**آزمايش پنجم پياده سازي شبكه عصبي RBF**

**اعضای گروه:**

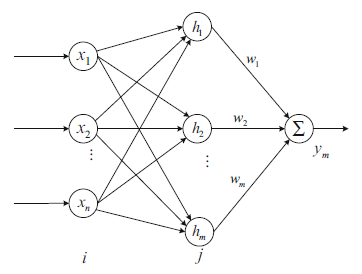
**ملیکا صالحیان ، محدثه قاسم مهرابی، مریم حیدری**

* **ساختار کلی شبکه عصبی RBF:**

شبکه‌های عصبی RBF سه لایه دارند :

لایه ورودی (Input Layer)،

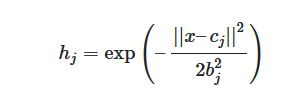
لایه پنهان یا مخفی (Hidden Layer)

 و لایه خروجی (Output Layer)

ساختار شبکه عصبی RBF

نورون‌های لایه مخفی با یک تابع پایه شعاعی (RBF) فعال می‌شوند. لایه مخفی از آرایه‌ای از واحدهای محاسباتی تشکیل شده که گره‌های مخفی (Hidden Nodes) نامیده می‌شوند. هر گره مخفی شامل یک بردار c مرکزی است که یک بردار پارامتری با طولی مشابه با بردار ورودی x است.

در شبکه عصبی RBF بردار ورودی x=[xi]T است. فرض می‌کنیم m نورون در لایه مخفی وجود داشته باشد، و بردار تابع پایه شعاعی در لایه مخفی وhj تابع گوسی نورون j در لایه مخفی باشد،



مقدار وزن RBF برابر است با:

w=[w1,⋯,wm]T

خروجی شبکه عصبی RBF به صورت زیر است:

y(t)=wTh=w1h1+w2h2+⋯+wm.hm

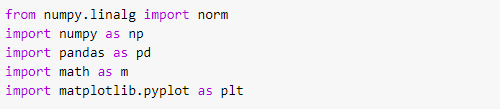
یک شبکه تابع شعاعی ( RBF )، نظیر MLP ، یک شبکه یادگیری نظارت شده است و از جهاتی به آن شباهت دارد. اما شبکه RBF فقط با یک لایه مخفی کار می کند. این کار با محاسبه مقدار هر واحد در لایه مخفی برای یک مشاهده به انجام می رسد. در واقع به جای مجموع مقادیر وزن دار واحد های سطح قبلی، از فاصله میان این مشاهدات و مرکز این واحد استفاده می کند.

بر خلاف وزن های یک پرسپترون چند لایه ، مراکز لایه پنهان یک شبکه RBF در طول یادگیری در هر تکرار تنظیم نمی شوند. در شبکه RBF ، نورون های مخفی ، فضا را به اشتراک می گذارند و عملا از یکدیگر مستقل هستند. این امر باعث ایجاد همگرایی سریع تر شبکه‌های RBF در مرحله یادگیری می شود، که یکی از نقاط قوت آن هاست.

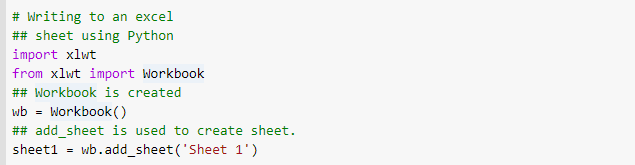
توضیحات کد پایتون کلاس RBF :

شبکه عصبی توابع شعاعی پایه(radial basis function) یک شبکه عصبی 3 لایه است که از یک‌لایه پنهان تشکیل‌شده است و برای حل مسائل پیچیده و غیرخطی استفاده می‌شود.

در ابتدا کتابخانه‌های لازم برای استفاده در کد معرفی‌شده‌اند.

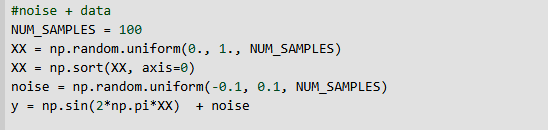


در این قسمت با استفاده از کد زیر یک فایل اکسل به منظور ذخیره سازی دیتا ایجاد می شود.

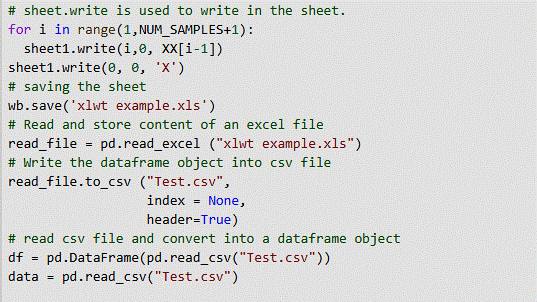


**سپس داده‌های ورودی و خروجی ساخته می‌شوند.**

**ابتدا 100 داده ورودی در بازه [0,1] و فواصل رندم ساخته و مرتب می‌شود. سپس خروجی‌ها مقدار sin هر داده تعریف شده و با نویز در محدوده [-0.1,0,1] جمع می شود و به این ترتیب یک تابع سینوسی نویزی تولید می شود.**



به منظور نوشتن در فایل اکسل ایجاد شده در قسمت قبل از دستور sheet.write استفاده می شود. در فایل اکسل ایجاد شده در سطر اول نام متغیر و در سطرهای بعدی به ترتیب تعداد 100 داده ورودی رندم نوشته می شوند. پس از ثبت فایل لازم است به منظور استفاده از کد k-mean فایل به فرمت .csv تبدیل شود.



**کد K mean مورد استفاده که به منظور پیدا کردن میانگین و انحراف معیار توابع گوسین که به تعداد دسته های انتخابی می باشند، به شرح زیر می باشد.**

**#finding the number of class**

**def CNUM(x):**

**s=0**

**for i in range (k):**

**A = float(x - center\_x[i])**

**#dist[i]= m.sqrt(A\*\*2 + B\*\*2)**

**dist[i]= (A\*\*2)**

**if dist[i] == min(dist):**

**s = i**

**return s**

**#classification**

**def classify(x):**

**for j in range (data\_number):**

**X = data.iloc[[j],[0]].values**

**X = float(X)**

**index = CNUM(X)**

**A[index].append([float(X)])**

**return A**

**# Data**

**data\_number = input('Enter the number of data')**

**data\_number = int(data\_number)**

**k = input('Enter the number of class')**

**k = int(k)**

**center\_x\_new = np.empty(k)**

**center\_x = np.empty(k)**

**dist = np.empty(k)**

**CLASS = [[]for \_ in range (k)]**

**E22= [0 for \_ in range (k)]**

**Error=[[]for \_ in range (k)]**

**A = [[]for \_ in range (k)]**

**flag = 1**

**data = pd.read\_excel("xlwt example.xls")**

**lr = 0.01**

**# Center initializing**

**for i in range (k):**

**center\_x[i] = np.random.uniform(size=(1,1))**

**CLASS= classify(data\_number)**

**#Finding new centers**

**for o in range (k):**

**if CLASS[o] != []:**

**m =np.mean(CLASS[o], axis=0)**

**#print(m)**

**center\_x\_new[o] = float(m[0])**

**#Reclassification by new centers**

**while flag:**

**for i in range (k):**

**C = center\_x\_new[i]- center\_x[i]**

**C = C.astype('float')**

**E2 = C\*\*2**

**E22[i] = C\*\*2**

**if (np.sum(E22) < (0.001)\*\*2):**

**flag = 0**

**if E2 > (0.001)\*\*2:**

**center\_x[i] = center\_x\_new[i]**

**CLASS= classify(data\_number)**

**MAX = []**

**MIN = []**

**d = []**

**for i in range(k):**

**if CLASS[i] == []:**

**d.append(0)**

**else:**

**MAX.append(np.max(np.abs(CLASS[i])))**

**MIN.append(np.min(np.abs(CLASS[i])))**

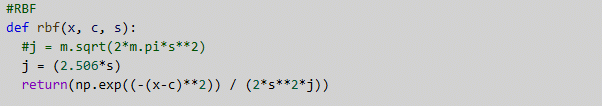
**d.append((MAX[i]-MIN[i])/(1.414\*k))**

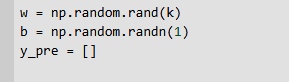
**s = d**

**c = center\_x**

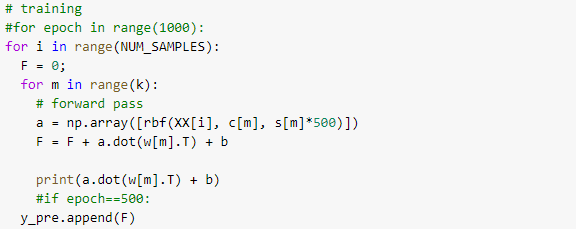
توضیحات کد پایتون بخش RBF:

ابتدا تابع rbf به منظور محاسبه تابع گوسی با میانگین و واریانس مشخص شده در قسمت قبل تعریف می شود.

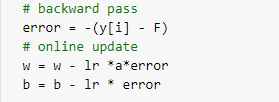


**وزن ها ، بایاس و خروجی پیش بینی شده توسط RBF:**

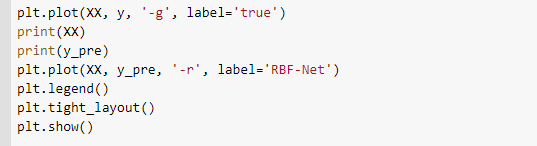
**در این قسمت مجموع توابع RBF با استفاده از تابع rbf محاسبه می شود. در اینجا به منظور طولانی شدن اجرای کد نمودار در حالت اجرای بار اول به نمایش درآمده است.**

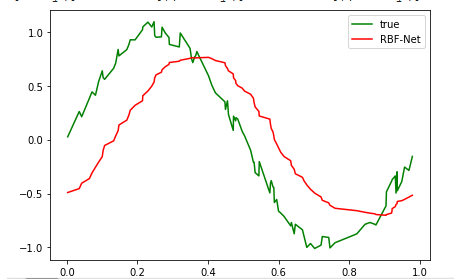


**به ترتیب زیر وزن ها و بایاس آپدیت می شوند.**



**نمودار خروجی سینوسی همراه با نویز و خروجی حاصل از کد به ترتیب با رنگ های سبز و قرمز ترسیم می شوند.**





**تمرین 1-کد متلب پیاده سازی شبکه عصبی RBF**

**%%computational Inteligence Lab RBF code%%**

**clear all**

**clc**

**close all**

**%%noise + data**

**NUM\_SAMPLES = 100**

**%building an array of 0 to 1 with 100 random numbers**

**XX = zeros(1,100);**

**i=0;**

**for i =1: 100**

**XX(i)=rand();**

**while XX(i)==0;**

**XX(i)=rand();**

**end**

**end**

**XX = sort(XX);**

**%producing a -1 to 1 noise**

**noise = rand(1,100)/10;**

**for i=1:50**

**noise(i)= -noise(i);**

**end**

**noise = sort(noise)**

**%y**

**y = sin(2\*pi\*XX) + noise**

**%% initializing %%**

**k = 50 %limit of the number of classes that can be chosen by user**

**g = 1**

**iteration = 10; %the limit of updationg**

**center\_x =zeros(1,k);**

**center\_x\_new =ones(1,k);**

**dist = zeros(1,k);**

**data\_number = 100**

**k = input('Enter the number of class')**

**dist = zeros(1,k)**

**A = []**

**s = []**

**flag = 1**

**lr = 0.01**

**%% initializing centers of classes by random numbers**

**for i = 1:k**

**center\_x(1,i) = rand();**

**while center\_x(1,i)==0**

**center\_x(1,i) = rand();**

**end**

**end**

**%% K-Mean %%**

**%input data & classification**

**for j = 1:length(XX)**

**index = CNUM(k,XX(j),center\_x(1,:));**

**CLASS\_X(index,j)= XX(j);**

**end**

**%Finding new centers**

**flag = 1;**

**while flag**

**for l = 1 : iteration %to limit the number of updating**

**m\_1 = 0;**

**sum\_1 = 0**

**for o = 1:k**

**for j = 1:length(XX)**

**CLASS\_X(o,j)**

**if CLASS\_X(o,j) ~= 0**

**m\_1 = m\_1+1;**

**sum\_1 = sum\_1 +CLASS\_X(o,j);**

**end**

**end**

**center\_x\_new(1,o) = sum\_1 / m\_1**

**end**

**% input data & classification**

**for j = 1:length(XX)**

**index = CNUM(k,XX(j),center\_x\_new(1,:));**

**CLASS\_X(index,j)= XX(j);**

**end**

**for i = 1:k**

**T=center\_x(1,i) - center\_x\_new(1,i)**

**if (sqrt(T^2) < 10)**

**flag = 0**

**end**

**end**

**end**

**end**

**% finding c & s**

**c= center\_x**

**%finding the max distance between centers**

**d = 0;**

**for i=1:k**

**d0 = 0;**

**for j = 1:k**

**d = abs(center\_x(i)-center\_x(j));**

**if d > d0**

**d0 = d;**

**end**

**end**

**s(i) = d0/sqrt(2\*k);**

**end**

**%%RBF**

**w = rand(1,k);**

**b = rand(1,1);**

**y\_pre = [];**

**%% training**

**j=0;**

**for epoch = 1:100**

**for i = 1 : NUM\_SAMPLES**

**F = 0;**

**for m =1:k**

**a = rbf(XX(i), c(m), s(m));**

**F = F + a\*(transpose(w(m))) + b**

**if epoch==100**

**y\_pre(i)=F;**

**end**

**end**

**loss = (y(i) - F)^ 2;**

**error = -(y(i) - F);**

**% update**

**w = w - lr \*a\*error;**

**b = b - lr \* error;**

**end**

**end**

**%%Plot**

**plot(XX,y,'g')**

**hold on;**

**plot(XX, y\_pre,'r')**

**legend('sin(x)+noise','RBF')**

**%%FUNCTIONS**

**function [s] = CNUM(k, x, center\_x)**

**s=0;**

**for i = 1:k**

**A = x - center\_x(i);**

**dist(i) = sqrt(A^2);**

**if (dist(i) == min(dist))**

**s = i ;**

**end**

**end**

**end**

**function[r] = rbf(x, c, s)**

**r = exp(-1 / (2 \* s^2) \* (x-c)^2);**

**end**

**نتیجه کد متلب با انتخاب 8 کلاس:**

