2]

梯度下降法是在已知一个可行的多项式系数向量的条件下，根据该点上损失函数的梯度寻求更优解的算法。由于梯度方向为在该点增加最快的方向，因此梯度的反方向为在该点下降最快的方向。在训练过程中只需每次将向梯度反方向移动，即可降低值。

设当前参数上求得函数梯度为，为了控制学习速率，设一超参数学习率，并定义为训练步向量。训练过程为：首先任取一可行的，之后求该点梯度，并根据训练步向量更新，即。重复这一训练过程直至一定步数或梯度足够小即可停止。

随机梯度下降法相比于梯度下降法，仅在计算梯度时不计算在整个训练集上的梯度，而是从训练集中随机选出一个batch，并计算batch中数据的梯度，由此可以加快速度。

### 共轭梯度法[3]

共轭梯度法本用于求线性方程组之解，在本实验中可用于求梯度为的解。该方法使用若干线性无关且共轭正交的向量张成参数空间，并在每个向量确定的方向上寻找使取最小值的分量大小。

该方法中求关于（加入正则化）共轭正交的基向量的过程可使用G-S正交化算法完成，求分量大小则可通过向量运算与二次函数性质直接完成，且两过程可以合并，迭代执行直至所有分量计算完毕，得出答案。

## 算法的实现

### 求解析解法

按照1.1节中的公式TODO直接计算解析解即可。

### 梯度下降法

首先将待训练参数初始化为，之后开始训练过程。

在每一步训练中，从训练数据中可重复地随机抽取batch\_size个训练样本（当batch\_size为时，将整个训练集作为训练样本，即不进行随机化），针对抽取的训练样本计算的梯度，并求出梯度的2-范数。若小于给定的，则训练结束，否则将减去，并进行下一个训练步。

### 共轭梯度法

首先将待训练参数初始化为，并计算CG算法中需要用到的参数：

同时计算初始误差，选定初始步进方向。之后开始步的训练过程。

在每一步训练中，按顺序计算下列表达式，对相应变量进行赋值，从而计算出每一步的步长、步进方向及新的误差等用于下一步的参数。

# 实验结果与分析

以下各次实验中所使用的数据均使用在上独立生成，且加入了均值为，方差为的高斯噪声。

## 求解析解法

### 无正则项

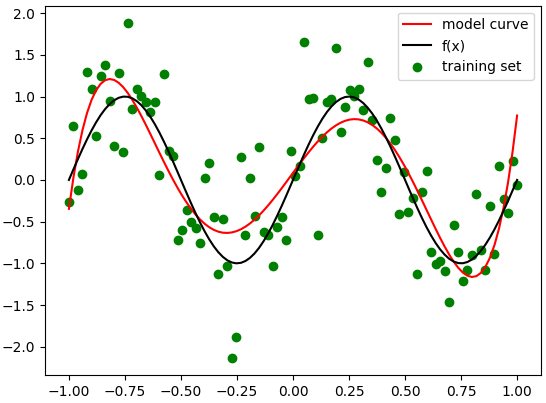


图1.1.1 训练集包含个点，多项式阶数，有明显欠拟合。

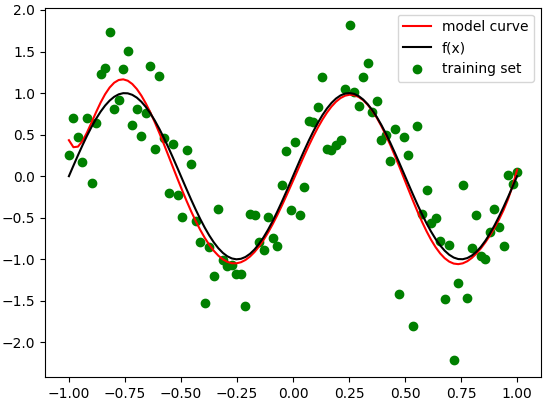


图1.1.2 训练集包含个点，多项式阶数，拟合效果较好。

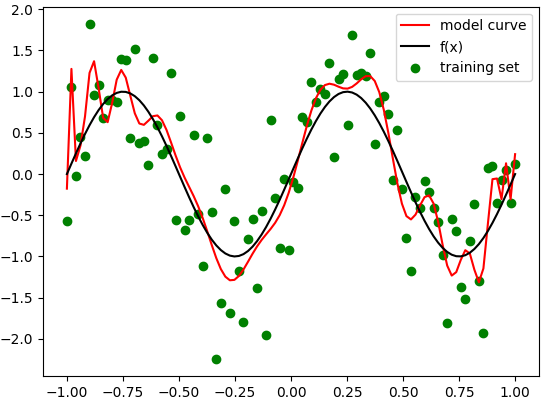


图1.1.3 训练集包含个点，多项式阶数，有明显过拟合。

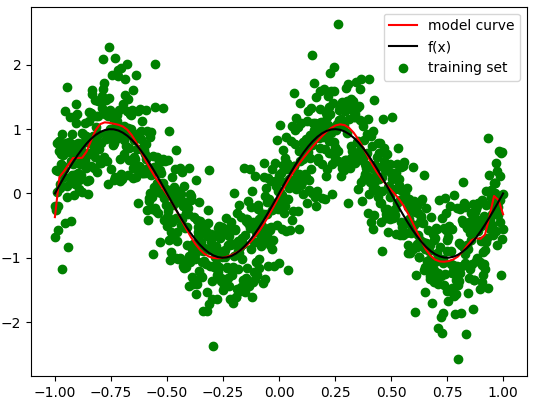


图1.1.4 训练集包含个点，多项式阶数，过拟合现象不严重。

可以发现，使用适当的模型大小（图1.1.2）能够在训练数据量不大的条件下得到较好的拟合结果。而当模型过小（图1.1.1）时，由于模型表达能力不够，训练结果将欠拟合；当模型过大（图1.1.3）时，由于模型表达能力过强，容易学得数据噪声的偶然性规律，训练极易发生过拟合。同时，在模型过大的场合，增大训练数据量（图1.1.4）能够一定程度地抑制过拟合现象。

然而并非任何时候都能获得大量的训练数据，而合适的模型大小也并不容易估计，这就要求一种实用的控制复杂模型过拟合程度的方法，即正则化方法。

### 有正则项

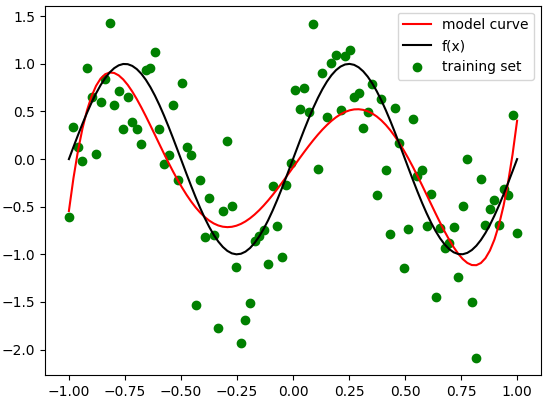


图1.2.1 训练集包含个点，多项式阶数，正则化系数，训练结果与图1接近。

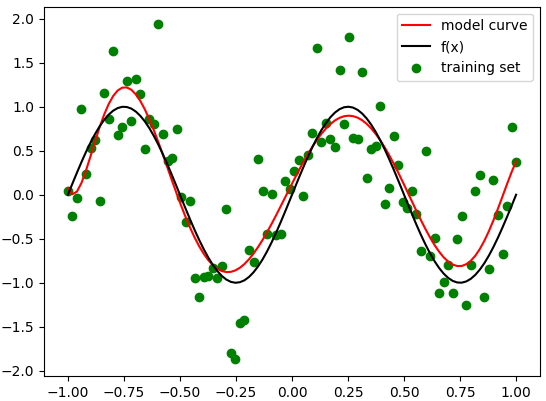


图1.2.2 训练集包含个点，多项式阶数，正则化系数，有一定程度欠拟合。

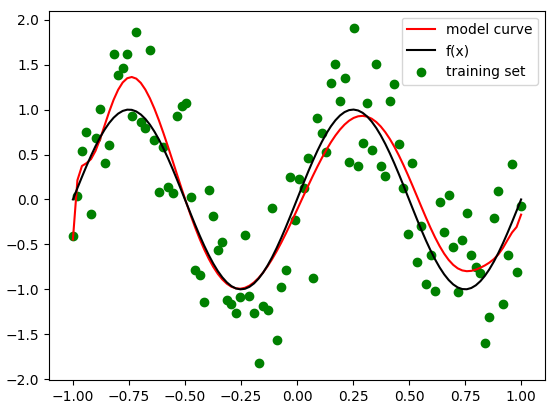


图1.2.3 训练集包含个点，多项式阶数，正则化系数。对比图1.1.3，过拟合程度显著降低。

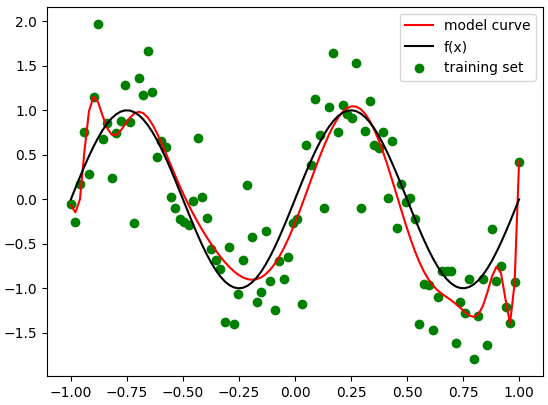


图1.2.4 训练集包含个点，多项式阶数，正则化系数，仍有明显过拟合。

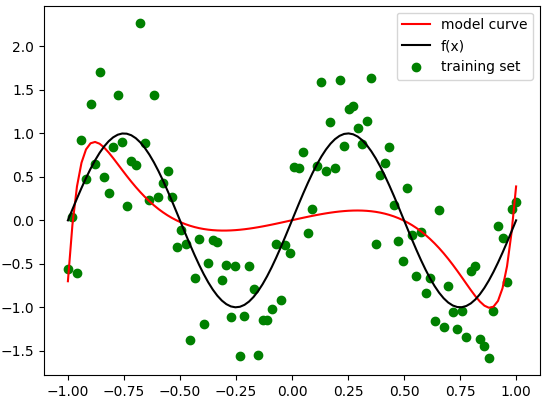


图1.2.5 训练集包含个点，多项式阶数，正则化系数，有严重欠拟合。

可以发现，在选择适当的正则项及正则化参数后，其对于未过拟合的模型（图1.2.1、图1.2.2）影响不大，但能显著抑制过拟合模型（图1.2.3）的过拟合程度。但若正则化参数过小（图1.2.4），则过拟合现象不能被很好地抑制；而若正则化参数过大（图1.2.5），则模型会欠拟合。因此，在添加了正则项的模型中，控制正则化系数十分重要。

## 梯度下降法（含正则项）

### 全训练集梯度下降

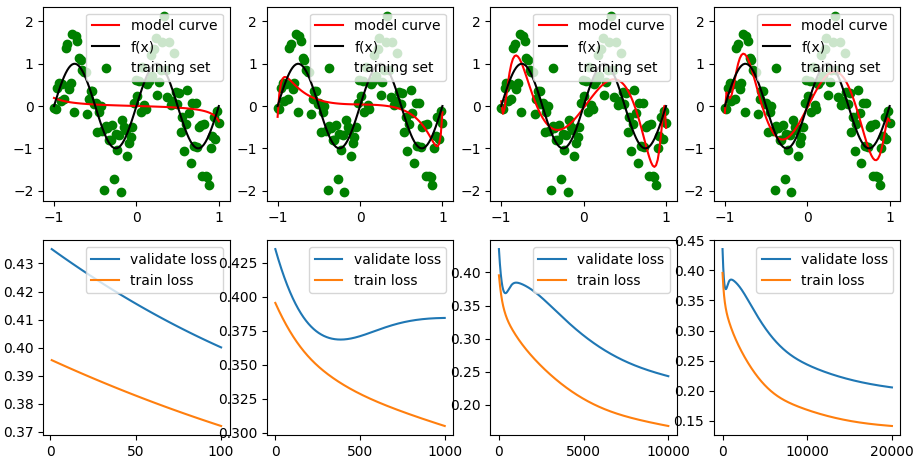


图2.1.1 训练集包含个点,验证集包含个点，多项式阶数，学习率为，正则化系数，从从左至右训练步数依次为。

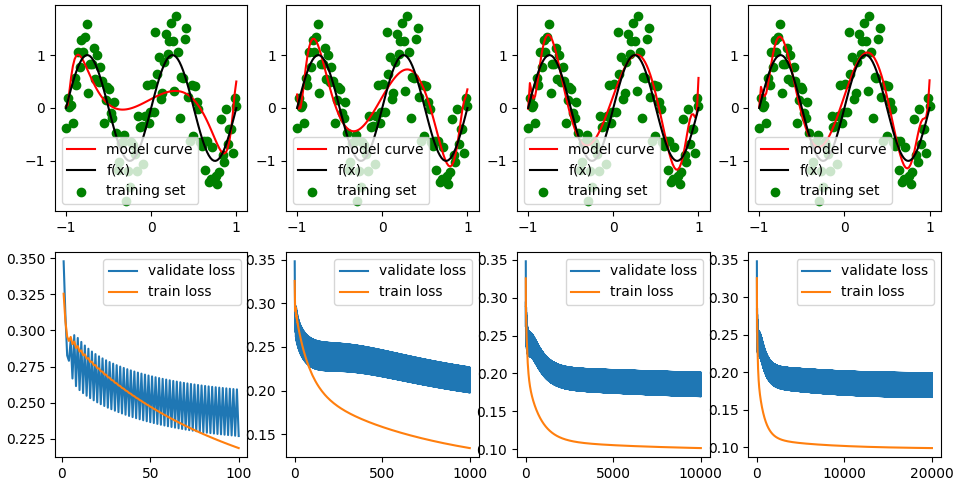


图2.1.2 训练集包含个点，验证集包含个点，多项式阶数，学习率为，正则化系数，从左至右训练步数依次为。

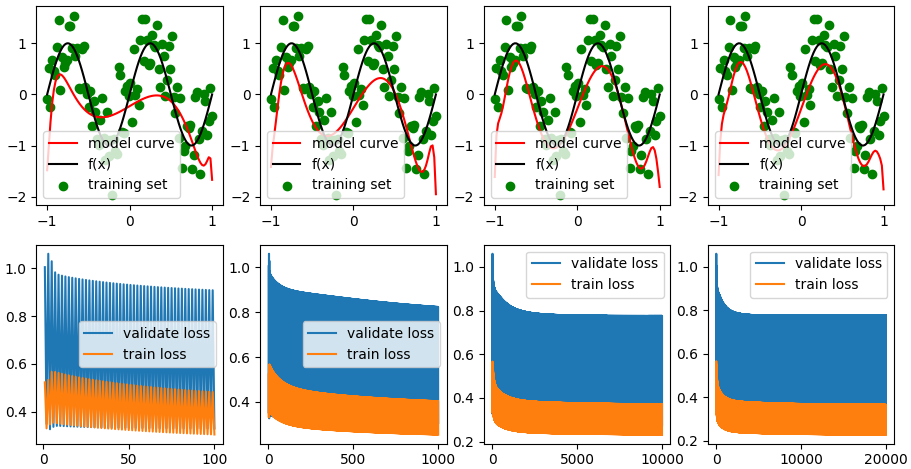


图2.1.3 训练集包含个点，验证集包含个点，多项式阶数，学习率为，正则化系数，从左至右训练步数依次为。

在每一组图中从左至右看，可以直观地发现随着训练步数增加，模型对训练数据的拟合效果越来越好，模型在训练集与验证集上的值也呈下降趋势。这表明梯度下降法作为模型训练方法是有效的。

但梯度下降法仅在学习率合适（图2.1.2）时，可以得到较好的训练结果。当学习率过小（图2.1.1）时，验证集loss值在训练结束时仍在下降，故模型将发生欠拟合；学习率过大（图2.1.3）时，训练集与验证集值在训练过程中均发生大幅度抖动，表明模型无法收敛至优化解。同时，当学习率合适时，过多的训练步数并不能为模型带来显著的性能提升，反而使模型发生过拟合，因此也应设置恰当的训练步数，例如在值变动小于给定阈值时停止训练。

### 随机梯度下降

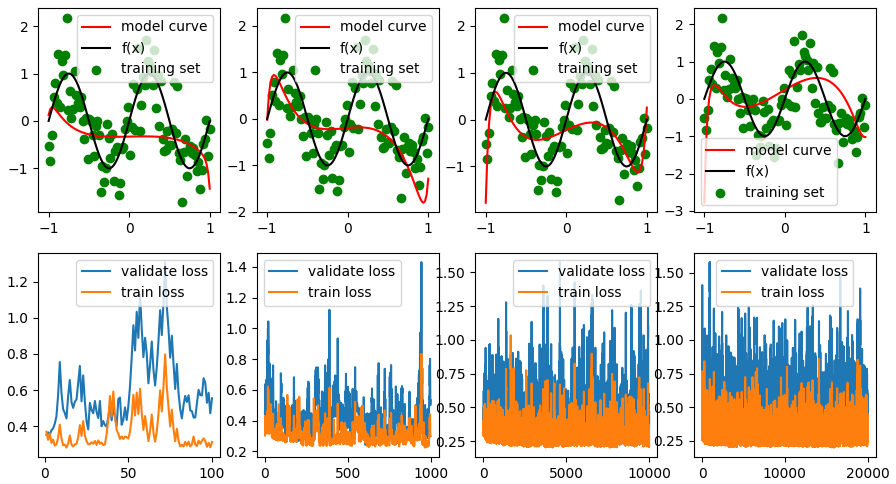


图2.2.1 训练集包含个点，验证集包含个点，多项式阶数，学习率为，每轮训练个样本，正则化系数，从左至右训练步数依次为，训练时间依次为0.13s，1.26s，12.62s，25.28s。

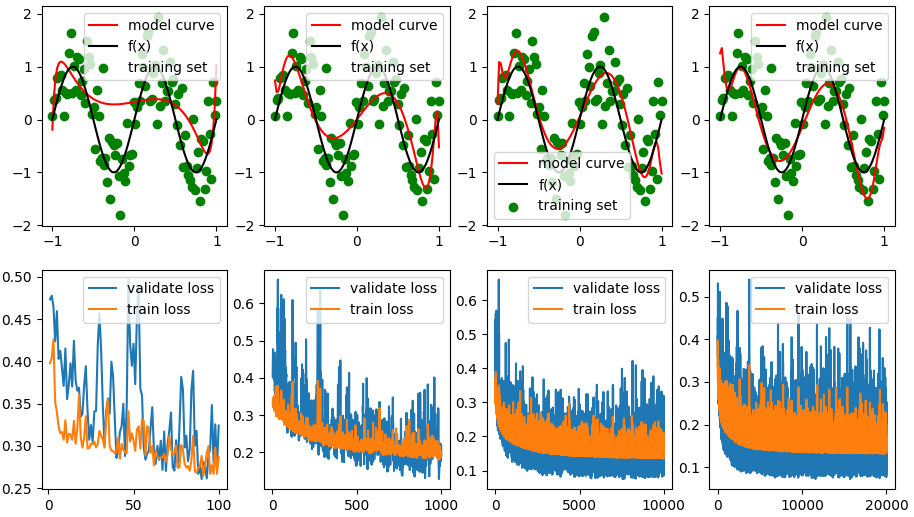


图2.2.2 训练集包含个点，验证集包含个点，多项式阶数，学习率为，每轮训练个样本，正则化系数，从左至右训练步数依次为，训练时间依次为0.13s，1.30s，12.90s，25.62s。

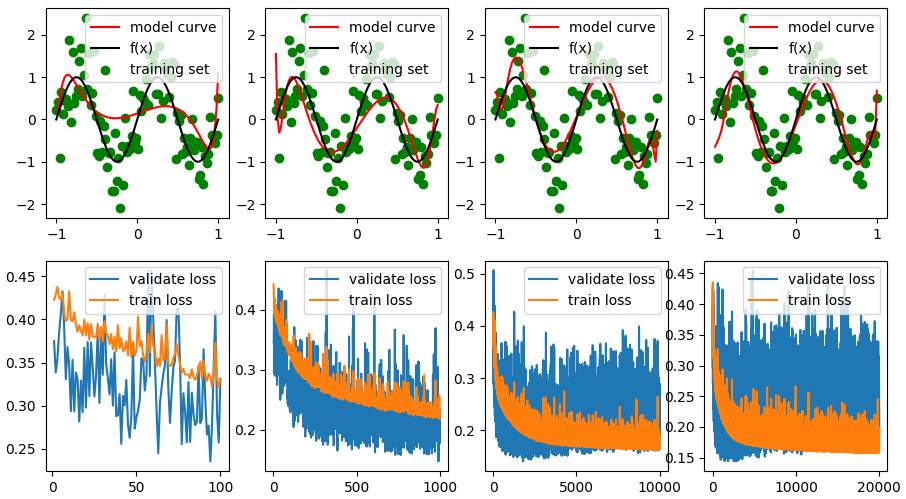


图2.2.3 训练集包含个点，验证集包含个点，多项式阶数，学习率为，每轮训练个样本，正则化系数，从左至右训练步数依次为，训练时间依次为0.15s，1.39s，13.60s，27.22s。

在每轮训练样本数很少（图2.2.1）时，随机梯度下降的训练效果较差，值抖动十分剧烈，且没有明显的下降趋势。这是因为每轮训练使用的样本无法充分表示训练数据的分布。每轮训练样本数适中（图2.2.2）时，值的抖动相比图2.2.1平缓，也呈现出一定下降趋势，这表明一定数量的样本点已能够体现原训练数据的分布情形。当每轮训练样本数较大（图2.2.3）时，训练集的值抖动更加平缓，而验证集值仍然抖动剧烈。对比全训练集梯度下降的值变化曲线，这表明随机梯度下降的稳定性不及全训练集梯度下降，但有一定可能强化模型的泛化能力。

此外，对训练时间的测量表明，随机梯度下降每轮训练样本数越多，所需时间越长。实际应用中，应在模型精度与训练时间中找寻平衡点。

## 共轭梯度法

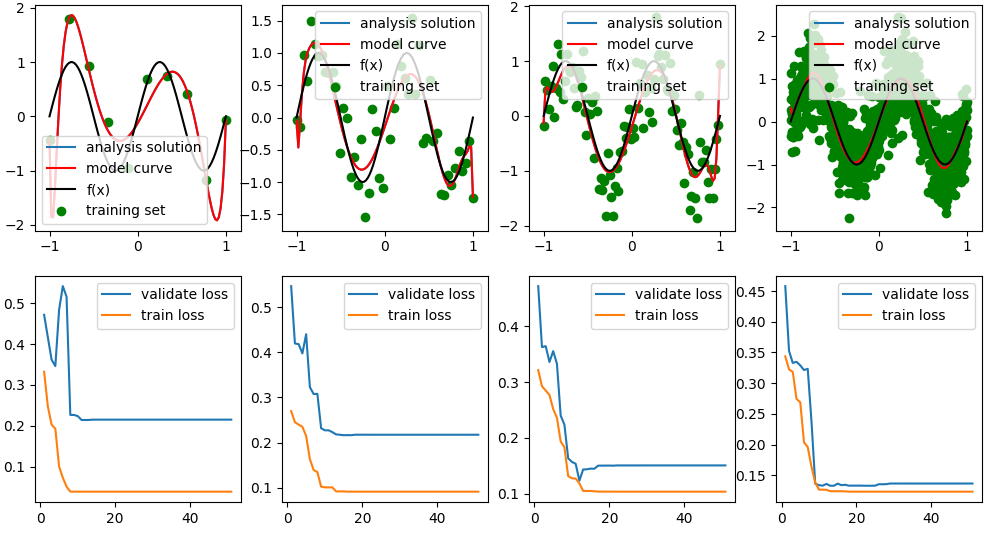


图3.1 验证集包含个点，多项式阶数，正则化系数。从左至右训练集大小依次为。图中模型曲线与解析解所得曲线几乎完全重合，对应参数之差不超过。

对参数的比较表明共轭梯度法求得的是与解析法相同的使训练集上函数最小的精确解。对训练过程中值变化的观察表明共轭梯度法中值首先迅速下降，接着保持近似不变，直至训练结束。因此当精度要求不高时，可以适当减少共轭梯度法的训练步数。

# 结论

本次实验主要采用了3种拟合方法，即解析解法、梯度下降法和共轭梯度法。解析解法为直接求解法，通过对函数各点求梯度以直接求得使训练集最小的参数，但其仅对函数梯度为0的方程可解的情况有效；梯度下降法为优化方法，通过求函数在一点的梯度以确定优化方向和优化步长，其适用于任意可求梯度的函数；共轭梯度法仅适用于优化由正定二次型确定的函数，其在训练步（为参数向量长度）后可求得与解析法相同的精确解，但在精度要求不高时可减少训练步数以求得粗略解。

模型复杂度是影响模型拟合程度的重要因素。当模型过于简单时，模型会欠拟合；模型过于复杂时，容易导致过拟合。为了防止欠拟合，可增加模型复杂度。而为了抑制过拟合，可降低模型复杂度或添加正则项（惩罚项），并使用正则化系数控制过拟合程度。

学习率和训练步数是影响梯度下降法学习效果的重要超参数。学习率过低或训练步数过少时，模型将无法接近优化解；学习率过高时，模型易无法收敛至优化解，而是在近似解附近振荡，甚至远离优化解。训练步数过多时，若模型复杂度足够高，模型极易发生过拟合，从而降低泛化能力。在进行随机梯度下降时，也应选择恰当的每轮训练样本数，以使得在保持一定训练效果的同时加快训练速度，同时一定程度上增强模型泛化能力。

# 参考文献

[1] Wikipedia. Polynomial regression. Wikipedia website (2018). http://en.wikipedia.org/wiki/Polynomial\_regression.

[2] Wikipedia. Gradient descent. Wikipedia website (2018). http://en.wikipedia.org/wiki/Gradient\_descent.

[3] Wikipedia. Conjugate gradient method. Wikipedia website (2018). http://en.wikipedia.org/wiki/Conjugate\_gradient\_method.

# 七、附录：源代码（带注释）

**# 文件：polymodel.py**

import numpy as np

class poly\_model:

plot\_N=100

def \_\_init\_\_(self,size,l=0):

"""

size size of the model

l the regularization coefficient

"""

self.M=size

self.l=l

self.w=np.zeros(self.M+1)

def \_getX(self,x):

"""

Returns the Vandermonde matrix of x.

"""

return np.array([x\*\*i for i in range(self.M+1)]).T

def \_get\_batch(self,x,y,batch\_size):

"""

Returns a random batch with size batch\_size from given data.

"""

if batch\_size==0:

return x,y

else:

index=list(np.random.randint(0,len(x),batch\_size))

return x[index],y[index]

def loss(self,x,y):

"""

loss(x,y)=(1/2)||Xw-y||\*\*2+(l/2)||w||\*\*2

"""

y=y-self.\_getX(x)@self.w

return (np.dot(y,y)/2+(self.l/2)\*np.dot(self.w,self.w))/len(y)

def dloss(self,x,y):

"""

(d/dw)loss=XT(Xw-y)+lw

"""

X=self.\_getX(x)

return X.T@(X@self.w-y)+self.l\*self.w

def train\_by\_analysis(self,train\_set,\*\*datasets):

"""

Trains the model by solving the equation directly.

datasets loss values on these datasets in training progress will be returned as a dict

"""

datasets['train\_set']=train\_set

loss\_dict=dict()

for key in datasets:

loss\_dict[key]=[]

x,y=train\_set

X=self.\_getX(x)

print("training...")

self.w=np.linalg.inv(X.T@X+self.l\*np.eye(self.M+1))@X.T@y

for key in datasets.keys():

loss\_dict[key].append(self.loss(datasets[key][0],datasets[key][1]))

print("training finished.")

return loss\_dict

def train\_by\_gradient\_descent(self,

train\_set,

learning\_rate,

step,

limit=1e-6,

batch\_size=0,

w=None,

\*\*datasets):

"""

Trains the model by gradient descent.

If the batch size is set to 0, the training method is normal gradient descent,

else the method is random gradient descent.

learning\_rate learning rate for every step

step number of training steps

limit gradient bottom limit

batch\_size size of each batch. 0 for all data, >0 for random batch

w initial value of w, default to 0

datasets loss values on these datasets in training progress will be returned as a dict

"""

datasets['train\_set']=train\_set

loss\_dict=dict()

for key in datasets:

loss\_dict[key]=[]

x,y=train\_set

self.w=np.zeros(self.M+1) if w is None else w

print("training...")

for i in range(step):

train\_x,train\_y=self.\_get\_batch(x,y,batch\_size)

d=self.dloss(train\_x,train\_y)

d2=np.linalg.norm(d,2)

if d2<limit: break

self.w-=learning\_rate\*(d/d2)

for key in datasets.keys():

loss\_dict[key].append(self.loss(datasets[key][0],datasets[key][1]))

print("\rtraining process: %d%% training loss=%.3f"%((i+1)\*100//step,self.loss(x,y)),end='')

print("\ntraining finished.")

return loss\_dict

def train\_by\_conjugate\_gradient(self,train\_set,step=None,w=None,\*\*datasets):

"""

Trains the model by conjugate gradient method.

w initial value of w, default to 0

step number of training steps

datasets loss values on these datasets in training progress will be returned as a dict

"""

datasets['train\_set']=train\_set

loss\_dict=dict()

for key in datasets:

loss\_dict[key]=[]

x,y=train\_set

self.w=np.zeros(self.M+1) if w is None else w

X=self.\_getX(x)

print("training...")

A=X.T@X+self.l\*np.eye(self.M+1)

b=X.T@y

r=b-A@self.w

p=r

for k in range(step if step is not None else self.M+1):

alpha=np.dot(r,r)/np.dot(p,A@p) # calculate the step length

self.w+=alpha\*p # step once

beta=np.dot(r,r)

r=r-alpha\*A@p

beta=np.dot(r,r)/beta

p=r+beta\*p # calculate the direction of the next step

for key in datasets.keys():

loss\_dict[key].append(self.loss(datasets[key][0],datasets[key][1]))

print("training finished.")

return loss\_dict

def predict(self,x):

"""

Returns the predict value(s) for the given x.

"""

return sum([self.w[i]\*x\*\*i for i in range(self.M+1)])

def curve(self,L=-1,R=1):

"""

Returns a curve of this model for plotting.

"""

x=np.linspace(L,R,self.plot\_N)

return x,self.predict(x)

**# 文件：main.py**

import numpy as np

import math

import time

import matplotlib.pyplot as plt

from polymodel import poly\_model

L,R=-1,1

def f(x):

return np.sin(2\*math.pi\*x)

def generate\_data(n,noise=0):

x=np.linspace(L,R,n)

y=np.array([f(i)+np.random.normal(scale=noise) for i in x])

return x,y

train\_set=generate\_data(100,noise=0.5)

test\_set=generate\_data(20,noise=0.5)

validate\_set=generate\_data(20,noise=0.5)

######################################################

print("Analysis solution without regularization test")

sizes=[5,10,50]

for i in range(3):

model=poly\_model(size=sizes[i])

loss\_dict=model.train\_by\_analysis(train\_set,

test\_set=test\_set,

validate\_set=validate\_set)

print("validate loss=%.3f test loss=%.3f\n"%(loss\_dict['validate\_set'][-1],loss\_dict['test\_set'][-1]))

plot\_x,plot\_y=model.curve()

plt.scatter(train\_set[0],train\_set[1],c='g',label='training set')

plt.plot(plot\_x,plot\_y,c='r',label='model curve')

plt.plot(np.linspace(-1,1,100),f(np.linspace(-1,1,100)),c='black',label='f(x)')

plt.legend()

plt.show()

train\_set=generate\_data(1000,noise=0.5)

model=poly\_model(size=50)

loss\_dict=model.train\_by\_analysis(train\_set,

test\_set=test\_set,

validate\_set=validate\_set)

print("validate loss=%.3f test loss=%.3f\n"%(loss\_dict['validate\_set'][-1],loss\_dict['test\_set'][-1]))

plot\_x,plot\_y=model.curve()

plt.scatter(train\_set[0],train\_set[1],c='g',label='training set')

plt.plot(plot\_x,plot\_y,c='r',label='model curve')

plt.plot(np.linspace(-1,1,100),f(np.linspace(-1,1,100)),c='black',label='f(x)')

plt.legend()

plt.show()

print("")

######################################################

print("Analysis solution with regularization test")

train\_set=generate\_data(100,noise=0.5)

sizes=[5,10,50,50,50]

ls=[1e-3,1e-3,1e-3,1e-6,1]

for i in range(5):

model=poly\_model(size=sizes[i],l=ls[i])

loss\_dict=model.train\_by\_analysis(train\_set,

test\_set=test\_set,

validate\_set=validate\_set)

print("validate loss=%.3f test loss=%.3f\n"%(loss\_dict['validate\_set'][-1],loss\_dict['test\_set'][-1]))

plot\_x,plot\_y=model.curve()

plt.scatter(train\_set[0],train\_set[1],c='g',label='training set')

plt.plot(plot\_x,plot\_y,c='r',label='model curve')

plt.plot(np.linspace(-1,1,100),f(np.linspace(-1,1,100)),c='black',label='f(x)')

plt.legend()

plt.show()

print("")

######################################################

print("Gradient descent test")

lrs=[1e-3,2e-1,1]

steps=[100,1000,10000,20000]

for i in range(3):

for j in range(4):

model=poly\_model(size=50,l=1e-3)

loss\_dict=model.train\_by\_gradient\_descent(train\_set,

learning\_rate=lrs[i],

step=steps[j],

test\_set=test\_set,

validate\_set=validate\_set)

print("validate loss=%.3f test loss=%.3f\n"%(loss\_dict['validate\_set'][-1],loss\_dict['test\_set'][-1]))

plot\_x,plot\_y=model.curve()

plt.subplot(2,4,j+1)

plt.scatter(train\_set[0],train\_set[1],c='g',label='training set')

plt.plot(plot\_x,plot\_y,c='r',label='model curve')

plt.plot(np.linspace(-1,1,100),f(np.linspace(-1,1,100)),c='black',label='f(x)')

plt.legend()

plt.subplot(2,4,j+5)

plt.plot(range(1,len(loss\_dict['validate\_set'])+1),loss\_dict['validate\_set'],label='validate loss')

plt.plot(range(1,len(loss\_dict['train\_set'])+1),loss\_dict['train\_set'],label='train loss')

plt.legend()

plt.show()

print("")

######################################################

print("Random gradient descent test")

batches=[1,10,30]

steps=[100,1000,10000,20000]

for i in range(3):

for j in range(4):

model=poly\_model(size=50,l=1e-3)

start\_time=time.time()

loss\_dict=model.train\_by\_gradient\_descent(train\_set,

learning\_rate=2e-1,

step=steps[j],

batch\_size=batches[i],

test\_set=test\_set,

validate\_set=validate\_set)

end\_time=time.time()

print("validate loss=%.3f test loss=%.3f"%(loss\_dict['validate\_set'][-1],loss\_dict['test\_set'][-1]))

print("training time: %.2fs\n"%(end\_time-start\_time))

plot\_x,plot\_y=model.curve()

plt.subplot(2,4,j+1)

plt.scatter(train\_set[0],train\_set[1],c='g',label='training set')

plt.plot(plot\_x,plot\_y,c='r',label='model curve')

plt.plot(np.linspace(-1,1,100),f(np.linspace(-1,1,100)),c='black',label='f(x)')

plt.legend()

plt.subplot(2,4,j+5)

plt.plot(range(1,len(loss\_dict['validate\_set'])+1),loss\_dict['validate\_set'],label='validate loss')

plt.plot(range(1,len(loss\_dict['train\_set'])+1),loss\_dict['train\_set'],label='train loss')

plt.legend()

plt.show()

print("")

######################################################

print("Conjugate gradient test")

sizes=[10,50,100,1000]

for i in range(4):

train\_set=generate\_data(sizes[i],noise=0.5)

model=poly\_model(size=50,l=1e-3)

loss\_dict=model.train\_by\_conjugate\_gradient(train\_set,

test\_set=test\_set,

validate\_set=validate\_set)

print("validate loss=%.3f test loss=%.3f"%(loss\_dict['validate\_set'][-1],loss\_dict['test\_set'][-1]))

plot\_x,plot\_y=model.curve()

w=model.w

model.train\_by\_analysis(train\_set)

analysis\_x,analysis\_y=model.curve()

plt.subplot(2,4,i+1)

plt.plot(analysis\_x,analysis\_y,label='analysis solution')

plt.scatter(train\_set[0],train\_set[1],c='g',label='training set')

plt.plot(plot\_x,plot\_y,c='r',label='model curve')

plt.plot(np.linspace(-1,1,100),f(np.linspace(-1,1,100)),c='black',label='f(x)')

plt.legend()

plt.subplot(2,4,i+5)

plt.plot(range(1,len(loss\_dict['validate\_set'])+1),loss\_dict['validate\_set'],label='validate loss')

plt.plot(range(1,len(loss\_dict['train\_set'])+1),loss\_dict['train\_set'],label='train loss')

if np.alltrue(np.abs(w-model.w)<1e-4):

print("The training result is same as the analysis solution.\n")

else:

print("The training result is different from the analysis solution.\n")

plt.legend()

plt.show()