*VII-2-3-1.数値微分を使ったニューラルネットワークモデルによる判別分析*

VII-2-3-1-i.準備とデータの読み込み

|  |
| --- |
| #必要なライブラリーの読み込み  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  %matplotlib inline  #データの読み込み  df =pd.read\_csv("sample2.csv")  xn,D=df.shape  K=5 #クラスの数入力  D=D-K #説明変数の数  M=D #　第１層の細胞の数  X=np.zeros((xn,D))  T=np.zeros((xn,K))  for i in range(D):  X[:,i]=df.iloc[:,K+i]  for i in range(K):  T[:,i]=df.iloc[:,i]  X\_range0=[-2,2] #項目1の範囲  X\_range1=[-2,2] #項目2の範囲  #データをテストデータとトレーニング用のデータに分割して保存  TrainingRatio=1  X\_n\_training=int(xn\*TrainingRatio)  X\_train=X[:X\_n\_training,:]  X\_test=X[X\_n\_training:,:]  T\_train=T[:X\_n\_training,:]  T\_test=T[X\_n\_training:,:] |

VII-2-3-1-ii.データ分布の確認

|  |
| --- |
| #作業内容の定義  import matplotlib.pyplot as plt  %matplotlib inline  def show\_data(x,t):  wk,N=t.shape  col=["b","r","g","y","w"]  for k in range (K):  plt.plot(x[t[:,k]==1,0],x[t[:,k]==1,1],linestyle='none',marker='o',markeredgecolor='black',color=col[k],alpha=0.8)  plt.grid(True)  #動作  plt.figure(1,figsize=(8,3.7))  plt.subplot(1,2,1)  show\_data(X\_train,T\_train)  plt.xlim(X\_range0)  plt.ylim(X\_range1)  plt.title('Training data')  plt.subplot(1,2,2)  show\_data(X\_test,T\_test)  plt.xlim(X\_range0)  plt.ylim(X\_range1)  plt.title('Test data')  plt.show() |

VII-2-3-1-iii.feed forward neural network modelの定義

|  |
| --- |
| #シグモイド関数の定義  def sigmoid(alfa):  y=1/(1+np.exp(-alfa))  return y  #net work  def FNN(ab,M,K,x):  N,D=x.shape #データサイズと項目数  a=ab[:D\*(D+1)] #中間層の係数の選択  a=a.reshape(M,(M+1)) #M行D+1列に配列  b=ab[D\*(D+1):] #出力層の係数選択  b=b.reshape(K,(M+1)) #K行M+１列に配列  alfa=np.zeros((N,M+1)) #中間層の入力総和の配列枠(定数項を含む)  y=np.zeros((N,M+1)) #中間層の出力シグナルの枠(定数項を含む)  beta=np.zeros((N,K)) #出力層の入力総和の配列枠  z=np.zeros((N,K))  for n in range (N):  y[n,0]=1 #出力層のダミー変数  for m in range (M): #中間層  alfa[n,m]=np.dot(a[m,:],np.r\_[1,x[n,:]]) #中間層の入力総和の計算  y[n,m+1]=sigmoid(alfa[n,m])  sum=0 #ソフトマック関数の分母  for k in range(K):  beta[n,k]=np.dot(b[k,:],y[n,:])  sum=sum+np.exp(beta[n,k])  for k in range(K):  z[n,k]=np.exp(beta[n,k])/sum  return z,beta,y,alfa  #リスト3-4.平均交差エントロピー誤差の定義  def CE\_FNN(ab,M,K,x,t):  N,D=x.shape  z,beta,y,alfa=FNN(ab,M,K,x)  ce=-np.dot(t.reshape(-1),np.log(z.reshape(-1)))/N  return ce |

VII-2-3-1-iv.数値微分の定義

|  |
| --- |
| #リスト3-4.数値微分  #平均交差エントロピー誤差を係数で微分する。  #数値微分  def dCE\_FNN\_num(ab,M,K,x,t):  epsilon=0.00001  dab=np.zeros\_like(ab)  for iab in range(len(ab)):  ab\_modified=ab.copy()  ab\_modified[iab]=ab[iab]-epsilon  mse1=CE\_FNN(ab\_modified,M,K,x,t)  ab\_modified[iab]=ab[iab]+epsilon  mse2=CE\_FNN(ab\_modified,M,K,x,t)  dab[iab]=(mse2-mse1)/(2\*epsilon)  return dab  #abの微分値の表示  def show\_ab(ab,M):  Nab=ab.shape[0]  plt.bar(range(1,M\*3+1),ab[:M\*3],align="center",color="black")  plt.bar(range(M\*3+1,Nab+1),ab[M\*3:],align="center",color="cornflowerblue")  plt.xticks(range(1,Nab+1))  plt.xlim(0,Nab+1) |

VII-2-3-1-v.勾配降下法の実行

|  |
| --- |
| #計算時間を記録する  import time  #勾配降下法の定義  def Fit\_FNN\_num(ab\_init,M,K,x\_train,t\_train,x\_test,t\_test,nstp,lr):  ab=ab\_init  err\_train=np.zeros(nstp) #nstpはステップの数  err\_test=np.zeros(nstp)  ab\_hist=np.zeros((nstp,len(ab\_init))) #abの記録  for n in range(nstp):  ab=ab-lr\*dCE\_FNN\_num(ab,M,K,x\_train,t\_train)  err\_train[n]=CE\_FNN(ab,M,K,x\_train,t\_train)  err\_test[n]=CE\_FNN(ab,M,K,x\_test,t\_test)  ab\_hist[n,:]=ab  return ab,ab\_hist,err\_train,err\_test  #実行  startTime=time.time()  np.random.seed(1)  ab\_init=np.random.normal(0,0.01,M\*3+K\*(M+1))  N\_step=1000 #1000回繰り返す  LR=0.5 #学習率  AB,AB\_hist,Err\_train,Err\_test=Fit\_FNN\_num(ab\_init,M,K,X\_train,T\_train,X\_test,T\_test,N\_step,LR)  calculation\_time=time.time()-startTime  print("caculation time:{0:.3f}sec".format(calculation\_time)) |

VII-2-3-1-vi.経過の表示

|  |
| --- |
| #リスト3-6.計算経過の表示  plt.figure(1,figsize=(3,3))  plt.plot(Err\_train,'black',label='training')  plt.plot(Err\_test,'cornflowerblue',label='test')  plt.legend()  plt.show() |

VII-2-3-1-vii.係数の変化の図示

|  |
| --- |
| #係数の変化  plt.figure(1,figsize=(3,3))  plt.plot(AB\_hist[:,:M\*3],'black')  plt.plot(AB\_hist[:,M\*3:],'cornflowerblue')  plt.show |

VII-2-3-1-viii.判別境界線の表示

|  |
| --- |
| #リスト3-8.判別境界の表示  def show\_FNN(ab,M,K):  xn=60 #解像度  gn=xn\*xn  x0=np.linspace(X\_range0[0],X\_range0[1],xn)  x1=np.linspace(X\_range1[0],X\_range1[1],xn)  xx0,xx1=np.meshgrid(x0,x1)  x=np.c\_[xx1.reshape(gn,1), xx0.reshape(gn,1)]  z,beta,y,alfa=FNN(AB,M,K,x)  plt.figure(1,figsize=(4,4))  for k in range(K):  f=z[:,k]  f=f.reshape(xn,xn)  f=f.T  cont=plt.contour(xx0,xx1,f,levels=[0.5,0.95],colors=['cornflowerblue','black'])  cont.clabel(fmt='%.2f',fontsize=9)  plt.xlim(X\_range0)  plt.ylim(X\_range1)  #表示  import matplotlib  matplotlib.use('Agg')  import matplotlib.pyplot as plt  plt.figure(1,figsize=(4,4))  show\_data(X\_train,T\_train)  show\_FNN(AB,M,K)  plt.show()  fig=plt.show()  plt.savefig("numericfig,png") |

VII-2-3-1-ix.結果の出力と保存

|  |
| --- |
| #推定値の出力  print(AB)  AA=AB[:D\*(D+1)] #中間層の係数の選択  AA=AA.reshape(M,(M+1)) #M行D+1列に配列eras]  BB=AB[M\*(D+1):] #出力層の係数選≅  BB=BB.reshape(K,(M+1)) #K行M+１列に配列  #係数の保存  df=pd.DataFrame(AA)  df.to\_csv('parameter1AS.csv')  df=pd.DataFrame(BB)  df.to\_csv('parameter1BŞ.csv') |