به نام خدا

نویسنده:

محمدعلی مجتهدسلیمانی

♦ توضیحات پیاده سازی سوال ٦:

❋موارد خواسته شده در موارد الف و ب را در حین توضیح پیاده سازی اشاره خواهم کرد و در آخر بعد از اتمام توضیحات مورد ج را پاسخ میدهم.

• بخش اضافه کردن کتابخانه:

```
import numpy as np
import matplotlib
matplotlib.use('TkAgg')
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
from scipy.optimize import minimize
```

با توجه به اینکه بنده از IDE، Pycharm استفاده میکنم نیاز دارم که برای استفاده از کتابخانه matplotlib، backend را روی TkAgg برای استفاده از این کتابخانه قرار دهم.

از کتابخانه scipy برای به حداقل رساندن تابع loss برای بهینه سازی عرض RBF استفاده کرده ایم.

- **پیاده سازی تابع RBF:**

```
def rbf(x, center, sigma):
    if sigma <= 0:
        sigma = abs(sigma)
    return np.exp(-((x - center) ** 2) / (2 * sigma ** 2))
```

برای شبکه RBF از تابع Gaussian استفاده کرده ایم، X بردار ورودی و مراکز و عرض شبکه را به عنوان ورودی به شبکه داده ایم. ضمنا با استفاده از قدر مطلق مطمئن میشویم عرض همان سیگما حتما مثبت باشد.

- **پیاده سازی تابع target:**

```
def target_function(x):
    return np.sin(x) + np.cos(2 * x) - 3 * x + 1
```

تابع هدفی که در خود سوال داده شده بود را به عنوان هدف قرار دادیم شبکه RBF قرار است این تابع را تخمین بزند.

- **بخش تنظیم ورودی:**

```
x_train = np.linspace(-5, 5, 250)
y_train = target_function(x_train)
centers = np.linspace(-5, 5, 5)
```

X_train ورودی ما هست که یک مجموعه ۲۵۰ تایی نقاط به طور مساوی بین -۵ تا ۵ هستند. y_train همان خروجی واقعی هست که توسط تابع target برای ورودی x تولید میشود. همچنین مراکز را هم تعریف کرده ایم.

- **بخش ساخت feature برای RBF:**

```
def create_rbf_features(x, centers, widths):
    features = np.zeros((len(x), len(centers)))
    for i, center in enumerate(centers):
        features[:, i] = rbf(x, center, widths[i])
    return features
```

ورودی را به عنوان ماتریس ویژگی تبدیل میکند. ابتدا یک ماتریس دو بعدی با مقدار دهی اولیه صفر میسازد ابعاد این ماتریس بر اساس ورودی و مراکز خواهند بود. ماتریس نهایی برای شبکه مهم است و بیانگر ورودی تبدیل شده برای سازگاری با وزن ها خواهد بود.

- **تابع loss**

```
def loss_function(params):
    widths = params
    X_rbf = create_rbf_features(x_train, centers, widths)
    weights = np.linalg.pinv(X_rbf) @ y_train
    y_pred = X_rbf @ weights
    return mean_squared_error(y_train, y_pred)
```

در اینجا میزان خطا را بین خروجی واقعی با مقدار پیشبینی شده ما محاسبه میکنیم بر اساس میانگین مربعات. این کار برای بهینه سازی عرض شبکه مورد نیاز است.

• بهینه سازی

```
initial_widths = np.ones(len(centers)) * 2.0

result = minimize(loss_function, initial_widths, method='BFGS')

optimized_widths = result.x
print("Optimized Widths:", optimized_widths)
```

مقدار ابتدایی عرض را ابتدا ۲ قرار میدهیم. بعد از روش BFGS به عنوان الگوریتم بهینه سازی استفاده میکنیم تا بتوانیم بهترین عرض را متناسب با هر نورون پیدا کنیم. خروجی عرض بهینه شده خواهد بود.

• بخش نهایی RBF

```
X_rbf = create_rbf_features(x_train, centers, optimized_widths)

weights = np.linalg.pinv(X_rbf) @ y_train
```

ماتریس RBF را با عرض بروز شده دوباره میسازد. و در نهایت با استفاده از PSEUDO-INVERSE خروجی وزن ها را محاسبه میکند.

• پیشبینی

```
def predict(x, centers, widths, weights):
    features = create_rbf_features(x, centers, widths)
    return features @ weights
```

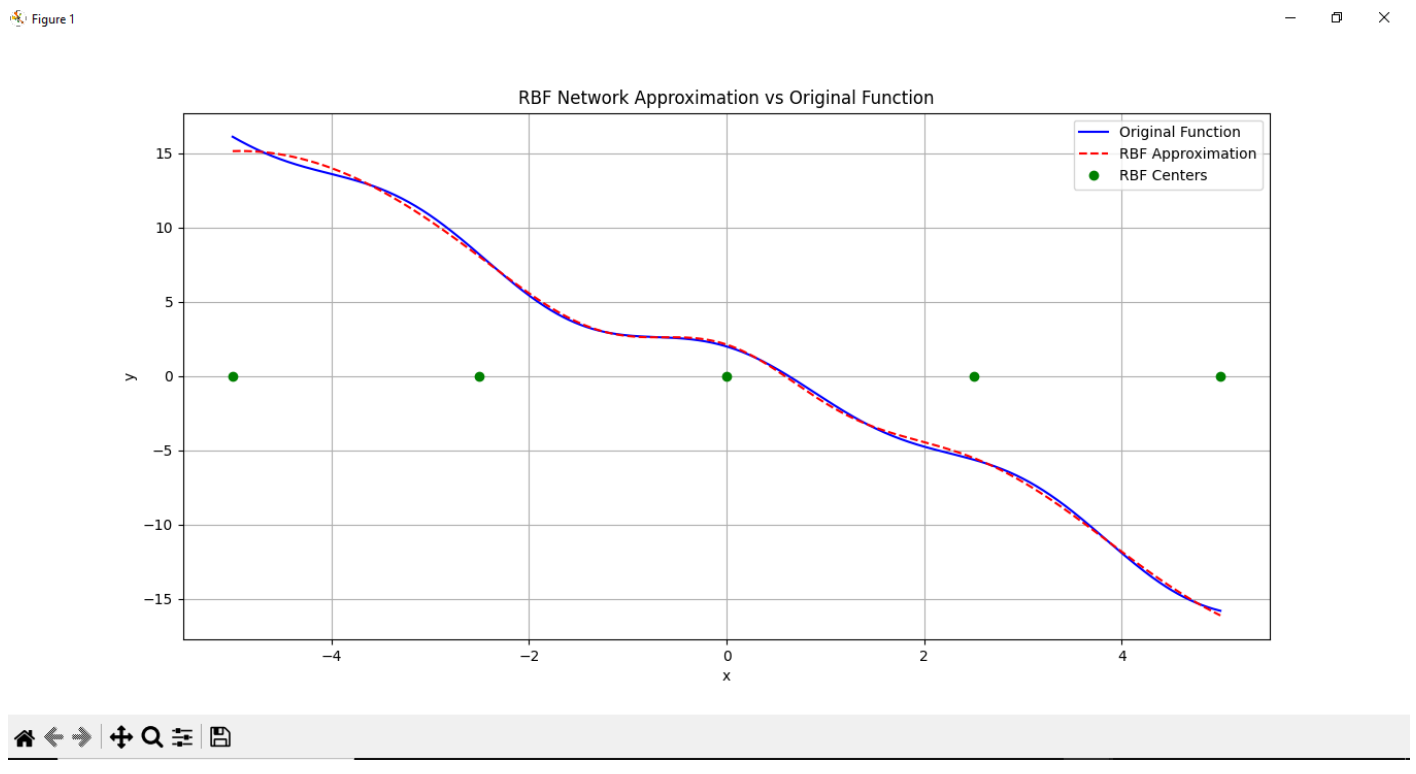
ورودی جدید را بر اساس شبکه ای داریم پیشبینی میکند.

در ادامه خروجی را نمایش میدهیم و خطا را محاسبه میکنیم. همانطور که در موارد الف و ب خواسته شده بود هم از پنج نورون با مراکز با فاصله ی مساوی در بازه ۵- تا ۵ استفاده کردیم هم پارامتر عرض را بهینه کردیم بر اساس هر نورون تا بتوانیم بهترین تقریب را در بازه های ورودی داشته باشیم.

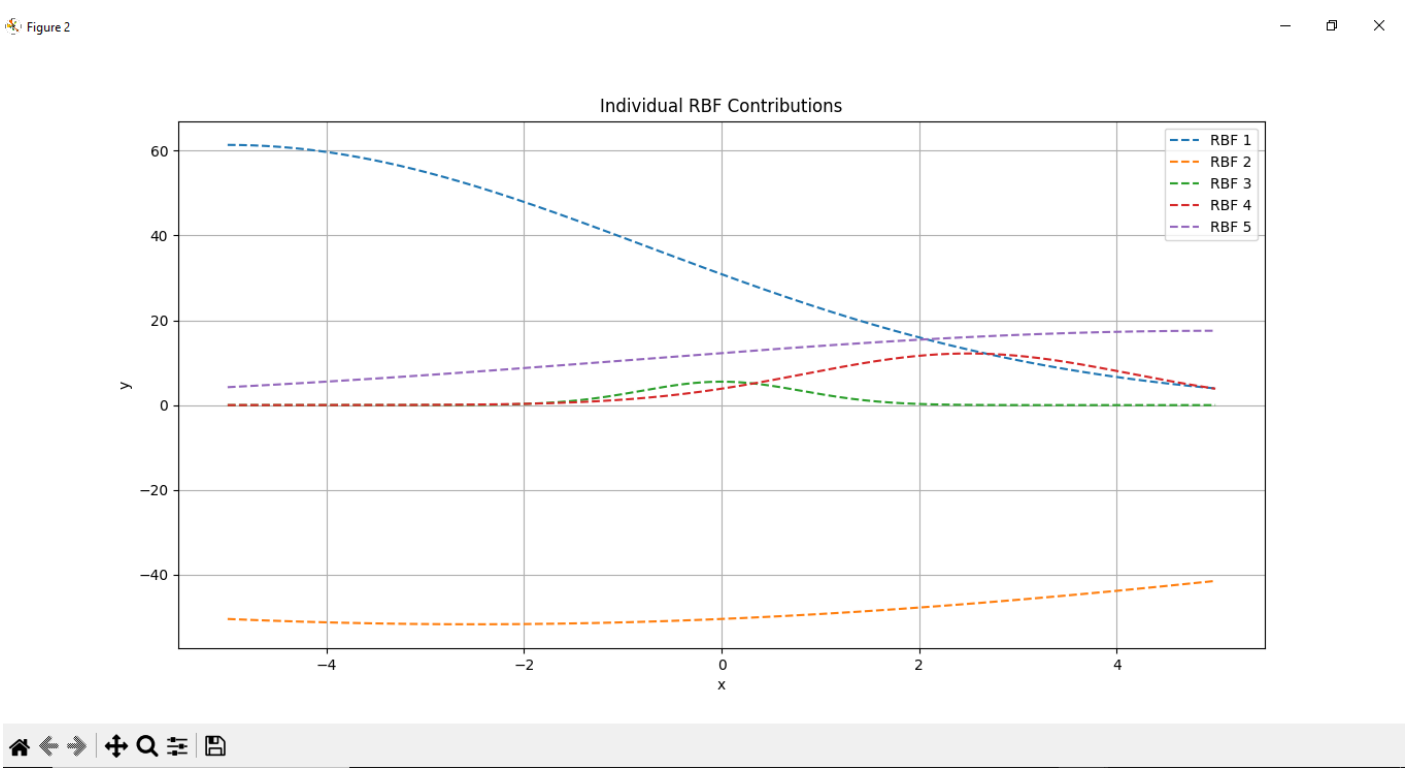
```
Optimized Widths: [ 4.26608489 11.29380683 -0.81383561  1.6513558   5.92829487]
Mean Squared Error: 0.050989
```

لیستی از عرض های بهینه شده همچنین خروجی MSE.

♦ بخش ج



خط آبی تابع اصلی ما و خط قرمز تقریب ما بوده است. نقاط سبز مراکز ما هستند.



خروجی های مختلف شبکه RBF بر اساس عرض های بهینه سازی شده. این خروجی ها تاثیر هر کدام از RBF ها را نشان میدهند و ترکیب اینها باعث رسیدن به مدل تقریبی نهایی شده است.

✓ تعداد نورون های RBF به ۲ طریق روی عملکرد شبکه تاثیر میگذارند:

۱. افزایش تعداد نورون ها باعث بهتر شدن دقت تخمین میشود اما در نهایت محاسبات را پیچیده میکند.

۲. کاهش تعداد نورون ها باعث میشود تخمین ما دقیق نباشد و ضعیف باشد گرچه که باعث کاهش محاسبات میشود.

✓ تحلیل نقاط قوت و ضعف:

به طور کلی شبکه توانسته است شکل کلی تابع را تخمین بزند. مراکز با فضای به طور مساوی به خوبی توانسته اند پوشش خوابی بر ورودی اعمال کنند. با استفاده از عرض بهینه سازی شده به خوبی توانسته ایم به تابع خواسته شده نزدیک شویم.

اگر چه که استفاده از ۵ نورون دقت تخمین را محدود کرده است. همچنین اگر مجموعه داده ای داشتیم که پراکندگی بیشتری داشت استفاده از تعداد ثابت مراکز با فاصله مساوی شاید مناسب نباشد و نقطه ضعف باشد و موجب عدم موفقیت در مدل کردن الگو شود. همچنین مدل ما مستعد بیش بردازش است زیرا که بهینه سازی به طور مستقیم روی مجموعه آموزشی صورت گرفته است و نسبت به داده های جدید دیده نشده ارزیابی نشده است.