simulation 2: RBFN

10867024 廖育賢

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

以Radial Basis Function Network學習函數y = x1^2 + x2^2。

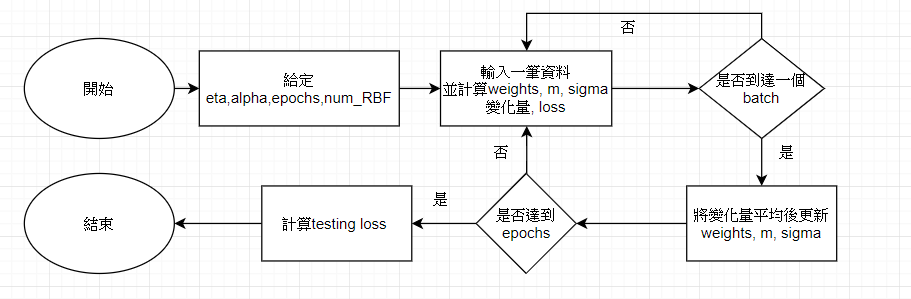
# **PRocrdure**

## 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，更新權重、平均值、變異數，並分析高斯函數個數、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

1. INPUTS: x1 , x2
2. RBF LAYER : gaussian function
3. OUTPUT : y

## calculation

### back propagation:

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

for j is output unit : - =

### generalized delta rule:

Δwji (n)=η\*δj\*yi (n)+α\*Δwji (n-1)

### batch mode:

### Gaussian:

, m : mean ,

(n)  ,

,

# **simulation results**

## program codes

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 to range 0.2~0.8

d\_max = np.max(d)

d\_min = np.min(d)

d = (d-d\_min)/(d\_max-d\_min)\*(0.8-0.2)+0.2

#---------------- Input vector -----------------------------

num\_in = 2

#---------------- Radial Basis phi -----------------------

num\_phi = 15

sigma = np.random.uniform(-0.5,0.5,[num\_phi,1])

phi\_out = np.zeros([num\_phi,1])

m = np.random.uniform(-0.5,0.5,[num\_phi,num\_in])

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

num\_out = 1

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

w\_out = np.random.uniform(-0.5,0.5,[num\_phi,num\_out])

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

eta = 0.01

mom = 0.95

epoch = 35000

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

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

dw\_out = temp1 = np.zeros([num\_phi,num\_out])

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

dm = temp3 = np.random.uniform(0,0.5,[num\_phi,num\_in])

dsigma = temp4 = np.random.uniform(0,0.5,[num\_phi,1])

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

t0 = timeit.default\_timer()

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

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)

XX = np.array(X)

for \_ in range(num\_phi-1):

XX = np.append(XX, X, axis=1)

XX = np.transpose(XX)

dx\_m = XX - m

phi\_out = Gaussian(XX,m,sigma)

out = sigmoid(np.dot(np.transpose(phi\_out),w\_out) + bias\_out)

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

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

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

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

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

temp1 = temp1 + mom \* dw\_out + eta \* locg\_k \* phi\_out

temp3 = temp3 + mom \* dm + eta \* e[j] \* w\_out \* phi\_out / sigma\*\*2 \* (dx\_m)

temp4 = temp4 + mom \* dsigma + eta \* e[j] \* w\_out \* phi\_out / sigma\*\*3 \* np.linalg.norm(dx\_m, axis=1, keepdims=True)

#---------- Average delta weight -----------------

dbias\_out = temp2/300

dw\_out = temp1/300

dm = temp3/300

dsigma = temp4/300

temp1 = np.zeros([num\_phi,num\_out])

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

temp3 = np.zeros([num\_phi,num\_in])

temp4 = np.zeros([num\_phi,1])

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

bias\_out = bias\_out + dbias\_out

w\_out = w\_out + dw\_out

m = m + dm

sigma = sigma + dsigma

#---------- 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)

XX = np.array(X)

for \_ in range(num\_phi-1):

XX = np.append(XX, X, axis=1)

XX = np.transpose(XX)

dx\_m = XX - m

phi\_out = Gaussian(XX,m,sigma)

out = sigmoid(np.dot(np.transpose(phi\_out),w\_out) + bias\_out)

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

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

if i % 1000 == 0 and i!=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)

XX = np.array(X)

for \_ in range(num\_phi-1):

XX = np.append(XX, X, axis=1)

XX = np.transpose(XX)

dx\_m = XX - m

phi\_out = Gaussian(XX,m,sigma)

out = sigmoid(np.dot(np.transpose(phi\_out),w\_out) + bias\_out)

y\_predict[j] = out

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

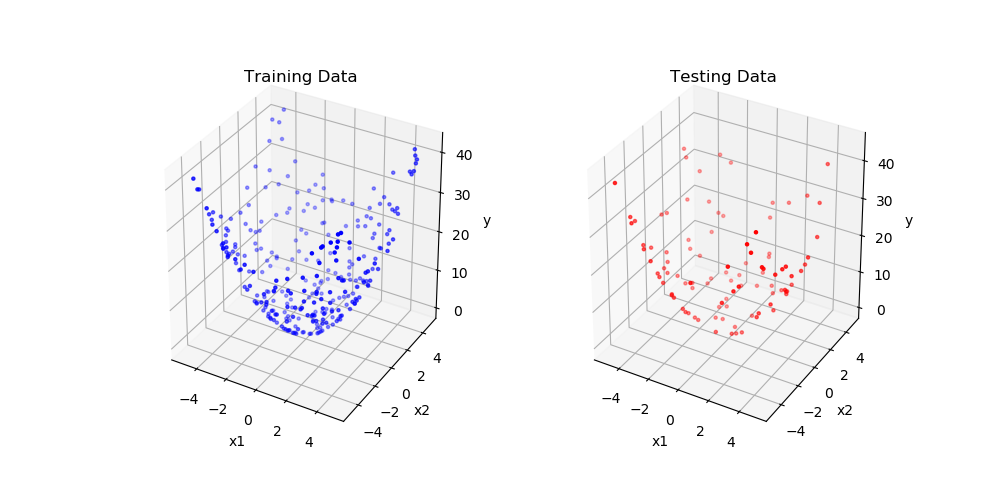
Eav\_predict = np.mean(E\_predict)

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

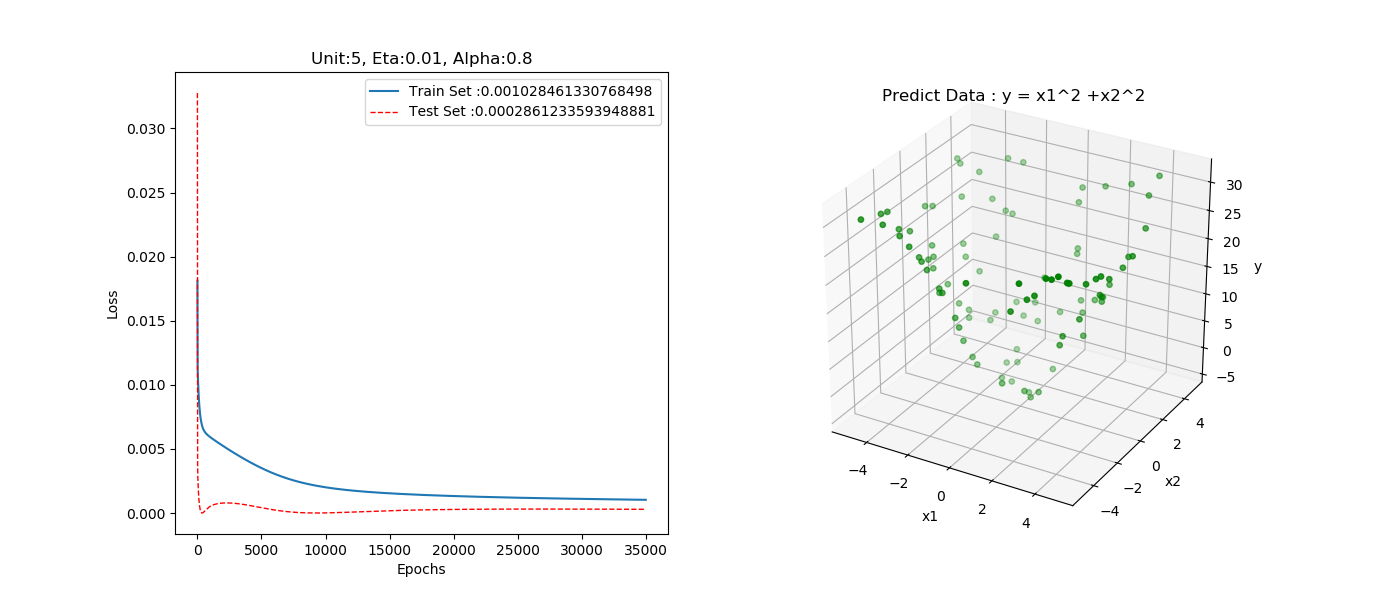
y\_predict = (y\_predict-0.2)/(0.8-0.2)\*(d\_max-d\_min)+d\_min

## graphs

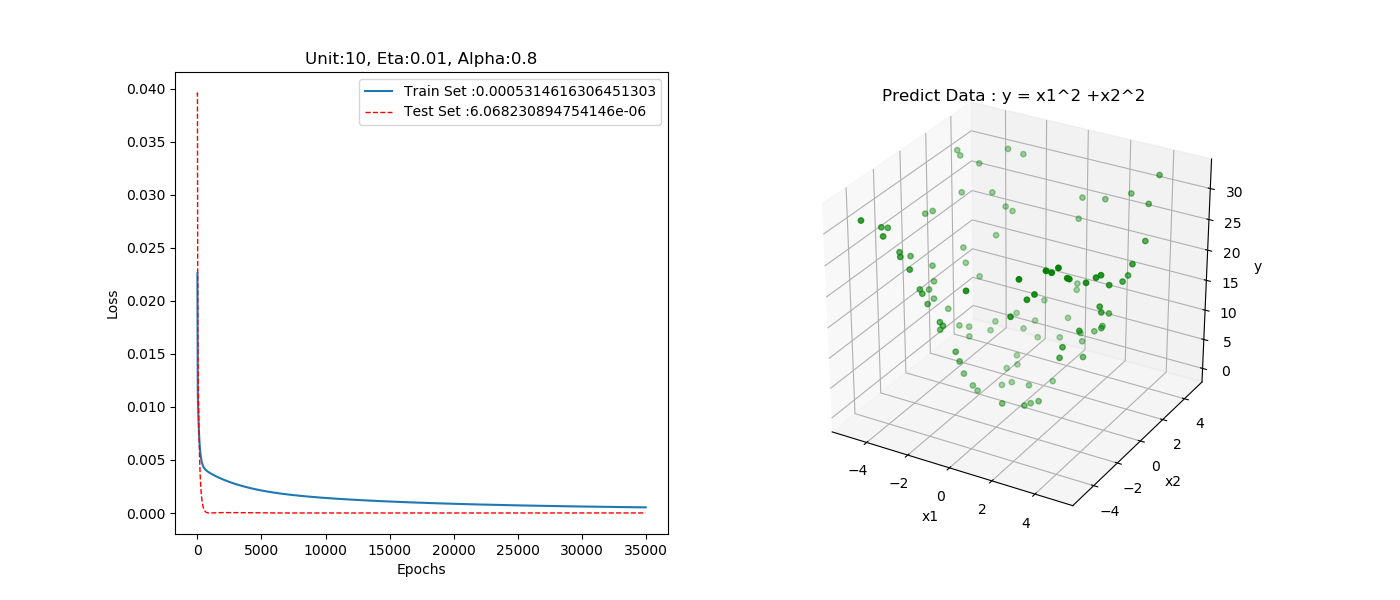
#### 400 data (300 training & 100 testing):

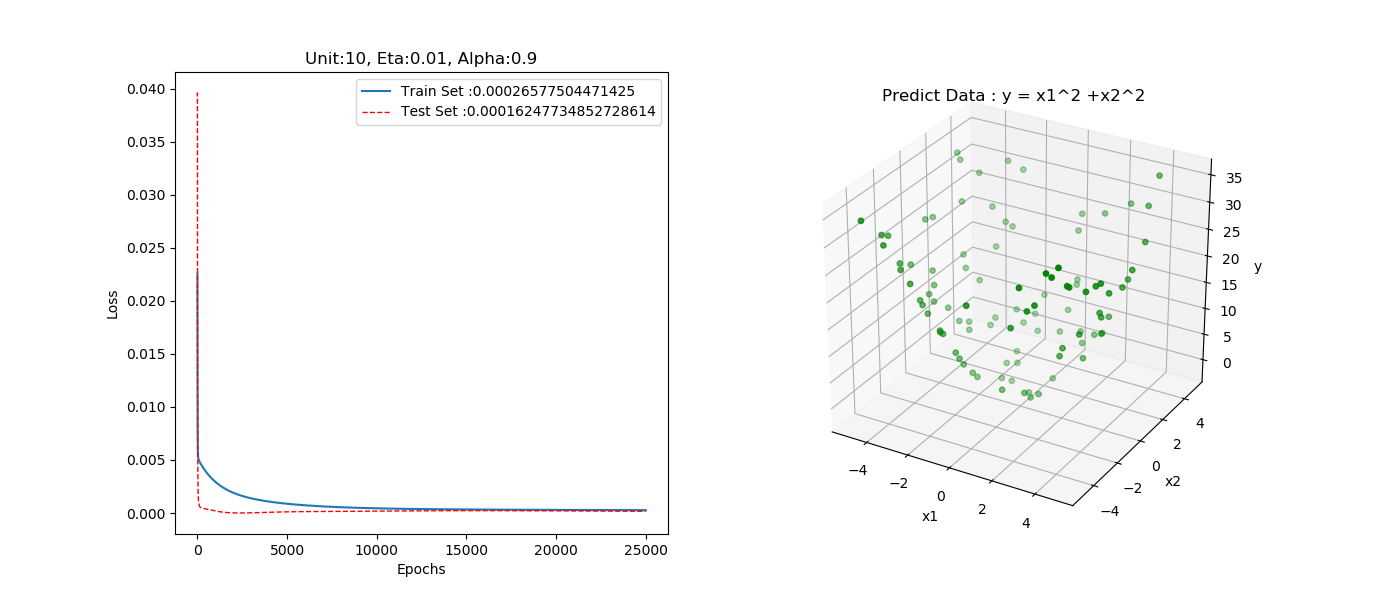


圖一

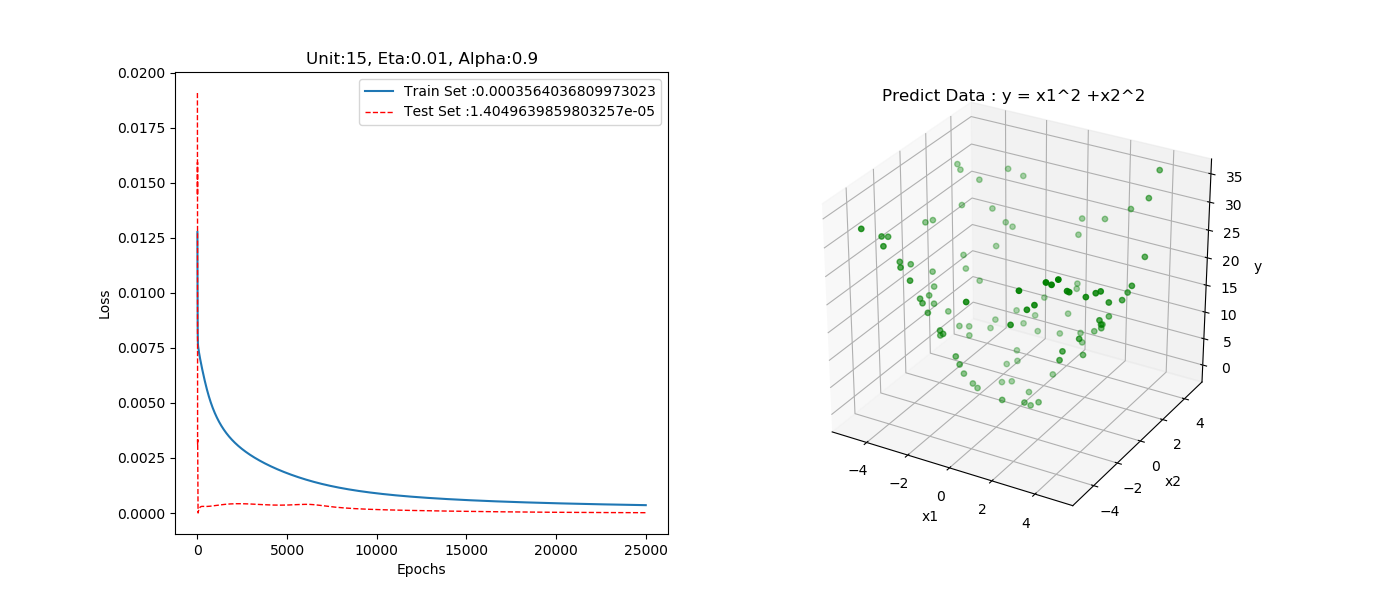


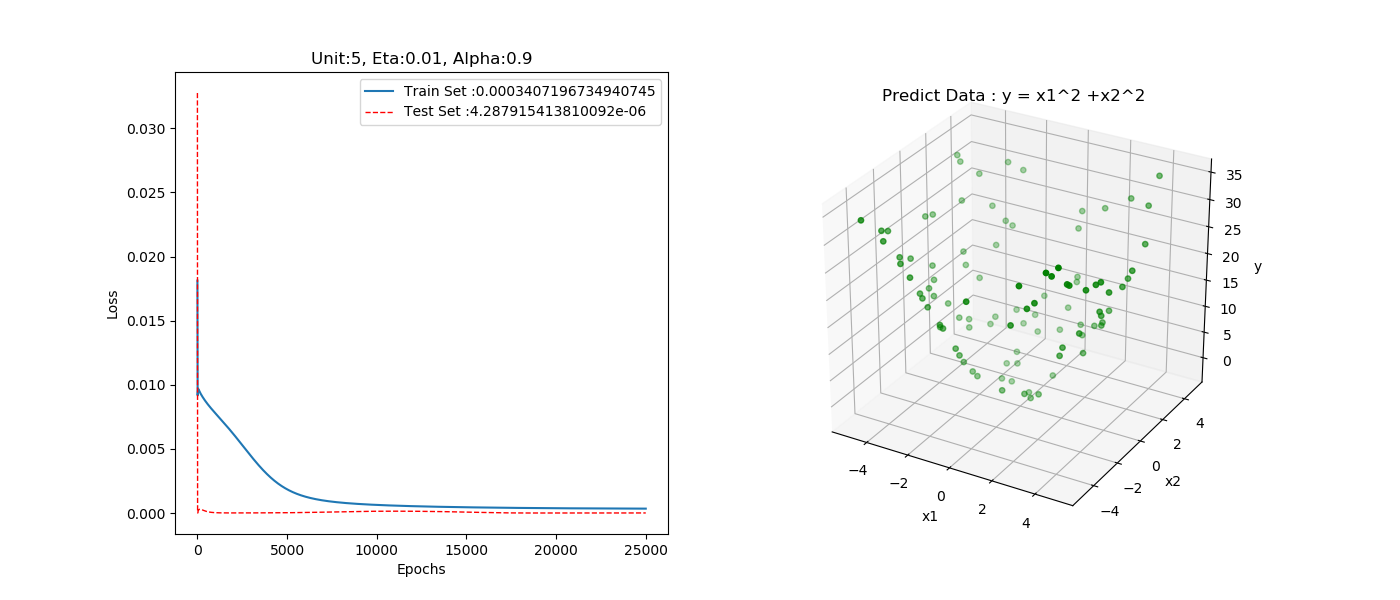
圖二

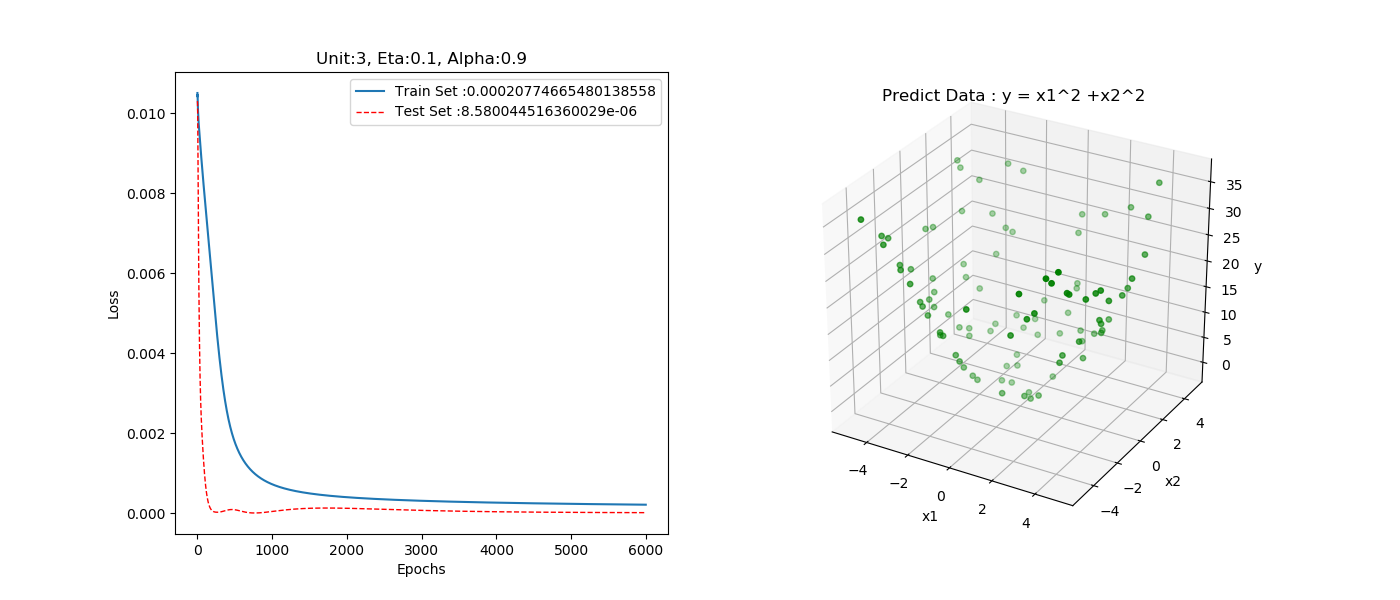
圖三

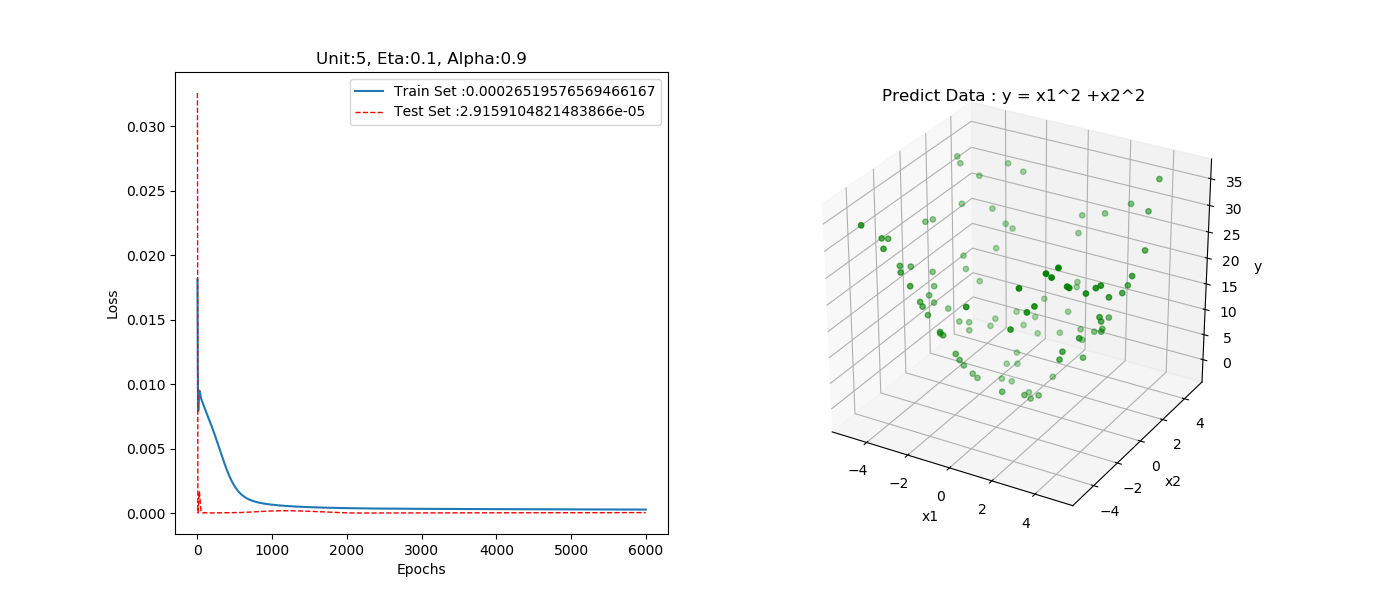
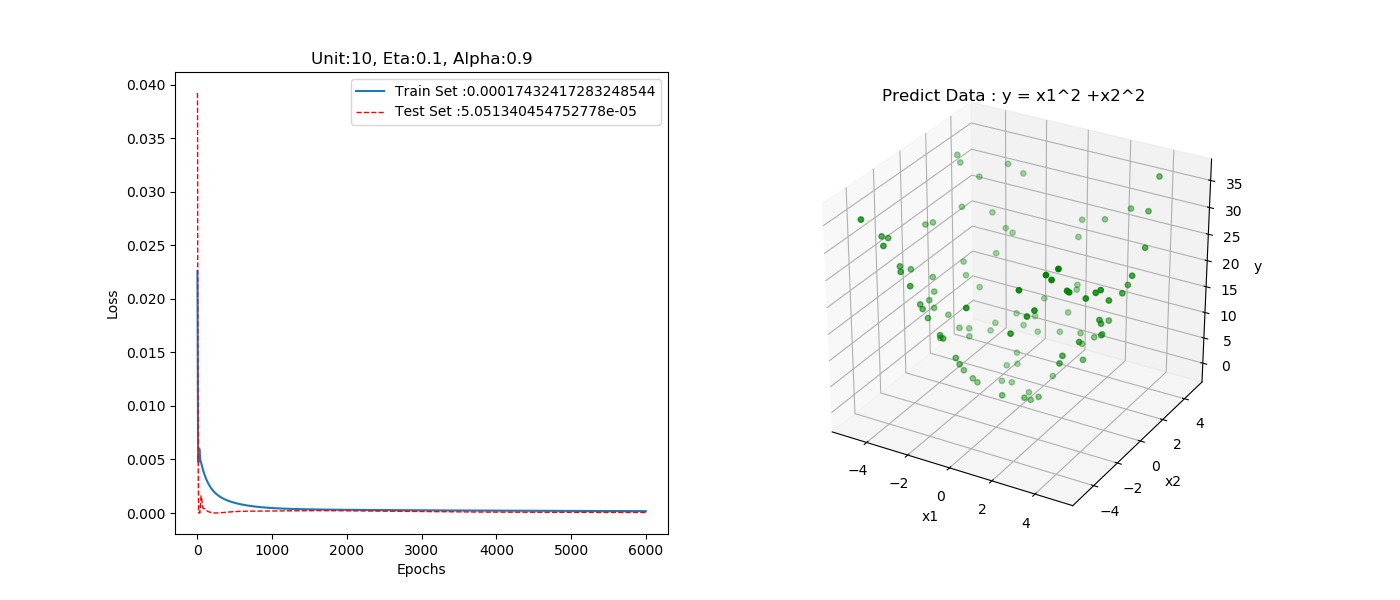


圖四

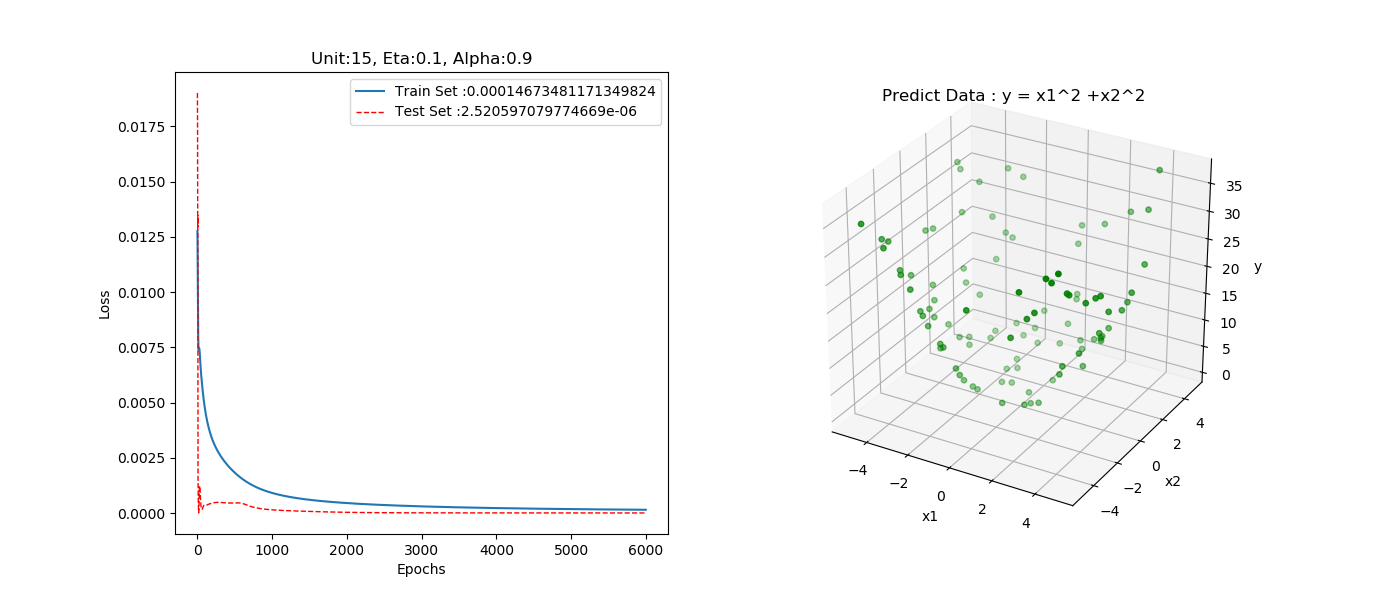
圖五

圖六

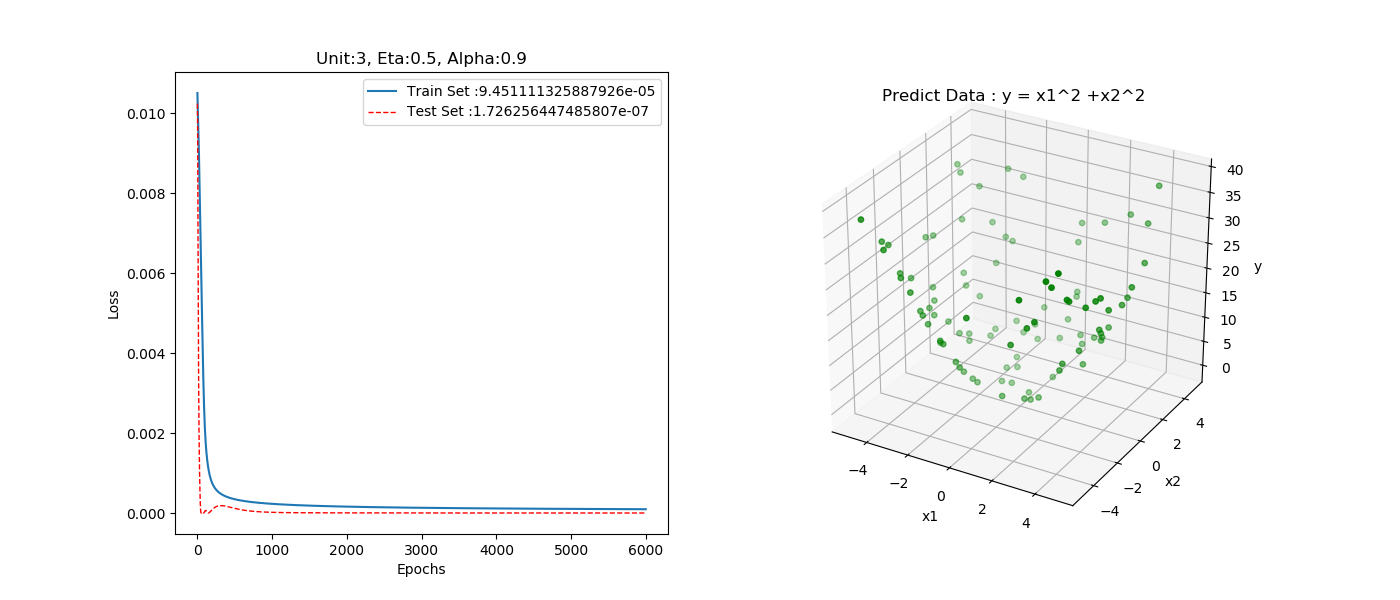
圖七

圖八

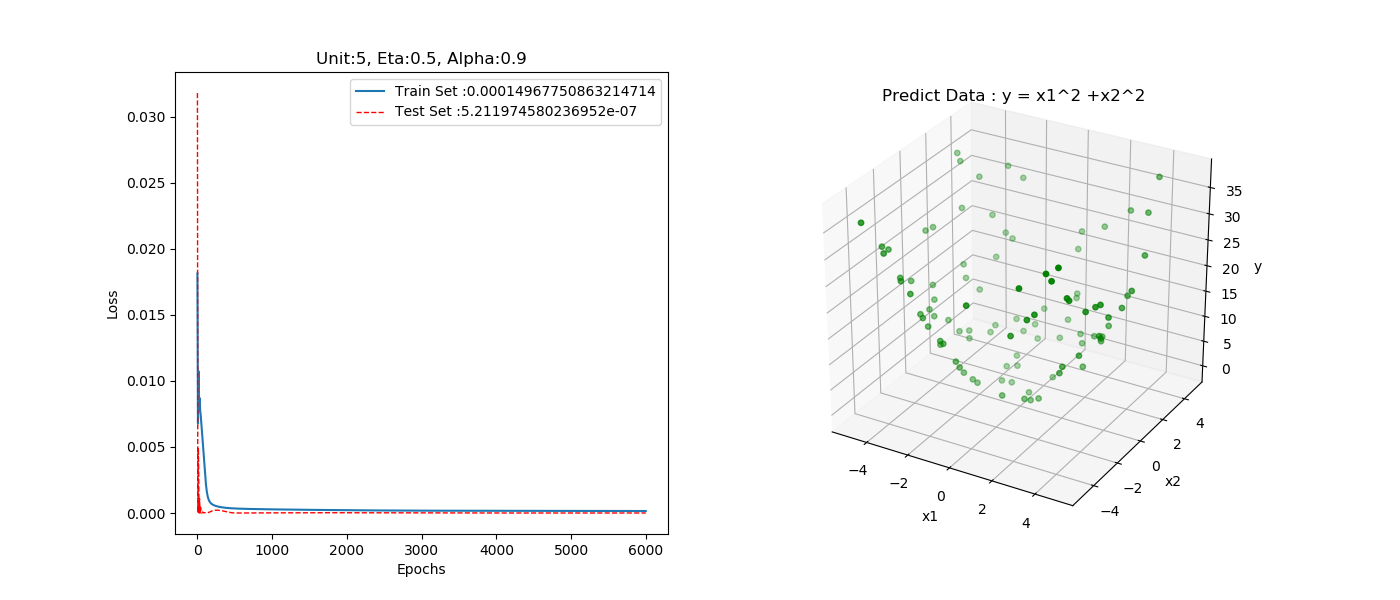
圖九

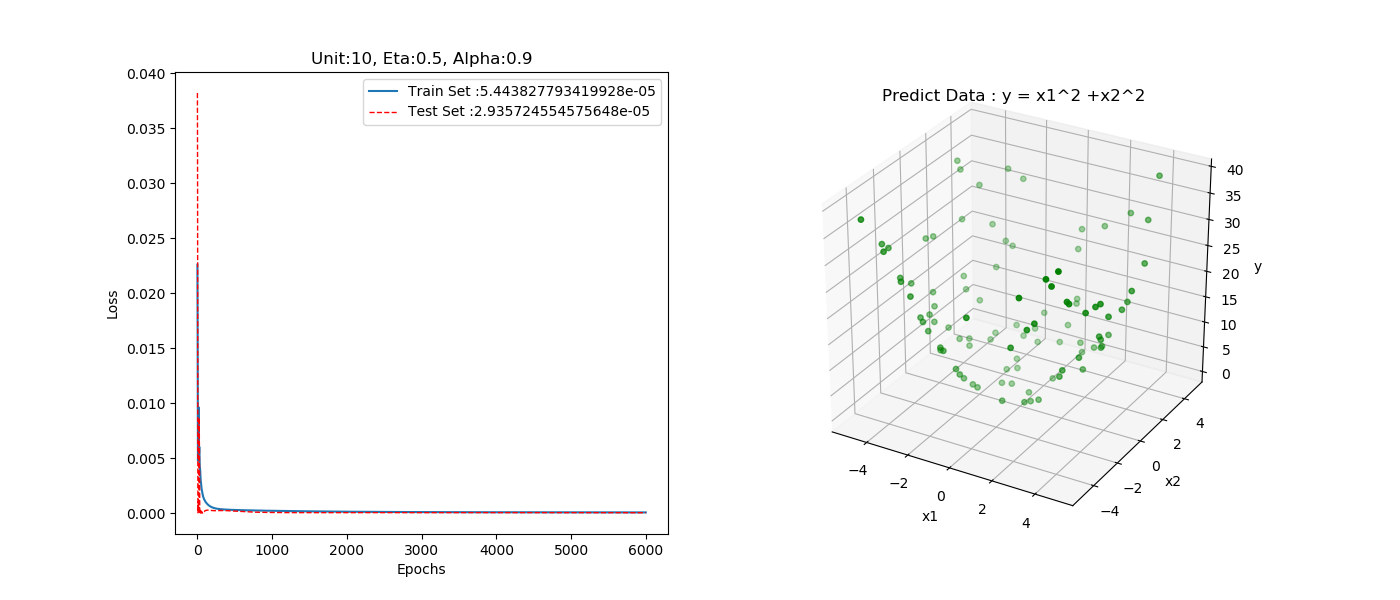


圖十

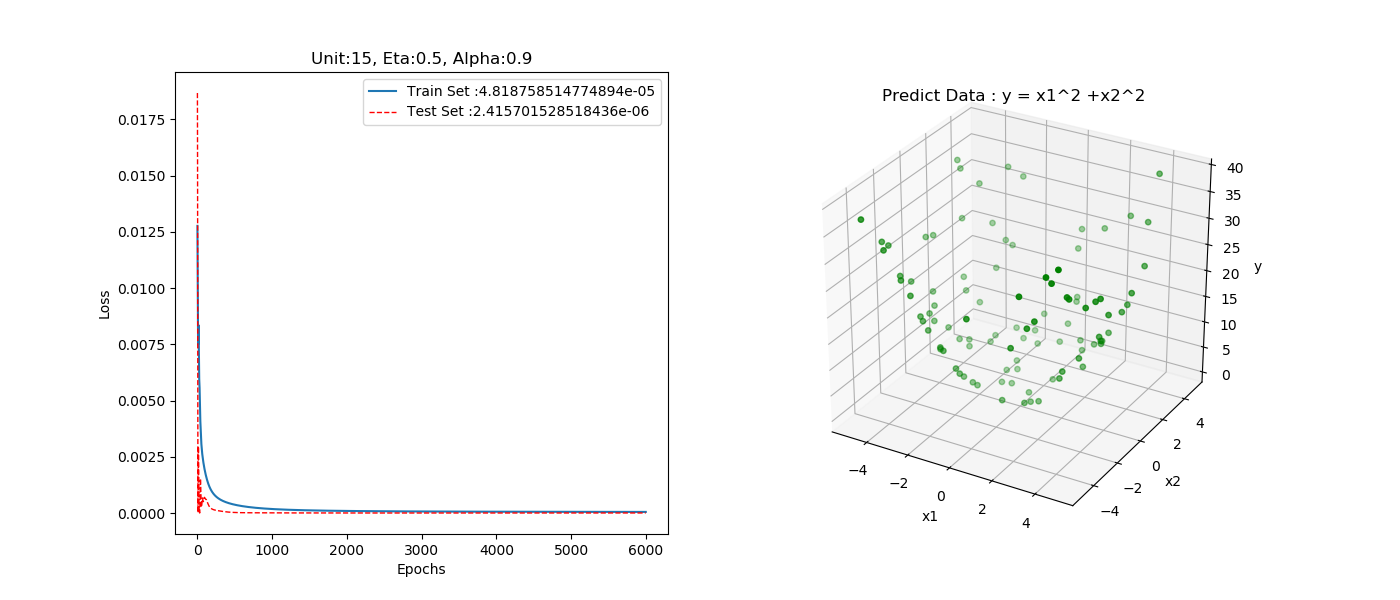


圖十一

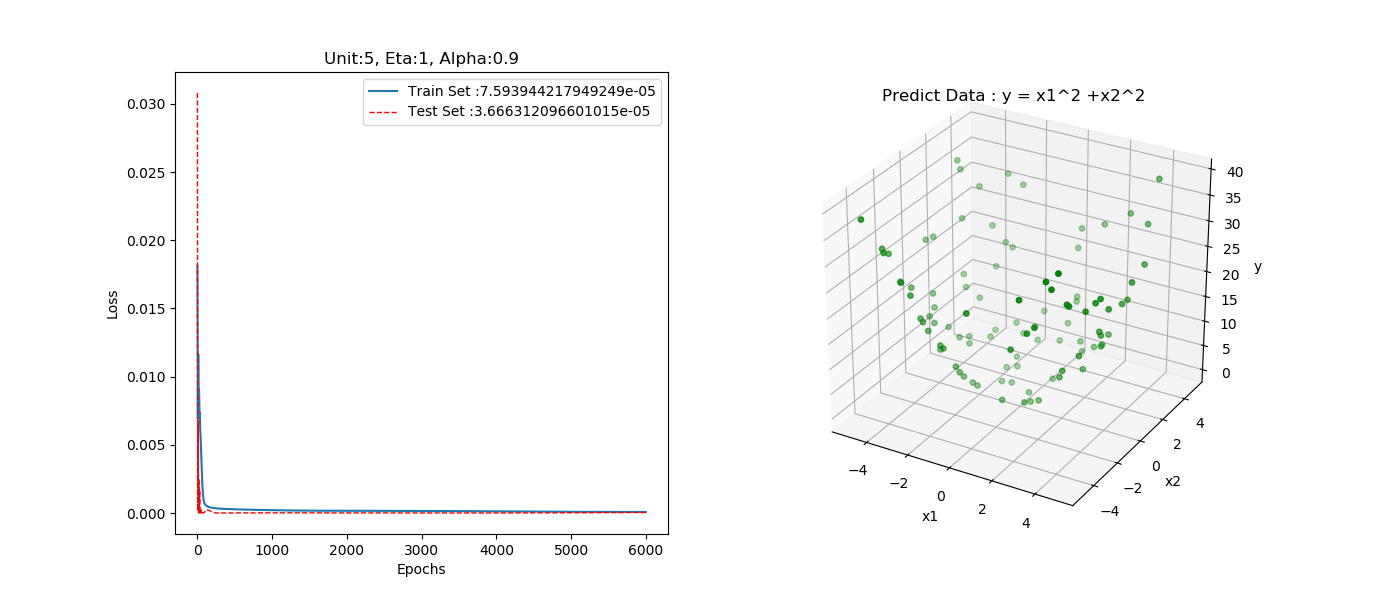


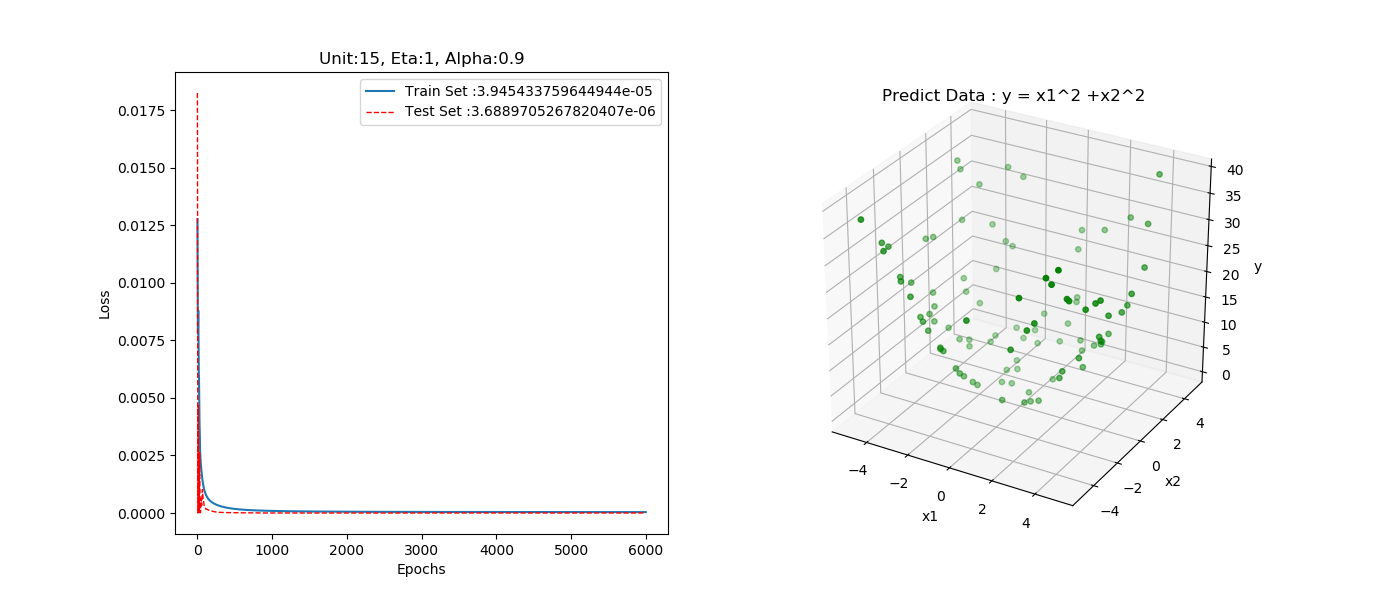
圖十二

圖十三

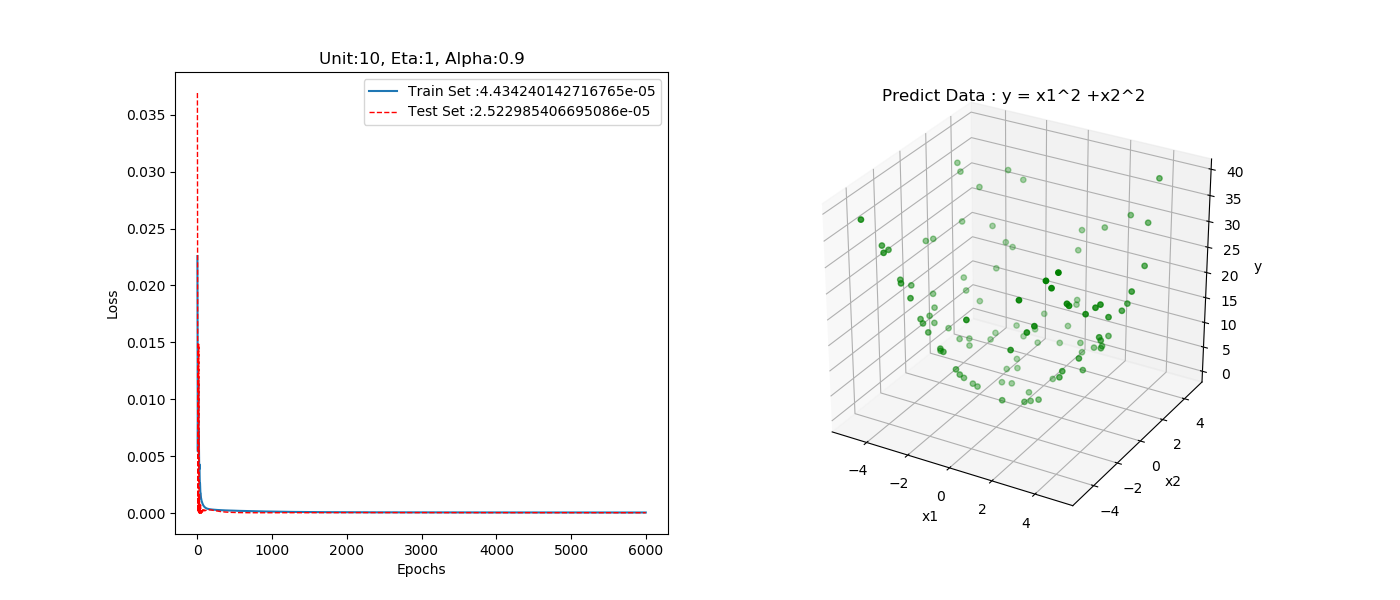


圖十四

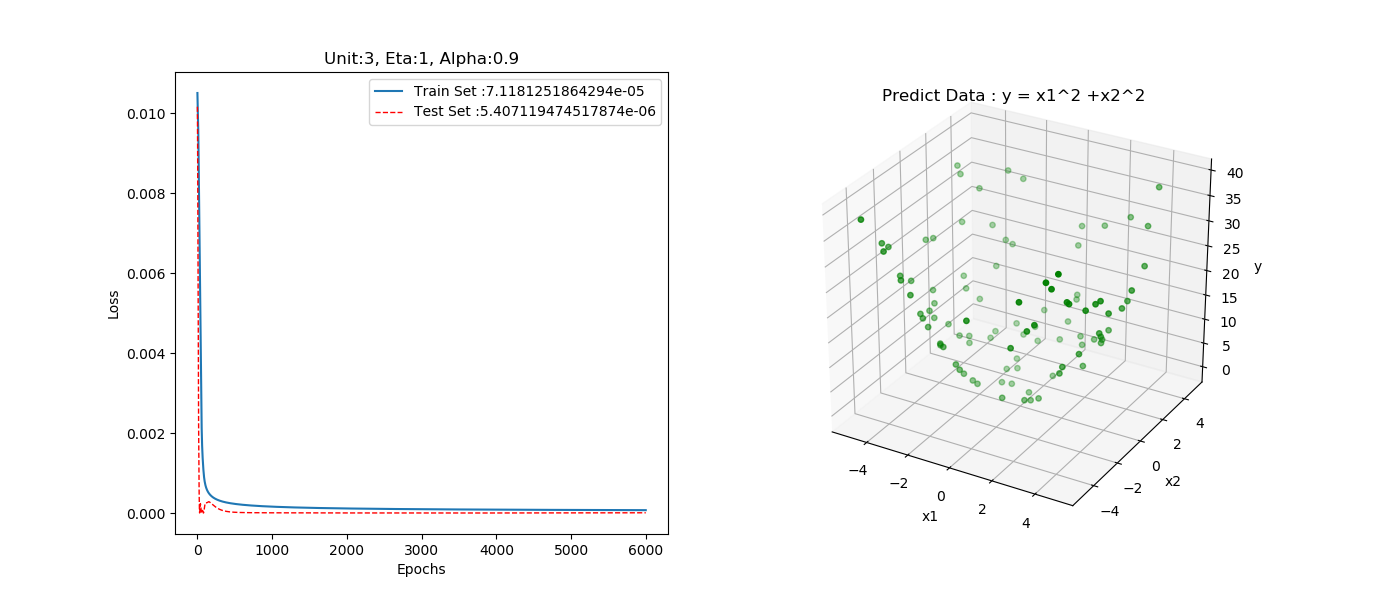


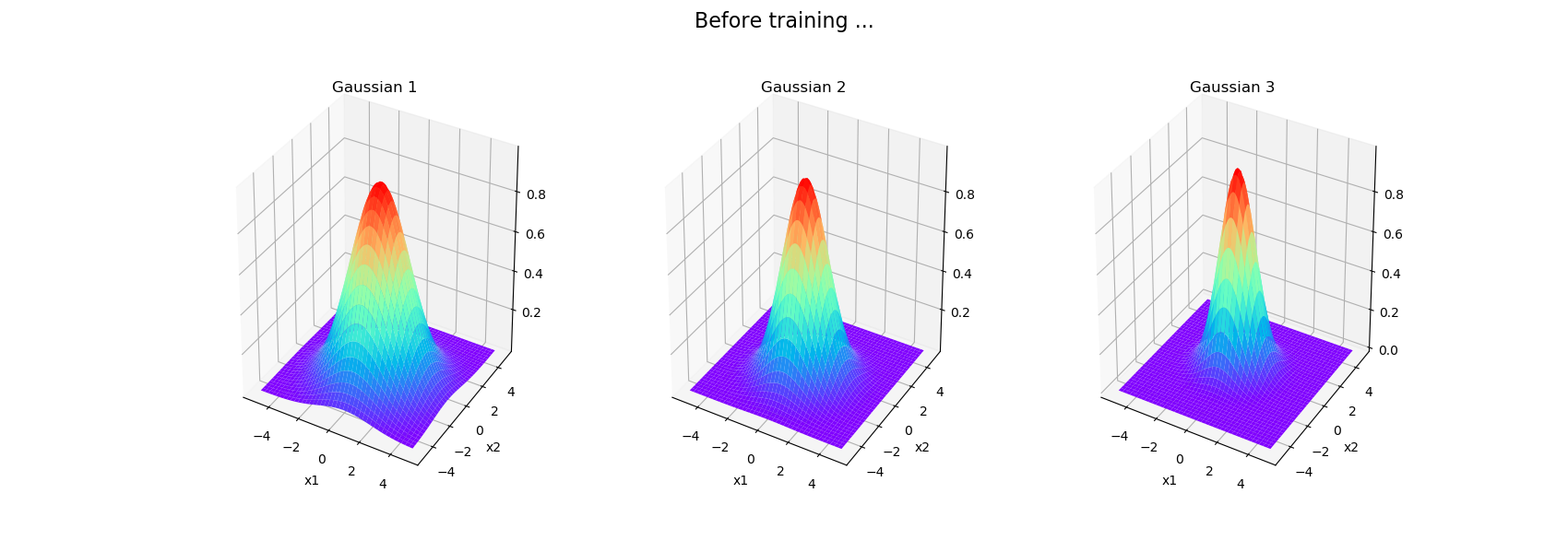
圖十五

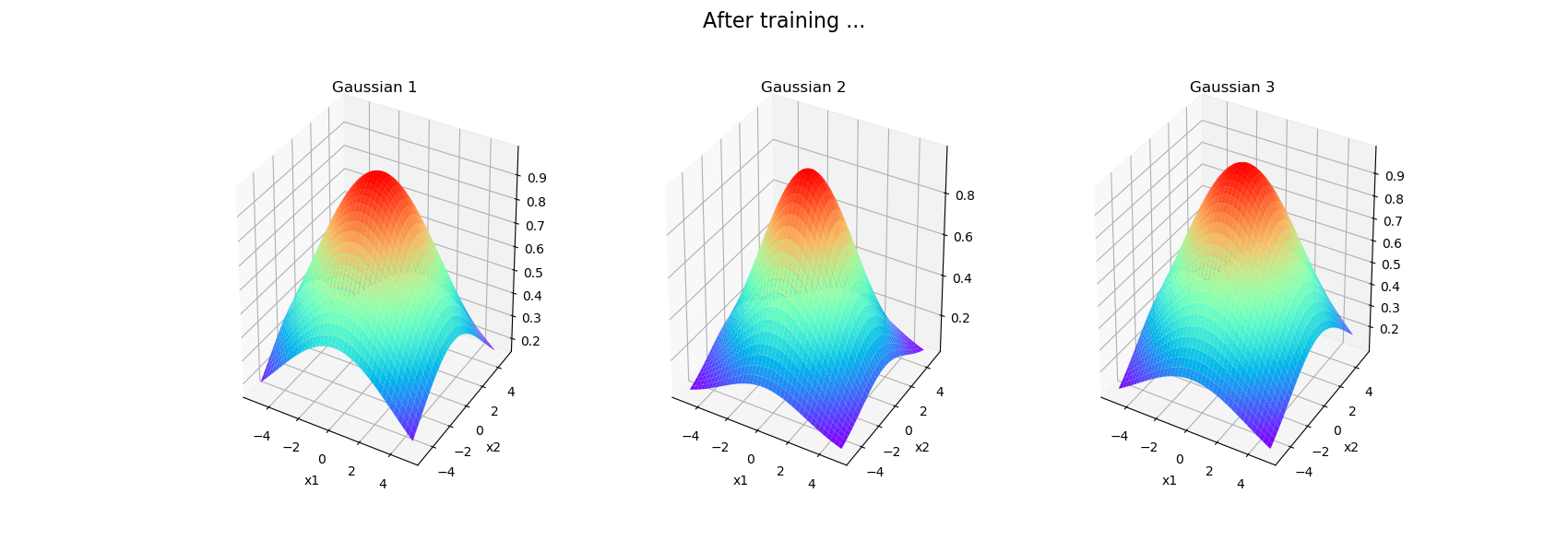
圖十六



圖十七

圖十八

圖十九



圖二十

圖二十一

圖二十二

## tables

#### batch learning:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Unit | Eta | Alpha | Training\_loss | Predict\_loss | Epoch |
| 3 | 1 | 0.9 | 7.12E-05 | 7.57E-05 | 6000 |
| 5 | 1 | 0.9 | 7.59E-05 | 9.13E-05 | 6000 |
| 10 | 1 | 0.9 | 4.43E-05 | 5.97E-05 | 6000 |
| 15 | 1 | 0.9 | 3.95E-05 | 6.24E-05 | 6000 |
| 3 | 0.5 | 0.9 | 9.45E-05 | 9.85E-05 | 6000 |
| 5 | 0.5 | 0.9 | 0.000149678 | 0.000158436 | 6000 |
| 10 | 0.5 | 0.9 | 5.44E-05 | 7.24E-05 | 6000 |
| 15 | 0.5 | 0.9 | 4.82E-05 | 7.34E-05 | 6000 |
| 3 | 0.1 | 0.9 | 0.000207747 | 0.000210934 | 6000 |
| 5 | 0.1 | 0.9 | 0.000265196 | 0.000268353 | 6000 |
| 10 | 0.1 | 0.9 | 0.000174324 | 0.000202364 | 6000 |
| 15 | 0.1 | 0.9 | 0.000146735 | 0.000188503 | 6000 |
| 3 | 0.01 | 0.9 | 0.000593376 | 0.000621084 | 12000 |
| 5 | 0.01 | 0.9 | 0.000532658 | 0.000577442 | 12000 |
| 10 | 0.01 | 0.9 | 0.000381139 | 0.000423956 | 12000 |
| 15 | 0.01 | 0.9 | 0.000737516 | 0.000932002 | 12000 |
| 3 | 0.01 | 0.9 | 0.000340487 | 0.000346204 | 25000 |
| 5 | 0.01 | 0.9 | 0.00034072 | 0.000345726 | 25000 |
| 10 | 0.01 | 0.9 | 0.000265775 | 0.000291238 | 25000 |
| 15 | 0.01 | 0.9 | 0.000356404 | 0.000434567 | 25000 |
| 3 | 0.01 | 0.8 | 0.000494071 | 0.00050685 | 35000 |
| 5 | 0.01 | 0.8 | 0.001028461 | 0.001148848 | 35000 |
| 10 | 0.01 | 0.8 | 0.000531462 | 0.000528898 | 35000 |
| 15 | 0.01 | 0.8 | 0.0005006 | 0.000545143 | 35000 |

表一

|  |  |  |
| --- | --- | --- |
| Mean | | Sigma |
| Before Training | | |
| 0.57852759 | **-0.95370448** | **1.71650483** |
| 0.19531336 | **-0.67779949** | **1.33621775** |
| -0.10899176 | **0.23392018** | **1.05920794** |
| After Training | | |
| -0.01876763 | **-0.271125** | **3.79853667** |
| -0.08042581 | **0.06549378** | **2.79223293** |
| -0.09161632 | **0.69190515** | **3.56217249** |

表二( 對應圖十八之RBF 參數 (m, sigma) )

# **conclusion - analysis and comparison**

此系統有兩輸入，故RBF之mean 有二，由( 圖十八、圖十九及表二) ，RBF個數為3時，Gaussian 函數經過訓練6000 epochs後，sigma值明顯增大，平均值也有稍許變動，loss 收斂在7E-5。

由圖二至圖十八來看，RBF個數少、alpha大、eta大收斂快，並從表一可觀察到，eta在0.01~1、alpha在0.8~0.95、RBF個數在3~15所訓練之結果皆可同時使traning loss 及testing loss 收斂。

RBFN與BPNN相比，由於RBFN 需更新 m, sigma，而BPNN只需更新weight，故RBFN之訓練時間約為BPNN之訓練時間三倍左右。