simulation 1 : BPNN

10867024 廖育賢

# **objective - the problem and the purpose**

以一層隱藏層之神經網路使用倒傳遞演算法offline學習函數y = x1^2 + x2^2。

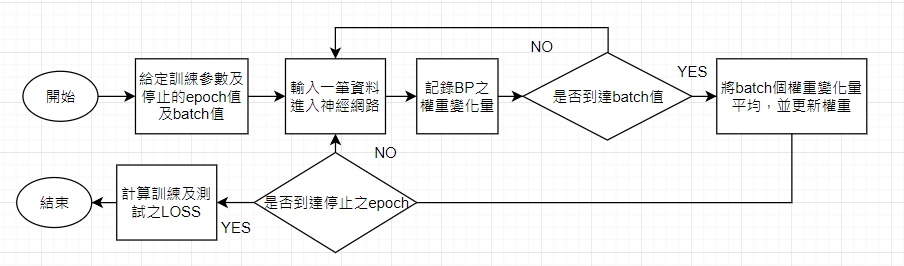
# **PRocrdure**

## method

首先隨機取樣400筆資料，分為300筆訓練資料及100筆測試資料，Batch Learning以倒傳遞演算法更新權重，分析神經元個數、學習速率、慣量等參數對於訓練結果之影響。

## program flow chart

### batch mode:



## network structure

INPUTS: x1 , x2

HIDDEN LAYER 1 : fully connected

OUTPUT : y

Activation function : sigmoid(y) =

## calculation

### back propagation

wji (n+1)=wji (n)+Δwji , Δwji (n)= η\*δj (n)\* yi (n) ,δj (n)= -

for j is output unit : - =

for j is hidden unit : - , k is output unit

### generalized delta rule

Δwji (n)=η\*δj\*yi (n)+α\*Δwji (n-1)

### batch mode

# **simulation results**

## program codes

def activation(input\_array,function='sigmoid'):

if function =='sigmoid':

return 1/(1 + np.exp(-input\_array))

np.random.seed(6666)

x1 = np.linspace(-5,5, 400)

x2 = np.linspace(-5,5, 400)

np.random.shuffle(x1)

np.random.shuffle(x2)

d = x1\*\*2 + x2\*\*2

# Normalize d 0.2~0.8

d = (d-np.min(d))/(np.max(d)-np.min(d))\*(0.8-0.2)+0.2

#---------------- Input data ------------------------------

num\_in = 2

#----------------Hiddent Layer 1 ---------------------

num\_L1 = 3

bias\_L1 = np.random.uniform(-0.5,0.5,[num\_L1,1])#5 1

w\_L1 = np.random.uniform(-0.5,0.5,[num\_in,num\_L1])#2 5

#---------------- Output -----------------------------

num\_out = 1

bias\_out = np.random.uniform(-0.5,0.5,[num\_out,1])# 1 1

w\_out = np.random.uniform(-0.5,0.5,[num\_L1,num\_out])# 5 1

#---------------- Parameter --------------------------

eta = 0.8

mom = 0.95

epoch = 250000

Eav\_train = np.zeros([epoch])

Eav\_test = np.zeros([epoch])

dw\_out = temp1 = np.zeros([num\_L1,num\_out]) #5 1

dbias\_out = temp2 = np.zeros([num\_out,1])#1 1

dw\_L1 = temp3 = np.zeros([num\_in,num\_L1])#2 5

dbias\_L1 = temp4 = np.zeros([num\_L1,1])# 5 1

#---------------- Traning ----------------------------

t0 = timeit.default\_timer()

now = datetime.datetime.now().strftime("%Y-%m-%d\_%H-%M-%S")

i = -1

pbar = tqdm(total =epoch)

for i in range(epoch):

#--------------- Feed Forward -------------------

e = np.zeros([300])

E\_train = np.zeros([300])

for j in range(300):

X = np.array([x1[j],x2[j]]).reshape(2,1)# 2 1

L1 = activation(np.dot(np.transpose(w\_L1),X) + bias\_L1,'sigmoid')#5 1

out = activation(np.dot(np.transpose(L1),w\_out) + bias\_out,'sigmoid')#1 1

#--------------- Back Propagation-----------------

e[j] = (d[j]-out) #1 1

E\_train[j] = 0.5 \* e[j]\*\*2

locg\_k = e[j] \* (out\*(1-out))# 1 1

temp2 = temp2 + mom \* dbias\_out + eta \* locg\_k \* 1 #1 1

temp1 = temp1 + mom \* dw\_out + eta \* locg\_k \* L1 #5 1

locg\_j = L1\*(1-L1) \* locg\_k \* w\_out# 5 1

temp4 = temp4 + mom \* dbias\_L1 + eta \* locg\_j \* 1 # 5 1

temp3 = temp3 + mom \* dw\_L1 + eta \* np.dot(X,np.transpose(locg\_j))#2 5

dbias\_out = temp2/300

dw\_out = temp1/300

dbias\_L1 = temp4/300

dw\_L1 = temp3/300

temp1 = np.zeros([num\_L1,num\_out]) #5 1

temp2 = np.zeros([num\_out,1])#1 1

temp3 = np.zeros([num\_in,num\_L1])#2 5

temp4 = np.zeros([num\_L1,1])# 5 1

#---------- New weight --------------

bias\_out = bias\_out + dbias\_out

w\_out = w\_out + dw\_out

bias\_L1 = bias\_L1 + dbias\_L1

w\_L1 = w\_L1 + dw\_L1

#---------- Eave\_train

Eav\_train[i] = np.mean(E\_train)

#---------- Test data loss ---------------

E\_test = np.zeros([100])

for j in range(100):

X = np.array([x1[300+j],x2[300+j]]).reshape(2,1)# 2 1

L1 = activation(np.dot(np.transpose(w\_L1),X) + bias\_L1,'sigmoid')#5 1

out = activation(np.dot(np.transpose(L1),w\_out) + bias\_out,'sigmoid')#1 1

E\_test = 0.5\*( d[300+j] - out )\*\*2

Eav\_test[i] = np.mean(E\_test)

if i % 1000==0:

pbar.update(1000)

pbar.close()

t1 =(timeit.default\_timer()-t0)

print('Training time: {} min'.format((t1/60)))

#--------- Predict data --------------

y\_predict = np.zeros([100])

E\_predict = np.zeros([100])

for j in range(100):

X = np.array([x1[300+j],x2[300+j]]).reshape(2,1)# 2 1

L1 = activation(np.dot(np.transpose(w\_L1),X) + bias\_L1,'sigmoid')#5 1

out = activation(np.dot(np.transpose(L1),w\_out) + bias\_out,'sigmoid')#1 1

y\_predict[j] = out

E\_predict[j] = 0.5\*( d[300+j] - out )\*\*2

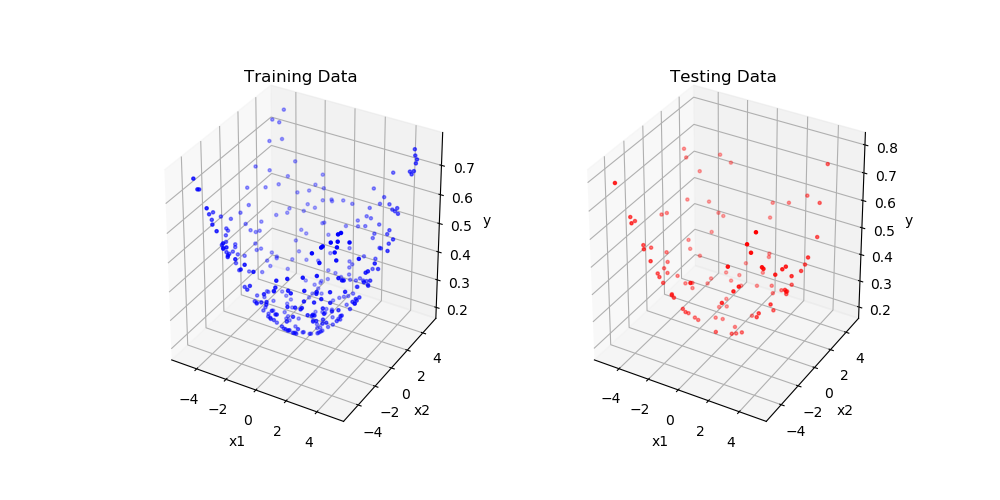
Eav\_predict = np.mean(E\_predict)

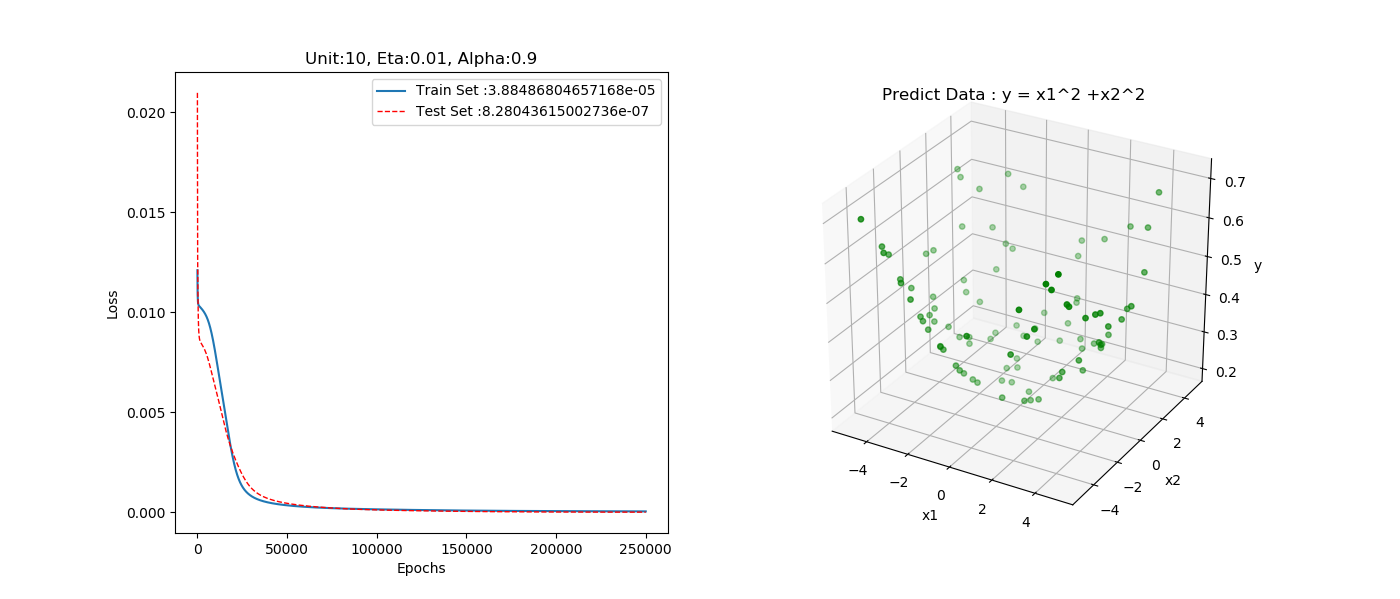
#----------- Return the data they were normolized before ----------------------

y\_predict = (y\_predict-0.2)/(0.8-0.2)\*(np.max(y\_predict)-np.min(y\_predict))+np.min(y\_predict)

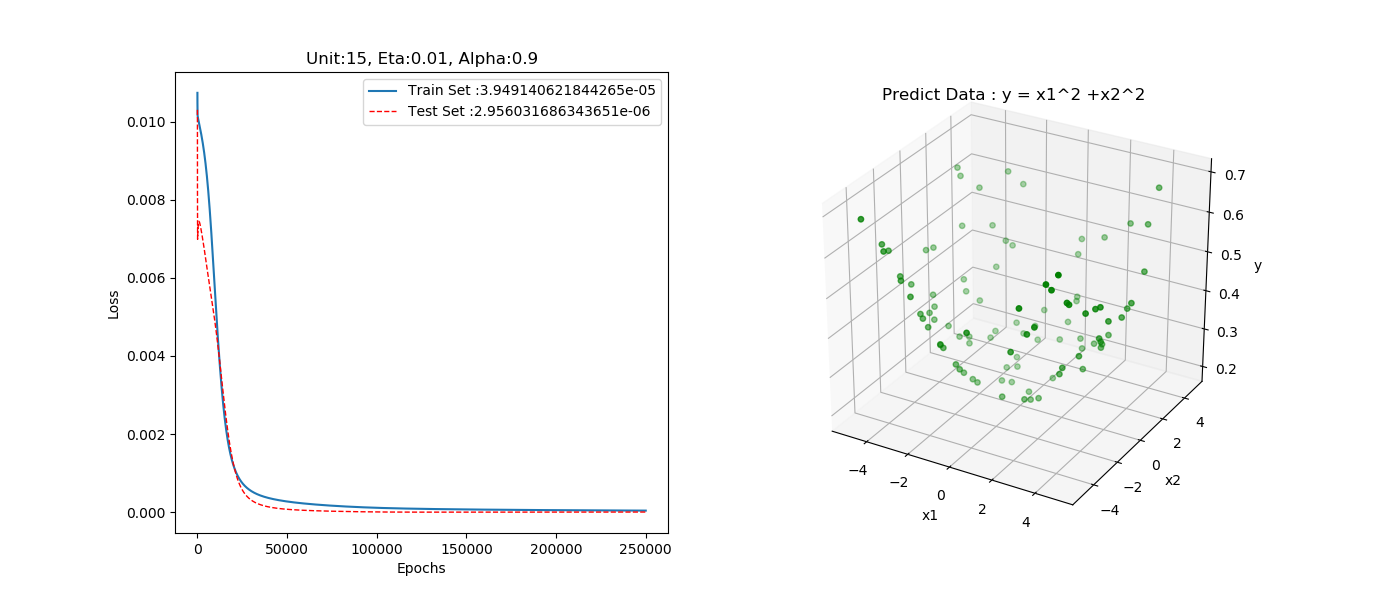
## graphs

#### 400 data (300 training & 100 testing):

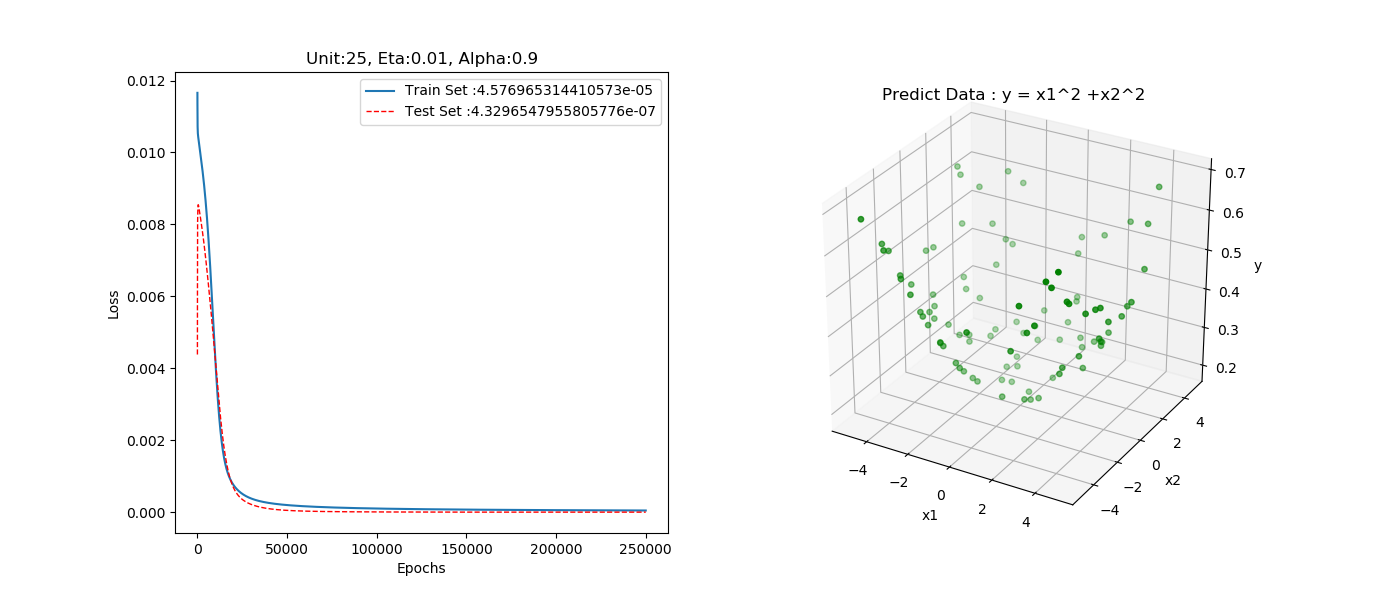


圖一

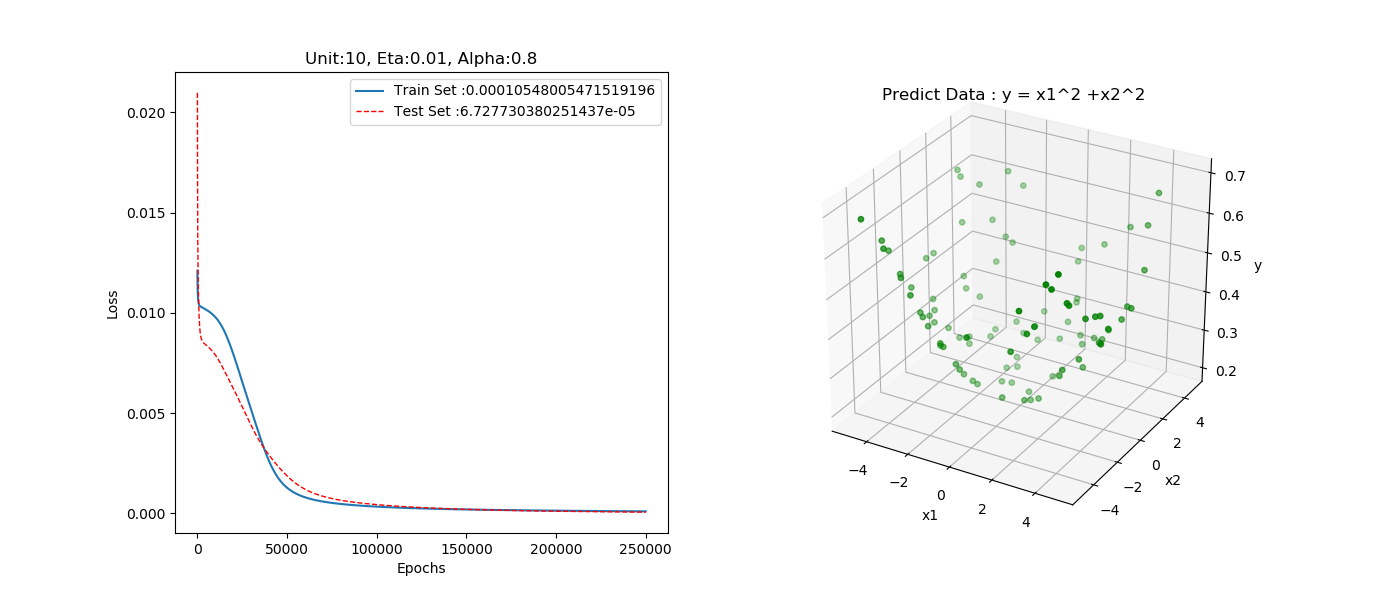
圖二

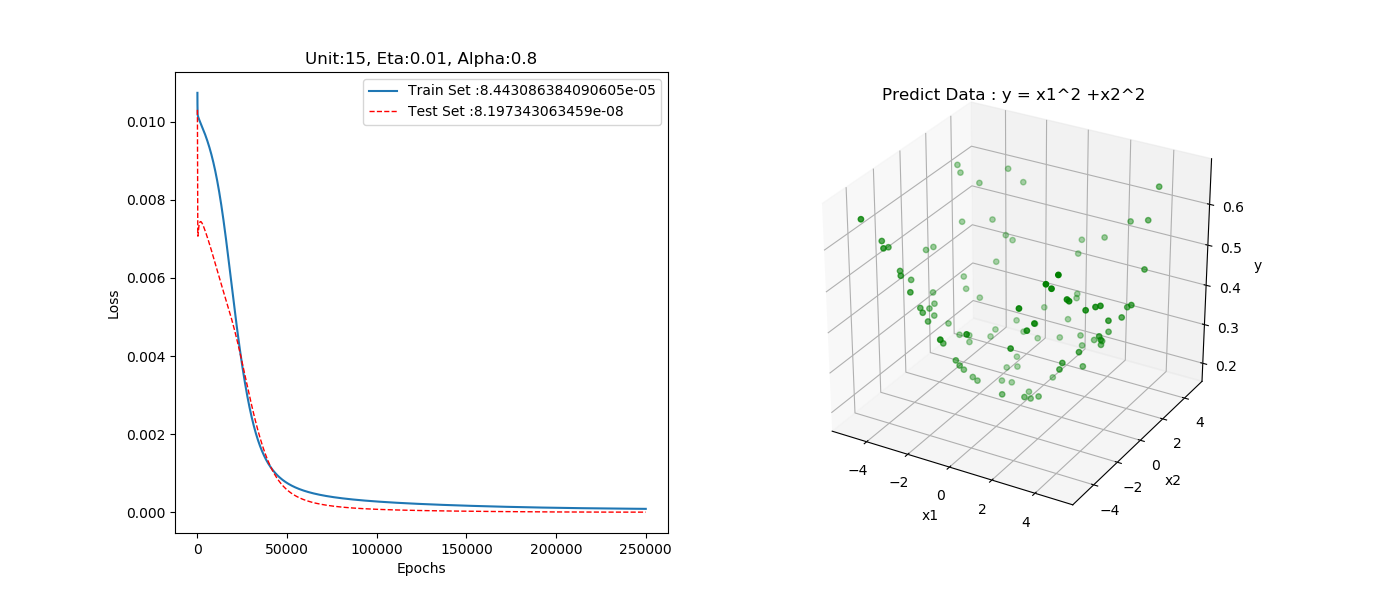


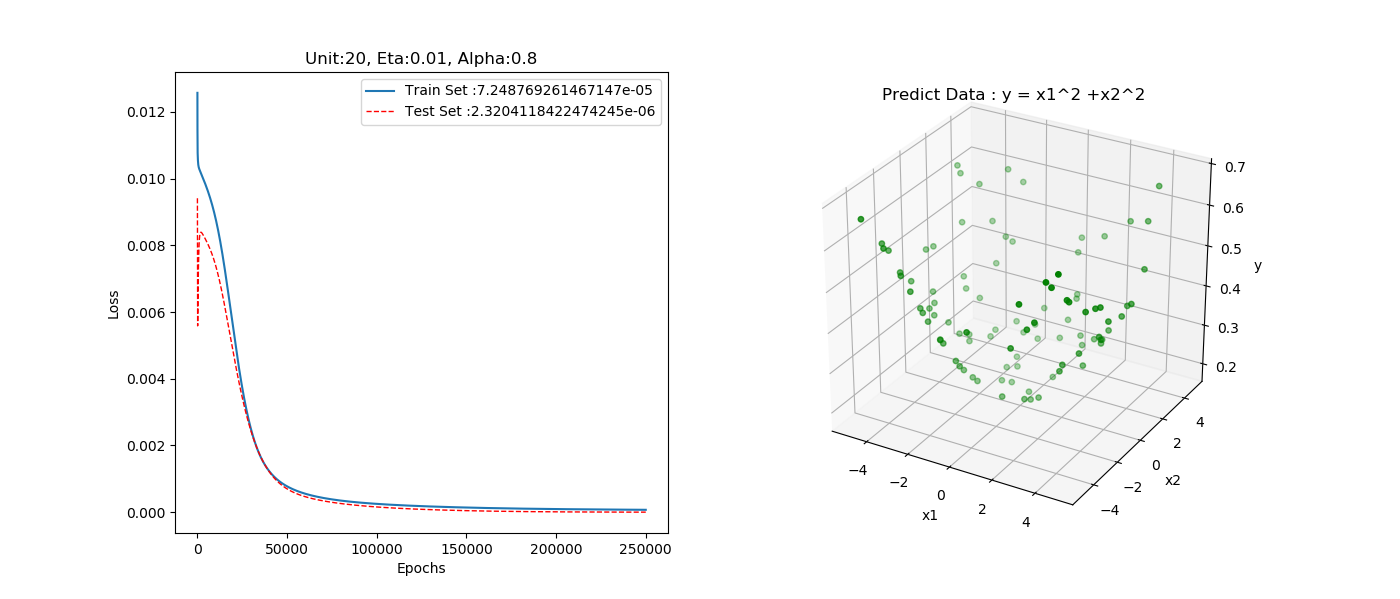
圖三



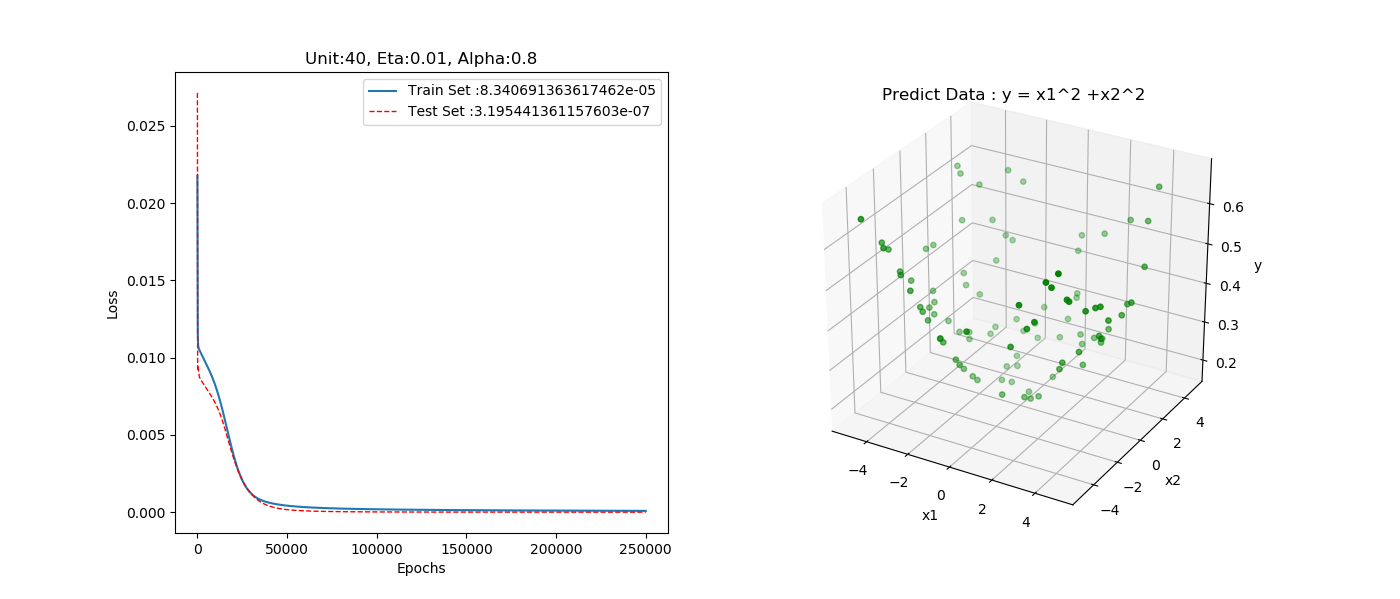
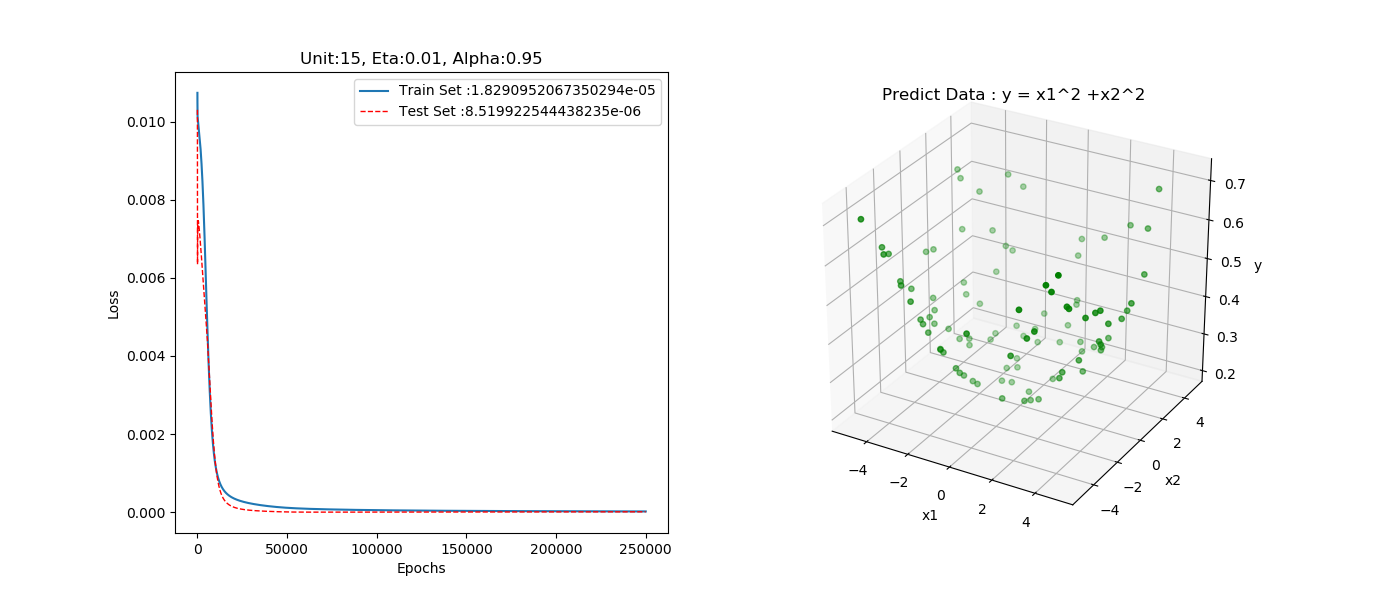
圖四

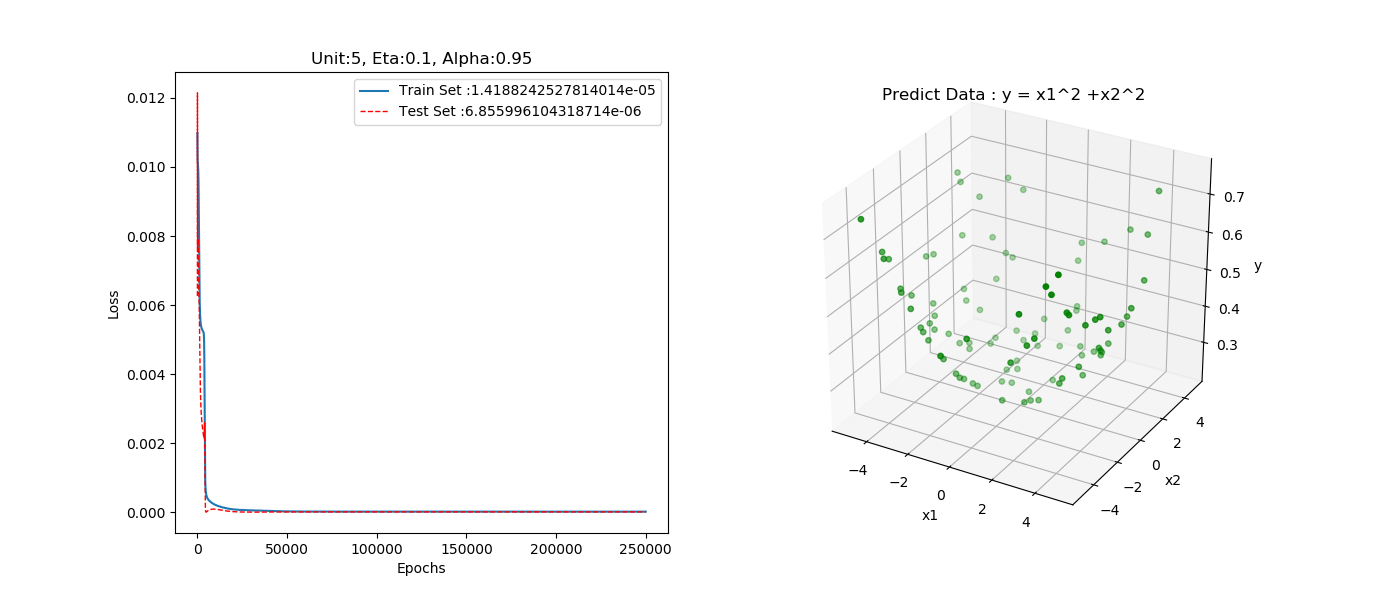
圖五



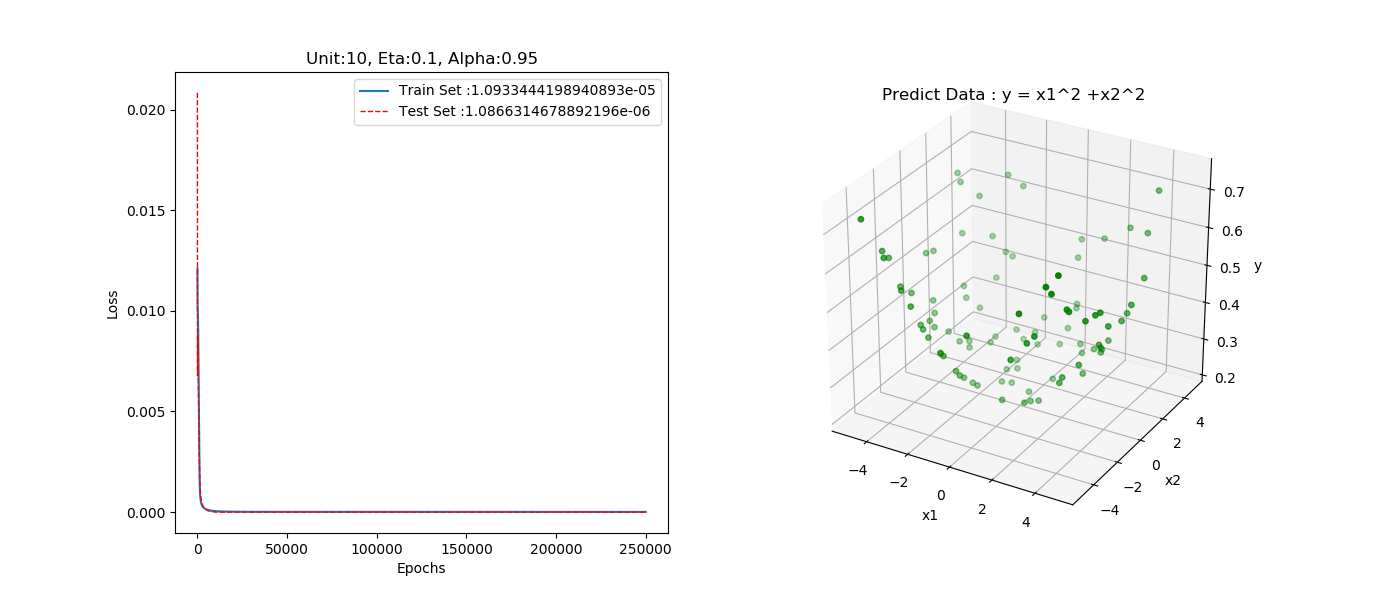
圖六

圖七

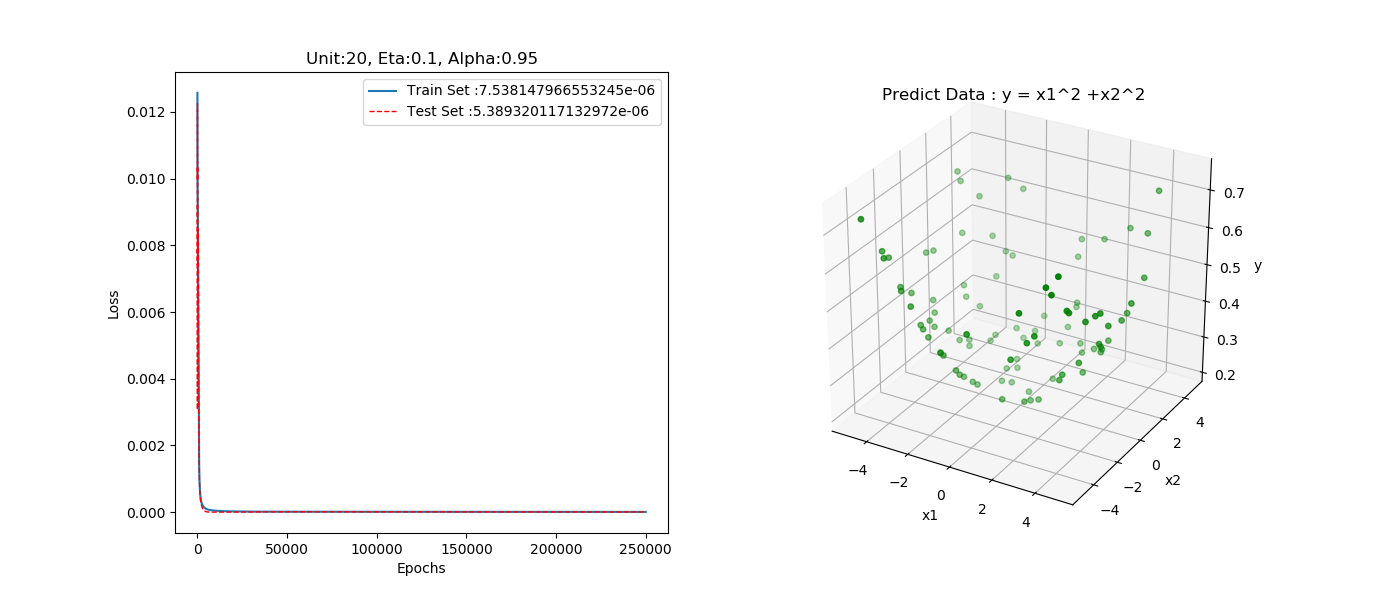
圖八圖九



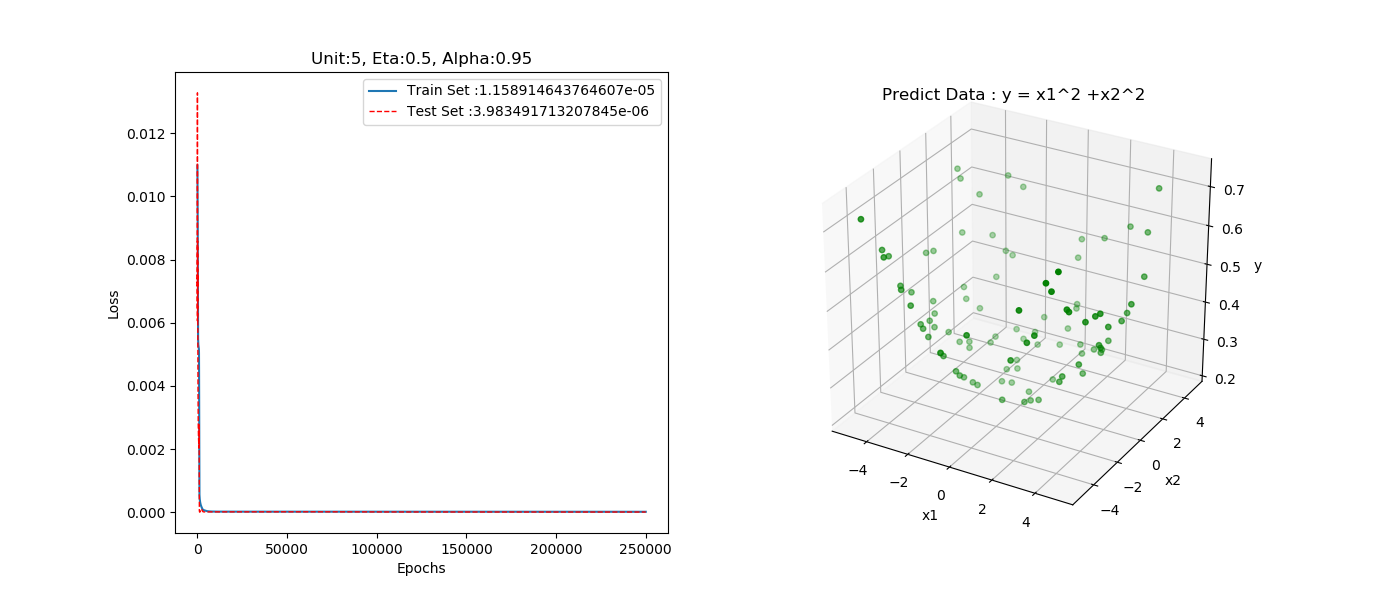
圖十



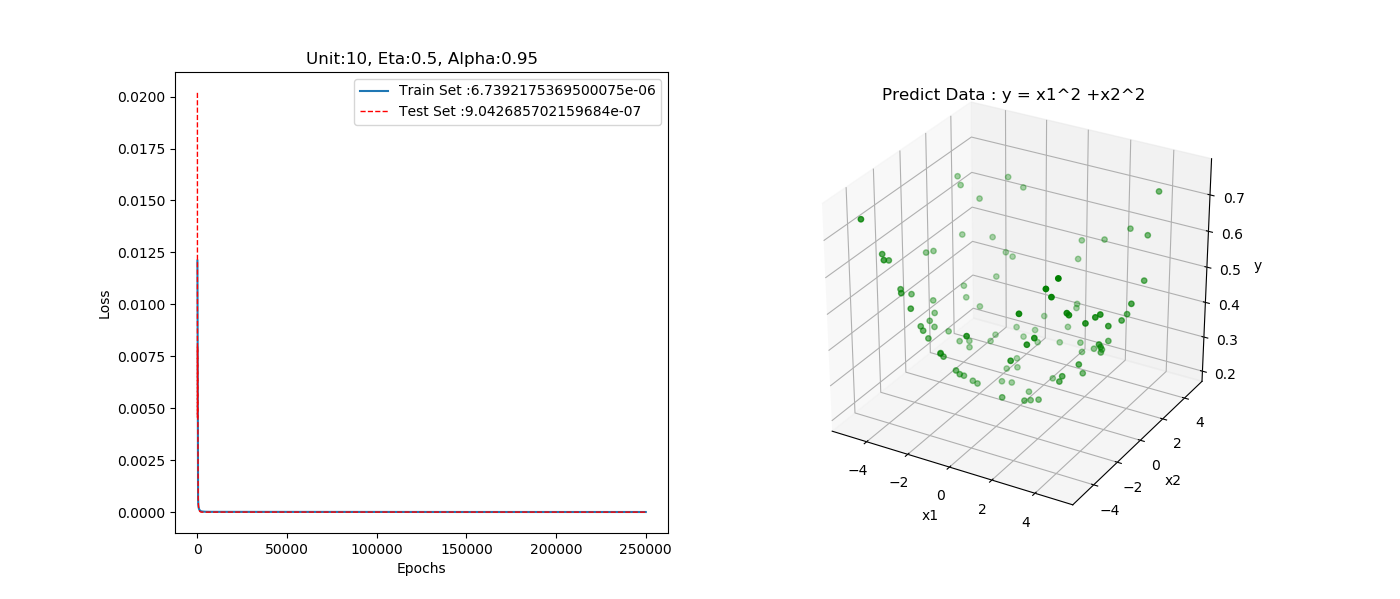
圖十一



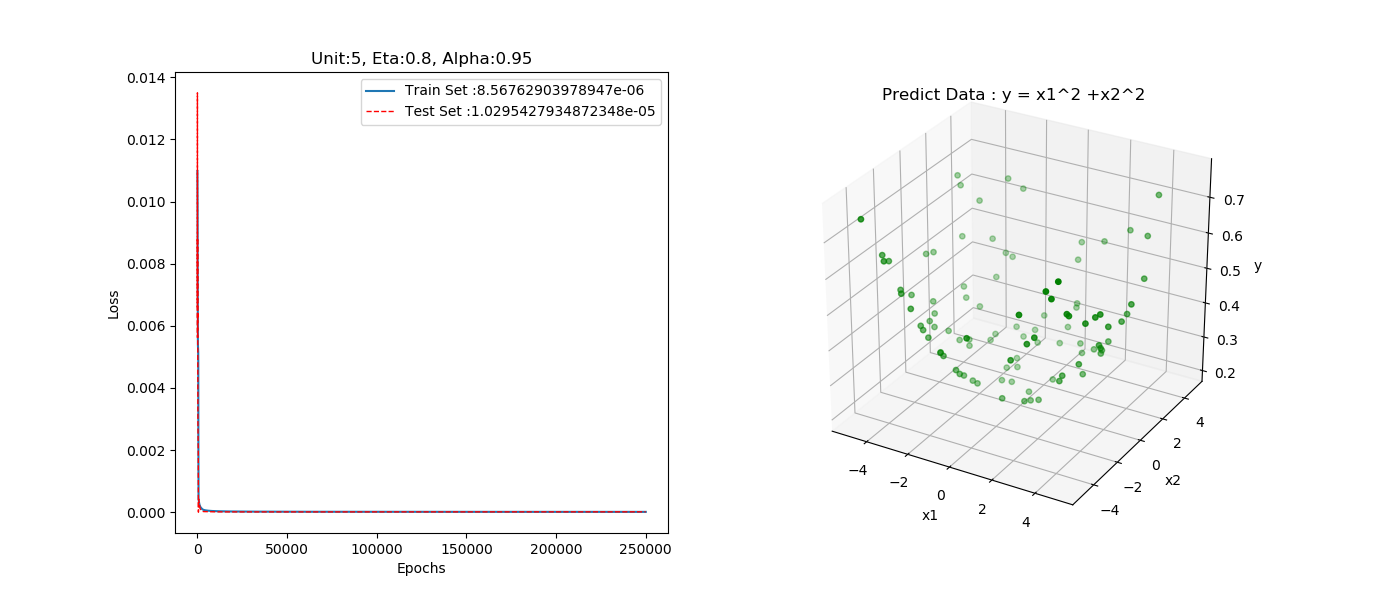
圖十二



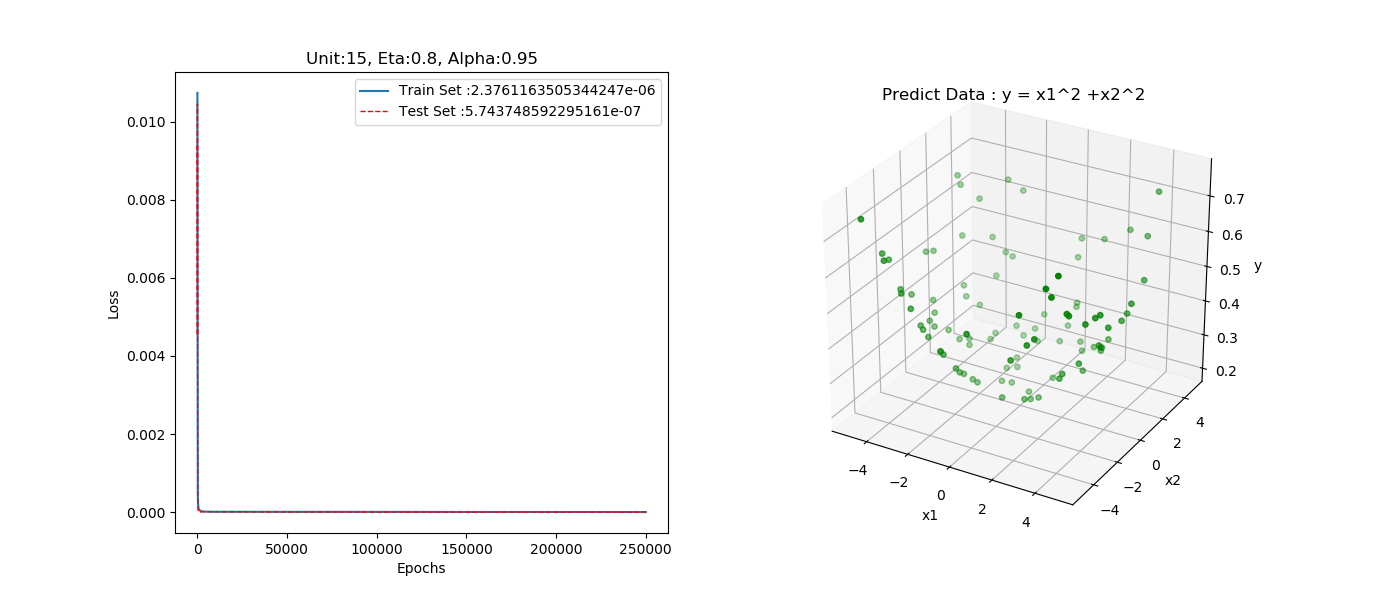
圖十三



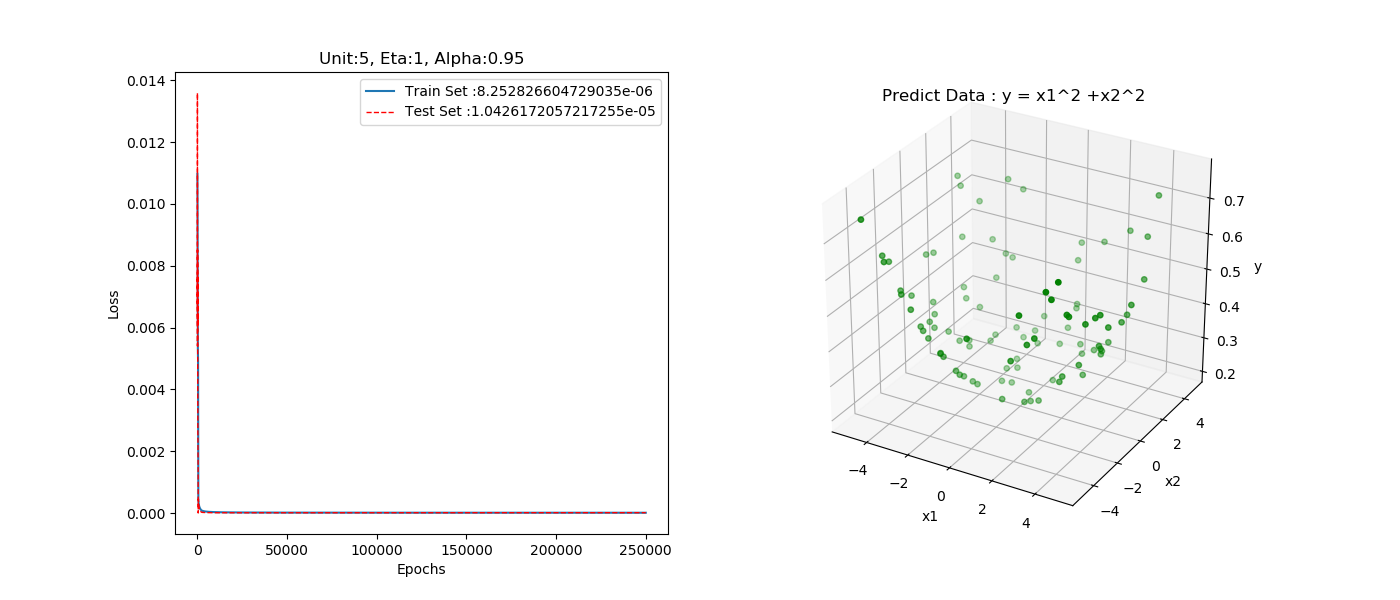
圖十四



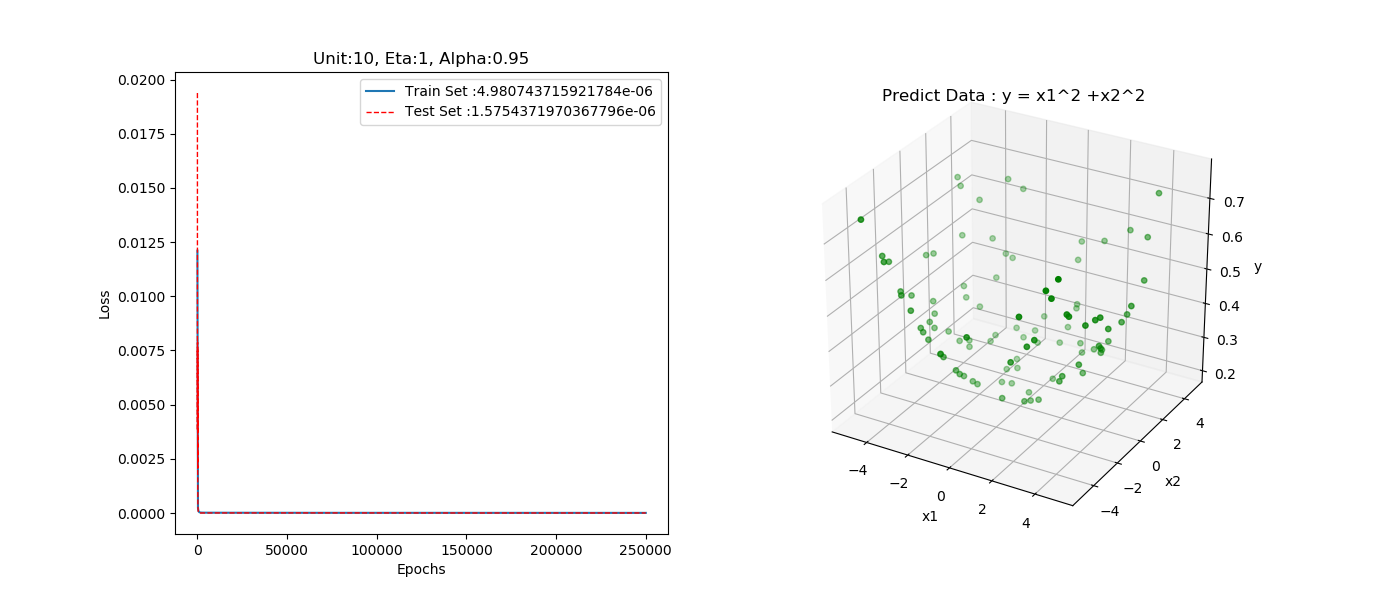
圖十五



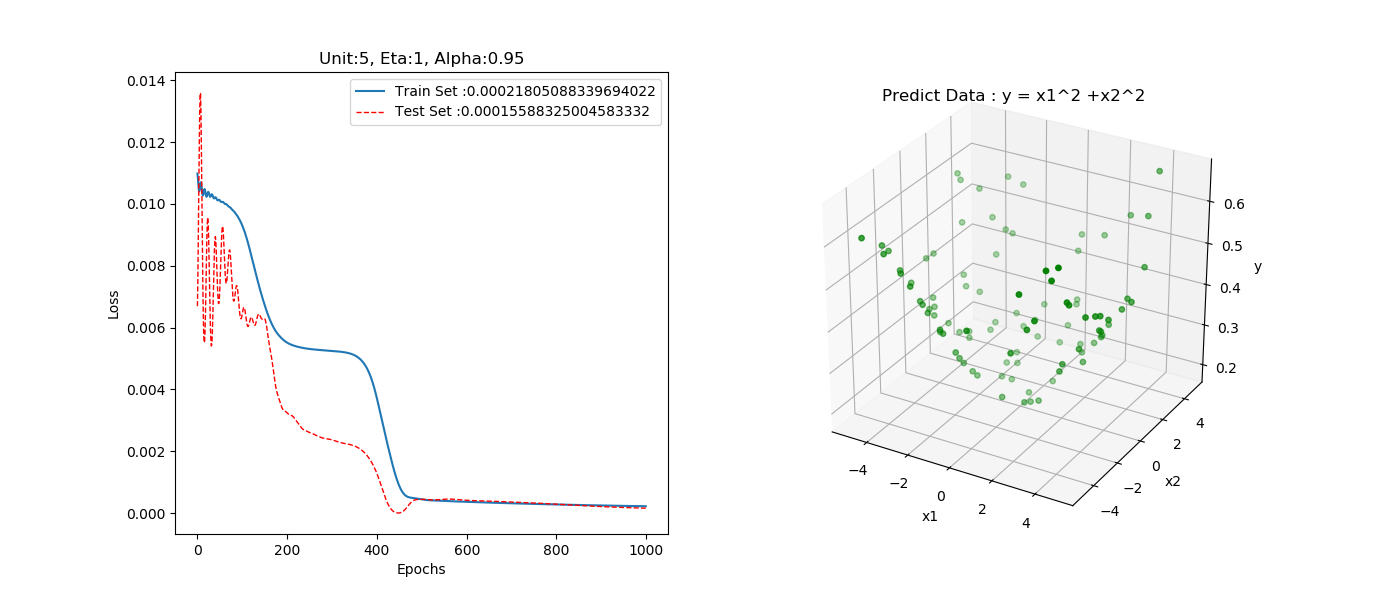
圖十六



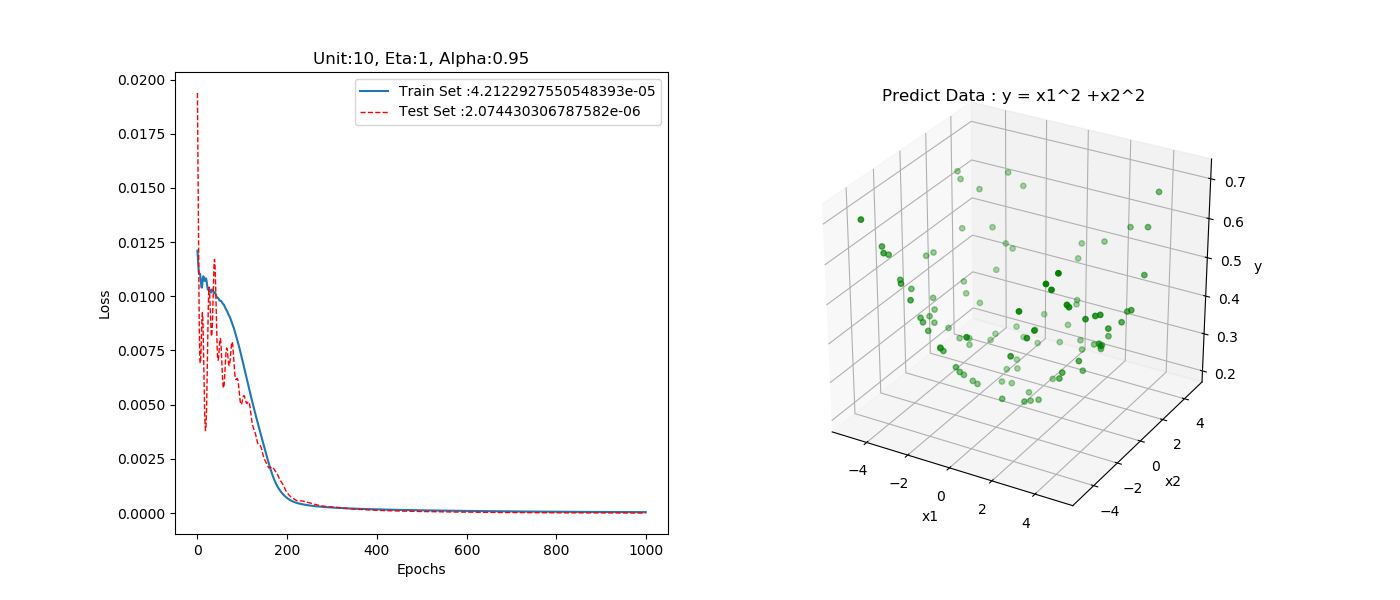
圖十七



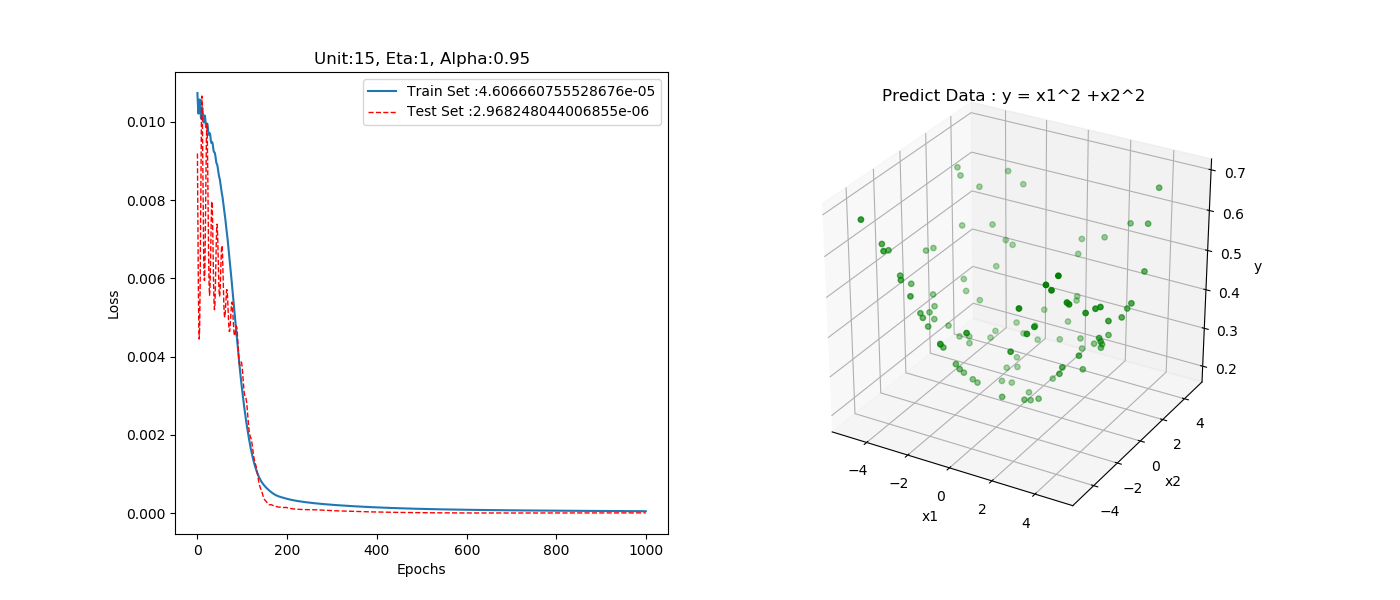
圖十八

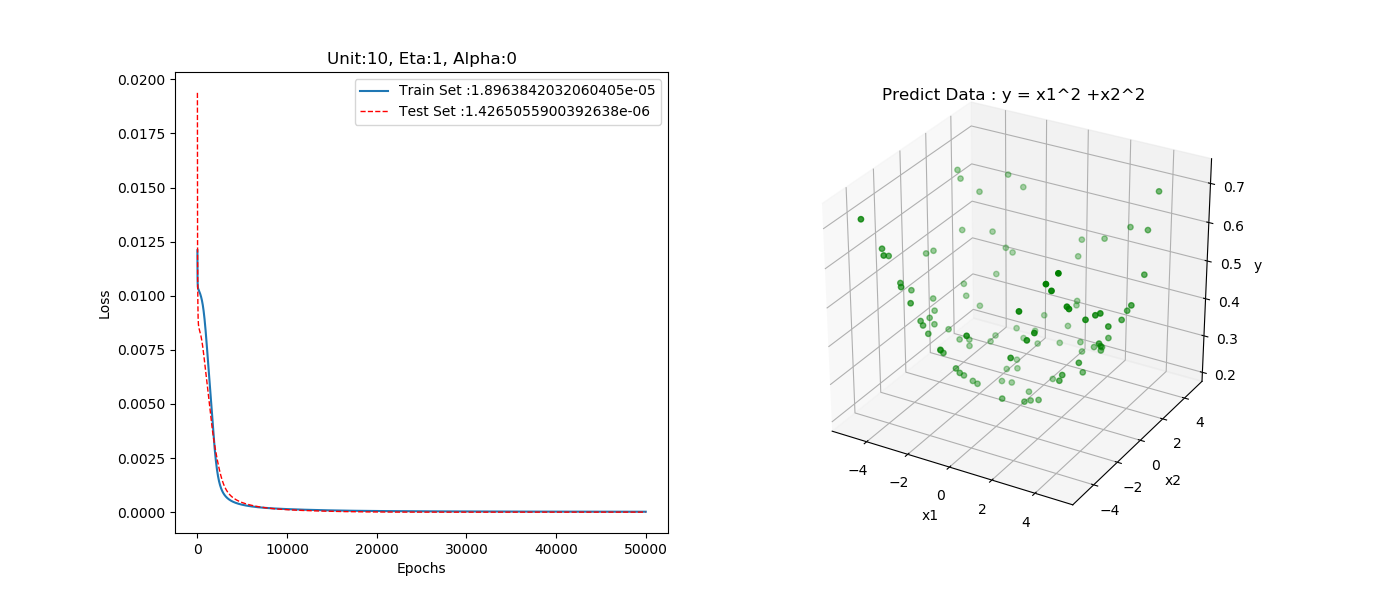


圖十九



圖二十

圖二十一

圖二十二

圖二十三

圖二十四

## tables

#### batch learning:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unit | Eta | Alpha | Training\_loss | Predict\_loss | Epoch |
| 5 | 0.01 | 0.9 | 0.000159 | 0.000226 | 250000 |
| 10 | 0.01 | 0.9 | 3.88E-05 | 4.99E-05 | 250000 |
| 20 | 0.01 | 0.9 | 3.50E-05 | 5.25E-05 | 250000 |
| 15 | 0.01 | 0.9 | 3.95E-05 | 5.91E-05 | 250000 |
| 30 | 0.01 | 0.9 | 4.78E-05 | 6.78E-05 | 250000 |
| 25 | 0.01 | 0.9 | 4.58E-05 | 6.27E-05 | 250000 |
| 35 | 0.01 | 0.9 | 4.76E-05 | 6.03E-05 | 250000 |
| 40 | 0.01 | 0.9 | 4.75E-05 | 6.38E-05 | 250000 |

表一

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unit | Eta | Alpha | Training\_loss | Predict\_loss | Epoch |
| 5 | 0.01 | 0.8 | 0.00035 | 0.000452 | 250000 |
| 10 | 0.01 | 0.8 | 0.000105 | 0.000131 | 250000 |
| 20 | 0.01 | 0.8 | 7.25E-05 | 9.90E-05 | 250000 |
| 15 | 0.01 | 0.8 | 8.44E-05 | 0.000115 | 250000 |
| 30 | 0.01 | 0.8 | 8.58E-05 | 0.000116 | 250000 |
| 35 | 0.01 | 0.8 | 8.56E-05 | 0.000105 | 250000 |
| 25 | 0.01 | 0.8 | 8.24E-05 | 0.000111 | 250000 |
| 40 | 0.01 | 0.8 | 8.34E-05 | 0.000105 | 250000 |

表二

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unit | Eta | Alpha | Training\_loss | Predict\_loss | Epoch |
| 5 | 0.01 | 0.95 | 6.06E-05 | 8.45E-05 | 250000 |
| 15 | 0.01 | 0.95 | 1.83E-05 | 2.96E-05 | 250000 |
| 10 | 0.01 | 0.95 | 1.89E-05 | 2.61E-05 | 250000 |
| 20 | 0.01 | 0.95 | 1.78E-05 | 2.88E-05 | 250000 |

表三

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unit | Eta | Alpha | Training\_loss | Predict\_loss | Epoch |
| 5 | 0.1 | 0.95 | 1.42E-05 | 1.82E-05 | 250000 |
| 10 | 0.1 | 0.95 | 1.09E-05 | 1.77E-05 | 250000 |
| 15 | 0.1 | 0.95 | 8.45E-06 | 1.60E-05 | 250000 |
| 20 | 0.1 | 0.95 | 7.54E-06 | 1.34E-05 | 250000 |

表四

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unit | Eta | Alpha | Training\_loss | Predict\_loss | Epoch |
| 5 | 0.5 | 0.95 | 1.16E-05 | 1.70E-05 | 250000 |
| 10 | 0.5 | 0.95 | 6.74E-06 | 1.14E-05 | 250000 |
| 15 | 0.5 | 0.95 | 3.07E-06 | 7.24E-06 | 250000 |
| 20 | 0.5 | 0.95 | 2.44E-06 | 6.09E-06 | 250000 |

表五

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *Unit* | Eta | Alpha | Training\_loss | Predict\_loss | Epoch |
| *5* | 0.8 | 0.95 | 8.57E-06 | 1.23E-05 | 250000 |
| *15* | 0.8 | 0.95 | 2.38E-06 | 5.43E-06 | 250000 |
| *20* | 0.8 | 0.95 | 1.48E-06 | 4.72E-06 | 250000 |

表六

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unit | Eta | Alpha | Training\_loss | Predict\_loss | Epoch |
| 5 | 0.9 | 0.95 | 8.40E-06 | 9.32E-06 | 250000 |
| 15 | 0.9 | 0.95 | 2.27E-06 | 3.00E-06 | 250000 |
| 20 | 0.9 | 0.95 | 1.37E-06 | 2.16E-06 | 250000 |

表七

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unit | Eta | Alpha | Training\_loss | Predict\_loss | Epoch |
| 5 | 1 | 0.95 | 8.25E-06 | 1.19E-05 | 250000 |
| 10 | 1 | 0.95 | 4.98E-06 | 7.54E-06 | 250000 |
| 15 | 1 | 0.95 | 2.19E-06 | 4.97E-06 | 250000 |
| 20 | 1 | 0.95 | 1.26E-06 | 4.21E-06 | 250000 |

表八

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unit | Eta | Alpha | Training\_loss | Predict\_loss | Epoch |
| 5 | 1 | 0.95 | 0.000218 | 0.000283 | 1000 |
| 10 | 1 | 0.95 | 4.21E-05 | 5.61E-05 | 1000 |
| 15 | 1 | 0.95 | 4.61E-05 | 6.85E-05 | 1000 |
| 20 | 1 | 0.95 | 4.03E-05 | 5.96E-05 | 1000 |
| 25 | 1 | 0.95 | 5.21E-05 | 7.09E-05 | 1000 |

表九

# **conclusion - analysis and comparison**

以Learning Mode 來看，Batch learning 因為將一批的權重更新量做平均，所以可使學習效果穩定、平滑，適用於處理細緻複雜的問題，缺點是收斂速度緩慢，耗費運算空間；Sequential learning 優點是可以做on-line學習，但因為每一筆資料皆會影響權重，如果資料選得不好，造成陷入局部最佳值。

我們可以加入Momentum將上一次的更新量保留一個比例，加入此次的權重更新量，圖二十中Loss收斂在30000 Epochs ，加入Momentum 0.95後 ( 如圖二十二 )，收斂於1000 Epochs，明顯提升收斂速度，

從learning rate的角度來看，理論上learing rate越小，loss將會越小，由表三與表八，將learning rate 0.01提高至1後，Traning Loss反而變小，因為欲訓練之系統，對於learning rate精度要求不高，選擇適合訓練模型的學習率才能有效降低loss。

從Hidden Layer 神經元個數來看，由圖五至圖八，固定learning rate 0.01、Momentum 0.8，改變神經元數10,15,20,40，觀察到隨著神經元個數提升，收斂速度越快；由圖二十三至圖二十四可看到神經元個數5顆所訓練的loss相較於其他10~40顆，明顯比較高，因此建議系統不適用五顆神經元訓練。