به نام خدا

1928
K. N. Toosi University of Technology

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

مبانی سیستم های هوشمند

پاسخنامه پایان ترم

محمدرضا جنیدی جعفری

۹۹۲۵۲۵۳

لینک کولب

استاد : آقای دکتر مهدی علیاری

پاییز ۱۴۰۳

فهرست مطالب

عنوان شماره صفحه

بخش ۱: سوالات هماهنگ شده

سوال اول

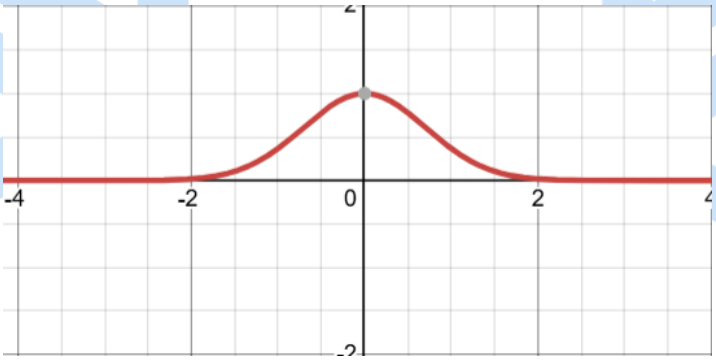
در یک مسئله طبقه‌بندی شده، اگر $y > 0$ باشد؛ کلاس 1 و در غیر این صورت کلاس 0 می‌باشد، مرز تصمیم‌گیری را رسم نمایید.

$$y = -e^{-\|x-t_1\|^2} - e^{-\|x-t_2\|^2} + 1$$

نیاز داریم یک مرز تصمیم گیری تعریف کنیم. می دانیم مرز تصمیم گیری لزوما یک خط مستقیم نیست؛ این مرز به صورت y = 0 تعریف می شود. پس داریم:

$$0 = -e^{-\|x-t_1\|^2} - e^{-\|x-t_2\|^2} + 1$$

این معادله، یک منحنی است نه یک خط مستقیم. در زیر، شکل تابع $e^{-\|x-t_1\|^2}$ رسم شده است:



معادله بالا را می توان به صورت زیر نیز نوشت:

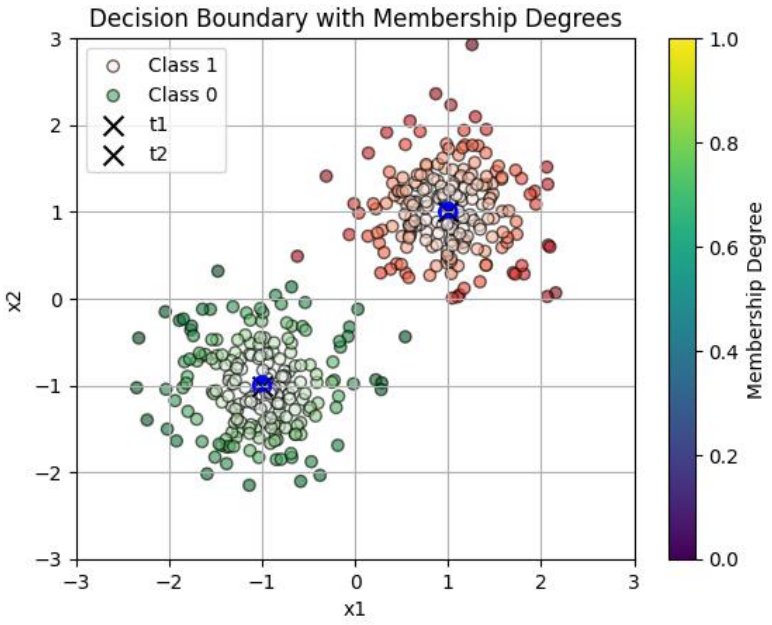
$$1 = e^{-\|x-t_1\|^2} + e^{-\|x-t_2\|^2}$$

این معادلع نشان دهنده مجموعه از نقاط x است که در آنها مجموع دو تابع گوسی برابر با یک است. شکل ما، نشان می دهد که اکر فاصله t1 یا t2 به x نزدیک باشند، مقدار $e^{-\|x-t_1\|^2} + e^{-\|x-t_2\|^2}$ به یک نزدیک می شود، بنابراین مجموع این دو مقدار ممکن است از یک بیشتر شود و y مثبت شود. (کلاس ۱)

اگر x از هر دو نقطه t1 و t2 دور باشد، مجموع تابع بالا از یک کمتر خواهد شد و بنابراین y منفی خواهد شد.(کلاس ۲)

اگر $t1$ و $t2$ به هم نزدیک باشند، مرز تصمیم‌گیری ممکن است به شکل یک دایره یا بیضی حول این دو نقطه باشد.

اگر $t1$ و $t2$ از هم دور باشند، مرز تصمیم‌گیری ممکن است به شکل دو دایره جداگانه حول هر یک از نقاط باشد.



داده های مصنوعی تولید کردیم که شامل دو کلاس هستند، که به طور واضح توسط مرز تصمیم گیری جدا شده اند و داده‌های کلاس ۱ در نزدیکی t1و داده‌های کلاس ۰ در نزدیکی t2متمرکز شده‌اند. درجه عضویت هر نقطه برای هر کلاس محاسبه می‌شود و این مقادیر به‌صورت رنگی روی نمودار نمایش داده می‌شوند. نقاطی که به مرکز کلاس‌ها نزدیک‌تر هستند، رنگ‌های پررنگ‌تری دارند و نقاط دورتر رنگ‌های کمرنگ‌تری خواهند داشت.

سوال دوم

چرا در شبکه RBF، تعیین میزان گستردگی توابع پایه شعاعی ($Spreads$ یا σ) از طریق رابطه زیر بهترین جواب را می‌دهد؟

$$\sigma = \frac{Maximum\ distance\ between\ any\ 2\ centers}{\sqrt{number\ of\ centers}} = \frac{d_{max}}{\sqrt{m_l}}$$

این فرمول از اصول تقریب توابع، روش‌های کرنل و تئوری یادگیری آماری نشأت می‌گیرد.

- فرمول σ بالا از ادبیات تخمین چگالی کرنل گرفته شده است، جایی که پهنای باند یک کرنل (مشابه σ) به‌گونه‌ای انتخاب می‌شود که تعادل بین بایاس و واریانس در تخمین برقرار شود. در

شبکه‌های RBF، این فرمول تضمین می‌کند که توابع پایه نه خیلی محلی (واریانس بالا، بیش‌برازش) و نه خیلی گسترده (بایاس بالا، کم‌برازش) باشند.

- m_l تعداد مراکز در شبکه RBF است. با افزایش تعداد مراکز، تراکم توابع پایه در فضای ورودی نیز افزایش می‌یابد. تقسیم بر m_l این اطمینان را می‌دهد که گستردگی هر تابع پایه به طور معکوس با جذر تعداد مراکز مقیاس شود. این ایده از این واقعیت نشأت می‌گیرد که در فضاهای با ابعاد بالا، فاصله میانگین بین نقاط به تعداد مراکز وابسته است.
- d_max حداکثر فاصله بین هر دو مرکز (فاصله بین دورترین جفت مراکز) است. این اطمینان می‌دهد که توابع پایه به اندازه کافی گسترش یابند تا فضای ورودی را به طور کامل پوشش دهند. اگر σنسبت به d_max خیلی کوچک باشد، توابع پایه به اندازه کافی همپوشانی نخواهند داشت و این منجر به تقریب ضعیف و تعمیم‌پذیری نامناسب خواهد شد. از طرفی، اگر σخیلی بزرگ باشد، توابع پایه بیش از حد همپوشانی خواهند داشت و شبکه توانایی خود را در شناسایی مناطق مختلف فضای ورودی از دست خواهد داد.

این فرمول از ادبیات تخمین چگالی کرنل گرفته شده است، جایی که پهنای باند یک کرنل (مشابه σ) به‌گونه‌ای انتخاب می‌شود که تعادلی بین بایاس و واریانس در تخمین برقرار شود. در شبکه‌های RBF، این فرمول تضمین می‌کند که توابع پایه نه خیلی محلی (که منجر به واریانس بالا و بیش‌برازش می‌شود) و نه خیلی گسترش‌یافته (که منجر به بایاس بالا و کم‌برازش می‌شود) باشند.

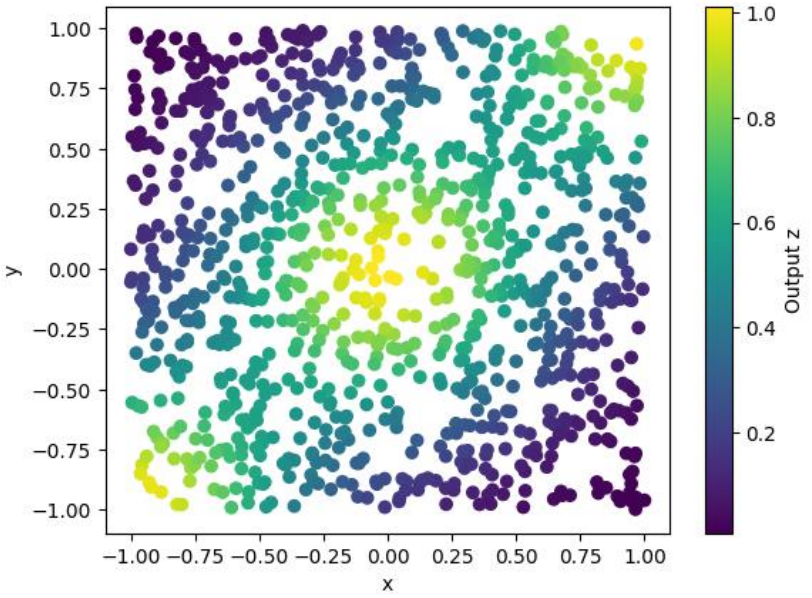
این فرمول به تغییرات در توزیع مراکز توابع پایه مقاوم است، زیرا تنها به حداکثر فاصله بین مراکز و تعداد مراکز بستگی دارد.

بخش۲: سوالات هماهنگ نشده

سوال دوم

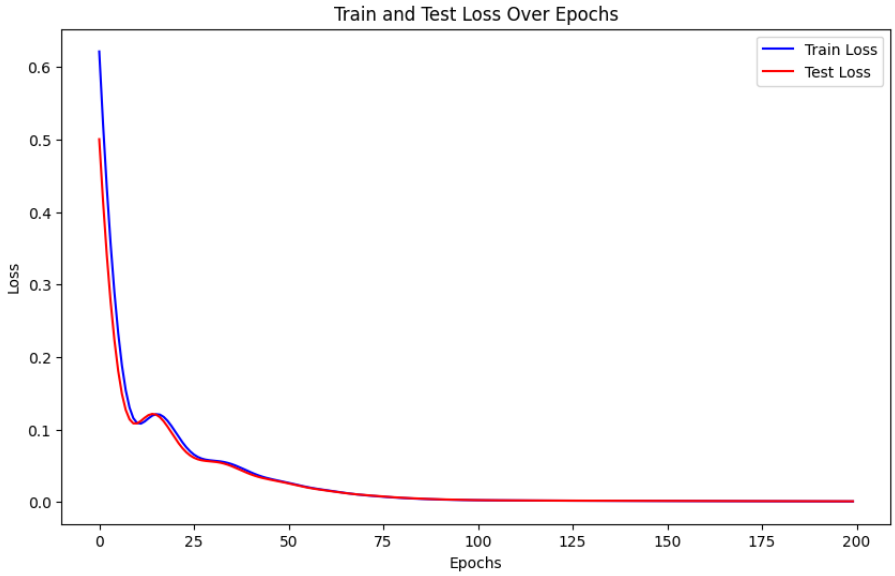
این سوال در محیط کولب و به زبان پایتون نوشته شده است، ابتدا داده های مصنوعی را تولید کردیم و با یک فازی ساز، خروجی فازی نمونه ها محاسبه شده است:

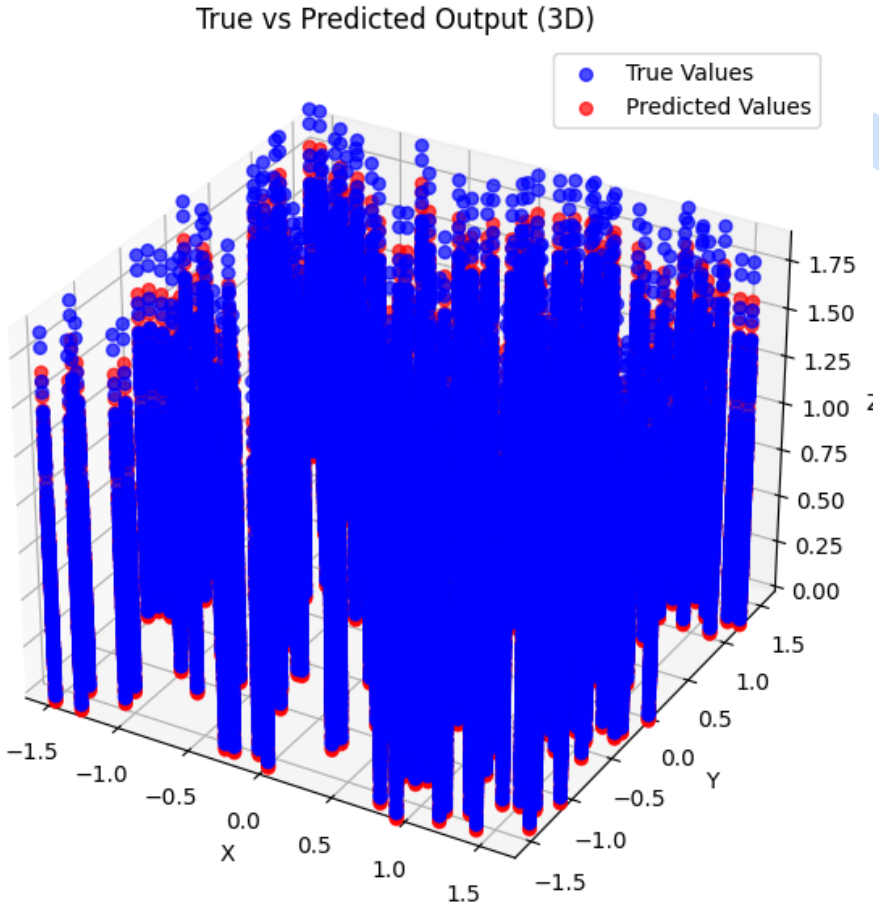
```
import numpy as np
import matplotlib.pyplot as plt

n_samples = 1000
x = np.random.uniform(-1, 1, n_samples)
y = np.random.uniform(-1, 1, n_samples)

# Gaussian membership functions
def gaussian(x, center, variance):
    return np.exp(- (x - center)**2 / (2 * variance))

# Centers and variances for Gaussian functions
centers_x = [-1, 0, 1]
centers_y = [-1, 0, 1]
variance = 0.2  # I prefered to adjust 0.2

# Compute membership values for each x and y
def get_membership_values(x, centers):
    return [gaussian(x, c, variance) for c in centers]

# Compute fuzzy outputs z
z = []
for xi, yi in zip(x, y):
    memberships_x = get_membership_values(xi, centers_x)
    memberships_y = get_membership_values(yi, centers_y)

    output = sum(m_x * m_y for m_x, m_y in zip(memberships_x,
memberships_y))
    z.append(output)

z = np.array(z)

# Visualize the generated data (optional)
plt.scatter(x, y, c=z, cmap='viridis')
plt.colorbar(label='Output z')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



مدل را آموزش می دهیم:

```
Epoch [10/200], Train Loss: 0.1158, Test Loss: 0.1086
Epoch [20/200], Train Loss: 0.1054, Test Loss: 0.0964
Epoch [30/200], Train Loss: 0.0578, Test Loss: 0.0563
Epoch [40/200], Train Loss: 0.0435, Test Loss: 0.0404
Epoch [50/200], Train Loss: 0.0277, Test Loss: 0.0269
Epoch [60/200], Train Loss: 0.0171, Test Loss: 0.0166
Epoch [70/200], Train Loss: 0.0100, Test Loss: 0.0102
Epoch [80/200], Train Loss: 0.0059, Test Loss: 0.0062
Epoch [90/200], Train Loss: 0.0037, Test Loss: 0.0039
Epoch [100/200], Train Loss: 0.0026, Test Loss: 0.0028
Epoch [110/200], Train Loss: 0.0022, Test Loss: 0.0023
Epoch [120/200], Train Loss: 0.0020, Test Loss: 0.0021
Epoch [130/200], Train Loss: 0.0018, Test Loss: 0.0019
Epoch [140/200], Train Loss: 0.0016, Test Loss: 0.0018
Epoch [150/200], Train Loss: 0.0015, Test Loss: 0.0016
Epoch [160/200], Train Loss: 0.0014, Test Loss: 0.0015
Epoch [170/200], Train Loss: 0.0013, Test Loss: 0.0014
Epoch [180/200], Train Loss: 0.0011, Test Loss: 0.0013
Epoch [190/200], Train Loss: 0.0010, Test Loss: 0.0012
Epoch [200/200], Train Loss: 0.0009, Test Loss: 0.0011
```


Train and Test Loss Over Epochs
Train Loss
Test Loss
Loss
Epochs
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
25
50
75
100
125
150
175
200


True vs Predicted Output (3D)
True Values
Predicted Values
X
Y
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75

- کاهش سریع خطا در ابتدا :در ابتدا، هر دو منحنی (آبی و قرمز) به سرعت کاهش می‌یابند، که نشان‌دهنده‌ی یادگیری سریع مدل است.
- توقف کاهش و تثبیت :در نهایت، هر دو منحنی به مقدار کم‌تری رسیده و تثبیت می‌شوند. این نشان‌دهنده‌ی این است که مدل به حد زیادی به حل مسئله نزدیک شده است و میزان خطا بیشتر از این کاهش نخواهد یافت.
- تفاوت کم بین خطای آموزشی و آزمایشی :اگر این دو منحنی نزدیک به هم باشند، این نشان‌دهنده‌ی عدم بیش‌برازش و عملکرد مناسب مدل بر روی داده‌های آزمایشی است.

در نهایت، این نمودار نشان‌دهنده‌ی این است که مدل به خوبی روی داده‌های آموزشی یادگیری کرده است و بر روی داده‌های آزمایشی هم عملکرد خوبی دارد، اما باید دقت کنیم که خطای آزمایشی کمی کاهش نیافته و به مقدار خیلی کمی رسیده است، که می‌تواند نشانه‌ای از ثبات مدل باشد.

```
Sample Differences Between True and Predicted Values:
Sample 1: True = 0.1895, Predicted = 0.1657, Difference = 0.0238
Sample 2: True = 0.3691, Predicted = 0.3766, Difference = 0.0075
Sample 3: True = 0.4968, Predicted = 0.5290, Difference = 0.0322
Sample 4: True = 0.6179, Predicted = 0.6130, Difference = 0.0049
Sample 5: True = 0.0285, Predicted = -0.0273, Difference = 0.0558
Sample 6: True = 0.2300, Predicted = 0.2264, Difference = 0.0037
Sample 7: True = 0.8064, Predicted = 0.8386, Difference = 0.0322
Sample 8: True = 1.2242, Predicted = 1.2418, Difference = 0.0176
Sample 9: True = 0.6459, Predicted = 0.6825, Difference = 0.0366
Sample 10: True = 1.2367, Predicted = 1.2284, Difference = 0.0083
```

سوال سوم

مجموعه داده را در محیط متلب می خوانیم. این مجموعه داده دارای ۶ ستون است که با استفاده از کد زیر می خوانیم. پنج ستون اول را ویژگی ها و ستون آخر، ستون هدف ما خواهد بود. (نرمال سازی و پیش پردازش انجام شده است)

```
data = readtable('evaporator.dat', 'Delimiter', '\t');

if any(all(ismissing(data), 1))
   data(:, end) = [];
end

% Nan Detector
for col = 1:width(data)
   if any(ismissing(data{:, col}))
      colMean = mean(data{:, col}, 'omitnan');
```

```
        data{ismissing(data{:, col}), col} = colMean;
    end
end

% col naming
numVars = width(data);
varNames = {'Feature_1', 'Feature_2', 'Feature_3', 'Feature_4', 'Feature_5', 'Target'};
if numVars == numel(varNames)
    data.Properties.VariableNames = varNames;
else
    error('Number of variables does not match the number of names provided.');
end

% Normalization
for col = 1:width(data)
    colMin = min(data{:, col});
    colMax = max(data{:, col});
    if colMin ~= colMax
        data{:, col} = (data{:, col} - colMin) / (colMax - colMin);
    else
        data{:, col} = 0.5;
    end
end

features = data{:, 1:5};
target = data{:, end};

disp('Features (First 5 Columns):');
disp(features);

disp('Target (Last Column):');
disp(target);
```

با اضافه کردن کد زیر، داده ها را با نسبت به ۷ به ۳، به آموزش و تست تقسیم می کنیم:

```
numSamples = size(features, 1);
trainSize = round(0.7 * numSamples);
randIndices = randperm(numSamples);

% Splitting
trainIndices = randIndices(1:trainSize); % 70% for train
testIndices = randIndices(trainSize+1:end); % 30% for test

X_train = features(trainIndices, :);  % Features
y_train = target(trainIndices, :);    % Target

X_test = features(testIndices, :);   % Features
y_test = target(testIndices, :);     % Target
```

```
disp(['Shape of X_train: ', num2str(size(X_train, 1)), ' samples, ', num2str(size(X_train, 2)), ' features']);
disp(['Shape of y_train: ', num2str(size(y_train, 1)), ' samples']);
```

خروجی:

Shape of X_train: 4414 samples, 5 features

Shape of y_train: 4414 samples

حالا سراغ ساخت شبکه RBF می رویم:

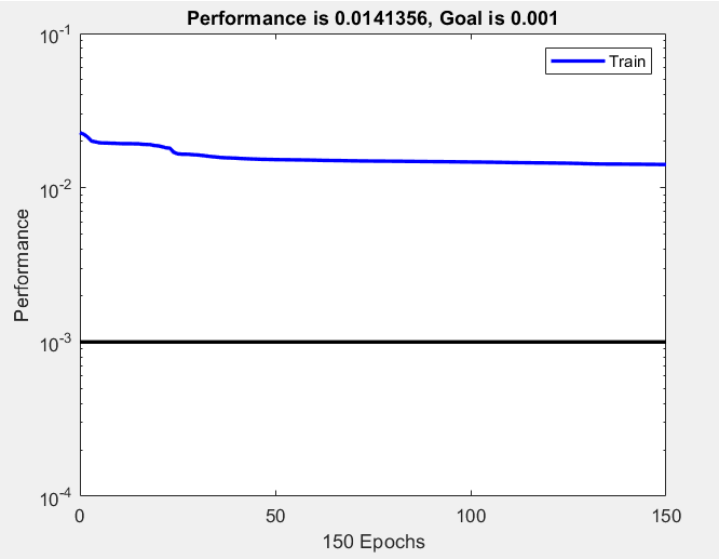
```
inputs = X_train';
targets = y_train';

% RBF net Config
spread = 1;
goal = 0.001;
max_neurons = 100;
increase_rate = 1;


net = newrb(inputs, targets, goal, spread, max_neurons, increase_rate);

% Prefricting
test_inputs = X_test';
predicted_outputs = net(test_inputs);

% MSE
predicted_outputs = predicted_outputs';
mse_test = mean((y_test - predicted_outputs).^2);

disp(['Mean Squared Error (MSE) on Test Data: ', num2str(mse_test)]);

figure;
plot(y_test, 'o-', 'DisplayName', 'Actual Outputs');
hold on;
plot(predicted_outputs, 'x-', 'DisplayName', 'Predicted Outputs');
title('Comparison of Actual and Predicted Outputs (RBF)');
xlabel('Sample Index');
ylabel('Output Value');
legend;
grid on;
```

ابتدا داده‌ها (شامل ویژگی‌های آموزشی و هدف‌ها) به‌صورت ماتریس ستونی تنظیم می‌شوند تا بتوانند به عنوان ورودی شبکه استفاده شوند. سپس با استفاده از تابع newrb یک شبکه RBF ایجاد و آموزش داده می‌شود. پارامترهای مهمی نظیر spread (پهنای تابع) RBF، goal(مقدار خطای مطلوب)، و max_neurons(حداکثر تعداد نرون‌های لایه مخفی) تنظیم می‌شوند. شبکه به صورت تدریجی گره‌هایی به لایه مخفی اضافه می‌کند تا زمانی که خطای آموزشی به مقدار هدف برسد یا تعداد نرون‌ها به حد مجاز

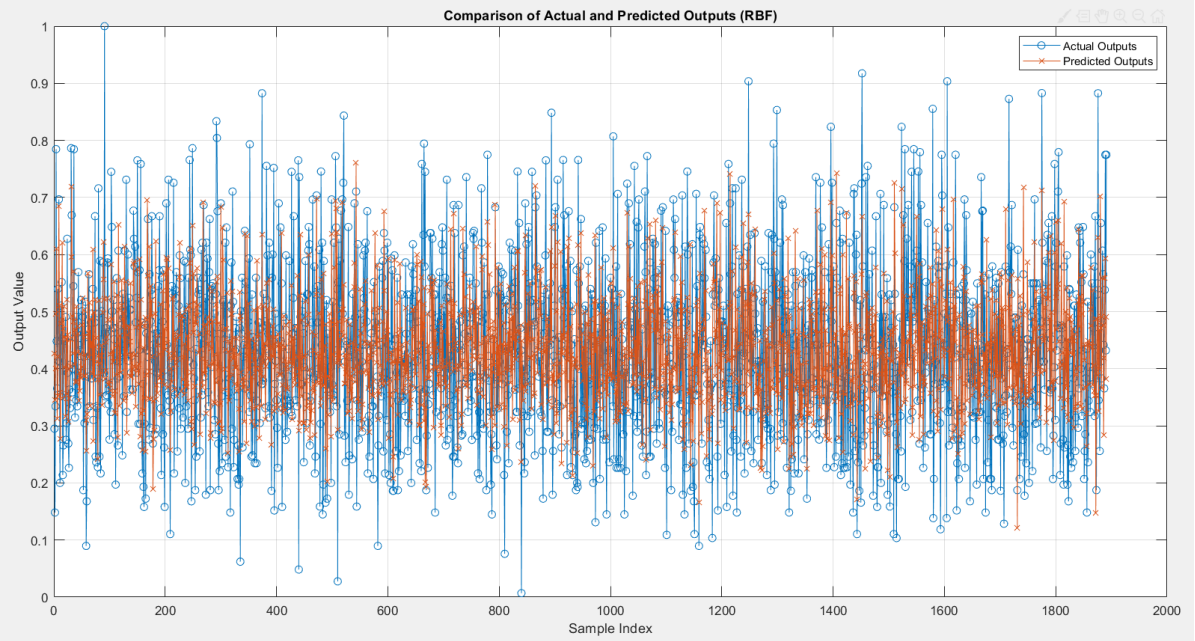
برسد. پس از آموزش، شبکه با داده‌های تست ارزیابی می‌شود. خروجی‌های پیش‌بینی شده توسط شبکه با داده‌های واقعی مقایسه شده و خطای میانگین مربعات (MSE) محاسبه می‌شود تا دقت مدل بررسی شود. در نهایت، نموداری ترسیم می‌شود که مقایسه‌ای بین خروجی‌های واقعی و پیش‌بینی‌شده را نمایش می‌دهد. این نمودار کمک می‌کند تا کیفیت مدل به صورت بصری ارزیابی شود .MSE پایین‌تر نشان‌دهنده عملکرد بهتر مدل است.

```
% RBF net Config
spread = 1.5;
goal = 0.001;
max_neurons = 150;
increase_rate = 1;
```

نتایج:



- خط آبی در طول ایپاک‌ها کاهش پیدا کرده و در نزدیکی مقدار هدف 0.001متوقف شده است. این نشان می‌دهد که شبکه به خوبی آموزش دیده و عملکرد مطلوبی دارد.
- شبکه زمانی متوقف شده است که خطای عملکرد دیگر بهبود چشم‌گیری نداشته یا به مقدار هدف رسیده است.



Mean Squared Error (MSE) on Test Data: 0.015255

- **دایره‌های آبی** :**(Actual Outputs)** مقادیر واقعی هدف (خروجی‌های مورد انتظار) از داده‌های تست.
- **ضربدرهای نارنجی** :**(Predicted Outputs)** خروجی‌هایی که توسط شبکه RBF پیش‌بینی شده‌اند.
- **محور افقی** :ایندکس نمونه‌ها در داده‌های تست.
- **محور عمودی** :مقادیر خروجی.

حالا در محیط متلب، ANFIS را پیاده سازی می کنیم:

کدش درون فایل متلب هست، حقیقتا ران نمیشه، بنابراین تحلیلی نمیتونم انجام بدم.

- ابتدا یک سیستم فازی (FIS) با استفاده از داده‌های آموزشی ایجاد می‌شود.
- سپس این سیستم فازی آموزش داده می‌شود.
- پیش‌بینی‌ها با استفاده از مدل ANFIS انجام می‌شود و خطای میانگین مربعات (MSE) محاسبه می‌شود.

سوال چهارم

با استفاده از محیط متلب، کد زیر را برای سیستم می نویسیم:

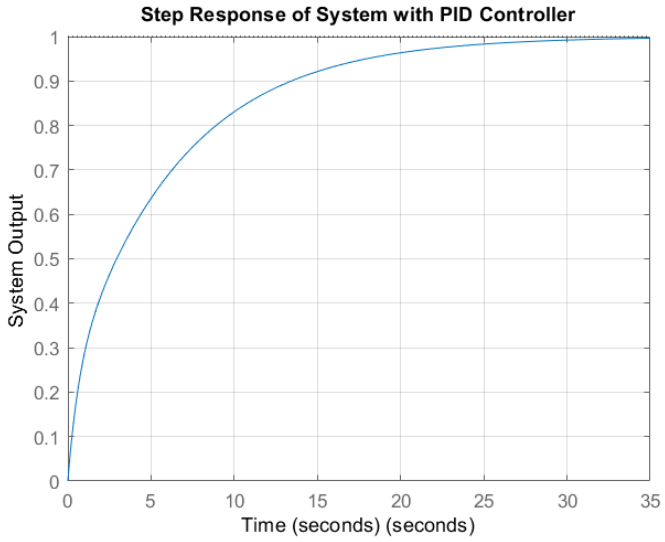
```
% Main Script: PID Controller Tuning using Ziegler-Nichols Method

% Define system parameters
num = 1;
den = [1 2 1];
sys = tf(num, den); % Create the system

Kp = 1; % Initial proportional controller gain
sys_cl = feedback(Kp * sys, 1); % Closed-loop system

[Ku, Pu] = find_critical_values(sys);

% PID controller coefficients using Ziegler-Nichols method
Kp = 0.6 * Ku;
Ti = Pu / 2;
Td = Pu / 8;

Ki = Kp / Ti;
Kd = Kp * Td;

% PID controller
C_pid = pid(Kp, Ki, Kd);

% Closed-loop system with PID controller
sys_cl_pid = feedback(C_pid * sys, 1);

% Step response of the closed-loop system
figure;
step(sys_cl_pid);
title('Step Response of System with PID Controller');
xlabel('Time (seconds)');
ylabel('System Output');
grid on;

% Display PID controller coefficients
fprintf('PID Controller Coefficients:\n');
fprintf('Kp = %.4f\n', Kp);
fprintf('Ki = %.4f\n', Ki);
fprintf('Kd = %.4f\n', Kd);

% Time response with stability display
[y, t] = step(sys_cl_pid);
figure;
plot(t, y, 'LineWidth', 1.5);
title('Time Response of System with PID Controller');
xlabel('Time (seconds)');
ylabel('System Output');
grid on;
```

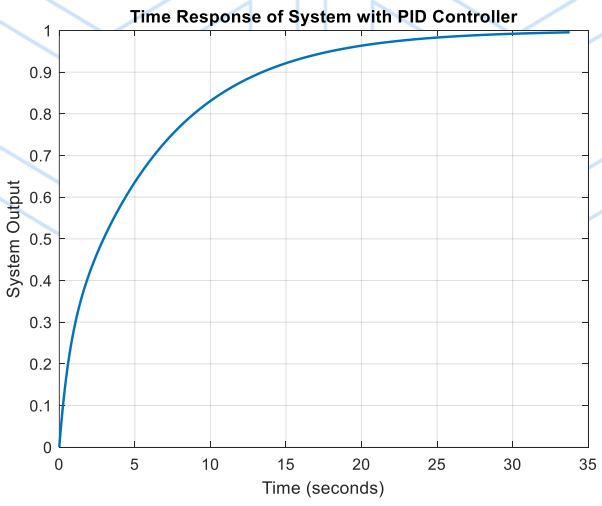
```
% Function to find Ku and Pu
function [Ku, Pu] = find_critical_values(sys)
    Ku = 1; % Initial guess for critical gain
    delta_K = 0.1; % Increment step for gain
    max_iter = 1000; % Maximum number of iterations

    % Initialize variables
    is_unstable = false;
    oscillation_detected = false;

    for i = 1:max_iter
        sys_cl = feedback(Ku * sys, 1); % Closed-loop system
        [y, t] = step(sys_cl, 0:0.01:100); % Step response for long enough time

        % Check stability by analyzing the poles
        poles = pole(sys_cl);
        if any(real(poles) >= 0)
            % System becomes unstable, stop incrementing gain
            is_unstable = true;
            break;
        end

        % Detect oscillations in the step response
        [~, locs] = findpeaks(y);
        if numel(locs) >= 2
            % Oscillations detected
            oscillation_detected = true;
            break;
        end

        % Increment the gain
        Ku = Ku + delta_K;
    end

    if ~oscillation_detected
        error('Unable to find Ku: System does not exhibit sustained oscillations.');
    end

    if ~is_unstable
        warning('Ku found, but system may not be fully critical.');
    end

    % Calculate critical period Pu from peaks in the step response
    [~, locs] = findpeaks(y);
    if numel(locs) > 1
        Pu = t(locs(2)) - t(locs(1));
    else
        error('Unable to determine Pu: Not enough oscillations detected.');
    end
```

```
end
```

در این کد، ابتدا سیستم تعریف می شود؛ سپس با تعریف تابع find_critical_values مقدار بحرانی ku را بدست می آوریم. این بهره مقداری است که سیستم در آن نوسانات پایدار را تجربه می کند. پس از این ضرایب PID را تنظیم(tune) می کنیم و سیستم حلقه بسته را شبیه سازی می کنیم.

(کد به صورتی است که در صورت عدن یافتن مقادیر، خطا نمایش دهد)





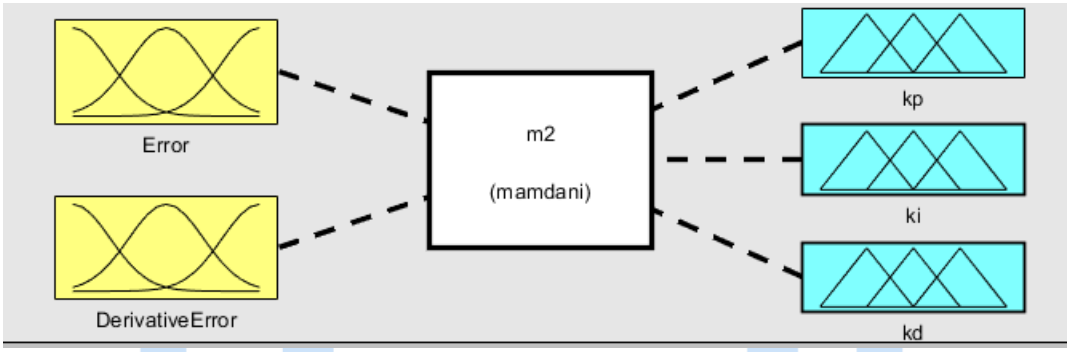
PID Controller Coefficients:

Kp = 0.6000

Ki = 0.1911

Kd = 0.4710

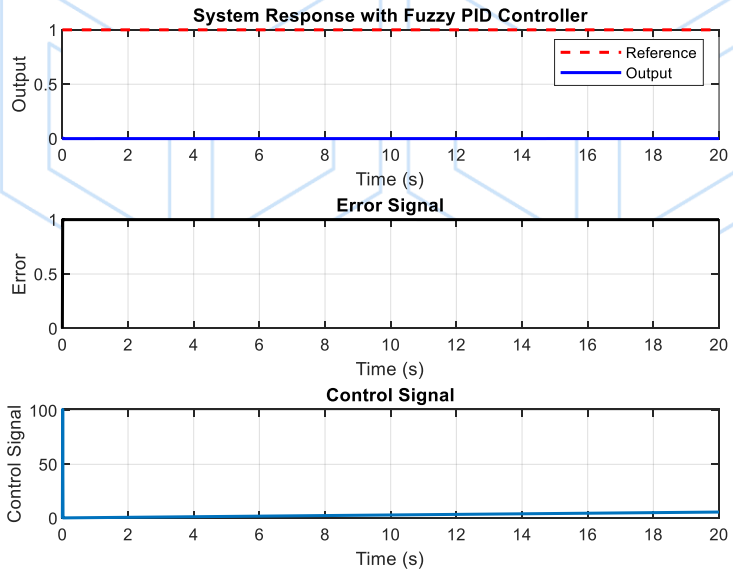
پاسخ پله نشان می‌دهد که سیستم به درستی با کنترل‌کننده PID تنظیم شده است.



یک سیستم فازی به صورت شکل بالا تعریف میکنیم(ورودی ها بین -۱ و ۱ و خروجی ها بین ۰ تا ۲)

قوانین را اضافه می کنیم.

دستی اضافه کردن قوانین خیلی زمان بر بود وفقط ۱۲ تا از قوانین اضافه شد، به هر حال خروجی با همان ۱۲ قانون به صورت زیر شد:



اگر سیستم ما خطی و ساده باشد، کنترل‌کننده PID معمولی عملکرد مناسبی ارائه داده و نیاز به طراحی پیچیده فازی PID نخواهد بود. اما اگر سیستم غیرخطی یا پیچیده باشد، کنترل‌کننده فازی PID گزینه بهتری است، زیرا ضرایب آن به صورت پویا تنظیم شده و برای تغییر شرایط کاری مانند تغییر بار، عدم قطعیت یا نویز مناسب‌تر است. همچنین، فازی PID معمولاً در کاهش زمان نشست، کاهش خطای حالت ماندگار، و بهبود پایداری کلی سیستم عملکرد بهتری نسبت به PID معمولی ارائه می‌دهد. در کل:

- کنترل‌کننده PID معمولی به‌راحتی می‌تواند پاسخ مناسبی ارائه دهد، زیرا سیستم خطی و ساده است.
- اگر نیاز به کنترل دقیق‌تر یا تطبیق‌پذیری در برابر نویز یا تغییرات سیستم دارید، می‌توانید از فازی PID استفاده کنید.