به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر

درس سیستم‌های هوشمند

تمرین شماره۳

نام و نام خانوادگی محمدمهدی رحیمی

شماره دانشجویی ۸۱۰۱۹۷۵۱۰

آذر ۱۴۰۰

فهرست سوالات

سوال ۲

در بخش اول با استفاده از روش گرادیان نزولی طی دو مرحله وزن ها را اصلاح می کنیم. در بخش بعدی به بررسی مسائلی که می توانند بازدهی و سرعت همگرایی و دقت ما را افزایش دهند می پردازیم و تحقیق می کنیم. در انتها نیز یک شبکه پرسپترون را پیاده سازی می کنیم.

الف: تحلیلی

در این بخش با توجه به شماره دانشجویی داریم:

$$w1 = \begin{bmatrix} 1.01 & -0.1 & 0.1 \\ 0.21 & -1 & 0.13 \end{bmatrix}$$

$$b1 = \begin{bmatrix} 0 \\ 0.11 \\ 1 \end{bmatrix}$$

$$w2 = [1.05 \quad -0.51 \quad 0.1]$$

$$b2 = 0.1$$

$$w3 = \begin{bmatrix} 0 \\ 1.1 \end{bmatrix}$$

$$b3 = 0.1$$

$$y = 5$$

اگر تابع هزینه را به شکل زیر در نظر بگیریم داریم:

$$L = \frac{1}{2}(\hat{y} - y)^2$$

حال با توجه به تابع هزینه و مقادیر ماتریس ها که با توجه به شماره دانشجویی بدست می آید و نرخ یادگیری داده شده وبا توجه به قاعده زنجیری که مشتق هر مرحله را با توجه به قسمت قبل محاسبه می کنیم داریم:

$$w_i^{k+1} = w_i^k - \alpha \nabla_{w_i} L \qquad b_i^{k+1} = b_i^k - \alpha \nabla_{b_i} L$$

$$\hat{y} = relu(w_2 \tanh(w_1 X + b_1) + b_2) + w_3 X + b_3$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_3} = (\hat{y} - y) X \qquad \frac{\partial relu(z)}{\partial z} = U(z)$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_2} = (\hat{y} - y) \times U(w_2 \tanh(w_1 X + b_1) + b_2) \times \tanh(w_1 X + b_1)$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_1} = (\hat{y} - y) \times U(w_2 \tanh(w_1 X + b_1) + b_2) \times w_2 \times \frac{1}{\cosh(w_1 X + b_1)^2} X$$

$$\frac{\partial L}{\partial b_3} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_3} = (\hat{y} - y)$$

$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_2} = (\hat{y} - y) \times U(w_2 \tanh(w_1 X + b_1) + b_2)$$

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial b_1} = (\hat{y} - y)\, U(w_2 \tanh(w_1 X + b_1) + b_2) \times w_2 \times \frac{1}{\cosh(w_1 X + b_1)^2}$$

$$w_1^1 = \begin{bmatrix} 1.0091 & -0.1009 & 0.0991 \\ 0.2087 & -1.0013 & 0.1287 \end{bmatrix} \qquad w_2^1 = [1.0362 \quad -0.4961 \quad 0.0872]$$

$$b_1^1 = \begin{bmatrix} -0.0004 \\ 0.1096 \\ 0.9996 \end{bmatrix} \qquad b_2^1 = 0.0861 \qquad b_3^1 = 0.0861$$

$$w_3^1 = [-0.0279 \quad 1.0582]$$

$$w_1^2 = \begin{bmatrix} 1.0098 & -0.1002 & 0.0998 \\ 0.2097 & -1.0003 & 0.1297 \end{bmatrix} \qquad w_2^2 = [1.047 \quad -0.5070 \quad 0.0972]$$

$$b_1^2 = \begin{bmatrix} -0.0001 \\ 0.1099 \\ 0.9999 \end{bmatrix} \qquad b_2^2 = 0.097 \qquad b_3^2 = 0.097$$

$$w_3^2 = [-0.006 \quad 1.0909]$$

شکل۲-۱ : حل دستی قسمت الف

نتایج این قسمت با کد متلب نیز بررسی شده است.

ب: تحقیق

۱:تابع هزینه رگرسیون

ابتدا شکل تابع هزینه های مختلف را نمایش می دهیم:

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

شکل۲-۲ تابع هزینه L1

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

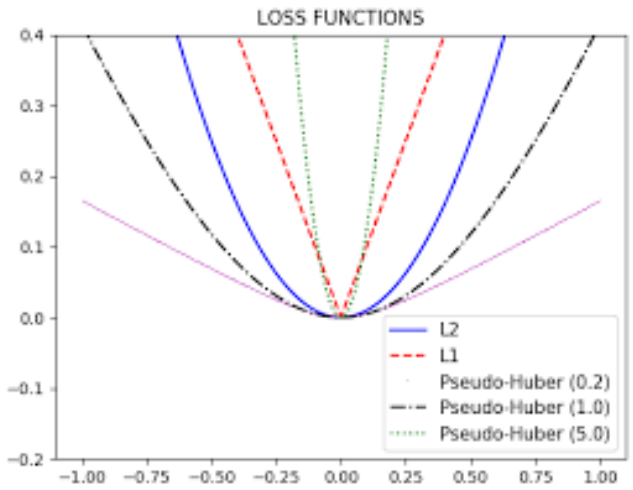
شکل ۲-۳ تابع هزینه L2

$$Huber = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}(y_i - \hat{y}_i)^2 \qquad |y_i - \hat{y}_i| \le \delta$$

$$Huber = \frac{1}{n}\sum_{i=1}^{n} \delta\left(|y_i - \hat{y}_i| - \frac{1}{2}\delta\right) \qquad |y_i - \hat{y}_i| > \delta$$

شکل ۲-۴ تابع هزینه هوبر

تابع هزینه هوبر در مقادیر کم مانند تابع هزینه L2 عمل می کند و در مقادیر دیگر مانند L1 می باشد بنابر این با توجه به تعیین پارامتر آن می توان آن را تغییر داد و با توجه به استفاده مورد نظر بهترین بهره را از آن برد. و تابه هزینه هوبر مانند ترکیب دو تابع هزینه دیگر می باشد که در شکل زیر مقایسه آن ها را مشاهده می کنیم.



شکل ۲-۵ مقایسه تابع هزینه ها

۲: استفاده از داده ارزیابی

یعنی مدل در موقیعت مناسبی می باشد که نه دچار بیش برازش و کم برازش نشدیم و در حالت مناسبی می باشد و بهتر است آموزش را ادامه ندهیم زیرا در حالتی که کم برازش باشد این خطا زیاد می باشد و با اموزش خطا کمتر می شود تا به نقطه بهینه برسد. اما اگر در نقطه بهینه آموزش را ادامه بدهیم خطا شروع به افزایش می کند و دچار بیش برازش می شویم.

۳: گرادیان نزولی به همراه تکانه

در روش گرادیان نزولی چون باید تمام داده ها در محاسبات باشند بسیار کند می باشد اما گرادیان نزولی تصادفی چون در هر بار از محاسبات به صورت تصادفی به تعداد محدودی از داده ها توجه می شود و در محاسبات می باشند پس سرعت آن بالا تر خواهد بود. و اما سریع ترین روش در بین گزینه ها بیشترین سرعت را دارد زیرا مانند روش گرادیان نزولی تصادفی تعداد محدودی در محاسبات می باشند اما به دلیل وجود ترمی که مقدار کاهش را اصلاح می کند نویز کمتر می شود و سریع تر همگرا می شود.

پ: پیاده سازی شبکه پرسپترون در کاربرد رگرسیون

۱: تولید دادگان

در این بخش با استفاده از کدی که در ادامه آمده است داده ای تولید می کنیم که با توجه به تعدادی که می خواهیم داده تولید شود به تابع مورد نظر می دهیم و اعداد نرمالایز شده در بازه مورد نظر تولید می کند سپس به سه دسته تست و آموزش و داده ارزیابی تقسیم می کند و خروجی متناسب نیز محاسبه می شود. توابع استفاده شده به شکل زیر می باشند:

```
def data_maker(num_of_data):
    x = np.random.uniform(0, 2 * np.pi, num_of_data)
    y = np.random.uniform(0, 2 * np.pi, num_of_data)
    tag = np.sin(x + y)
    x = normalize(x)
    y = normalize(y)
    return x,y,tag

def split_data(x,y,tag):
    df = pd.DataFrame([x, y]).T
    data_train_with_valid = df.sample(frac=0.8)
    data_test = df.drop(data_train_with_valid.index)
    data_valid = data_train_with_valid.sample(frac=0.2)
    data_train = data_train_with_valid.drop(data_valid.index)
    tag_train = tag[data_train.index]
    tag_test = tag[data_test.index]
    tag_valid = tag[data_valid.index]
    return data_train,data_test,data_valid,tag_train,tag_test,tag_valid
```

۲: پیش پردازش

این قسمت نیز در قسمت قبل توسط تابعی انجام می شود که به شکل زیر می باشد:

```
def normalize (x):
    return((x - min(x)) / (max(x) - min(x)))
```

۳:پیاده سازی مدل

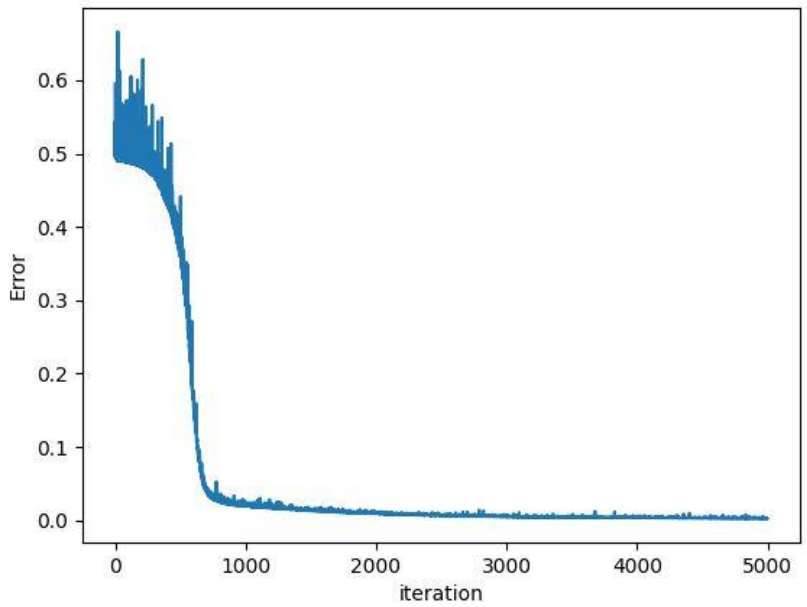
کل کد در انتها قرار داده می شود که شامل این بخش نیز می شود. در ابتدا از تابع فعال ساز Relu استفاده شد که نتیجه مطلوبی نداشت سپس از تابع فعال ساز tanh استفاده شد که نتیجه بسیار مطلوب تر شد. در کد موارد مختلفی هست که می توان آن ها را تغییر داد که در حالت فعلی روی مقادیر معقولی قرار دارند. برای مثال تعداد نود ها و تعداد تکرار حلقه برای آموزش شبکه و تعداد داده ها نیز قابل تغییر می باشد

۴: ارزیابی

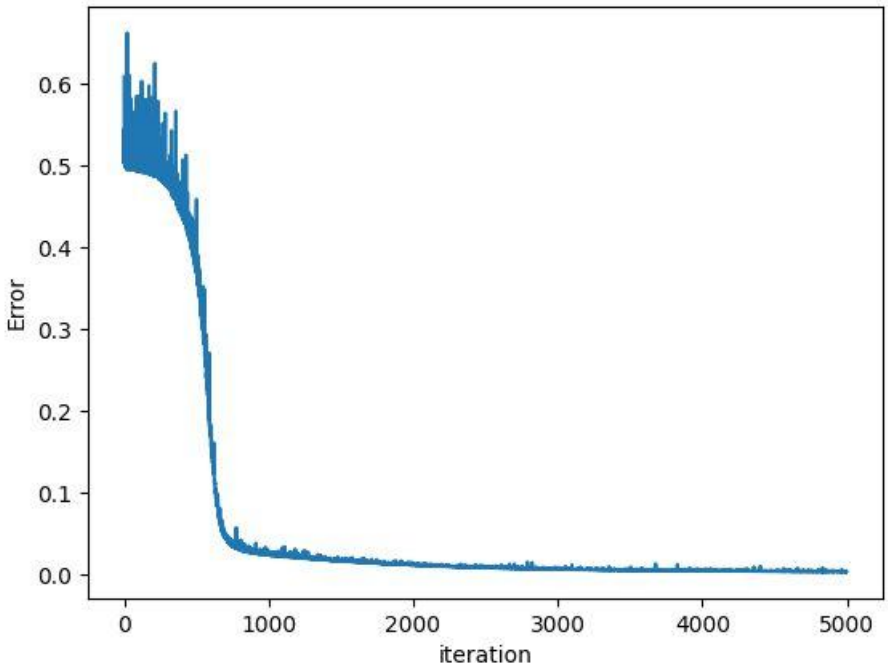
در انتهای کد موارد خواسته شده نمایش داده می شوند که به شکل زیر می باشد.

	Error
Train	0.0034136139100287 11
Validation	0.0032744946088567 06
Test	0.0028234047857023754

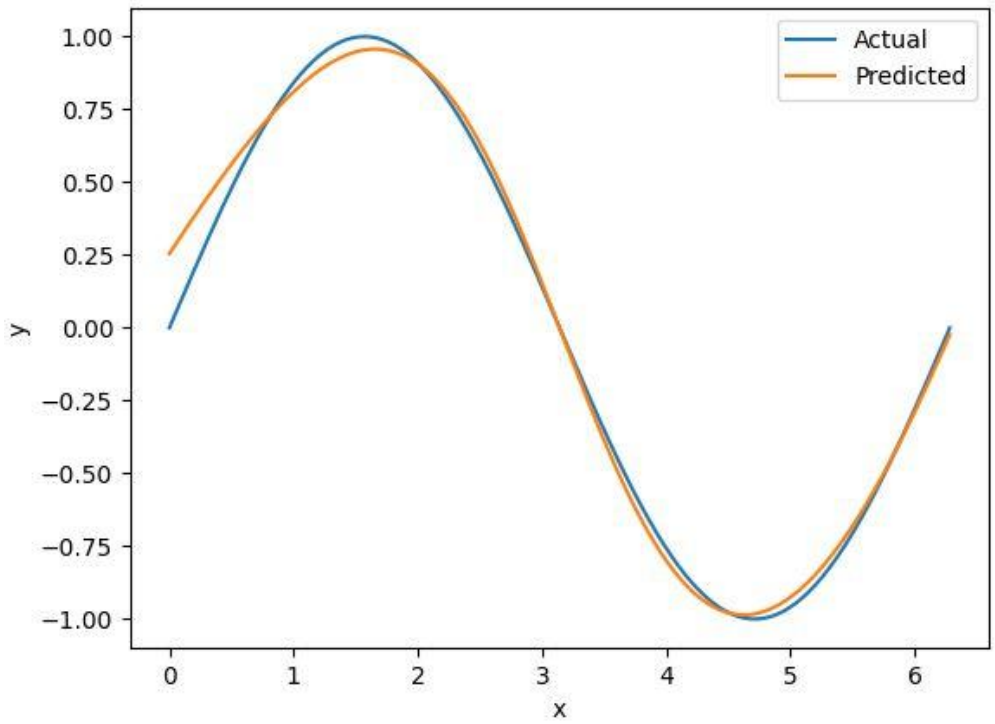
شکل ۲-۶ مقدار خطا برای داده آموزش و داده ارزیاب و داده تست



شکل ۲-۷ نمودار خطا شبکه برای داده تست برحسب تعداد تکرار



شکل۲-۸ نمودار خطاشبکه برای داده آموزش برحسب تعداد تکرار



شکل۲-۹ تابع sin و خروجی شبکه پرپترون که تخمینی از sin می باشد

همانطور که مشاهده شد نتیجه تا حد قابل قبولی مطلوب بود. و کد کامل در انتها در قسمت پیوست می باشد و ضمیمه نیز خواهد شد که در آن می توان با تغییر متغیر های مختلف نتیجه متفاوت گرفت که البته اگر هر بار کد را اجرا کنید به دلیل رندوم بودن قسمت هایی نتایج مختلف خواهند بود.

پیوست:

کد سوال ۲ قسمت پ:

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

def normalize (x):      #function for normalization
    return((x - min(x)) / (max(x) - min(x)))

def d_tanh(x):          #function of derivative tanh
    return (1 - np.tanh(x) ** 2)

def data_maker(num_of_data):        #this function make data which you can change number
    x = np.random.uniform(0, 2 * np.pi, num_of_data)
    y = np.random.uniform(0, 2 * np.pi, num_of_data)
    tag = np.sin(x + y)
    x = normalize(x)
    y = normalize(y)
    return x,y,tag

def split_data(x,y,tag):            #split data to validation , train and test with tags
    df = pd.DataFrame([x, y]).T
    data_train_with_valid = df.sample(frac=0.8)
    data_test = df.drop(data_train_with_valid.index)
    data_valid = data_train_with_valid.sample(frac=0.2)
    data_train = data_train_with_valid.drop(data_valid.index)
    tag_train = tag[data_train.index]
    tag_test = tag[data_test.index]
    tag_valid = tag[data_valid.index]
    return data_train,data_test,data_valid,tag_train,tag_test,tag_valid

def predict( x,w1,w2,b1,b2):        #use weight to calculate output
    x = x/1.0
    y1 = np.tanh(np.dot(x,w1) +b1.T)
    y = np.dot(y1, w2) + b2
    return [y, y1]



epo=5000            #number of iteration
num_of_data = 10000     #number of data
num_of_nodes = 15       #number of nodes
x,y,tag = data_maker(num_of_data)
data_train, data_test, data_valid, tag_train, tag_test, tag_valid = split_data(x,y,tag)
data_train = np.array(data_train)
```

```
tag_train = np.array(tag_train)

w1 = np.random.rand(2, num_of_nodes)        #initial weights
w2 = np.random.rand(num_of_nodes, 1)
b1 = np.random.rand(num_of_nodes, 1)
b2 = 1

train_error = np.zeros(epo)     #array of each iteration error for train data
test_error = np.zeros(epo)      #array of each iteration error for test data

for i in range(epo):        #update weights with gradient descent method
    for iter in np.random.randint(len(data_train), size=100):
        temp_train = data_train[iter]
        temp_tag = tag_train[iter]
        pred1,pred2 = predict(temp_train,w1,w2,b1,b2)
        error = temp_tag - pred1
        #gradient of weights
        delta_w1 = np.dot(temp_train.reshape(2, 1), error *d_tanh(np.dot(temp_train,w1) + b1.T) * w2.T)
        delta_w2 = np.dot(pred2.T, error)
        delta_b1 = error * d_tanh(np.dot(temp_train, w1) + b1.T) * w2.T
        delta_b2 = error
        #update weights
        w2 = w2 + 0.01 * delta_w2
        w1 = w1 + 0.01 * delta_w1
        b1 = b1 + 0.01 * delta_b1.T
        b2 = b2 + 0.01 * delta_b2
    #find predict with weights and calculate errror
    train_pred = predict(data_train,w1,w2,b1,b2)[0]
    test_pred = predict(data_test,w1,w2,b1,b2)[0]
    train_error[i] = ((train_pred.T - tag_train) ** 2).mean()
    test_error[i] = ((test_pred.T - tag_test) ** 2).mean()

#print errors of validation, train and test
valid_pred = predict(data_valid,w1,w2,b1,b2)[0]
valid_error = ((valid_pred.T - tag_valid) ** 2).mean()
print('              Error')
print('Train        ',train_error[-1])
print('Validation   ', valid_error)
print('Test         ',test_error[-1])
#plot errors of train and test
plt.plot(test_error)
plt.ylabel('Error')
plt.xlabel('iteration')
plt.show()
plt.plot(train_error)
plt.ylabel('Error')
plt.xlabel('iteration')
plt.show()
```

```
#next part which we should compare model with sine
xx = np.linspace(0, 2 * np.pi, 2000)
xx_normalize = normalize(xx)
yy = np.zeros(2000)
data = pd.DataFrame([xx_normalize, yy])
predicted = []
predicted = predict(data.T,w1,w2,b1,b2)[0]
predicted = np.array(predicted)
plt.plot(xx, np.sin(xx), label="Actual")
plt.plot(xx, predicted, label="Predicted")
plt.legend()
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```