تمرین سری 5 مبانی سیستم‌های هوشمند

ریحانه هادی پور

9731123

Question 1:

این دیتاست از 150 نمونه(داده) با 4 ویژگی تشکیل شده است. این 4 ویژگی عبارتند از: طول کاسبرگ(sepal length)، عرض کاسبرگ (sepal width)، طول گلبرگ (petal length)و عرض گلبرگ (petal width)تشکیل شده است که همگی بر حسب سانتی‌متر در دیتاست ذخیره شده است. همچنین از این 150 نمونه، 50 نمونه اول برچسب(کلاس) Iris setosa و 50 نمونه دوم برچسب Iris Versicolour و 50 نمونه سوم نیز در کلاس Iris Virginica ذخیره شده است.

دو نکته در بدست آوردن خروجی مطلوب در شبکه عصبی RBF حائز اهمیت می‌باشد. اولین نکته فضای داده‌های آموزشی است که باید کل فضای ورودی را پوشش دهند به منظور کلاس‌بندی داده‌های ورودی به دلیل آنکه داده‌های هر کلاس پشت سر هم می‌باشند باید به طریقی داده‌های ورودی را از نظر ترتیب بهم ریخت تا در داده‌های آموزش هر کلاس دارای فراوانی کافی باشد. پس ابتدا داده ها را فراخوانی کرده و به صورت سطری شافل میکنیم:

```
function y = shuffle(data)
n = size(data,1); % number of rows
m = size(data,2); % number of columns
t = zeros(n,m);
index = randperm(n);
for i = 1:n
    t(i,:) = data(index(i),:);
end
y = t;
end
```

سپس داده های شافل شده را 75 درصد ترین و مابقی را تست میگیریم. ستون آخر که تارگت ها هستند نیز جدا میکنیم.

Load data

```
data = xlsread('iris.xlsx');

data = shuffle(data);

rate_train = 0.75;

n = size(data,2); % number of columns
num_data = size(data,1); % number of rows

num_train  = round(num_data*rate_train);  % number of rows of train data
num_test   = num_data - num_train;        % number of rows of test data
data_train = data(1: num_train, :);       % train data
data_test  = data(1 + num_train: end, :); % test data

X_train = data_train(:,1:n-1);
X_test = data_test(:,1:n-1);

target_tarin = data_train(:,n);
target_test  = data_test(:,n);

target_tarin_onehot = make_onehot(target_tarin,3);
target_test_onehot = make_onehot(target_test,3);
```

برای رسم confusion matrix با دستور plotconfusion لازم است که تارگت ها به صورت وان هات باشند تا تعداد کلاس ها به درستی تشخیص داده شوند. برای همین شبکه را با تارگت های وات هات شده ترین میکنیم.

Make OneHot

```
function y = make_onehot(data,m)
w = size(data,1); % number of rows
temp = zeros(w,m);
for i = 1:w
    temp(i,data(i))=1;
end
y = temp;
end
```

Train the Network

```
goal=0;

spread=1;

MN=15;

DF=1;

net = newrb(X_train',target_tarin_onehot',goal,spread,MN,DF);

y_train = sim(net, X_train');
y_train = round(y_train);

y_test = sim(net, X_test');
y_test = round(y_test);

figure
plotconfusion(target_tarin_onehot',y_train)

title('Confusion Matrix of Train Data')

figure

plotconfusion(target_test_onehot',y_test)

title('Confusion Matrix of Test Data')

view(net);
```

برای ترین کردن شبکه با RBF از دستور newrb استفاده میکنیم. این دستور مرحله به مرحله نورون به لایه میانی شبکه عصبی rbf اضافه میکند تا به خطای مورد نظر برسد. و یک new radial basis network ترین شده برمیگرداند. پارامتر های آن دیتای ترین، تارگت های آن، خطای نهایی مورد نظر، واریانس تابع گوسی مورد استفاده، ماکسیمم تعداد نورون های لایه ی میانی هستند. پارامتر آخر تنها برای نمایش این است که هر نرونی که اضافه میشود چه خطایی داریم. این پارامتر تعداد این نمایش ها را مشخص میکند.

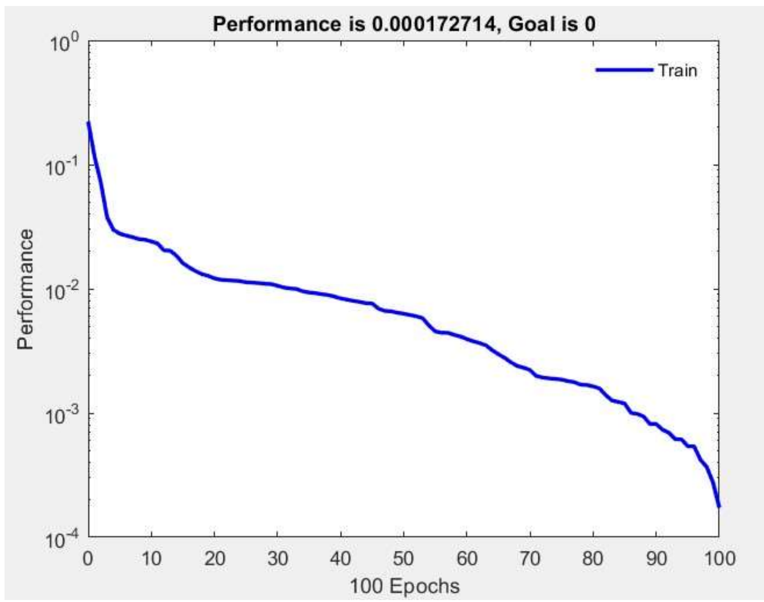
در اینجا چون به صورت اتومات مراکز دسته تعیین شده و شبکه ترین میشود احتیاجی به استفاده از k means برای پیدا کردن مراکز دسته ها نیست.

حال goal را صفر گذاشته و سپس تعداد ماکسیمم نورون را تغییر میدهیم و 4بار برای هر کدام ترین میکنیم و نتایج را مقایسه میکنیم. به دلیل اینکه goal صفر قرار داده میشود تعداد نورون ها به همان ماکسیمم مقداری که گذاشتیم میرسد و سعی میکند تا خطای خود را صفر کند. ستون های جدول زیر را با مقادیر overall accuracy پر میکنیم. ستون آخر نیز MSE نهایی به صورت میانگین بین 4 بار ران گرفتن است. داده های کلاس اول همیشه به درستی تشخیص داده میشدند.

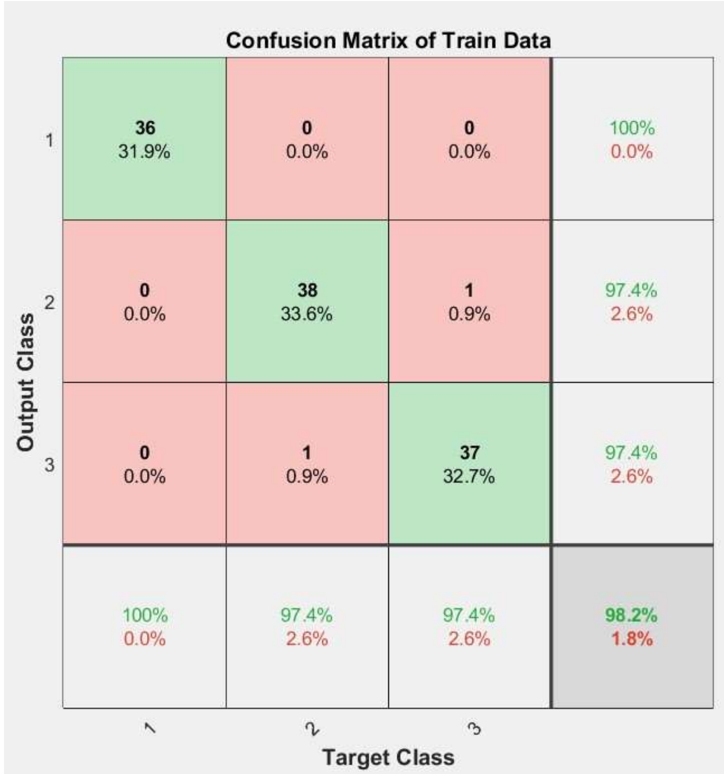
MSE	Min Test	Max Test	Min Train	Max Train	MN
0.05	82.3	89.2	86.7	94.6	**2**

0.04	91.9	97.3	93.8	95.6	**3**
0.03	91.9	97.3	96.5	98.2	**5**
0.02	94.6	97.3	97.3	100	**10**
0.01	91.9	100	97.3	99.1	**15**
0.01	94.6	99.1	98.2	100	**20**
0.005	86.5	94.6	99.1	100	**50**
0.0001	64.9	86.5	100	100	**100**

شبکه با 100 نرون واضح است که overfit شده است. تعداد نرون مناسب که خطای خوبی هم داشته باشد را میتوان حوالی 15 نرون در نظر گرفت.



Confusion Matrix برای 15 نورون:



نتایج حاصل از ماتریس Confusion داده‌های آموزش:

۱) ٪۱۰۰ داده‌های کلاس یک را درست تشخیص داده است.(Target)

۲) ٪۹۷.۴ داده‌های کلاس دو را درست تشخیص داده و ٪۲.۶ (۱ داده) آن را به اشتباه کلاس سه تشخیص داده است.(Target)

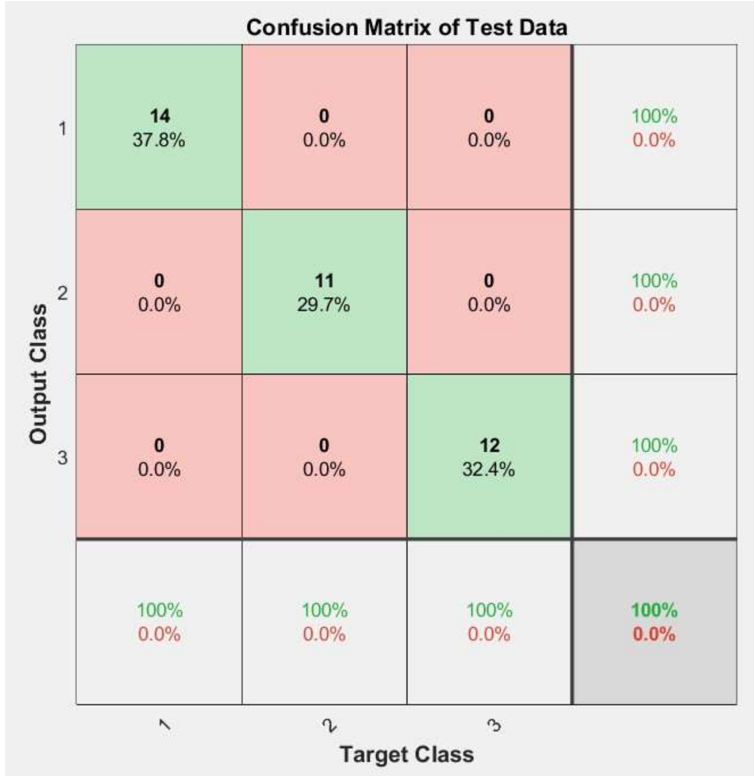
۳)٪۹۷.۴ داده‌های کلاس سه را درست تشخیص داده ٪۲.۶ (۱ داده) آن را به اشتباه کلاس دو تشخیص داده است.(Target)

۴) ٪۱۰۰ داده‌ها را به درستی کلاس یک تشخیص داده است. (Output)

۵) ٪۹۷.۴ داده‌ها را به درستی کلاس دو تشخیص داده و ٪۲.۶ (۱ داده) آن را به اشتباه کلاس دو تشخیص داده است. (Output)

۶) ٪۹۷.۴ داده‌ها را به درستی کلاس سه تشخیص داده ٪۲.۶ (۱ داده) آن را به اشتباه کلاس سه تشخیص داده است. (Output)

۷) ٪۹۸.۲ از کل داده‌های آموزش را درست تشخیص داده و ٪۱.۸(۲ داده از ۱۱۳داده) از آن را اشتباه تشخیص داده است.



نتایج حاصل از ماتریس Confusion داده‌های تست:

۱) ٪۱۰۰ داده‌های کلاس یک را درست تشخیص داده است.(Target)

۲) ٪۱۰۰ داده‌های کلاس دو را درست تشخیص داده است.(Target)

۳) ٪۱۰۰ داده‌های کلاس سه را درست تشخیص داده است.(Target)

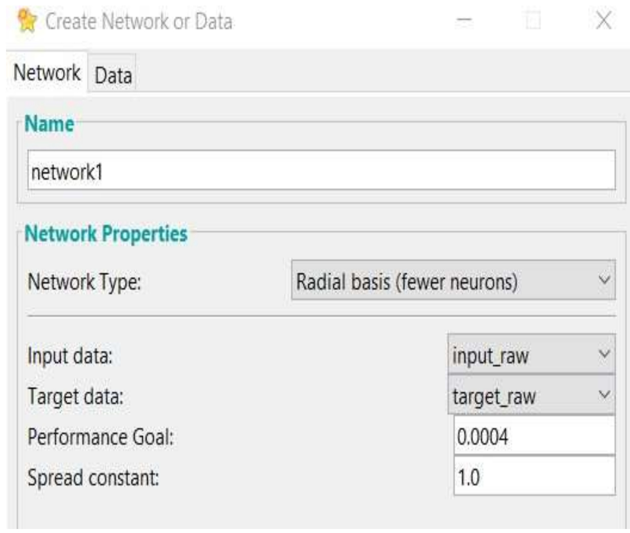
۴) ٪۱۰۰ داده‌ها را به درستی کلاس یک تشخیص داده است.(Output)

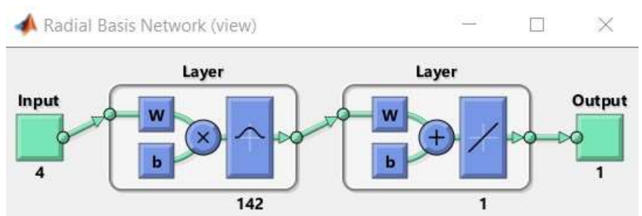
۵) ٪۱۰۰ داده‌ها را به درستی کلاس دو تشخیص داده است.(Output)

۶) ٪۱۰۰ داده‌ها را به درستی کلاس سه تشخیص داده است.(Output)

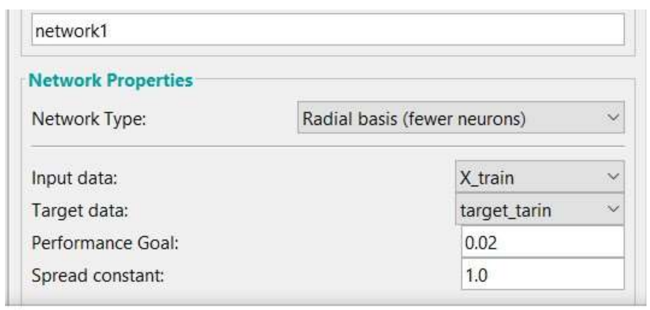
۷) ٪۱۰۰ کل داده‌های تست(۳۷ داده) را درست تشخیص داده است.

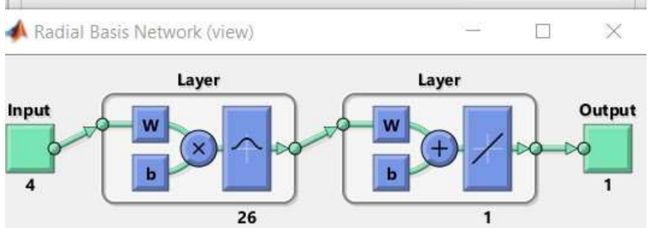
حال این سوال را با استفاده از nntool نیز حل میکنیم.





میبینیم که با خطای انتخابی تعداد نرون زیادی را درنظر گرفته است.





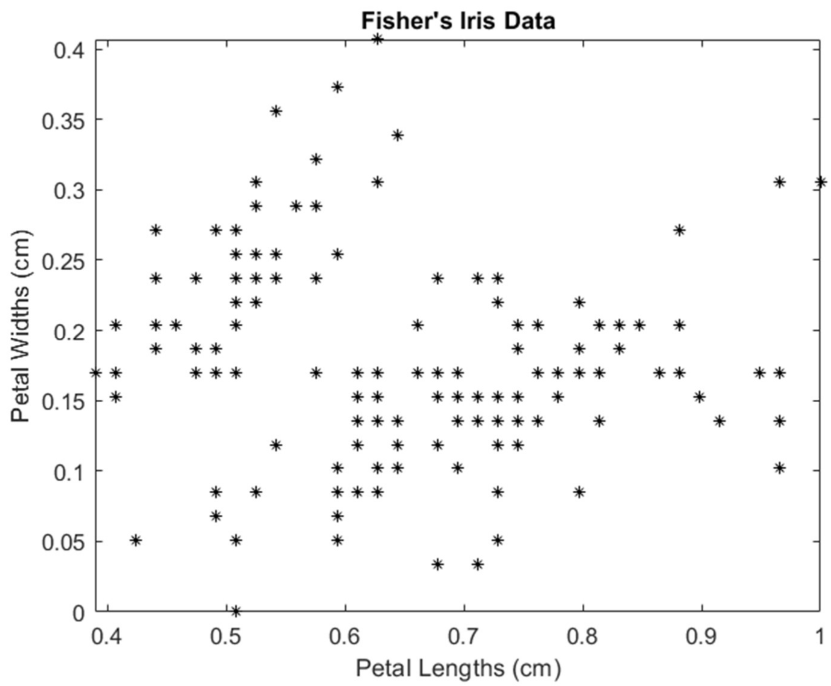
از شبکه خروجی میگیریم و روی دیتای تست آن را امتحان میکنیم.

```
Command Window
New to MATLAB? See resources for Getting Started.
ans =

  Columns 1 through 19

     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0

  Columns 20 through 37

     0     0     0     0     0     0     0     0     0     0     1     0     0     0     1     0     0     0
```

تنها به یک بار تست اکتفا کرده. که نتایج قابل قبولی نیز داشت.

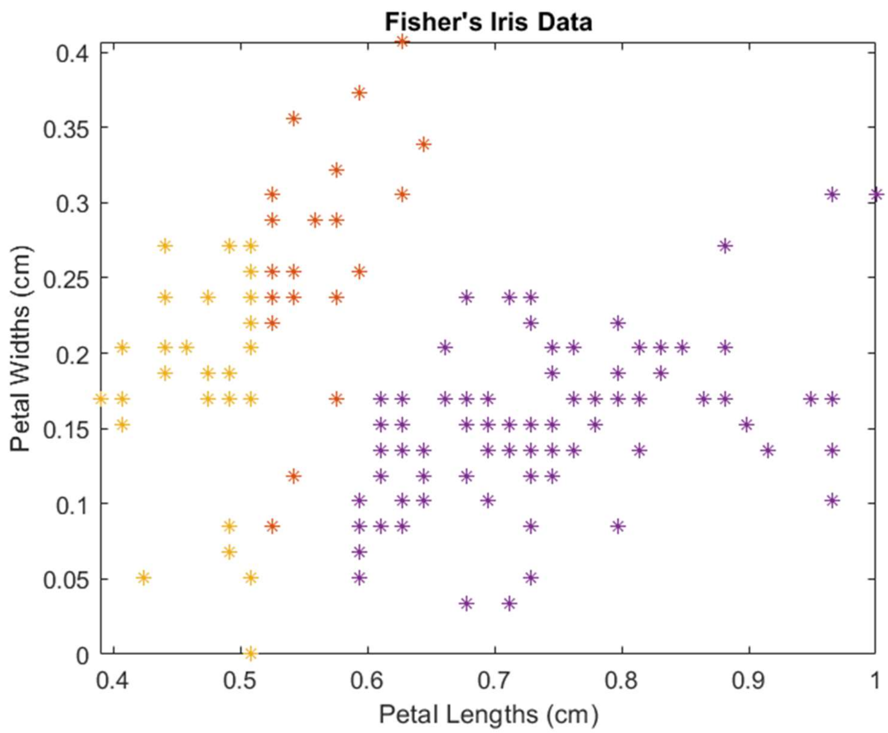
Question 2:

```
clc
clear all
% load Data
 load iris.dat
X = iris(:,1:2);    % column:samples, line:feature vector
[n,d] = size(X);    % n=number of samples, d:dimension of feature vector
% range transration [0~1]
 [a,b] = normalization(X);
 X = (X-a)/b;
%  when X is two-dimension, Draw to the plane
if d==2
   figure(1)
   plot(X(:,1),X(:,2),'k*','MarkerSize',5);
   title 'Fisher''s Iris Data';
   xlabel 'Petal Lengths (cm)';
   ylabel 'Petal Widths (cm)';
   axis([min(X(:,1)) max(X(:,1)) min(X(:,2)) max(X(:,2))])
   hold on
end
```



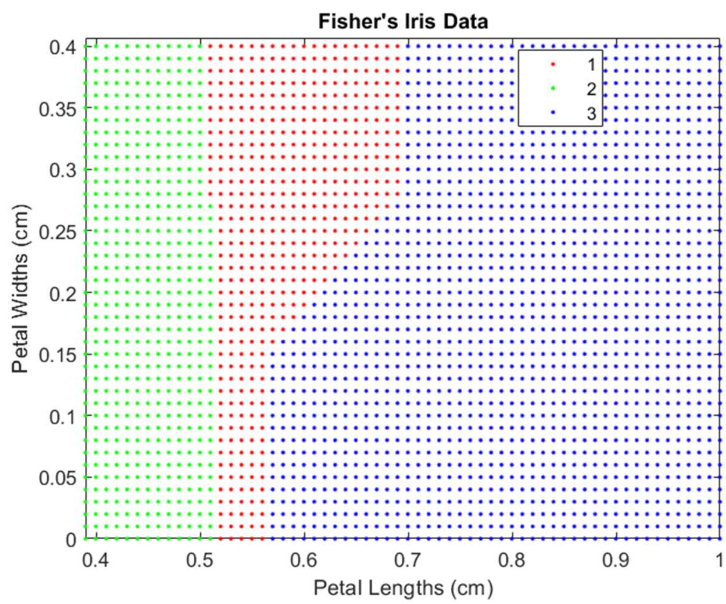

```
K = 3;  % number of crass
max_iter = 40;  % Maximum number of attempts
%  k-means clustering function
```

```
[idx, mean_val] = kmeansc(X,K,max_iter);
% Add color based on k-means clustering
if d == 2
   for j=1:K
      plot(X(idx == j,1),X(idx == j,2),'*')
   end
```




```
   x1 = min(X(:,1)):1e-2:max(X(:,1));
   x2 = min(X(:,2)):1e-2:max(X(:,2));
   [x1G,x2G] = meshgrid(x1,x2);
```

```
XGrid = [x1G(:),x2G(:)];
  for i = 1:size(XGrid,1)
  for j = 1:K
    % To calculate the distance between?mean_val created by kmeansc function
    distance_XGrid(i,j) = min(sum(abs(XGrid(i,:) - mean_val(j,:)))');
  end
  [D_X_Grid(i),idx_XGrid(i)] = min(distance_XGrid(i,:)');
end
figure
gscatter(XGrid(:,1),XGrid(:,2),idx_XGrid)
axis([min(X(:,1)) max(X(:,1)) min(X(:,2)) max(X(:,2))])
title 'Fisher''s Iris Data';
xlabel 'Petal Lengths (cm)';
ylabel 'Petal Widths (cm)';
```




```
function [idx, mean_val] = kmeansc(X,K,iteration)
% n=number of samples, d:dimension of feature vector
[n,d] = size(X);
% Decide randomly an initial value from X
P = randperm(n);
% initial value of center vector
mean_val = X(P(1,1:K)',:);
for iter = 1:iteration
  for i = 1:n
    for j = 1:K
      distance(i,j) = sum(abs(X(i,:) - mean_val(j,:)));
    end %EOF j
    [D(i,:),idx(i,:)] = min(distance(i,:)');
  end %EOF i

  for j=1:K
    mean_val(K,:) =  sum(X(idx==K,:)) / sum(idx==j);
  end

end
end
function [a,b] = normalization(x)
a = min(min(x));
b = max(max(x-a));
end
```

```
Question 3:
```

شناسایی سیستم عمل استفاده از مدل ریاضی مناسب و الگوریتم های یادگیری می باشد به منظور نگاشت داده های تجربی به طوری که خطای بین مدل و خروجی مطلوب سیستم حداقل شود .

مدل های ARMA مدل های رگرسیون خطی هستند که از معادلات مختلفی برای ارتباط خروجی مدل با ورودی حال و گذشته و همچنین خروجی های گذشته استفاده می کنند. یک مدل ARMA زمان گسسته به طور کلی به صورت زیر است:

$$y(k) = a_1 y(k-1) + \cdots + a_n y(k-n) + b_0 x(k) + \cdots + b_m x(k-m) \qquad (1)$$

که به صورت زیر نیز قابل نمایش است :

$$y(k) = \sum_{i=0}^{m} b_i x(k-i) + \sum_{j=1}^{n} a_j y(k-j) \qquad (2)$$

که در این روابط $x(k)$ ورودی مدل،$y(k)$ خروجی مدل می باشند و همچنین k شماره نمونه و b_i و a_j پارامترهای مدل هستند. تابع تبدیل به فرم زمان گسسته و به کمک تبدیل z به صورت زیر می باشد:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_m z^{-m}}{1 - a_1 z^{-1} - \cdots - a_n z^{-n}} \qquad (3)$$

معادله (2) را می توان به فرم برداری به صورت $y(k) = \theta^T \vartheta(k)$ نمایش داد که در آن :

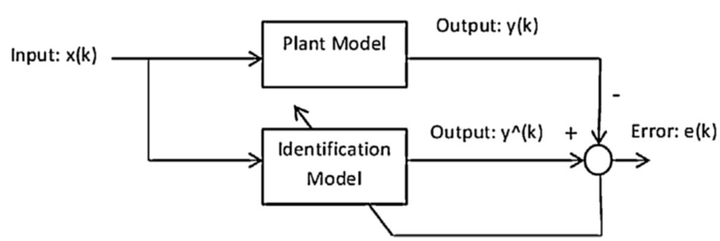
$$\theta^T = [a_1, \dots, a_n, b_0, \dots, b_m]$$

بردار پارامتر،و:

$$\vartheta(k) = [y(k-1), \dots, y(k-n), x(k), \dots, x(k-m)]^T$$

بردار اندازه گیری می باشد. شناسایی سیستم پارامتری منجر به تقریب پارامترهای b_i و a_j خواهد شد که با داشتن بردار پارامتر تخمین زده شده:

$$\hat{\theta}^T = [\hat{a}_1, \dots, \hat{a}_n, \hat{b}_0, \dots, \hat{b}_m]$$



شکل ۱ - بلوک دیاگرام کلی برای برای شناسایی سیستم

می توان خروجی تخمینی $\hat{y}(k) = \hat{\theta}^T \vartheta(k)$ را محاسبه کرد. هدف یافتن پارامترهای تخمینی مناسبی می باشد به منظور حداقل نمودن خطا بین خروجی $y(k)$ و خروجی تخمین زده شده $\hat{y}(k)$. آ

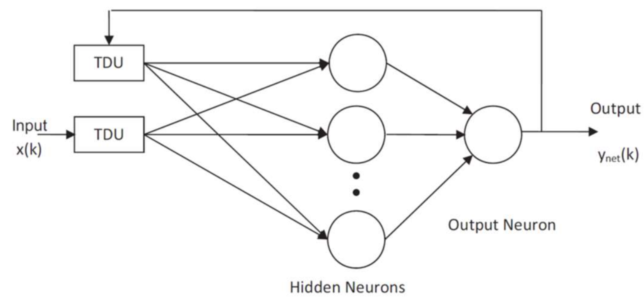
از الگوریتم هایی چون حداقل مربعات (LS)،حداقل مربعات بازگشتی (RLS)،خطای پیش بینی بازگشتی (RPEM) برای مینیمم کردن خطا بین خروجی واقعی و تخمینی بهره برد .

اغلب سیستم های فیزیکی غیرخطی هستند،بنابراین برای شناسایی چنین سیستم هایی نیازمند مدل سازی غیرخطی می باشیم،اما به دلیل ارتباط غیرخطی بین ورودی و خروجی سیستم های دینامیکی،شناسایی چنین سیستم هایی با مدل های خطی دشوار است. شناسایی چنین سیستم هایی اغلب با مدل هایی چون Hammerstein و سری های voltra صورت می گیرد .

یک راه دیگر در این رابطه،اصلاح مدل ARMA می باشد به طوری که شامل جملاتی غیرخطی شود و در واقع مدل ARMA غیرخطی،NARMA تولید شود :

$$y(k)=\sum_{i=0}^{m}b_i x(k-i)+\sum_{j=1}^{n}a_j y(k-j)+\sum_{i=0}^{m}\sum_{j=0}^{n}b_{ij}x(k-i)x(k-j)+\sum_{i=1}^{m}\sum_{j=1}^{n}a_{ij}y(k-i)y(k-j)+\sum_{i=0}^{m}\sum_{j=1}^{n}c_{ij}x(k-i)y(k-j) \qquad (4)$$

در این مدل ها پارامترهای جدیدی،در این جا $(a_{ij}, b_{ij}, c_{ij}$ اضافه می شوند. هم چنین دشواری های پیش بینی حدود مناسب برای هر Σ مشخص می شود. علاوه بر این،این مدل ها به فرم مرسوم تابع تبدیل نمی باشند. به همین دلیل سر و کار داشتن با آن ها دشوار تر می باشد. تمام این موانع موجب شده تا شبکه های عصبی به عنوان یک انتخاب مناسب برای مدل کردن سیستم غیرخطی استفاده شود .



شکل ۲ - ساختار عمومی برای شبکه های دینامیکی

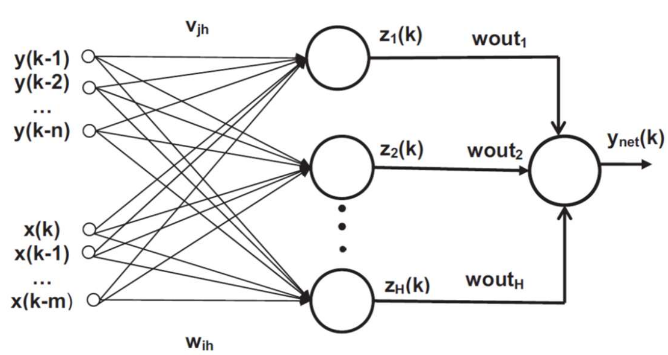
شکل ۲ یک شبکه دینامیکی با یک ورودی $x(k)$ یک خروجی $y(k)$ نشان می دهد، که k شماره نمونه می باشد. TDU تاخیر زمانی می باشد که معادل z^{-n} است و نمونه تاخیر خورده را ایجاد می کند. بنابراین شبکه به صورت چند ورودی در می آید که ورودی های آن، ورودی ها و خروجی های تاخیر خورده می باشند. میزان تاخیر به مرتبه مدل وابسته است .

یک انتخاب دیگر استفاده از خروجی پلنت تاخیر خورده (به جای استفاده از خروجی شبکه تاخیر خورده) به عنوان فیدبک شبکه می باشد. بنابراین خروجی شبکه تابعی خواهد بود از ورودی ها و خروجی های گذشته که در معادله زیر نشان داده شده است :

$$y_{net}(k)=f(x(k),x(k-1),\dots,y(k-1),y(k-2),\dots) \qquad (5)$$

این رابطه به صورت یک شبکه *FeedForward* با ورودی های:

$$y(k-1),\dots,y(k-2),x(k),\dots,x(k-m)$$



در شکل ۳ نشان داده شده است. هریک از نمونه های ورودی در وزن های مربوطه شان ضرب شده و وارد نورون میانی می شوند. ورودی های $x(k-i)$ در وزن های w_{ih} و ورودی های $y(k-j)$ در وزن های v_{jh} ضرب می شوند که h تعداد نورون های میانی می باشد. خروجی

نورون ها $z_1(k), \dots, z_h(k)$ هستند که آن ها نیز به نوبه خود در وزن های $w_{out_1}, \dots, w_{out_H}$ ضرب شده و نهایتا خروجی شبکه $y_{net}(k)$ را تشکیل می دهند.

توجه: خروجی شبکه به صورت $y_{net}(k)$ نوشته شده می شود تا با خروجی پلنت یعنی $y(k)$ متمایز شود. از وزن های بایاس در این شبکه به منظور جلوگیری از معرفی ورودی های جدید، استفاده نشده است .

معادله ریاضی شبکه به صورت زیر می باشد :

$$y_{net}(k) = g\left(\sum_{h=1}^{H} z_h(k) wout_h\right) \quad (6)$$

$$z_h(k) = f\left(\sum_{i=0}^{m} w_{ih} x(k-i) + \sum_{j=1}^{n} v_{jh} y(k-j)\right) \qquad (7)$$

در این جا f و g به ترتیب توابع فعالیت نورون های میانی و خروجی می باشند. توابع فعالیت مختلفی وجود دارند که می توان از آن ها استفاده کرد ؛ مثل سیگموید، تانژانت هیپربولیک.

هدف یافتن وزن های بهینه شبکه می باشد یعنی: $w_{ih}, v_{ih}, w_{out_h}$. به طوری که خطای بین خروجی شبکه، $y_{net}(k)$، و خروجی پلنت $y(k)$ حداقل شود. خطای معمول مورد استفاده، مجموع مربعات خطا (SSE) می باشد .

$$SSE = \sum_{k=1}^{K} \big(e(k)\big)^2 = \sum_{k=1}^{K} \big(y_{net}(k) - y(k)\big)^2 \qquad (8)$$

الگوریتم های مختلفی چون Backpropagation یا آموزش Levenberg-Marquardt می تواند برای آموزش شبکه (یافتن وزن های بهینه) به کار گرفته شود. Levenberg-Marquardt مشتق مرتبه دوم را به کمک مششتق مرتبه اول تقریب می زند به طوری که :

$$NewWeights = OldWeights - [\,J^T J + \mu I\,]^{-1} J^T e \quad (9)$$

که در آن e بردار خطا، I ماتریس همانی و J ماتریس ژاکوبین (مشتقات مرتبه اول خطاهای شبکه نسبت به وزن ها) می باشند. برای لایه خروجی، المان های ژاکوبین به صورت زیر محاسبه می شوند :

$$\frac{\partial e}{\partial w_{out_h}} = \frac{\partial e}{\partial y_{net}} \frac{\partial y_{net}}{\partial w_{out_h}} \qquad (10)$$

هم چنین برای وزن های لایه ابتدایی یعنی w_{ih}, v_{ih} ماتریس ژاکوبین می تواند با استفاده از تکنیک Backpropagation استاندارد محاسبه شود :

$$\frac{\partial e}{\partial w_{ih}} = \frac{\partial e}{\partial y_{net}} \frac{\partial y_{net}}{\partial z_h} \frac{\partial z_h}{\partial w_{ih}} \qquad (11)$$

این مقاله در رابطه با همگرایی وزن نمی باشد. بلکه مربوط به استفاده از وزن های شبکه همگرا شده برای تقریب تابع تبدیل می باشد. به همین دلیل بحث جزییات آموزش شبکه مطرح نمی شود .

۳- الگوریتم تبدیل شبکه عصبی به تابع تبدیل (NN2TF)

در بخش های قبل، روابط ریاضی مربوط به مدل های ARMA معرفی شد. در این بخش رابطه ریاضی بین مدل های ANN و ARMA بررسی می شود و یک الگوریتم تیدیل شبکه عصبی به تابع تبدیل مطرح خواهد شد .

یک تابع فعالیت مورد استفاده در شبکه های عصبی تابع تانژانت هیپربولیک است که خروجی را بین ۱- و ۱+ تنظیم می کند. بنابراین خروجی گره یا نورون میانی از معادله (7) به صورت زیر خواهد بود :

$$z_h(k) = \frac{1 - e^{-\alpha net_h(k)}}{1 + e^{-\alpha net_h(k)}} \quad (13)$$

$$net_h(k) = \sum_{i=0}^{m} w_{ih} x(k-i) + \sum_{j=1}^{n} v_{jh} y(k-j) \quad (14)$$

نتایج زیر برای موارد خاصی که تابع فعالیت نورون های میانی (f)، تانژانت هایپربولیک ($\alpha = 1$) و تابع فعالیت نورون خروجی (g) خطی است، استنتاج می شوند .

بسط تیلور را می توان برای تقریب معادله (13) حول نقطه صفر با $net_h(k) = 0$ در نظر گرفت. دقت شود که رنج تانژانت هایپربولیک بین ۱- و ۱+ می باشد، بنابراین حدود تقریب نقطه میانی صفر معقولانه است .

$$z_h(k) = \frac{1 - e^0}{1 + e^0} + \frac{2e^0}{(1 + e^0)^2}(net_h(k) - 0) + \frac{-2e^0(1 - e^0)}{(1 + e^0)^3}(net_h(k) - 0)^2 + \cdots \quad (15)$$

برای جلوگیری از غیرخطی گری، تنها از دو جمله اول معادله (15) استفاده می کنیم :

$$z_h(k) = 0.5 \sum_{i=0}^{m} w_{ih} x(k-i) + 0.5 \sum_{j=1}^{n} v_{jh} y(k-j) \quad (16)$$

نورون خروجی به صورت زیر خواهد شد:

$$y_{net}(k) = \sum_{h=1}^{H} \left(w_{out_h} z_h(k) \right) \quad (17)$$

با جایگذاری معادله (16) در معادله (17)خواهیم داشت :

$$y_{net}(k) = 0.5 \sum_{h=1}^{H} \left(w_{out_h} \left(\sum_{i=0}^{m} w_{ih} x(k-i) + \sum_{j=1}^{n} v_{jh} y(k-j) \right) \right) \quad (18)$$

با مقایسه معادله (18) با معادله (2) پارامترهای مدل ARMA را می توان به صورت زیر تخمین زد :

$$\hat{a}_j = 0.5 \left(\sum_{h=1}^{H} \left(w_{out_h} v_{jh} \right) \right) \quad (19)$$

$$\hat{b}_i = 0.5 \left(\sum_{h=1}^{H} \left(w_{out_h} w_{ih} \right) \right) \quad (20)$$

این معادلات یک رابطه ریاضی بین وزن های شبکه و پارامترهای ARMA را نشان می دهند. بنابراین اطلاعات شبکه می تواند برای شناسایی پارامتری و تقریب تابع تبدیل مورد استفاده قرار گیرد.

جدول ۱ توابع فعالیت مختلف را با مشتقاتشان، بسط تیلور و نتایج تقریب نشان می دهد.

Function name	Math function $\boldsymbol{f}(\boldsymbol{x})$	Function derivative $\boldsymbol{f'}$	1st two terms of the Taylor expansionat 0	Approximation result
Hyperbolic tangent with $\boldsymbol{\alpha = 1}$	$\frac{1-e^{-x}}{1+e^{-x}}$	$0.5(1-f^2(x))$	$f(0)+0.5(1-f^2(0))x$	$0.5x$
Hyperbolic tangent with $\boldsymbol{\alpha = 2}$	$\frac{1-e^{-2x}}{1+e^{-2x}}$	$1-f^2(x)$	$f(0)+(1-f^2(0))x$	x
sigmoid	$\frac{1}{1+e^{-x}}$	$f(x)(1-f(x))$	$f(0)+\ f(0)(1-f(0))x$	$0.5+0.25x$
Gaussian	$e^{-(x-c)^2}$	$-2(x-c)e^{-(x-c)^2}$	$e^{-c^2}+2ce^{-c^2}x$	c_1+c_2x

جدول ۱- توابع فعالیت همراه با بسط تیلور و مشتق مرتبه اول

با توجه به نتایج جدول ۱ و معادله (17) می توان معادلات زیر را برای تانژانت هیپربولیک با $\alpha = 2$، سیگموید و گوسی نوشت :

$$y_{net}(k) = \sum_{h=1}^{H}\left(w_{out_h}\left(\sum_{i=0}^{m} w_{ih}x(k-i) + \sum_{j=1}^{n} v_{jh}y(k-j)\right)\right) \qquad (21)$$

$$y_{net}(k) = \sum_{h=1}^{H}\left(w_{out_h}\left(0.5 + 0.25\sum_{i=0}^{m} w_{ih}x(k-i) + 0.25\sum_{j=1}^{n} v_{jh}y(k-j)\right)\right) \qquad (22)$$

$$y_{net}(k) = \sum_{h=1}^{H}\left(w_{ou\ h}\left(c_1 + c_2\sum_{i=0}^{m} w_{ih}x(k-i) + c_2\sum_{j=1}^{n} v_{jh}y(k-j)\right)\right) \qquad (23)$$

دقت شود که توابع سیگموید و گوسی یک جمله اضافه تر دارند، که با ورودی ها و خروجی های سیستم ضرب نشده اند. این جملات بخشی از پارامتر های تابع تبدیل نیستند بلکه یک آفست DC می باشند (در مثال دوم بخش شبیه سازی به این موضوع پرداخته شده است). با استفاده از این معادلات نتایج مدل ARMA تخمین زده شده استنتاج و در جدول ۲ نمایش داده شده است .

Function name	$\boldsymbol{\hat{a}_j}$	$\boldsymbol{\hat{b}_i}$	offset
Hyperbolic tangent with $\boldsymbol{\alpha = 1}$	$0.5\left(\sum_{h=1}^{H}(w_{out_h}v_{jh})\right)$	$0.5\left(\sum_{h=1}^{H}(w_{out_h}w_{ih})\right)$	$-$
Hyperbolic tangent with $\boldsymbol{\alpha = 2}$	$\sum_{h=1}^{H}(w_{out_h}v_{jh})$	$\sum_{h=1}^{H}(w_{out_h}w_{ih})$	$-$
sigmoid	$0.25\left(\sum_{h=1}^{H}(w_{out_h}v_{jh})\right)$	$0.25\left(\sum_{h=1}^{H}(w_{out_h}w_{ih})\right)$	$0.5\left(\sum_{h=1}^{H}(w_{out_h})\right)$
Gaussian	$c_2\left(\sum_{h=1}^{H}(w_{ou\ h}v_{jh})\right)$	$c_2\left(\sum_{h=1}^{H}(w_{ou\ h}w_{ih})\right)$	$c_1\left(\sum_{h=1}^{H}(w_{ou\ h})\right)$

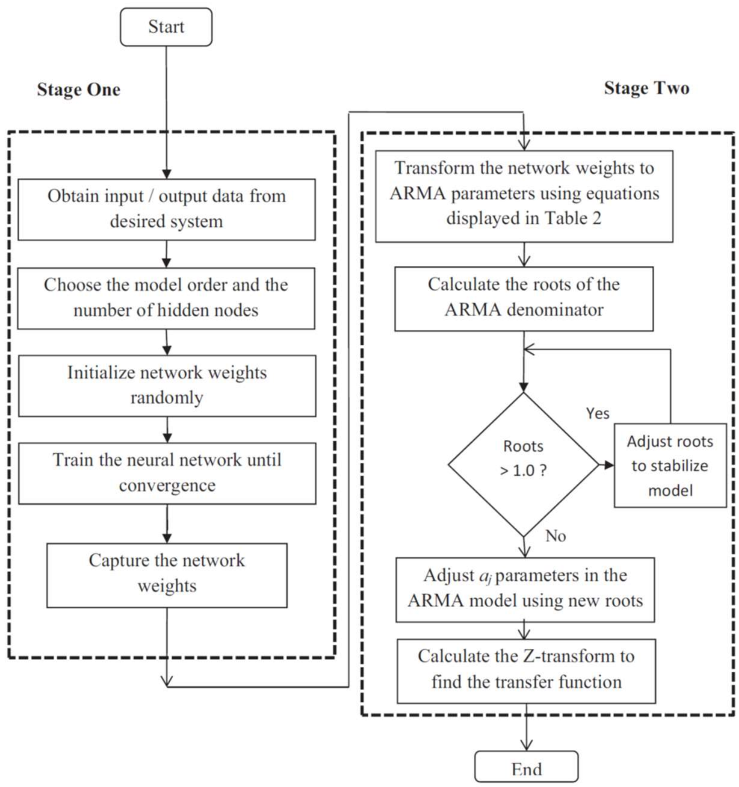
جدول ۲- تخمین پارامترهای مدل ARMA به ازای توابع فعالیت مختلف

پایداری مدل های ARMA زمان گسسته مطرح شده در معادله (2) وابسته به مقادیر پارامترهای a_j می باشد. پارامترهای a_j موقعیت قطب های تابع تبدیل معادله (3) را تعیین می کنند. در صورتی که قطب ها خارج دایره واحد باشند سیستم ناپایدار خواهد بود. بنابراین می توان یک

مدل شبکه عصبی با وزن های محدود پیدا کرد که به مدل ARMA نامحدود تبدیل شود .چنین مدلی در مثال ۴ قسمت شبیه سازی مطرح شده است .

یک روش ساده برای حل چنین مشکلی حرکت قطب ها به داخل دایره واحد با تغییر دامنه شان می باشد به طوری که زاویه قطب ها ثابت بماند:

الگوریتم مطرح شده در شکل ۴ نشان می دهد که چگونه می توان یک مدل شبکه عصبی را به یک مدل ARMA تبدیل کرد . مزیت این تبدیل قابلیت استنتاج یک تابع تبدیل از وزن های شبکه می باشد و بنابراین دید بهتری از سیستم تحت مطالعه بدست می آید .



شکل ۴ – فلوچارت الگوریتم NN2TF