به نام خدا

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

مبانی سیستم های هوشمند

گزارش مینی پروژه شماره دو

[ایمان فکری اسکی]

[۹۹۲۹۰۸۳]

استاد : آقای دکتر مهدی علیاری

آذرماه ۱۴۰۲

فهرست مطالب

عنوان | شماره صفحه

چکیده :

شبکه‌های عصبی به عنوان یک ابزار قوی در حوزه یادگیری عمیق و پردازش سیگنال‌های پیچیده شناخته می‌شوند. در این پروژه، ما به بررسی و بهبود عملکرد شبکه‌های عصبی متمرکز شده‌ایم. از روش‌های متنوعی برای آموزش شبکه‌ها استفاده کرده‌ایم، از جمله الگوریتم‌های بهینه‌سازی جدید و تکنیک‌های نورونی نوآورانه. در این گزارش، به تحلیل پیچیدگی مسائل پیش‌بینی، تعیین پارامترهای اثربخش در شبکه‌ها و بهینه‌سازی فرایند آموزش می‌پردازیم. همچنین، نتایج کاربردی این پروژه در پیش‌بینی الگوهای زمانی و تصمیم‌گیری‌های پیچیده را ارائه خواهیم داد. این پروژه نه تنها به افزایش دقت پیش‌بینی‌ها منجر شده است بلکه به بهبود قابلیت تفسیرپذیری و عملکرد کلی شبکه‌های عصبی نیز انجامیده است. این پروژه یک نگاه جامع به نحوه بهینه‌سازی و توسعه شبکه‌های عصبی فراهم می‌کند و ارتقاء مدل‌های آینده در حوزه یادگیری عمیق را تسهیل می‌کند.

مقدمه :

قاعده پرسپترون:

در زمینه یادگیری ماشین، قاعده پرسپترون به عنوان یک ابتدایی‌ترین الگوریتم‌ها برای دسته‌بندی داده‌ها شناخته می‌شود. این قاعده به وسیلهٔ فرآیند یادگیری از داده‌ها، توانمندی دارد تا الگوها و تفاوت‌های بین دسته‌ها را شناسایی کرده و تصمیم‌گیری را براساس ویژگی‌های ورودی انجام دهد. در این گزارش، به مفهوم و ابعاد مختلف قاعده پرسپترون پرداخته و کاربردهای آن در مسائل دسته‌بندی را بررسی خواهیم کرد.

پردازش تصویر:

علم پردازش تصویر به عنوان یک حوزه فرعی از علوم کامپیوتر، مطالعه و تحلیل تصاویر و ویدئوها با استفاده از الگوریتم‌ها و تکنیک‌های متنوع است. در این گزارش، به معرفی اهمیت و کاربردهای پردازش تصویر در حوزه‌های مختلف از جمله پزشکی، صنعت، و تشخیص اجسام خواهیم پرداخت.

شبکه‌های MLP (Multilayer Perceptron):

شبکه‌های MLP به عنوان یکی از پیشرفته‌ترین مدل‌های شبکه‌های عصبی معرفی می‌شوند. این شبکه‌ها با دارا بودن لایه‌های مختلف و توانمندی در یادگیری انواع توابع پیچیده، در بسیاری از مسائل یادگیری عمیق کاربرد دارند. در این گزارش، به ساختار، عملکرد، و کاربردهای شبکه‌های MLP در زمینه‌های گوناگون می‌پردازیم.

شبکه‌های RBF (Radial Basis Function):

شبکه‌های RBF یک دسته از شبکه‌های عصبی هستند که بر اساس توابع پایه شعاعی عمل می‌کنند. این شبکه‌ها به خصوص در مسائل الگویابی و تشخیص الگوهای ناهمگن مؤثر هستند. در این گزارش، به معرفی اصول و کاربردهای شبکه‌های RBF در تشخیص الگوها و تصمیم‌گیری خواهیم پرداخت.

بخش۱: سوالات تحلیلی

سوال اول :

برای حل این سوال تا یک بخشی با استفاده از کتابخانه پیش رفته ایم و برای حل دقیق تر از کلاس استفاده کرده ایم :

```
!pip install --upgrade --no-cache-dir gdown
!gdown 1iPrPZZKjKAwEumQ3s8kGCvIMURt1TaF-
import pandas as pd
data = pd.read_csv("/content/Perceptron.csv")
data
# The first and second columns of the CSV file related to this data set
are related to the features and the third column is related to the class
of each data.
```

در ابتدا با توجه به کدی که در بالا مشاهده می کنید باید فایل داده های مورد نظر را در محیط کولب import کنیم و سپس باید داده ها را بخوانیم. در انتها داده های مورد نظر را در داخل data ذخیره می کنیم.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_decision_regions
```

در ادامه کتابخانه های مختلف را فراخوانی می کنیم.

```
x = data.iloc[:, :-1].values # x is features
y = data.iloc[:, -1].values  # y is target
# Transforming y values from {-1, 1} to {0, 1}
y = np.where(y == -1, 0, 1)
y
```

این کد به یک مجموعه داده (data) دسترسی دارد و دو متغیر x و y را از این داده استخراج می‌کند.

X مشخصه‌های ویژگی (feature) مجموعه داده را نمایش می‌دهد و از تمام ستون‌های مجموعه داده به جز آخرین ستون (ستون متغیر وابسته) برداشته می‌شود.

yمتغیر وابسته (target) مجموعه داده را نمایش می‌دهد و فقط از آخرین ستون مجموعه داده برداشته می‌شود. سپس، مقادیر متغیر y از مقادیر {-۱، ۱} به مقادیر {۰، ۱} تبدیل می‌شوند. این تبدیل به وسیله تابع np.where

انجام می‌شود، به گونه‌ای که هر جایگاهی که y برابر با -۱ باشد، به ۰ تبدیل می‌شود و در غیر این صورت به ۱ تبدیل می‌شود.

```
# PART1
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

این کد به دو مجموعه آموزشی و آزمون از داده‌های x و y اصلی شما تقسیم می‌کند. این تقسیم بر اساس یک نسبت آزمون به آموزش (test_size) به اندازه ۲۰٪ از کل داده انجام می‌شود.

x_train : مجموعه ویژگی‌های آموزشی

x_test : مجموعه ویژگی‌های آزمون

y_train : مجموعه برچسب‌های آموزشی

y_test : مجموعه برچسب‌های آزمون

در انتها، ابعاد (تعداد سطرها و ستون‌ها) هر یک از این چهار مجموعه نمایش داده می‌شود.

```
# Initialize Perceptron classifier with a different threshold (example:
0.5)
model = Perceptron()

# Train the perceptron with the new threshold
model.fit(x_train, y_train)
```

در این قسمت، یک مدل Perceptron ایجاد شده و با استفاده از داده‌های آموزشی x_train و y_train آموزش داده می‌شود. از آنجا که یک آستانه جدید مقداردهی شده است (با مقدار ۰٫۵ به عنوان مثال)، ممکن است این مدل تفاوتی در تصمیم‌گیری نسبت به یک مدل با آستانه متفاوت نشان دهد.

```
# PART2
# Accuracy on train and test data with the new threshold
train_accuracy = model.score(x_train, y_train)
test_accuracy = model.score(x_test, y_test)

print(f"Accuracy on train set with new threshold: {train_accuracy}")
print(f"Accuracy on test set with new threshold: {test_accuracy}")
# Convert y_test to integer type
y_test = y_test.astype(np.int)
```

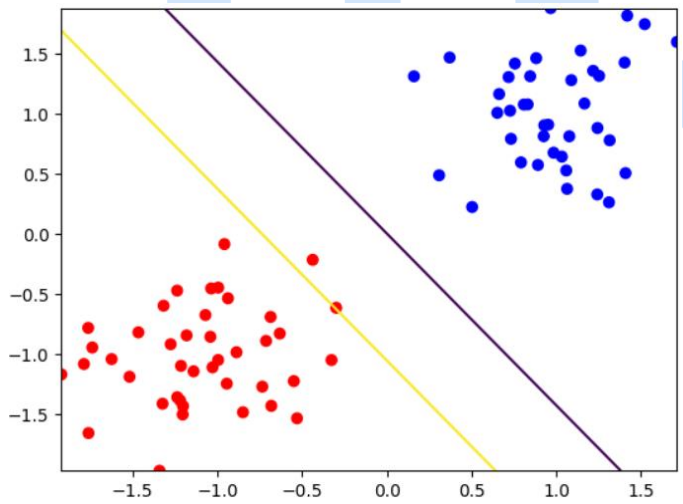
در این بخش از کد، دقت (accuracy) مدل Perceptron بر روی داده‌های آموزشی و آزمون با استفاده از آستانه threshold جدید محاسبه شده و نتایج چاپ می‌شوند. همچنین، برچسب‌های y_test به نوع داده‌ای integer

تبدیل می‌شوند. دقت نشان دهنده نسبت تعداد پیش‌بینی‌های صحیح به تعداد کل نمونه‌ها است و مقادیر بیشتر به معنای دقت بالاتر هستند. در نهایت، برچسب‌های y_test به نوع داده‌ای integer تبدیل می‌شوند تا از آنها برای محاسبه معیارهای ارزیابی استفاده شود.

```
Accuracy on train set with new threshold: 1.0
Accuracy on test set with new threshold: 1.0
```

```
x1_min, x2_min = x_test.min(0)
x1_max, x2_max = x_test.max(0)
n=400
x1r = np.linspace(x1_min, x1_max, n)
x2r = np.linspace(x2_min, x2_max, n)
x1m, x2m = np.meshgrid(x1r, x2r)
xm = np.stack((x1m.flatten(), x2m.flatten()),axis=1)
ym = model.decision_function(xm)
colors = np.array(['blue', 'red'])
plt.scatter(x_test[:, 0], x_test[:, 1], c=colors[y_test])
plt.contour(x1m, x2m, ym.reshape(x1m.shape), levels=[0,1])
plt.show()
```

این بخش از کد برای تصویرسازی مرز تصمیم‌گیری مدل Perceptron بر روی داده‌های آزمون x_test به کمک دو ویژگی (x1 و x2) انجام می‌شود. ابتدا دامنه‌های مقادیر ویژگی‌های x1 و x2 برای نقاط آزمون مشخص می‌شوند. سپس با استفاده از این دامنه‌ها و تعدادی نقطه تشکیل شده توسط آنها، یک ماتریس مربعی ایجاد می‌شود. مدل Perceptron بر روی این نقاط اعمال شده و مقادیر تصمیم‌گیری برای هر نقطه محاسبه می‌شود. در نمودار نهایی، نقاط آزمون با رنگهای مختلف بر اساس برچسب‌های واقعی (آبی برای ۰ و قرمز برای ۱) نمایش داده شده‌اند. همچنین، خطوط مرز تصمیم‌گیری بین دو دسته با استفاده از توابع scatter برای نمایش نقاط و contour برای نمایش خطوط مرز رسم شده‌اند.



تا به اینجای کار به صورت اماده از ککتابخانه های آماده استفاده کرده ایم اما برای تغییر بایاس و ترشهولد باید حتما سیستم مورد نظرمان را به صورت دستی و با تعریف کلاس تعریف کنیم. برای همین منظور از روش scratch استفاده کرده ایم که به صورت زیر می باشد :

```
def relu(x):
    return np.maximum(0, x)
def sigmoid(x):
    return 1/(1+np.exp(-x))
def tanh(x):
    return np.tanh(x)
def bce(y, y_hat):
    return np.mean(-(y*np.log(y_hat) + (1-y)*np.log(1-y_hat)))
def mse(y, y_hat):
    return np.mean((y - y_hat)**2)
def accuracy(y, y_hat, t=0.5):
    y_hat = np.where(y_hat<t, 0, 1)
    acc = np.sum(y == y_hat) / len(y)
    return acc
class Neuron:

    def __init__(self, in_features, threshold, af=None, loss_fn=mse,
n_iter=100, eta=0.1, verbose=True):
        self.in_features = in_features
        # weight & bias
        self.w = np.random.randn(in_features, 1)
        self.threshold = threshold
        self.af = af
        self.loss_fn = loss_fn
        self.loss_hist = []
        self.w_grad = None
        self.n_iter = n_iter
        self.eta = eta
        self.verbose = verbose

    def predict(self, x):
        # x: [n_samples, in_features]
        y_hat = x @ self.w + self.threshold
        y_hat = y_hat if self.af is None else self.af(y_hat)
        return y_hat

    def decision_function(self, x):
        # x: [n_samples, in_features]
        y_hat = x @ self.w + self.threshold
        return y_hat
```

```
    def fit(self, x, y):
        for i in range(self.n_iter):
            y_hat = self.predict(x)
            loss = self.loss_fn(y, y_hat)
            self.loss_hist.append(loss)
            self.gradient(x, y, y_hat)
            self.gradient_descent()
            if self.verbose & (i % 10 == 0):
                print(f'Iter={i}, Loss={loss:.4}')

    def gradient(self, x, y, y_hat):
        self.w_grad = (x.T @ (y_hat - y)) / len(y)

    def gradient_descent(self):
        self.w -= self.eta * self.threshold

    def __repr__(self):
        af_name = self.af.__name__ if self.af is not None else None
        loss_fn_name = self.loss_fn.__name__ if self.loss_fn is not None
else None
        return f'Neuron({self.in_features}, {self.threshold}, {af_name},
{loss_fn_name}, {self.n_iter}, {self.eta}, {self.verbose})'

    def parameters(self):
        return {'w': self.w, 'threshold': self.threshold}
```

این کد یک کلاس به نام Neuron تعریف می‌کند که یک نورون (یا یک واحد شبکه عصبی) را با ویژگی‌های مختلفی مانند تعداد ویژگی‌های ورودی in_features آستانه threshold تابع فعال‌سازی af تابع هزینه loss_fn تعداد تکرارها n_iter نرخ یادگیری eta و قابلیت چاپ اطلاعات مراحل آموزش verbose ایجاد می‌کند. توابع فعال‌سازی اعمال شده در این کد شامل Rectified Linear Unit (ReLU)، Sigmoid و Hyperbolic Tangent (Tanh) هستند. علاوه بر این، سه تابع تابع هزینه نیز تعریف شده‌اند: Binary Cross Entropy (`bce`)، Mean Squared Error (`mse`) و تابع دقت accuracy.

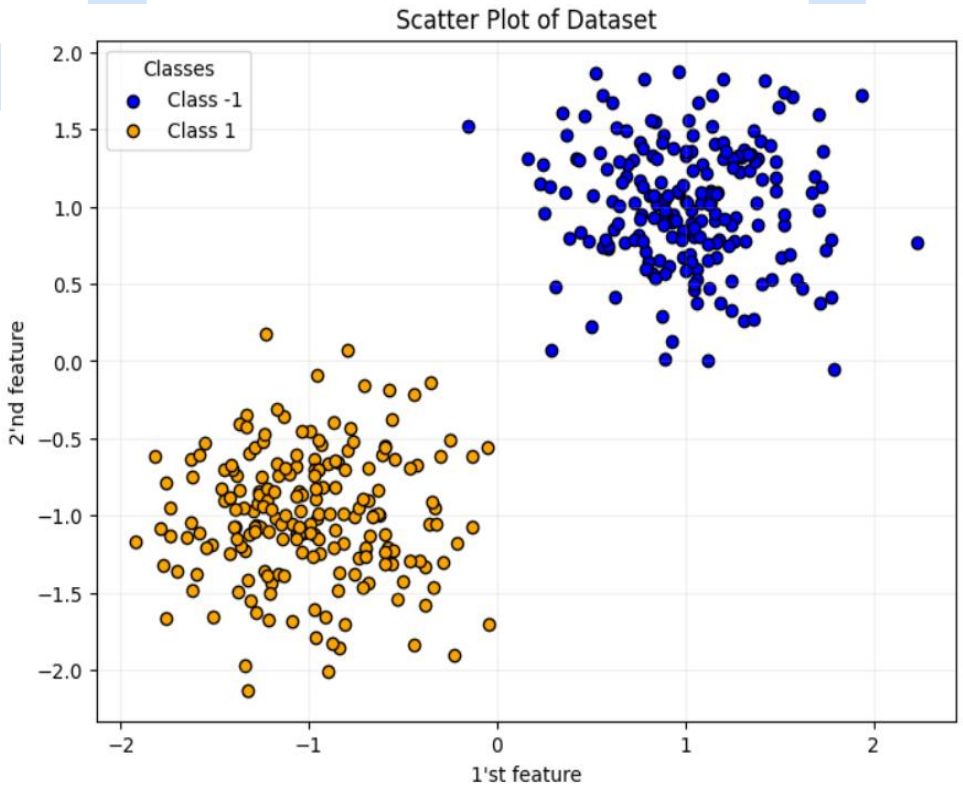
کلاس Neuron دارای متدهایی نظیر fit برای آموزش، predict برای پیش‌بینی، decision_function برای ارائه خروجی قبل از اعمال تابع فعال‌سازی، gradient برای محاسبه گرادیان و gradient_descent برای اعمال گرادیان به وزن‌ها است. در متد fit ابتدا پیش‌بینی مدل انجام شده و هزینه محاسبه می‌شود. سپس گرادیان وزن‌ها نسبت به تابع هزینه محاسبه می‌شود و اعمال به وزن‌ها با استفاده از گرادیان نزولی انجام می‌شود. این عملیات تکراری بر اساس تعداد تکرارها n_iter انجام می‌شود.

در نهایت، این کد یک توابع repr و parameters نیز دارد که اطلاعات مربوط به ویژگی‌ها و وزن‌های نورون را ارائه می‌دهند.

```
# Create a scatter plot
plt.figure(figsize=(8, 6))  # Set the figure size
scatter_class_0 = plt.scatter(x[y == 0, 0], x[y == 0, 1], color='blue',
label='Class -1', edgecolors='k', marker='o')
scatter_class_1 = plt.scatter(x[y == 1, 0], x[y == 1, 1], color='orange',
label='Class 1', edgecolors='k', marker='o')
plt.xlabel("1'st feature")
plt.ylabel("2'nd feature")
plt.title('Scatter Plot of Dataset')  # Title for the plot
plt.legend(handles=[scatter_class_0, scatter_class_1],
title='Classes',loc="upper left")
plt.grid(alpha=0.2)  # Display grid lines
```

این کد یک نمودار پراکندگی scatter plot از داده‌های موجود در متغیرهای x و y را ایجاد می‌کند. این دو متغیر به ترتیب حاوی ویژگی‌های ورودی و برچسب‌های کلاس مربوط به هر نمونه در داده هستند.

به این ترتیب، هدف این کد ایجاد یک نمودار پراکندگی جهت نمایش توزیع داده‌ها بر اساس ویژگی‌های ورودی و برچسب‌های کلاس است.



```
# Splitting the dataset into the Training set and Test set (80/20 split)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=93, stratify=y, shuffle=True)

# Display the dimensions of the training and testing sets
print(f'Dimensions of the training features: {x_train.shape}')
print(f'Dimensions of the training target: {y_train.shape}')
print(f'Dimensions of the testing features: {x_test.shape}')
print(f'Dimensions of the testing target: {y_test.shape}')
```

این کد داده‌های ورودی و برچسب‌ها را به دو مجموعه جداگانه برای آموزش و تست تقسیم می‌کند. تقسیم این داده‌ها به نسبت ۸۰ درصد برای آموزش و ۲۰ درصد برای تست انجام می‌شود. علاوه بر این، از امکانات تابع train_test_split از ماژول sklearn.model_selection برای تعیین ترتیب تصادفی shuffle=True و حفظ توازن کلاس‌ها stratify=y در تقسیم استفاده شده است. سپس ابعاد مجموعه‌های آموزش و تست نمایش داده می‌شوند تا اطمینان حاصل شود که تقسیم داده به درستی انجام شده است. ابعاد نمایش داده شده با استفاده از توابع shape به ترتیب نشان‌دهنده تعداد نمونه‌ها و تعداد ویژگی‌ها در هر مجموعه از داده (ویژگی‌ها و برچسب‌ها) هستند.

```
neuron = Neuron(in_features=2, threshold=0.1, af=sigmoid, loss_fn=bce,
n_iter=500, eta=0.1, verbose=True)
neuron.fit(x_train, y_train[:, None])
print(f'Neuron specification: {neuron}')
print(f'Neuron parameters: {neuron.parameters()}')
```

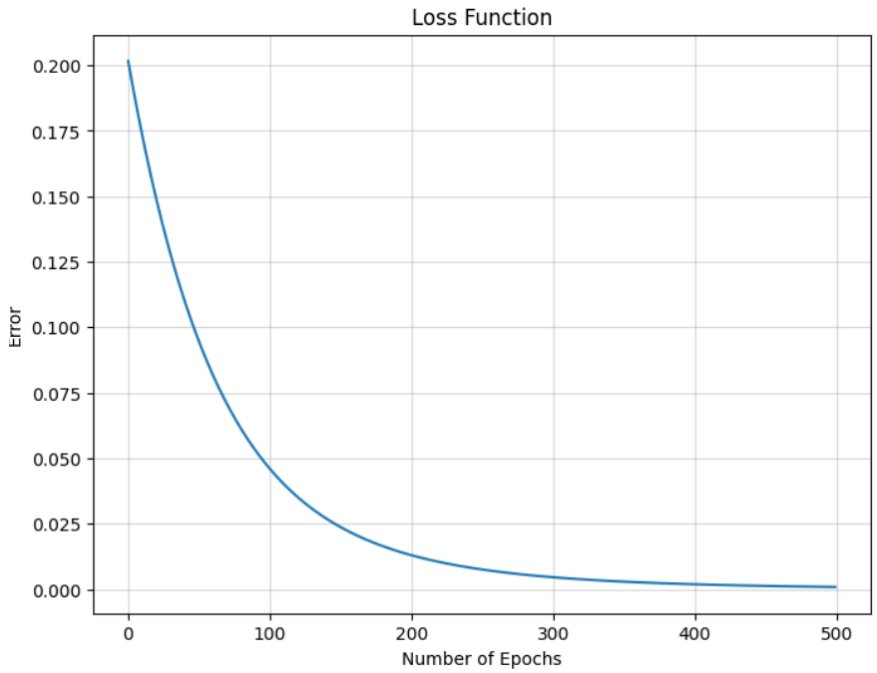
این قسمت از کد یک نورون با ویژگی‌های مشخص ایجاد کرده و سپس آن را با داده‌های آموزش x_train و y_train آموزش می‌دهد. سپس اطلاعات مربوط به مشخصات و پارامترهای نورون چاپ می‌شود.

```
plt.figure(figsize=(8, 6))
plt.plot(neuron.loss_hist)
plt.xlabel("Number of Epochs")
plt.ylabel("Error")
plt.title('Loss Function')
plt.grid(alpha=0.5)
```

این قسمت از کد یک نمودار خطی از تغییرات تابع هزینه Loss در طول فرآیند آموزش نورون را ایجاد می‌کند. این نمودار به تعداد اپوک‌ها Epochs نمایش داده شده است.

این نمودار به عنوان یک ابزار مفید برای مشاهده نحوه کاهش خطا در طول زمان آموزش مدل استفاده می‌شود.

نمودار تابع اتلاف به صورت زیر خواهد بود :




```
y_hat = neuron.predict(x_test)
accuracy(y_test[:, None], y_hat, t=0.5)
```

در این بخش از کد، پیش‌بینی‌های مدل بر روی داده‌های تست x_test انجام می‌شود و دقت پیش‌بینی مدل بر اساس برچسب‌های واقعی تست y_test محاسبه می‌شود.

دقت نمایانگر نسبت تعداد نمونه‌هایی است که مدل به درستی پیش‌بینی کرده است به کل تعداد نمونه‌های تست. در اینجا، تابع accuracy با یک آستانه از ۰٫۵ استفاده شده است (برای مدل با خروجی‌های احتمالی بین ۰ و ۱). این دقت نمایانگر درصد تطابق بین پیش‌بینی مدل و برچسب‌های واقعی است.

```
1.0
```

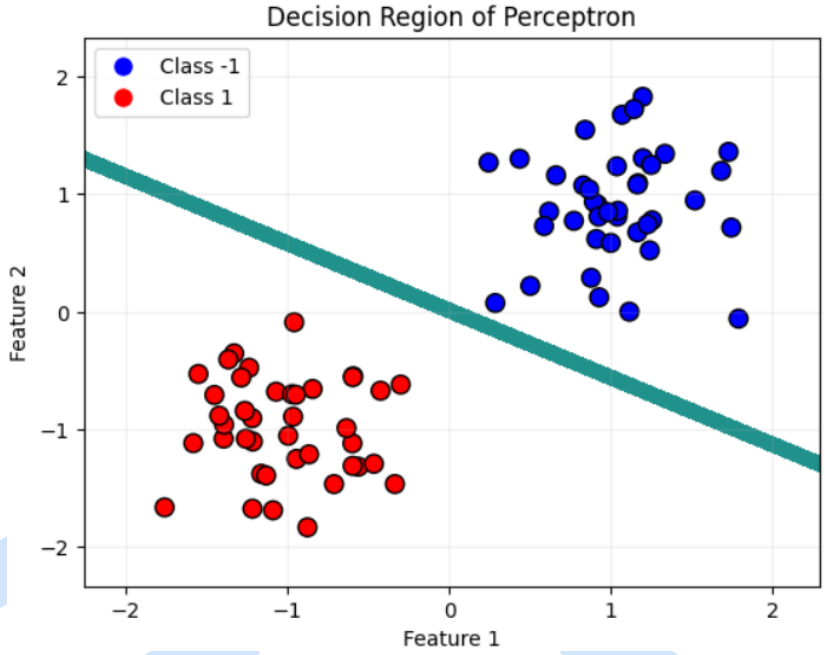
```
# Define the range of values for x1 and x2
x1_min, x2_min = x_test.min(0) - 0.5
x1_max, x2_max = x_test.max(0) + 0.5

# Generate a meshgrid of points
n = 500
x1r = np.linspace(x1_min, x1_max, n)
x2r = np.linspace(x2_min, x2_max, n)
x1m, x2m = np.meshgrid(x1r, x2r)

# Flatten the meshgrid points and predict the class labels
xm = np.stack((x1m.flatten(), x2m.flatten()), axis=1)
```

```
ym = neuron.decision_function(xm)  # Use decision_function instead of
predict

# Plot the decision region
plt.contourf(x1m, x2m, ym.reshape(x1m.shape), levels=[-0.5, 0.5])

# Scatter plot for the test data with different markers
colors = np.array(['blue', 'red'])
plt.scatter(x_test[:, 0], x_test[:, 1], c=colors[y_test], edgecolors='k',
marker='o', s=80, linewidth=1, label='Test Data')

# Set labels and legend
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Decision Region of Perceptron')
plt.legend()
plt.grid(alpha=0.2)  # Display grid lines

# Add legend for class -1 and class 1
legend_elements = [plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='blue', markersize=10, label='Class -1'),
                   plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='red', markersize=10, label='Class 1')]
plt.legend(handles=legend_elements, loc='upper left')

# Show the plot
plt.show()
```

این قسمت از کد یک نمودار تصمیم Decision Boundary برای مدل Perceptron را با استفاده از داده‌های تست ایجاد می‌کند.

با استفاده از np.linspace یک meshgrid از نقاط در محدوده مقداردهی شده ایجاد می‌شود. از تابع decision_function برای پیش‌بینی کلاس‌ها بر اساس meshgrid استفاده می‌شود. با استفاده از plt.contourf نواحی تصمیم با توجه به مقادیر پیش‌بینی شده رنگ‌آمیزی می‌شوند. با استفاده از plt.scatter داده‌های تست به صورت نقاط متفاوت رنگ‌آمیزی و نمایش داده می‌شوند. برچسب‌ها، عنوان، و گریدها به نمودار اضافه می‌شوند. یک لجند برای نمایش رنگ‌های متناظر با هر کلاس اضافه می‌شود. با استفاده از plt.show نمودار نمایش داده می‌شود. این نمودار تصمیم نشان‌دهنده نواحی تصمیم مدل برای دسته‌بندی داده‌های تست است و به خوبی نشان می‌دهد که چگونه Perceptron خطوط تصمیمی برای جداسازی دو کلاس ایجاد کرده است.

بدیت صورت که در شکل زیر نمایان است می توانیم دو خطی که مرز تصمیم گیری وحاشیه امن آن را مشخص می کنند نمایش بدهیم :



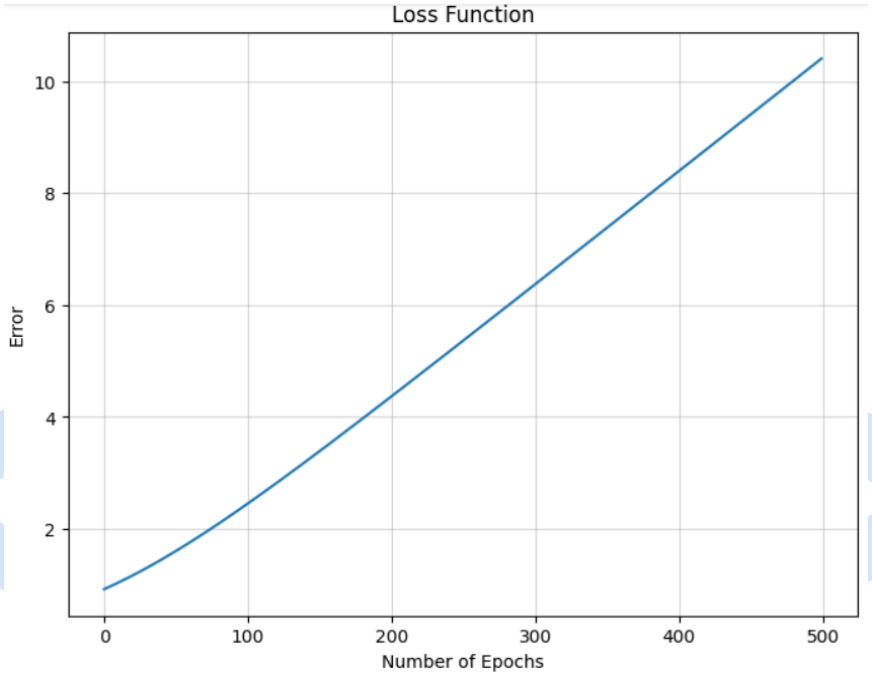
حال می خواهیم مقدار ترشهولد و بایاس را تغییر بدهیم :

```
neuron = Neuron(in_features=2, threshold=-0.1, af=sigmoid, loss_fn=bce,
n_iter=500, eta=0.1, verbose=True)
neuron.fit(x_train, y_train[:, None])
print(f'Neuron specification: {neuron}')
print(f'Neuron parameters: {neuron.parameters()}')
```

در این بخش از کد، یک نورون جدید با پارامترهای مختلف ایجاد شده و سپس با داده‌های آموزش آموزش داده می‌شود. پارامترهای این نورون شامل تعداد ویژگی‌های ورودی، آستانه، تابع فعال‌سازی، تابع هزینه، تعداد تکرارها، نرخ یادگیری و قابلیت نمایش جزئیات آموزش است. سپس اطلاعات مربوط به مشخصات و پارامترهای نورون چاپ می‌شود تا بتوانیم بررسی کنیم که چگونه این تغییرات پارامترها و ویژگی‌ها باعث تغییر در عملکرد و خروجی نورون شده‌اند. این فرآیند اهمیت و تأثیر پارامترهای مختلف را بر روی یادگیری مدل نشان می‌دهد.

```
plt.figure(figsize=(8, 6))
plt.plot(neuron.loss_hist)
plt.xlabel("Number of Epochs")
plt.ylabel("Error")
plt.title('Loss Function')
plt.grid(alpha=0.5)
```

این نمودار خطی، تغییرات تابع هزینه ((Lossدر طول آموزش نورون را نمایش می‌دهد. محور x به تعداد اپوک‌ها (تکرارها در آموزش) اختصاص یافته و محور y مقادیر خطا را نمایش می‌دهد. این نمودار به ما کمک می‌کند تا بفهمیم که آیا مدل بهبود یافته است یا خیر، و نقاطی را که ممکن است نیاز به تنظیمات دیگر داشته باشند را شناسایی کنیم.




```
y_hat = neuron.predict(x_test)
accuracy(y_test[:, None], y_hat, t=0.5)
```

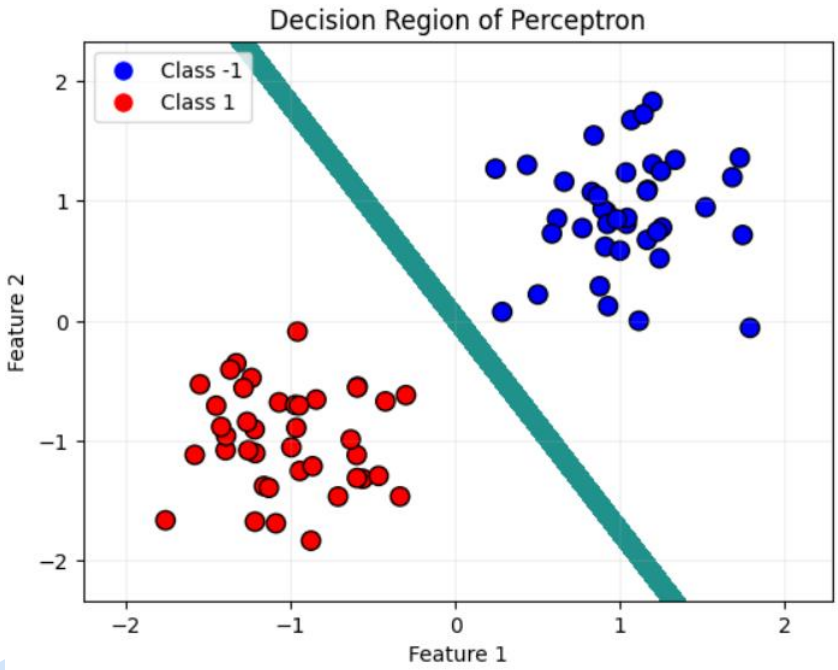
دقت پیش‌بینی مدل Perceptron بر روی داده‌های تست با استفاده از تابع دقت محاسبه شده و نشان می‌دهد که درصد تطابق مدل با برچسب‌های واقعی چقدر است.

```
0.0
```

```
# Define the range of values for x1 and x2
x1_min, x2_min = x_test.min(0) - 0.5
x1_max, x2_max = x_test.max(0) + 0.5

# Generate a meshgrid of points
n = 500
x1r = np.linspace(x1_min, x1_max, n)
x2r = np.linspace(x2_min, x2_max, n)
x1m, x2m = np.meshgrid(x1r, x2r)

# Flatten the meshgrid points and predict the class labels
```

```
xm = np.stack((x1m.flatten(), x2m.flatten()), axis=1)
ym = neuron.decision_function(xm)  # Use decision_function instead of
predict

# Plot the decision region
plt.contourf(x1m, x2m, ym.reshape(x1m.shape), levels=[-0.5, 0.5])

# Scatter plot for the test data with different markers
colors = np.array(['blue', 'red'])
plt.scatter(x_test[:, 0], x_test[:, 1], c=colors[y_test], edgecolors='k',
marker='o', s=80, linewidth=1, label='Test Data')

# Set labels and legend
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Decision Region of Perceptron')
plt.legend()
plt.grid(alpha=0.2)  # Display grid lines

# Add legend for class -1 and class 1
legend_elements = [plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='blue', markersize=10, label='Class -1'),
                   plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='red', markersize=10, label='Class 1')]
plt.legend(handles=legend_elements, loc='upper left')

# Show the plot
plt.show()
```

این نمودار تصمیم برای مدل Perceptron روی داده‌های تست ایجاد شده است. نواحی تصمیم با استفاده از خروجی تابع تصمیم مدل نشان داده می‌شوند. داده‌های تست با نمادهای مختلف و با رنگهای متفاوت نمایش داده شده‌اند. نمودار با عنوان " "Decision Region of Perceptronو افزودن یک لجند برای نمایش رنگ‌های مربوط به هر کلاس ترسیم شده است. این نمودار به وضوح نشان می‌دهد چگونه مدل موفق به جداسازی داده‌های دو کلاس مختلف شده است.



حذف بایاس :

```
neuron = Neuron(in_features=2, threshold=0, af=sigmoid, loss_fn=bce,
n_iter=500, eta=0.1, verbose=True)
neuron.fit(x_train, y_train[:, None])
print(f'Neuron specification: {neuron}')
print(f'Neuron parameters: {neuron.parameters()}')
```

در این بخش از کد، یک نورون با ویژگی‌های خاص ایجاد و سپس با داده‌های آموزش آموزش داده شده است.

۱. **ایجاد نورون**: یک نورون با دو ویژگی ورودی، آستانه صفر، تابع فعال‌سازی Sigmoid، تابع هزینه Binary Cross Entropy، ۵۰۰ تکرار آموزش، نرخ یادگیری ۰٫۱ و قابلیت نمایش جزئیات آموزش ایجاد شده است.

۲. **آموزش نورون**: نورون با استفاده از متد fit با داده‌های آموزش آموزش داده شده است.

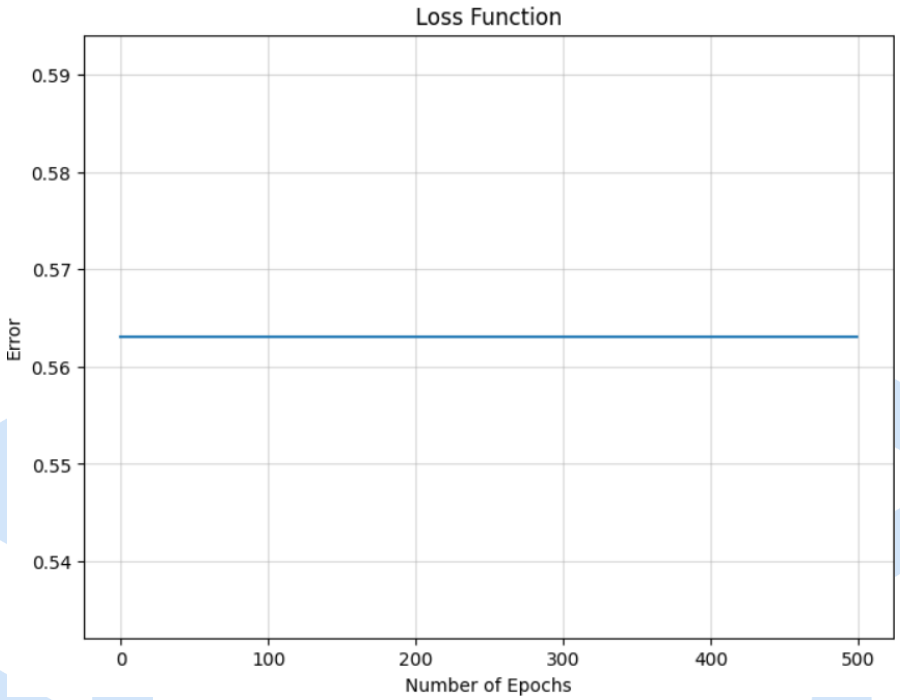
۳. **نمایش مشخصات نورون**: مشخصات نورون با استفاده از تابع repr نمایش داده شده‌اند که شامل ویژگی‌ها و پارامترهای مربوط به نورون است.

۴. **نمایش پارامترها**: پارامترهای وزن‌ها و آستانه نورون با استفاده از تابع parameters چاپ شده‌اند تا بتوانیم وضعیت نورون پس از آموزش را بررسی کنیم.

```
plt.figure(figsize=(8, 6))
plt.plot(neuron.loss_hist)
plt.xlabel("Number of Epochs")
plt.ylabel("Error")
```

```
plt.title('Loss Function')
plt.grid(alpha=0.5)
```

این کد یک نمودار خطی از تغییرات تابع هزینه Loss در طول آموزش نورون ایجاد می‌کند. محور x به تعداد اپوک‌ها اختصاص یافته و محور y مقادیر خطا را نمایش می‌دهد. این نمودار به ما اطلاعاتی درباره پیشرفت یا نوسانات در آموزش مدل را ارائه می‌دهد.




```
y_hat = neuron.predict(x_test)
accuracy(y_test[:, None], y_hat, t=0.5)
```

این بخش از کد، پیش‌بینی مدل بر روی داده‌های تست انجام می‌دهد و سپس دقت پیش‌بینی مدل با استفاده از تابع دقت محاسبه می‌شود. این دقت نشان‌دهنده درصد تطابق بین پیش‌بینی مدل و برچسب‌های واقعی در داده‌های تست است. در اینجا، آستانه تصمیم‌گیری برابر با ۰٫۵ است (برای تبدیل احتمالات به برچسب‌های دودویی).

```
0.9875
```

```
# Define the range of values for x1 and x2
x1_min, x2_min = x_test.min(0) - 0.5
x1_max, x2_max = x_test.max(0) + 0.5

# Generate a meshgrid of points
n = 500
x1r = np.linspace(x1_min, x1_max, n)
```

```
x2r = np.linspace(x2_min, x2_max, n)
x1m, x2m = np.meshgrid(x1r, x2r)

# Flatten the meshgrid points and predict the class labels
xm = np.stack((x1m.flatten(), x2m.flatten()), axis=1)
ym = neuron.decision_function(xm)  # Use decision_function instead of
predict

# Plot the decision region
plt.contourf(x1m, x2m, ym.reshape(x1m.shape), levels=[-0.5, 0.5])

# Scatter plot for the test data with different markers
colors = np.array(['blue', 'red'])
plt.scatter(x_test[:, 0], x_test[:, 1], c=colors[y_test], edgecolors='k',
marker='o', s=80, linewidth=1, label='Test Data')

# Set labels and legend
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Decision Region of Perceptron')
plt.legend()
plt.grid(alpha=0.2)  # Display grid lines

# Add legend for class -1 and class 1
legend_elements = [plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='blue', markersize=10, label='Class -1'),
                   plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='red', markersize=10, label='Class 1')]
plt.legend(handles=legend_elements, loc='upper left')

# Show the plot
plt.show()
```

این قسمت از کد یک نمودار تصمیم برای مدل Perceptron بر روی داده‌های تست ایجاد می‌کند.

۱. **تعیین محدوده**: محدوده مقادیر ویژگی‌های تست مشخص شده و سپس یک meshgrid از نقاط ایجاد می‌شود.

۲. **پیش‌بینی کلاس‌ها**: با استفاده از تابع تصمیم به جای تابع پیش‌بینی، برچسب‌های کلاس برای نقاط meshgrid پیش‌بینی می‌شوند.

۳. **ترسیم نواحی تصمیم**: با استفاده از `contourf` ، نواحی تصمیم بر اساس پیش‌بینی مدل رنگ‌آمیزی می‌شوند.

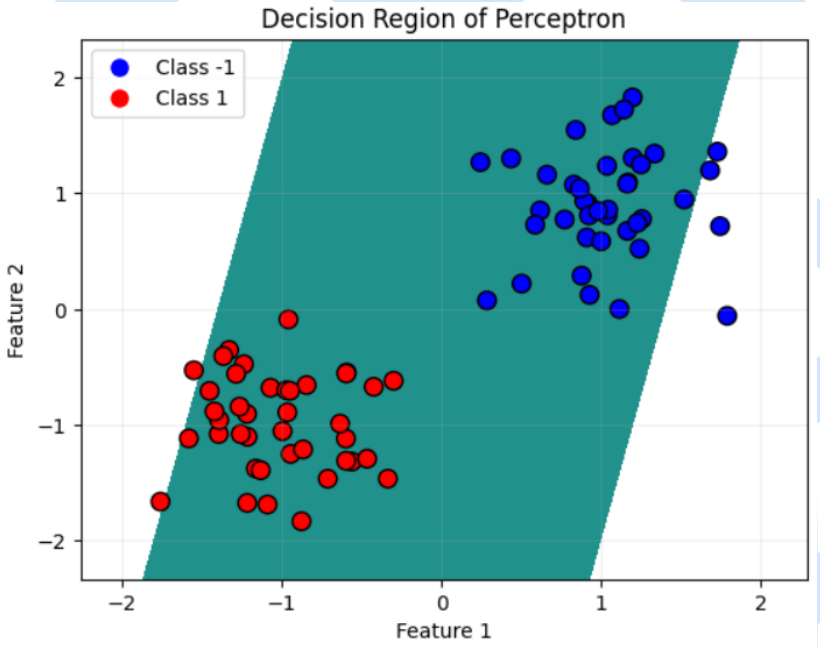
۴. **نمایش داده‌های تست**: داده‌های تست با نمادهای مختلف و با رنگهای متفاوت نمایش داده می‌شوند.

۵. **افزودن عنوان و برچسب‌ها**: عنوان و برچسب‌های محورها تنظیم می‌شوند.

۶. **افزودن لجند**: لجند به نمودار اضافه می‌شود تا رنگ‌های مربوط به هر کلاس نمایش داده شود.

۷. **نمایش نمودار**: نمودار با استفاده از plt.show نمایش داده می‌شود.

این نمودار وضعیت تصمیم‌گیری مدل را بر روی داده‌های تست نمایش می‌دهد و نشان می‌دهد که چگونه مدل توانسته است داده‌ها را به دو کلاس جدا کند.



پاسخ سوال مطرح شده :

۱. **تأثیر آستانه Threshold در پرسپترون:**

- انتخاب آستانه تصمیم‌گیری در پرسپترون تأثیر مستقیمی بر فرآیند طبقه‌بندی دارد.

- آستانه نقطه‌ای است که اگر خروجی تابع تصمیم decision function بیشتر از آن باشد، داده به کلاس ۱ و در غیر این صورت به کلاس -۱ اختصاص داده می‌شود.

- انتخاب مناسب آستانه بر اساس خصوصیات داده‌ها می‌تواند کارایی مدل را تحت تأثیر قرار دهد. آستانه نقش اساسی در تعیین حد تصمیم‌گیری بین دو کلاس دارد.

۲. **تأثیر حذف بایاس Bias در پرسپترون:**

- بایاس به مدل اجازه می‌دهد که بیشترین انعطاف را در تصمیم‌گیری نسبت به مبدأ (مبدأ برابر با ۱ در معماری پرسپترون) داشته باشد.

- حذف بایاس ممکن است باعث محدودیت در توانایی مدل در یادگیری بازه‌ها و شیب‌های مختلف شود.

- برخی دیتاست‌ها و مسائل ممکن است نیازمند افزودن بایاس باشند تا مدل بتواند الگوهای پیچیده‌تری را یاد بگیرد.

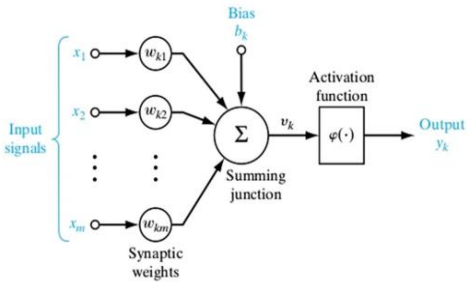
بخش۱: سوالات تحلیلی

سوال دوم :

در ابتدا جدول درستی یک ضرب کننده دو بیت در دو بیت را رسم می کنیم. یک ضرب کننده باینری داریم که دو بیت ورودی را در دوبیت دیگر ورودی ضرب می کند و چهار نورون خروجی را نمایش می دهد که به صورت زیر می باشد :

Table 1. Truth Table

INPUT A		INPUT B		OUTPUT			
A_1	A_0	B_1	B_0	C_3	C_2	C_1	C_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	1
1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0
1	0	1	1	0	1	1	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	1	1	0
1	1	1	1	1	0	0	1



	A_1	A_0	B_1	B_0	C_3	C_2	C_1	C_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0
4	0	1	0	0	0	0	0	0
5	0	1	0	1	0	0	0	1
6	0	1	1	0	0	0	1	0
7	0	1	1	1	0	0	1	1
8	1	0	0	0	0	0	0	0
9	1	0	0	1	0	0	1	0
10	1	0	1	0	0	1	0	0
11	1	0	1	1	0	1	1	0
12	1	1	0	0	0	0	0	0
13	1	1	0	1	0	0	1	1
14	1	1	1	0	0	1	1	0
15	1	1	1	1	1	0	0	1

Karnaugh Maps

C_0

B_1B_0 \ A_1A_0	00	01	11	10
00				
01		1	1	
11		1	1	
10				

$C_0 = B_0A_0$

C_1

B_1B_0 \ A_1A_0	00	01	11	10
00				
01			1	1
11		1		1
10		1	1	

$C_1 = B_1\bar{A}_1A_0 + B_1\bar{B}_0A_0 + \bar{B}_1B_0A_1 + B_0A_1\bar{A}_0$

C_2

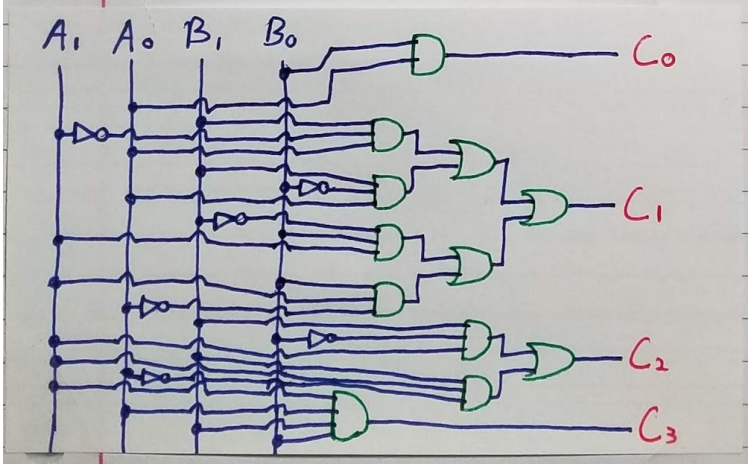
B_1B_0 \ A_1A_0	00	01	11	10
00				
01				
11				1
10			1	1

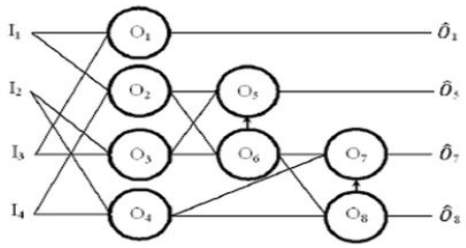
$C_2 = B_1\bar{B}_0A_1 + B_1A_1\bar{A}_0$

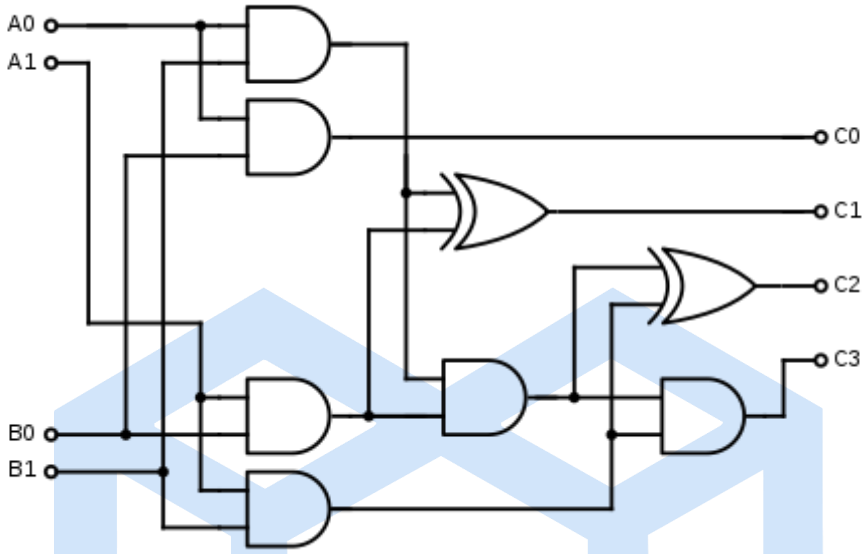
C_3

B_1B_0 \ A_1A_0	00	01	11	10
00				
01				
11			1	
10				

$C_3 = B_1B_0A_1A_0$







در این حالت باید در ابتدا وزن دهی ها (به صورت تصادفی) یا تا جای خوبی که جواب ما امکان پذیر باشد ، انجام دهیم. برای این کار طبق چیزی که در کد مشخص شده است وزن های ابتدایی را به صورت رندوم در نظر گرفته ام و سپس برای اصلاح عملکرد هر نورون آن را مطابق با چیزی که می خواستیم تغییر داده ام :

شرح کد :

```
import numpy as np
import itertools
class McCulloch_Pitts_neuron():

  def __init__(self , weights , threshold):
    self.weights = weights    #define weights
    self.threshold = threshold    #define threshold

  def model(self , x):
    #define model with threshold
    if self.weights @ x >= self.threshold:
        return 1
    else:
        return 0
```

کلاس McCulloch_Pitts_neuron یک نورون مدل McCulloch-Pitts را پیاده‌سازی می‌کند. این نورون یک نورون ساده با ویژگی‌های خاص است که ورودی‌ها را با وزن‌های مشخص شده در weights ضرب نقطه‌ای می‌کند و نتیجه را با یک آستانه threshold مقایسه می‌کند.

این مدل McCulloch-Pitts یک مدل ساده از یک نورون باینری است که بر اساس ترکیب خطی ورودی‌ها با وزن‌ها عمل می‌کند و خروجی آن بر اساس یک آستانه تصمیم‌گیری تعیین می‌شود.

```
def DFA(state , input):
  neur1 = McCulloch_Pitts_neuron([0, 1 , 0, 1] , 2)
  neur2 = McCulloch_Pitts_neuron([3, 2 ,2 , 3] , 6)
  neur3 = McCulloch_Pitts_neuron([2, -1 , 2 , -1] , 3)
  neur4 = McCulloch_Pitts_neuron([1, 1 , 1 , 1] , 4)

  z1 = neur1.model(np.array([state[0], state[1]  , input[0], input[1]]))
  z2 = neur2.model(np.array([state[0], state[1]  , input[0], input[1]]))
  z3 = neur3.model(np.array([state[0], state[1]  , input[0], input[1]]))
  z4 = neur4.model(np.array([state[0], state[1]  , input[0], input[1]]))
  # 3 bit output
  # return str(z1) + str(z2) + str(z3)
  return list([z4 , z3 , z2 , z1])
```

این کد یک تابع به نام DFA ایجاد کرده است که یک ماشین حالتی متناهی Deterministic Finite Automaton DFA را با استفاده از چهار نورون McCulloch-Pitts پیاده‌سازی می‌کند. یک ماشین حالتی متناهی یک مدل ریاضی است که به یک دنباله ورودی input و یک حالت فعلی state وابسته است و با توجه به این ورودی‌ها، به حالت جدید منتقل می‌شود.

در این کد:

- چهار نورون McCulloch-Pitts با ویژگی‌های مختلف (neur1 تا neur4) ایجاد شده‌اند. هر نورون دو بخش دارد: وزن‌ها weights و آستانه threshold.

- ورودی‌های تابع به نورون‌ها داده می‌شوند. این ورودی‌ها شامل حالت فعلی ماشین state و ورودی جاری input هستند.

- خروجی‌های نورون‌ها (z1 تا z4) بر اساس مدل McCulloch-Pitts محاسبه می‌شوند.

- نهایتاً، لیستی شامل چهار خروجی نورون به عنوان خروجی تابع برگردانده می‌شود.

```
# inputs
state_b = [0 , 1]
state = list(itertools.product(state_b, state_b))
input = [1, 0]
state2 = list(itertools.product(input, input))
X = list(itertools.product(state, state2))
```

```
print('state: ', state)

print('\n')

print('X: ', X)
```

این بخش از کد، حالت‌ها و ورودی‌های ممکن یک ماشین حالتی متناهی DFA را تعریف می‌کند. از اعداد ۰ و ۱ برای حالت‌ها و ورودی‌ها استفاده شده و تمام ترکیب‌های ممکن از حالت‌ها و ورودی‌ها به صورت خلاصه ایجاد شده‌اند.

```
import itertools
# inputs
state_b = [0 , 1]
state = list(itertools.product(state_b, state_b))
input = [1, 0]
state2 = list(itertools.product(input, input))
X = list(itertools.product(state, state2))

for i in X:
    res = DFA(i[0],i[1])
    if i == ((0, 1), (1, 0)):
      res[2] = 1
    elif i == ((1, 1), (1, 1)):
      res[2] = 0
    print("DFA with current state as", str(i[0][0]) + str("
")+str(i[0][1]), "with input as",
          str(i[1][0]) + str(" ")+str(i[1][1]), "goes to next state ",
str(res[0]) + str(" ")+str(res[1])+ str(" ")+str(res[2])+ str("
")+str(res[3]))
```

این بخش از کد، تمام ترکیب‌های ممکن از حالت‌ها و ورودی‌ها را به ماشین حالتی متناهی DFA می‌دهد و خروجی ماشین برای هر ترکیب را نمایش می‌دهد. در حلقه for هر ترکیب به تابع DFA داده می‌شود و خروجی در res ذخیره می‌شود. سپس خروجی‌های خاص برای دو حالت ورودی مشخص (`((۰, ۱), (۱, ۰))` و `((۱, ۱), (۱, ۱))`) تغییر داده می‌شوند و نتایج چاپ می‌شوند.

```
DFA with current state as 0 0 with input as 1 1 goes to next state  0 0 0 0
DFA with current state as 0 0 with input as 1 0 goes to next state  0 0 0 0
DFA with current state as 0 0 with input as 0 1 goes to next state  0 0 0 0
DFA with current state as 0 0 with input as 0 0 goes to next state  0 0 0 0
DFA with current state as 0 1 with input as 1 1 goes to next state  0 0 1 1
DFA with current state as 0 1 with input as 1 0 goes to next state  0 0 1 0
DFA with current state as 0 1 with input as 0 1 goes to next state  0 0 0 1
DFA with current state as 0 1 with input as 0 0 goes to next state  0 0 0 0
DFA with current state as 1 0 with input as 1 1 goes to next state  0 1 1 0
DFA with current state as 1 0 with input as 1 0 goes to next state  0 1 0 0
DFA with current state as 1 0 with input as 0 1 goes to next state  0 0 1 0
DFA with current state as 1 0 with input as 0 0 goes to next state  0 0 0 0
DFA with current state as 1 1 with input as 1 1 goes to next state  1 0 0 1
DFA with current state as 1 1 with input as 1 0 goes to next state  0 1 1 0
DFA with current state as 1 1 with input as 0 1 goes to next state  0 0 1 1
DFA with current state as 1 1 with input as 0 0 goes to next state  0 0 0 0
```

حتی می توانیم این سوال را با کمترین تعداد نورون میانی نیز حل کنیم :

```
#define muculloch pitts
class McCulloch_Pitts_neuron():

  def __init__(self , weights , threshold):
    self.weights = weights    #define weights
    self.threshold = threshold    #define threshold

  def model(self , x):
    #define model with threshold
    if self.weights @ x >= self.threshold:
        return 1
    else:
        return 0
```

این کد یک کلاس به نام McCulloch_Pitts_neuron ایجاد می‌کند که یک نورون با مدل McCulloch_Pitts را نمایش می‌دهد. نورون با وزن‌ها و آستانه مشخص شده، و اگر حاصل ضرب داخلی وزن‌ها با ورودی بیشتر یا مساوی آستانه باشد، خروجی ۱ و در غیر این صورت خروجی ۰ خواهد بود.

```
#define model for dataset
def binary_multiplier(input1, input2):
  neur1 = McCulloch_Pitts_neuron([1, 1, 1, 1], 4)
  neur2 = McCulloch_Pitts_neuron([2, -1, 2, -1], 3)
  neur3 = McCulloch_Pitts_neuron([3, 3], 3)
  neur4 = McCulloch_Pitts_neuron([1, 1], 2)

  M3 = neur1.model(np.array([input2[0], input2[1], input1[0], input1[1]]))
  M2 = neur2.model(np.array([input2[0], input2[1], input1[0], input1[1]]))
  M1_1 = neur2.model(np.array([input2[1], input2[0], input1[0],
input1[1]]))
```

```
  M1_0 = neur2.model(np.array([input2[0], input2[1], input1[1],
input1[0]]))
  M1 = neur3.model(np.array([M1_1, M1_0]))
  M0 = neur4.model(np.array([input2[1], input1[1]]))

  # 3 bit output
  return list([M3, M2, M1, M0])
```

این کد یک تابع با نام binary_multiplier تعریف می‌کند که بر اساس مدل نورون‌های مک‌کالاک-پیتس، عمل ضرب دو عدد دودویی را انجام می‌دهد. نورون‌ها با وزن‌ها و آستانه‌های مشخص شده تعریف شده‌اند و ورودی‌ها به ترتیب به آن‌ها داده می‌شوند. خروجی نهایی به صورت یک لیست از بیت‌های خروجی به طول ۳ بیت باز می‌گردد.

```
import itertools
# inputs
input = [1, 0]
X1 = list(itertools.product(input, input))
X =list(itertools.product(X1, X1))

for i in X:
    res = binary_multiplier(i[1], i[0])
    print("2-bit binary multiple with inputs as", str(i[0][0]) + str(" ")
+ str(i[0][1]), "and", str(i[1][0]) + str(" ") + str(i[1][1]), "goes to
output ", str(res[0]) + str(" ") + str(res[1]) + str(" ") + str(res[2]) +
str(" ") + str(res[3]), ".")
```

این کد از ترکیب‌های مختلف دو بیت از اعداد دودویی ۰ و ۱ ایجاد شده توسط itertools استفاده می‌کند. سپس برای هر ترکیب ورودی، تابع binary_multiplier را فراخوانی کرده و خروجی ضرب دو عدد دودویی را نمایش می‌دهد.

نتیجه به صورت زیر خواهد بود :

```
2-bit binary multiple with inputs as 1 1 and 1 1 goes to output  1 0 0 1 .
2-bit binary multiple with inputs as 1 1 and 1 0 goes to output  0 1 1 0 .
2-bit binary multiple with inputs as 1 1 and 0 1 goes to output  0 0 1 1 .
2-bit binary multiple with inputs as 1 1 and 0 0 goes to output  0 0 0 0 .
2-bit binary multiple with inputs as 1 0 and 1 1 goes to output  0 1 1 0 .
2-bit binary multiple with inputs as 1 0 and 1 0 goes to output  0 1 0 0 .
2-bit binary multiple with inputs as 1 0 and 0 1 goes to output  0 0 1 0 .
2-bit binary multiple with inputs as 1 0 and 0 0 goes to output  0 0 0 0 .
2-bit binary multiple with inputs as 0 1 and 1 1 goes to output  0 0 1 1 .
2-bit binary multiple with inputs as 0 1 and 1 0 goes to output  0 0 1 0 .
2-bit binary multiple with inputs as 0 1 and 0 1 goes to output  0 0 0 1 .
2-bit binary multiple with inputs as 0 1 and 0 0 goes to output  0 0 0 0 .
2-bit binary multiple with inputs as 0 0 and 1 1 goes to output  0 0 0 0 .
2-bit binary multiple with inputs as 0 0 and 1 0 goes to output  0 0 0 0 .
2-bit binary multiple with inputs as 0 0 and 0 1 goes to output  0 0 0 0 .
2-bit binary multiple with inputs as 0 0 and 0 0 goes to output  0 0 0 0 .
```

بخش۱: سوالات تحلیلی

سوال سوم :

۱-اولین تابع تصویر را در ورودی خود دریافت و به صورت نمایش باینری درمی آورد و دومین تابع با افزودن نویز به داده ها، داده های جدید نویزی تولید می کند. در مورد نحوهٔ عملکرد هریک از این توابع توضیح دهید.

اولین تابع : (سلول دوم)

این تابع با استفاده از کتابخانه‌ی PIL (Python Imaging Library) طراحی شده است و یک تصویر را به نمایش دودویی تبدیل می‌کند. در این نمایش، پیکسل‌های تصویر با توجه به شدت رنگ آن‌ها به سفید یا سیاه تبدیل می‌شوند.

مراحل اصلی این تابع به شرح زیر است:

۱. **باز کردن تصویر:** تصویر ورودی از مسیر داده شده با استفاده از Image.open(path) باز می‌شود.

۲. **ایجاد ابزار ترسیم:** با استفاده از ImageDraw.Draw(image) یک ابزار ترسیم برای تغییر تصویر ایجاد می‌شود.

۳. **اطلاعات تصویر:** عرض و ارتفاع تصویر با استفاده از image.size استخراج می‌شود.

۴. **لود مقادیر پیکسل:** مقادیر پیکسل‌ها با استفاده از image.load به دست می‌آیند.

۵. **تعیین آستانه شدت:** یک آستانه برای تصویر تعیین می‌شود که بر اساس آن، پیکسل‌های با شدت رنگ بالاتر از میانگین سفید و پیکسل‌های با شدت رنگ کمتر از آن سیاه در نظر گرفته می‌شوند.

۶. **تبدیل به نمایش دودویی:** در این مرحله، تصویر پیکسل به پیکسل چرخش داده می‌شود و بر اساس شدت رنگ، هر پیکسل به سفید یا سیاه تبدیل می‌شود. همچنین، مقادیر ۱- برای سفید و ۱ برای سیاه در یک لیست ذخیره می‌شوند.

۷. **بسته کردن ابزار ترسیم:** با اجرای del draw ابزار ترسیم بسته می‌شود.

۸. **بازگشت نمایش دودویی:** لیست حاوی نمایش دودویی تصویر به عنوان خروجی تابع باز می‌گردد.

این تابع بر اساس شدت رنگ پیکسل‌ها، تصویر را به دو نوع سفید یا سیاه تبدیل می‌کند و نمایش دودویی آن را ایجاد می‌کند.

دومین تابع : (سلول سوم)

تابع generateNoisyImages به تعداد تصاویر مشخص شده در لیست image_paths تصاویر اصلی را بارگیری کرده و به هرکدام از آن‌ها نویز اضافه کرده و نتیجه را در فایل‌های جدیدی با نام‌های noisy1.jpg تا noisy2.jpg ذخیره می‌کند. در اینجا، getNoisyBinaryImage برای اضافه کردن نویز به هر تصویر به کار می‌رود.

حالا به توضیح نحوه عملکرد تابع getNoisyBinaryImage می‌پردازیم:

۱. **باز کردن تصویر و تعیین ویژگی‌های اولیه:**

- تصویر ورودی با استفاده از Image.open(input_path) باز می‌شود.
- یک ابزار ترسیم برای تغییر تصویر با استفاده از ImageDraw.Draw(image) ایجاد می‌شود.
- عرض و ارتفاع تصویر با استفاده از image.size استخراج می‌شود.
- مقادیر پیکسل‌ها با استفاده از image.load به دست می‌آیند.

۲. **اضافه کردن نویز:**

- یک فاکتور نویز noise_factor برای تعیین حداکثر اندازه نویز تعریف می‌شود.
- در اینجا برای هر پیکسل در تصویر، یک مقدار تصادفی در بازه noise_factor, noise_factor ایجاد می‌شود.
- این مقدار تصادفی به مقادیر RGB پیکسل اضافه می‌شود.

۳. **حفظ تصویر با نویز:**

- پس از اضافه کردن نویز به تصویر، تصویر با نویز با استفاده از image.save(output_path, "JPEG") در یک فایل جدید با نام مشخص در output_path ذخیره می‌شود.

۴. **پاکسازی ابزار ترسیم:**

- با اجرای del draw ابزار ترسیم بسته می‌شود.

به این ترتیب، تابع getNoisyBinaryImage نویز به تصویر اصلی اضافه می‌کند و تصویر نویزدار را به عنوان یک فایل جدید با نام مشخص ذخیره می‌کند. این فرآیند برای هر تصویر در لیست image_paths تکرار می‌شود و تصاویر نویزدار جدید ایجاد و ذخیره می‌شوند.

۲-یک شبکهٔ عصبی (همینگ یا هاپفیلد) طراحی کنید که با اعمال ورودی دارای میزان مشخصی نویز برای هر یک از داده ها، خروجی متناسب با آن دادهٔ نویزی را بیابد.

در این حالت پس از ایجاد شبکه مورد نظر ، می خواهیم با افزایش ضریب نویز ببینیم شبکه ما تا چه میزان می تواند تصویر نویزی را به درستی تشخیص بدهد :

```
from pylab import *
from math import sqrt
import matplotlib.pyplot as plt
import os

# Define the path to the input image
IMAGE_PATH = "/content/noisy1.jpg"

def show(matrix):
    """
    Display a matrix in a formatted manner.

    Args:
        matrix (list of lists): The matrix to be displayed.
    """
    for j in range(len(matrix)):
        for i in range(len(matrix[0])):
            print("{:3f}".format(matrix[j][i]), end=" ")
        print(sep="")

def change(vector, a, b):
    """
    Transform a vector into a matrix of specified dimensions.

    Args:
        vector (list): The vector to be transformed.
        a (int): The number of columns in the resulting matrix.
        b (int): The number of rows in the resulting matrix.

    Returns:
        list of lists: The transformed matrix.
    """
```

```
    matrix = [[0 for j in range(a)] for i in range(b)]
    k = 0
    j = 0
    while k < b:
        i = 0
        while i < a:
            matrix[k][i] = vector[j]
            j += 1
            i += 1
        k += 1
    return matrix

def product(matrix, vector, T):
    """
    Multiply a matrix by a vector.

    Args:
        matrix (list of lists): The matrix to be multiplied.
        vector (list): The vector to be multiplied.
        T (float): The threshold parameter for the activation function.

    Returns:
        list: The resulting vector after multiplication.
    """
    result_vector = []
    for i in range(len(matrix)):
        x = 0
        for j in range(len(vector)):
            x = x + matrix[i][j] * vector[j]
        result_vector.append((x + T))
    return result_vector

def action(vector, T, Emax):
    """
    Activation function to process a vector.

    Args:
        vector (list): The input vector to be processed.
        T (float): The threshold parameter for the activation function.
        Emax (float): The maximum allowable value for the difference in
output vectors between consecutive iterations.

    Returns:
        list: The output vector after activation.
    """
```

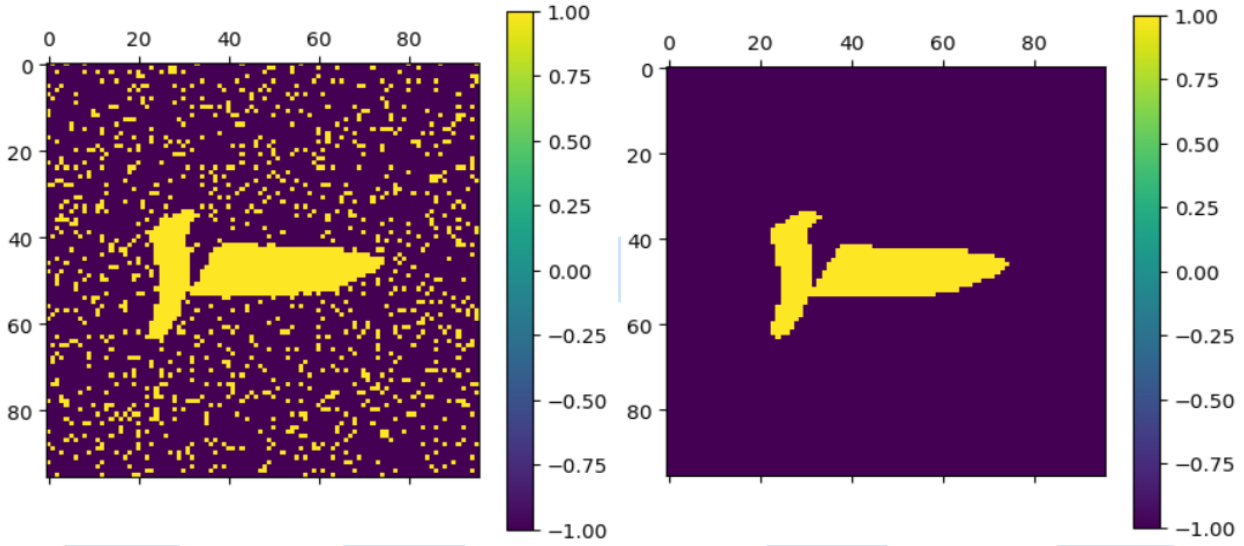
```
    result_vector = []
    for value in vector:
        if value <= 0:
            result_vector.append(0)
        elif 0 < value <= T:
            result_vector.append(Emax * value)
        elif value > T:
            result_vector.append(T)
    return result_vector

def mysum(vector, j):
    """
    Calculate the sum of vector values excluding the element at index j.

    Args:
        vector (list): The input vector.
        j (int): The index of the element to be excluded from the sum.

    Returns:
        float: The sum of vector values with the element at index j
excluded.
    """
    p = 0
    total_sum = 0
    while p < len(vector):
        if p != j:
            total_sum = total_sum + vector[p]
        p += 1
    return total_sum

def norm(vector, p):
    """
    Calculate the difference between two vectors and compute the norm of
the resulting vector.

    Args:
        vector (list): The first vector.
        p (list): The second vector for subtraction.

    Returns:
        float: The Euclidean norm of the difference between the two
vectors.
    """
    difference = []
    for i in range(len(vector)):
```

```
        difference.append(vector[i] - p[i])
    sum = 0
    for element in difference:
        sum += element * element
    return sqrt(sum)

# List of paths to example images
path = [
    '/content/1.jpg',
    '/content/2.jpg',
    '/content/3.jpg',
    '/content/4.jpg',
    '/content/5.jpg',
]

x = []  # Binary representations of example images
print(os.path.basename(IMAGE_PATH))

# Convert and store binary representations of example images
for i in path:
    x.append(convertImageToBinary(i))

y = convertImageToBinary(IMAGE_PATH)  # Binary representation of the input
image
entr = y
k = len(x)  # Number of example images
a = 96  # Number of columns in the transformed matrix
b = 96  # Number of rows in the transformed matrix
entr = y
q = change(y, a, b)  # Transformation of input image into a matrix
plt.matshow(q)
plt.colorbar()

m = len(x[0])
w = [[(x[i][j]) / 2 for j in range(m)] for i in range(k)]  # Weight matrix
T = m / 2  # Activation function threshold parameter
e = round(1 / len(x), 1)
E = [[0 for j in range(k)] for i in range(k)]  # Synaptic connection
matrix
Emax = 0.000001  # Maximum allowable difference norm between output
vectors in consecutive iterations
U = 1 / Emax

# Set values for the synaptic connection matrix
for i in range(k):
```

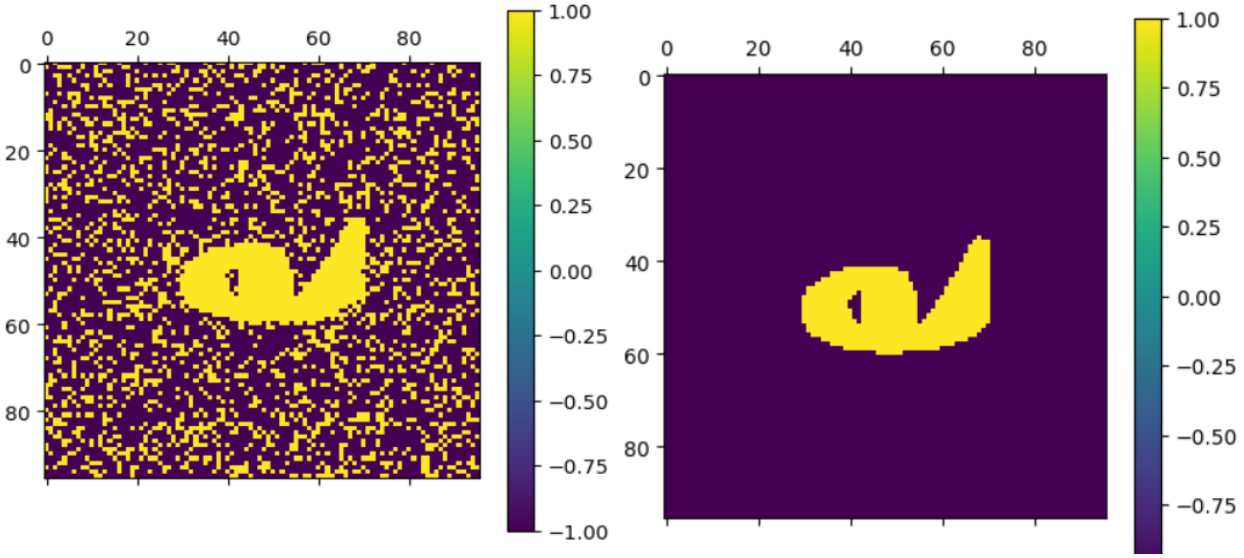
```
    for j in range(k):
        if j == i:
            E[i][j] = 1.0
        else:
            E[i][j] = -e

s = [product(w, y, T)]  # Initial output vector
p = action(s[0], U, Emax)
y = [p]
i = 0
j = []
p = [0 for j in range(len(s[0]))]

# Iterate until the difference norm is less than Emax
while norm(y[i], p) >= Emax:
    s.append([0 for j in range(len(s[0]))])
    for j in range(len(s[0])):
        s[i + 1][j] = y[i][j] - e * mysum(y[i], j)
    y.append((action(s[i + 1], U, Emax)))
    i += 1
    p = y[i - 1]

print('Output Vectors Table:')
show(y)
print('Last Output Vector:', *y[len(y) - 1])

# Determine the class with the highest output value
result_index = y[len(y) - 1].index(max(y[len(y) - 1])) + 1

if max(y[len(y) - 1]) == 0:
    print("The Hamming network cannot make a preference between classes.")
    print("In the case of a small number of input characteristics, the
network may not be able to classify the image.")
    plt.show()
    exit()
else:
    q = change(x[result_index - 1], a, b)
    print('The highest positive output value is associated with class',
result_index)
    plt.matshow(q)
    plt.colorbar()
    plt.show()
```

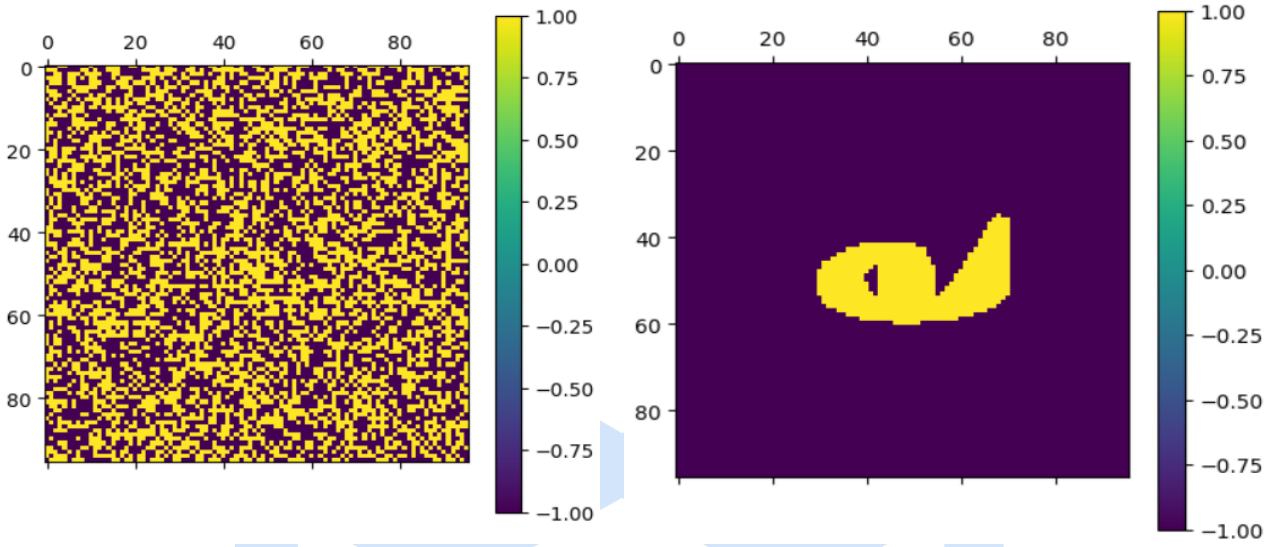
این کد یک شبکه همینگ را برای تشخیص الگوها در تصاویر دودویی پیاده‌سازی کرده است. با استفاده از توابع مختلف، تصویر ورودی به یک ماتریس تبدیل می‌شود. سپس وزن‌ها و ماتریس اتصال سیناپسی تنظیم می‌شوند.

شبکه با اعمال فعال‌سازی و آموزش به مرور خروجی تولید می‌کند تا زمانی که فاصله بین خروجی‌های متوالی کمتر از یک حداکثر مشخص Emax باشد. در نهایت، کلاس با بیشترین خروجی مشخص شده و تصویر متناظر با این کلاس نمایش داده می‌شود.

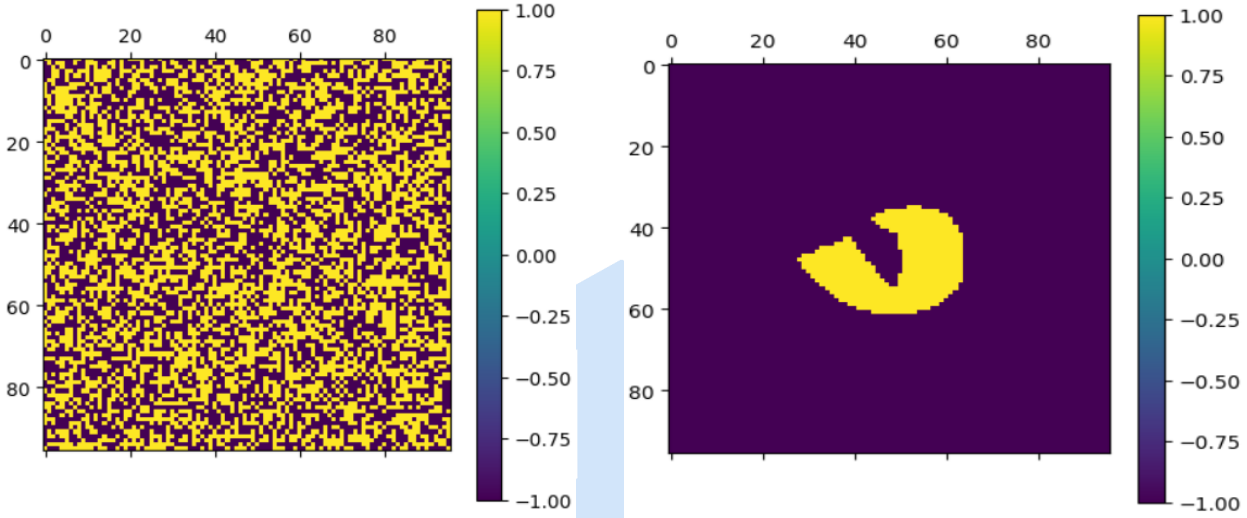


حال تنها ضریب نویز را تغییر می دهیم : (روی یک تصویر دیگر امتحان می کنیم.)





در این حالت انقدر ضریب نویز را افزایش داده ایم که شبکه دچار اختلال شد و تصویر را به اشتباه حدس زده است:



با اینکه افزایش نویز تصویر می‌تواند در برخی موارد برای آزمون و ارزیابی مدل‌ها یا الگوریتم‌های پردازش تصویر مفید باشد، اما میزان دقت و کارایی واقعی این وظیفه‌ها می‌تواند به شدت وابسته به نوع مسئله، الگوریتم‌ها، و حتی نحوه افزودن نویز باشد.

افزایش نویز تصویر به میزان مشخصی که منجر به افت شدید در دقت یا توانایی مدل یا الگوریتم شود، ممکن است بستگی به موارد زیر داشته باشد:

۱. **نوع مسئله:**

برخی مسائل پردازش تصویر حساس‌تر به نویز هستند. به عنوان مثال، در وظایف تشخیص اشیاء یا تفکیک کلاس‌ها، افزایش نویز ممکن است توانایی مدل را به شدت کاهش دهد.

۲. **نوع الگوریتم:**

الگوریتم‌ها با پایه ریاضیاتی و علم داده مختلف به نویز ورودی واکنش متفاوتی نشان می‌دهند. الگوریتم‌های قوی‌تر ممکن است توانایی بهتری در مقابل نویز داشته باشند.

۳. **میزان و نوع نویز:**

مقدار و نوع نویزی که به تصویر افزوده می‌شود، تأثیر زیادی بر نتایج نهایی دارد. نویزهای پراکنده، نویزهای گوسی، یا نویزهای ساختاری ممکن است به شکل‌های مختلف بر توانایی الگوریتم تاثیر بگذارند.

۴. **حساسیت مدل به نویز:**

بعضی مدل‌ها حساسیت کمتری به نویز دارند و ممکن است در شرایط نویزی بهتر عمل کنند. در عین حال، مدل‌های پیچیده‌تر ممکن است به نویز حساس‌تر باشند.

۳-با الهام گرفتن از تابع نوشته شده برای تولید داده های نویزی، یک تابع بنویسید که از داده های ورودی، خروجی های دارای Point Missing تولید کند. سپس عملکرد شبکهٔ خود را با مقدار مشخصی Point Missing آزمایش و تحلیل کنید.

```
from PIL import Image, ImageDraw
import random

def getNoisyBinaryImage(input_path, output_path, num_missing_points,
conversion_percentage):
    """
    Add noise to an image, generate missing points, and save it as a new
file.

    Args:
        input_path (str): The file path to the input image.
        output_path (str): The file path to save the noisy image.
        num_missing_points (int): The number of missing points to
generate.
        conversion_percentage (float): The percentage of black pixels to
convert to white.
```

```
    """
    # Open the input image.
    image = Image.open(input_path)

    # Create a drawing tool for manipulating the image.
    draw = ImageDraw.Draw(image)

    # Determine the image's width and height in pixels.
    width = image.size[0]
    height = image.size[1]

    # Load pixel values for the image.
    pix = image.load()

    # Define a factor for introducing noise.
    noise_factor = 50

    # Loop through all pixels in the image.
    for i in range(width):
        for j in range(height):
            # Generate a random noise value within the specified factor.
            rand = random.randint(-noise_factor, noise_factor)

            # Add the noise to the Red, Green, and Blue (RGB) values of
the pixel.
            red = pix[i, j][0] + rand
            green = pix[i, j][1] + rand
            blue = pix[i, j][2] + rand

            # Ensure that RGB values stay within the valid range (0-255).
            if red < 0:
                red = 0
            if green < 0:
                green = 0
            if blue < 0:
                blue = 0
            if red > 255:
                red = 255
            if green > 255:
                green = 255
            if blue > 255:
                blue = 255

            # Convert some black pixels to white based on the conversion
percentage.
```

```
            if (red, green, blue) == (0, 0, 0) and random.random() <
conversion_percentage:
                red, green, blue = 255, 255, 255

            # Set the pixel color accordingly.
            draw.point((i, j), (red, green, blue))

    # Generate missing points in the image.
    for _ in range(num_missing_points):
        x = random.randint(0, width - 1)
        y = random.randint(0, height - 1)
        draw.point((x, y), (255, 255, 255))  # Set the missing point to
white

    # Save the noisy image as a file.
    image.save(output_path, "JPEG")

    # Clean up the drawing tool.
    del draw

from PIL import Image, ImageDraw
import random

def generateNoisyImages():
    # List of image file paths
    image_paths = [
        "/content/1.jpg",
        "/content/2.jpg",
        "/content/3.jpg",
        "/content/4.jpg",
        "/content/5.jpg"
    ]

    for i, image_path in enumerate(image_paths, start=1):
        noisy_image_path = f"/content/noisy{i}.jpg"
        # Specify the number of missing points and conversion percentage
here
        getNoisyBinaryImage(image_path, noisy_image_path,
num_missing_points=500, conversion_percentage=0.1)
        print(f"Noisy image for {image_path} generated and saved as
{noisy_image_path}")

# Generate noisy images with missing points and black-to-white conversion
generateNoisyImages()
```

این تابع یک تصویر ورودی را با افزودن نویز به پیکسل‌ها تغییر می‌دهد. همچنین، تعدادی از پیکسل‌های سیاه را به سفید تبدیل می‌کند. علاوه بر این، تعدادی نقطه گم‌شده به تصویر اضافه می‌شود. این عملیاتها با استفاده از پارامترهای num_missing_points تعداد نقاط گم‌شده و conversion_percentage درصد تبدیل پیکسل‌های سیاه به سفید قابل تنظیم هستند.

```
from pylab import *
from math import sqrt
import matplotlib.pyplot as plt
import os

# Define the path to the input image
IMAGE_PATH = "/content/noisy5.jpg"

def show(matrix):
    """
    Display a matrix in a formatted manner.

    Args:
        matrix (list of lists): The matrix to be displayed.
    """
    for j in range(len(matrix)):
        for i in range(len(matrix[0])):
            print("{:3f}".format(matrix[j][i]), end=" ")
        print(sep="")

def change(vector, a, b):
    """
    Transform a vector into a matrix of specified dimensions.

    Args:
        vector (list): The vector to be transformed.
        a (int): The number of columns in the resulting matrix.
        b (int): The number of rows in the resulting matrix.

    Returns:
        list of lists: The transformed matrix.
    """
    matrix = [[0 for j in range(a)] for i in range(b)]
    k = 0
    j = 0
    while k < b:
        i = 0
        while i < a:
```

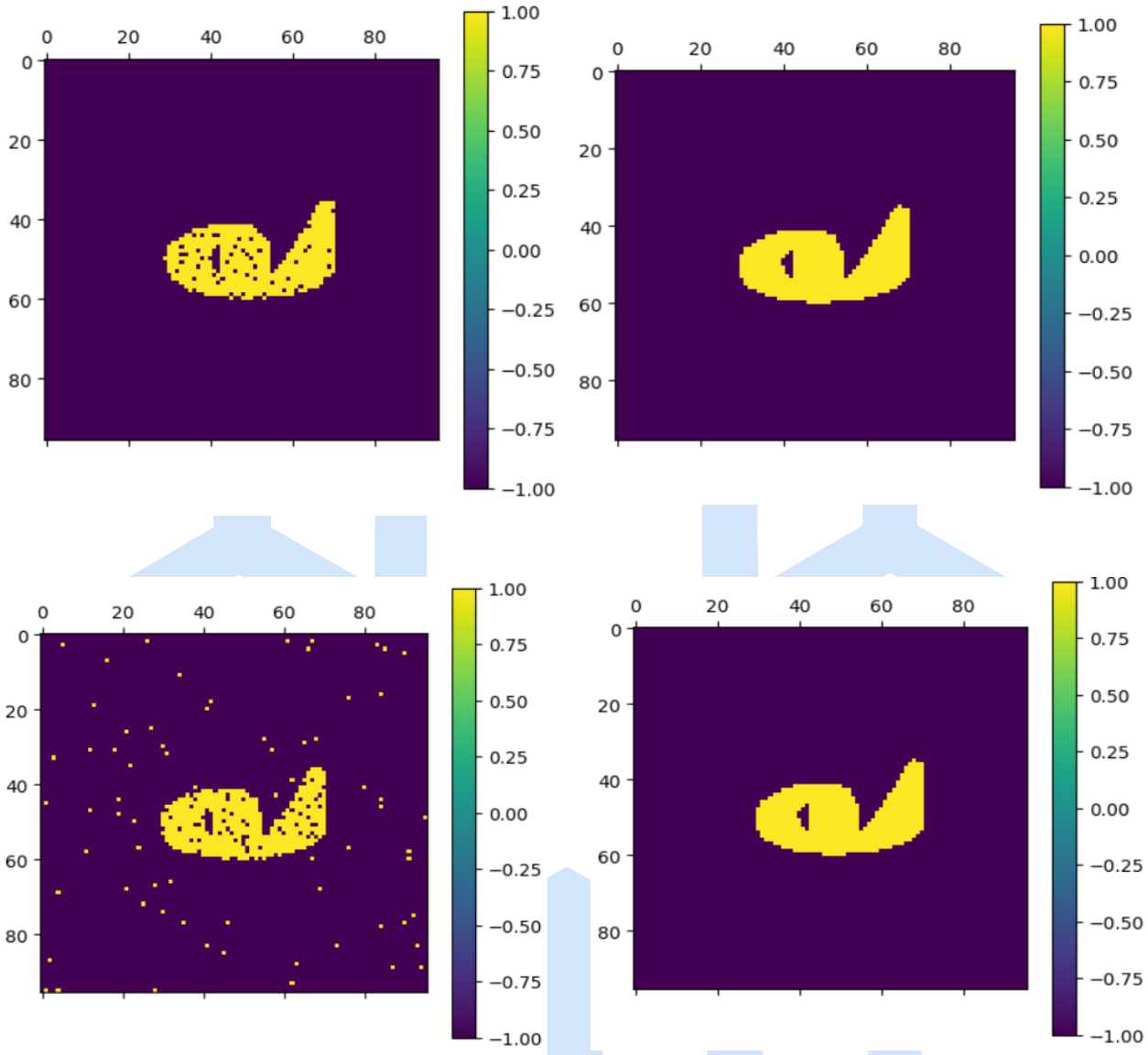
```
            matrix[k][i] = vector[j]
            j += 1
            i += 1
        k += 1
    return matrix

def product(matrix, vector, T):
    """
    Multiply a matrix by a vector.

    Args:
        matrix (list of lists): The matrix to be multiplied.
        vector (list): The vector to be multiplied.
        T (float): The threshold parameter for the activation function.

    Returns:
        list: The resulting vector after multiplication.
    """
    result_vector = []
    for i in range(len(matrix)):
        x = 0
        for j in range(len(vector)):
            x = x + matrix[i][j] * vector[j]
        result_vector.append((x + T))
    return result_vector

def action(vector, T, Emax):
    """
    Activation function to process a vector.

    Args:
        vector (list): The input vector to be processed.
        T (float): The threshold parameter for the activation function.
        Emax (float): The maximum allowable value for the difference in
output vectors between consecutive iterations.

    Returns:
        list: The output vector after activation.
    """
    result_vector = []
    for value in vector:
        if value <= 0:
            result_vector.append(0)
        elif 0 < value <= T:
            result_vector.append(Emax * value)
```

```
        elif value > T:
            result_vector.append(T)
    return result_vector

def mysum(vector, j):
    """
    Calculate the sum of vector values excluding the element at index j.

    Args:
        vector (list): The input vector.
        j (int): The index of the element to be excluded from the sum.

    Returns:
        float: The sum of vector values with the element at index j
excluded.
    """
    p = 0
    total_sum = 0
    while p < len(vector):
        if p != j:
            total_sum = total_sum + vector[p]
        p += 1
    return total_sum

def norm(vector, p):
    """
    Calculate the difference between two vectors and compute the norm of
the resulting vector.

    Args:
        vector (list): The first vector.
        p (list): The second vector for subtraction.

    Returns:
        float: The Euclidean norm of the difference between the two
vectors.
    """
    difference = []
    for i in range(len(vector)):
        difference.append(vector[i] - p[i])
    sum = 0
    for element in difference:
        sum += element * element
    return sqrt(sum)
```

```
# List of paths to example images
path = [
    '/content/1.jpg',
    '/content/2.jpg',
    '/content/3.jpg',
    '/content/4.jpg',
    '/content/5.jpg',
]

x = []  # Binary representations of example images
print(os.path.basename(IMAGE_PATH))

# Convert and store binary representations of example images
for i in path:
    x.append(convertImageToBinary(i))

y = convertImageToBinary(IMAGE_PATH)  # Binary representation of the input
image
entr = y
k = len(x)  # Number of example images
a = 96  # Number of columns in the transformed matrix
b = 96  # Number of rows in the transformed matrix
entr = y
q = change(y, a, b)  # Transformation of input image into a matrix
plt.matshow(q)
plt.colorbar()

m = len(x[0])
w = [[(x[i][j]) / 2 for j in range(m)] for i in range(k)]  # Weight matrix
T = m / 2  # Activation function threshold parameter
e = round(1 / len(x), 1)
E = [[0 for j in range(k)] for i in range(k)]  # Synaptic connection
matrix
Emax = 0.000001  # Maximum allowable difference norm between output
vectors in consecutive iterations
U = 1 / Emax

# Set values for the synaptic connection matrix
for i in range(k):
    for j in range(k):
        if j == i:
            E[i][j] = 1.0
        else:
            E[i][j] = -e
```

```
s = [product(w, y, T)]  # Initial output vector
p = action(s[0], U, Emax)
y = [p]
i = 0
j = []
p = [0 for j in range(len(s[0]))]

# Iterate until the difference norm is less than Emax
while norm(y[i], p) >= Emax:
    s.append([0 for j in range(len(s[0]))])
    for j in range(len(s[0])):
        s[i + 1][j] = y[i][j] - e * mysum(y[i], j)
    y.append((action(s[i + 1], U, Emax)))
    i += 1
    p = y[i - 1]

print('Output Vectors Table:')
show(y)
print('Last Output Vector:', *y[len(y) - 1])

# Determine the class with the highest output value
result_index = y[len(y) - 1].index(max(y[len(y) - 1])) + 1

if max(y[len(y) - 1]) == 0:
    print("The Hamming network cannot make a preference between classes.")
    print("In the case of a small number of input characteristics, the
network may not be able to classify the image.")
    plt.show()
    exit()
else:
    q = change(x[result_index - 1], a, b)
    print('The highest positive output value is associated with class',
result_index)
    plt.matshow(q)
    plt.colorbar()
    plt.show()
```

در نتیجه شکل نهایی به صورت زیر خواهد شد :



میزان Point Missing یا تعداد نقاط گم‌شده بیش از حد می‌تواند به اختلال در عملکرد شبکه یا الگوریتم‌هایی که از تصاویر مشابه به عنوان ورودی استفاده می‌کنند، منجر شود. این اختلالات ممکن است به دلیل از دست رفتن اطلاعات مهم در نقاطی که اضافه شده‌اند، یا تغییرات ناخواسته در محتوای تصویر ایجاد شوند.

راه‌حل‌های ممکن برای مقابله با این اختلالات عبارتند از:

۱. **میزان معقول Point Missing

- انتخاب یک حد معقول برای تعداد نقاط گم‌شده به نحوی که تاثیر بر عملکرد شبکه کم باشد. این مقدار باید با توجه به خصوصیات و نیازهای مساله تنظیم شود.

۲. **استفاده از تصاویر کنترل

- برای ارزیابی عملکرد شبکه، از تصاویر کنترل (تصاویر بدون نویز یا تغییرات) نیز استفاده کنید و نتایج را با تصاویر نویزدار مقایسه کنید.

۳. **تنظیمات دقت نویز

- کنترل دقت نویز افزوده شده به تصویر. مثلاً مقدار `noise_factor` را به یک حد معقول تنظیم کنید تا نویز اضافه شده به حد مفید باقی بماند.

۴. **تصویربرداری دقیق

- از روش‌های تصویربرداری دقیق‌تری برای اضافه کردن نویز و تولید نقاط گم‌شده استفاده کنید تا از از دست رفتن اطلاعات حیاتی جلوگیری شود.

۵. **ارزیابی همواره

- عملکرد شبکه را با مقدار مختلف نقاط گم‌شده ارزیابی کنید و تاثیر آن بر دقت و عملکرد کلی شبکه را مشاهده کنید.

۶. **استفاده از روشهای پیشرفته‌تر

- در صورت نیاز، از روش‌های پیشرفته‌تری برای مدیریت نویز و نقاط گم‌شده استفاده کنید، مثل استفاده از شبکه‌های مولد (GANs) برای تولید تصاویر نویزدار و گم‌شده.

یا می توانیم برای این حالت از کد زیر بهره بگیریم :

```
from pylab import *
from math import sqrt
import matplotlib.pyplot as plt
import os

# Define the path to the input image
IMAGE_PATH = "/content/noisy1.jpg"

def show(matrix):
    """
```

```
    Display a matrix in a formatted manner.

    Args:
        matrix (list of lists): The matrix to be displayed.
    """
    for j in range(len(matrix)):
        for i in range(len(matrix[0])):
            print("{:3f}".format(matrix[j][i]), end=" ")
        print(sep="")

def change(vector, a, b):
    """
    Transform a vector into a matrix of specified dimensions.

    Args:
        vector (list): The vector to be transformed.
        a (int): The number of columns in the resulting matrix.
        b (int): The number of rows in the resulting matrix.

    Returns:
        list of lists: The transformed matrix.
    """
    matrix = [[0 for j in range(a)] for i in range(b)]
    k = 0
    j = 0
    while k < b:
        i = 0
        while i < a:
            matrix[k][i] = vector[j]
            j += 1
            i += 1
        k += 1
    return matrix

def product(matrix, vector, T):
    """
    Multiply a matrix by a vector.

    Args:
        matrix (list of lists): The matrix to be multiplied.
        vector (list): The vector to be multiplied.
        T (float): The threshold parameter for the activation function.

    Returns:
        list: The resulting vector after multiplication.
```

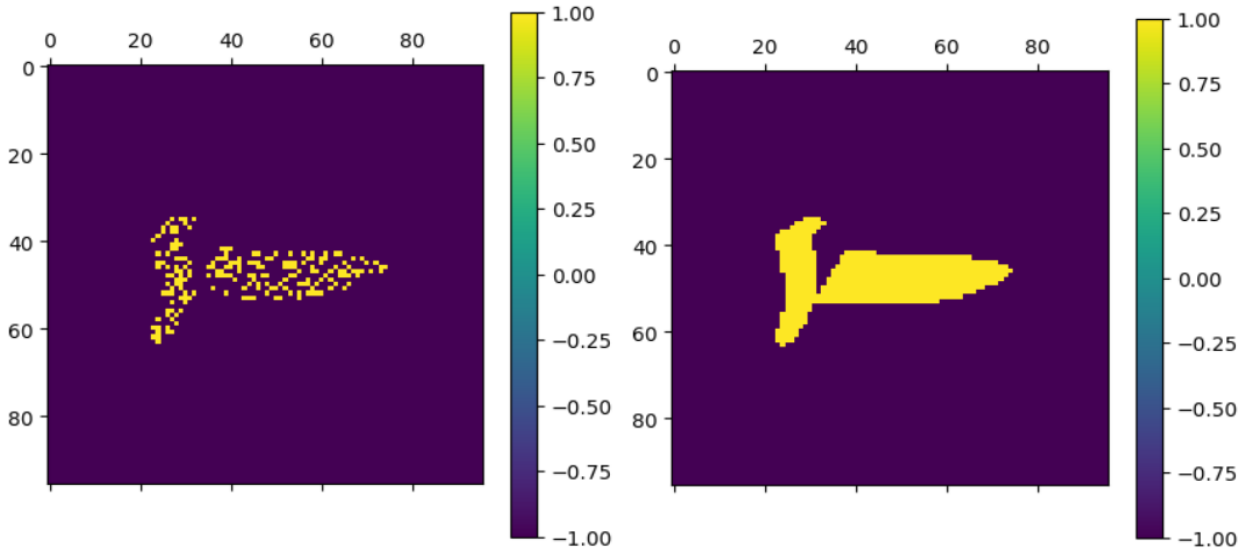
```
    """
    result_vector = []
    for i in range(len(matrix)):
        x = 0
        for j in range(len(vector)):
            x = x + matrix[i][j] * vector[j]
        result_vector.append((x + T))
    return result_vector

def action(vector, T, Emax):
    """
    Activation function to process a vector.

    Args:
        vector (list): The input vector to be processed.
        T (float): The threshold parameter for the activation function.
        Emax (float): The maximum allowable value for the difference in
output vectors between consecutive iterations.

    Returns:
        list: The output vector after activation.
    """
    result_vector = []
    for value in vector:
        if value <= 0:
            result_vector.append(0)
        elif 0 < value <= T:
            result_vector.append(Emax * value)
        elif value > T:
            result_vector.append(T)
    return result_vector

def mysum(vector, j):
    """
    Calculate the sum of vector values excluding the element at index j.

    Args:
        vector (list): The input vector.
        j (int): The index of the element to be excluded from the sum.

    Returns:
        float: The sum of vector values with the element at index j
excluded.
    """
    p = 0
```

```
    total_sum = 0
    while p < len(vector):
        if p != j:
            total_sum = total_sum + vector[p]
        p += 1
    return total_sum

def norm(vector, p):
    """
    Calculate the difference between two vectors and compute the norm of
the resulting vector.

    Args:
        vector (list): The first vector.
        p (list): The second vector for subtraction.

    Returns:
        float: The Euclidean norm of the difference between the two
vectors.
    """
    difference = []
    for i in range(len(vector)):
        difference.append(vector[i] - p[i])
    sum = 0
    for element in difference:
        sum += element * element
    return sqrt(sum)

# List of paths to example images
path = [
    '/content/1.jpg',
    '/content/2.jpg',
    '/content/3.jpg',
    '/content/4.jpg',
    '/content/5.jpg',
]

x = []  # Binary representations of example images
print(os.path.basename(IMAGE_PATH))

# Convert and store binary representations of example images
for i in path:
    x.append(convertImageToBinary(i))
```

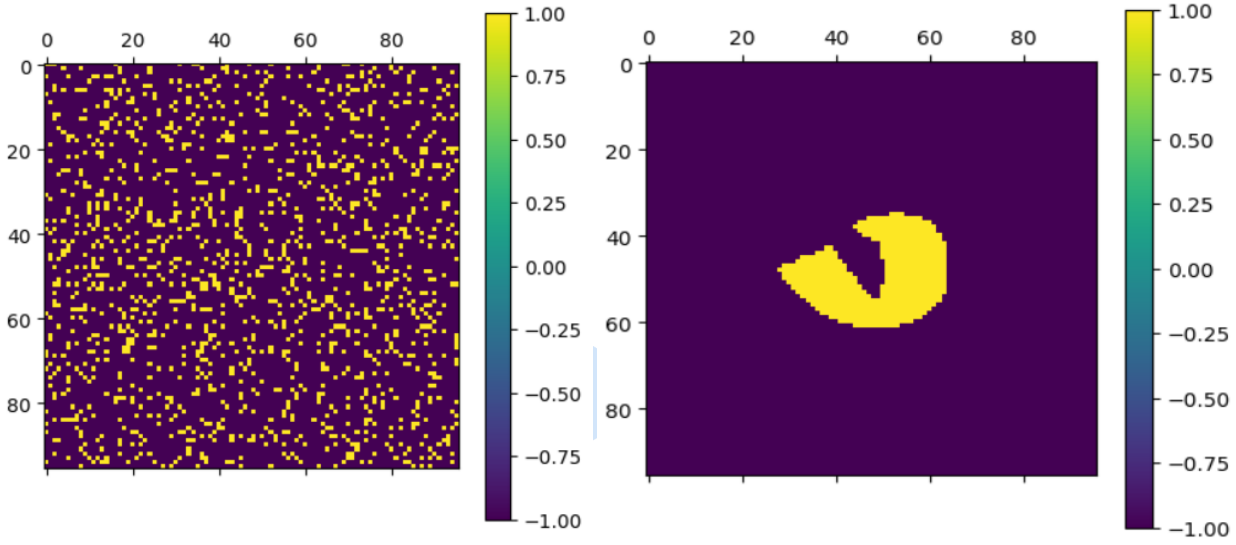
```
y = convertImageToBinary(IMAGE_PATH)  # Binary representation of the input
image
entr = y
k = len(x)  # Number of example images
a = 96  # Number of columns in the transformed matrix
b = 96  # Number of rows in the transformed matrix
entr = y
q = change(y, a, b)  # Transformation of input image into a matrix
plt.matshow(q)
plt.colorbar()

m = len(x[0])
w = [[(x[i][j]) / 2 for j in range(m)] for i in range(k)]  # Weight matrix
T = m / 2  # Activation function threshold parameter
e = round(1 / len(x), 1)
E = [[0 for j in range(k)] for i in range(k)]  # Synaptic connection
matrix
Emax = 0.000001  # Maximum allowable difference norm between output
vectors in consecutive iterations
U = 1 / Emax

# Set values for the synaptic connection matrix
for i in range(k):
    for j in range(k):
        if j == i:
            E[i][j] = 1.0
        else:
            E[i][j] = -e

s = [product(w, y, T)]  # Initial output vector
p = action(s[0], U, Emax)
y = [p]
i = 0
j = []
p = [0 for j in range(len(s[0]))]

# Iterate until the difference norm is less than Emax
while norm(y[i], p) >= Emax:
    s.append([0 for j in range(len(s[0]))])
    for j in range(len(s[0])):
        s[i + 1][j] = y[i][j] - e * mysum(y[i], j)
    y.append((action(s[i + 1], U, Emax)))
    i += 1
    p = y[i - 1]
```

```
print('Output Vectors Table:')
show(y)
print('Last Output Vector:', *y[len(y) - 1])

# Determine the class with the highest output value
result_index = y[len(y) - 1].index(max(y[len(y) - 1])) + 1

if max(y[len(y) - 1]) == 0:
    print("The Hamming network cannot make a preference between classes.")
    print("In the case of a small number of input characteristics, the
network may not be able to classify the image.")
    plt.show()
    exit()
else:
    q = change(x[result_index - 1], a, b)
    print('The highest positive output value is associated with class',
result_index)
    plt.matshow(q)
    plt.colorbar()
    plt.show()
```

این کد یک شبکه همینگ را پیاده‌سازی می‌کند که برای تشخیص الگوها در تصاویر دودویی استفاده می‌شود. شبکه از توابعی مانند ضرب ماتریسی و فعال‌سازی برای آموزش و تشخیص الگوها استفاده می‌کند. پس از آموزش، شبکه توانایی تشخیص الگوها در تصویر و تصمیم‌گیری در مورد کلاس متناظر با الگوها را داراست.



به طور با بالاتر بردن ضریب نویز مدل در حالت زیر مدل اشتباه تصویر نویزی را حدس زده است :



بخش۱: سوالات تحلیلی

سوال چهارم :

```
# PART1
# Load the dataset from the specified file path
data = pd.read_csv('/content/data.csv')
# Display the first few rows of the DataFrame
data.head()
```

کد بالا برای بارگیری مجموعه داده از مسیر فایل مشخص شده و نمایش چند ردیف اول از داده‌ها در یک DataFrame استفاده می‌شود.

```
data.isnull().sum()
```

این دستور به تعداد مقادیر ناپردازشگر در هر ستون از مجموعه داده اشاره دارد. به عبارت دیگر، برای هر ستون در DataFrame، تعداد مقادیر ناپردازشگر مقادیر خالی یا NaNشمارش می‌شود.

```
# Remove rows with any null values
data.dropna(inplace=True)
```

این دستور برای حذف سطرهایی از مجموعه داده استفاده می‌شود که حداقل یک مقدار ناپردازشگر دارند. با قرار دادن inplace=True تغییرات مستقیماً در DataFrame اعمال می‌شوند، به این معنی که DataFrame اصلی تغییر می‌کند و نیازی به ایجاد یک DataFrame جدید نیست. این کار معمولاً برای پاکسازی داده‌ها از مقادیر ناپردازشگر و حذف سطرهایی که حاوی آنها هستند، انجام می‌شود.

```
# Select columns with numerical data types
num = data.select_dtypes(exclude=['object']).columns
num
```

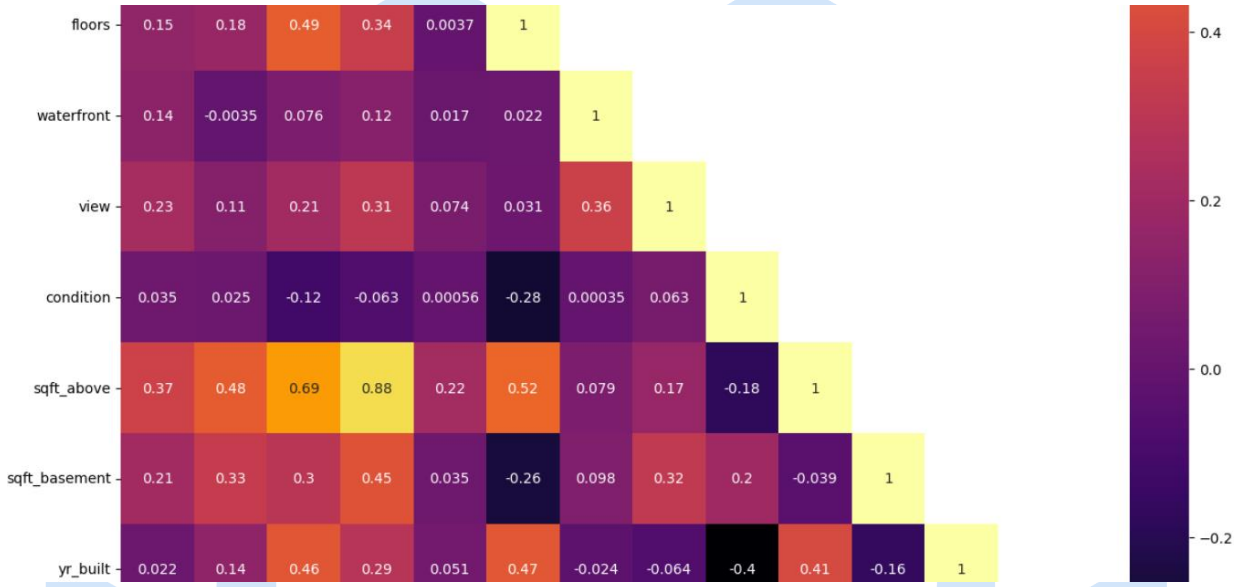
این کد برای انتخاب ستون‌هایی از مجموعه داده استفاده می‌شود که داده‌های عددی (اعشاری یا صحیح) دارند. تابع select_dtypes بر اساس نوع داده‌ها، انتخاب ستون‌های DataFrame را انجام می‌دهد. در اینجا، با استفاده از پارامتر exclude=['object'] ، ستون‌هایی که داده‌هایشان نوع رشته‌ای object نیستند (بلکه داده‌های عددی دارند) انتخاب می‌شوند. نتیجه این انتخاب در متغیر num ذخیره شده و نمایش داده می‌شود.

```
# PART2
# Create a heatmap to visualize the correlation matrix of numerical
columns
plt.figure(figsize=(15, 15))
```

```
sns.heatmap(data[num].corr(), annot=True, cmap='inferno',
mask=np.triu(data[num].corr(), k=1))
```

این بخش از کد یک نمودار حرارت از ماتریس همبستگی بین اعداد دیتافریم ایجاد می‌کند و مقادیر واقعی همبستگی‌ها را در هر خانه نمایش می‌دهد. استفاده از رنگهای inferno و حذف نواحی تکراری در heatmap این تصویرسازی را بهبود می‌بخشد.

به طور مثال بخشی از این جدول را در زیر می توانیم مشاهده کنیم :



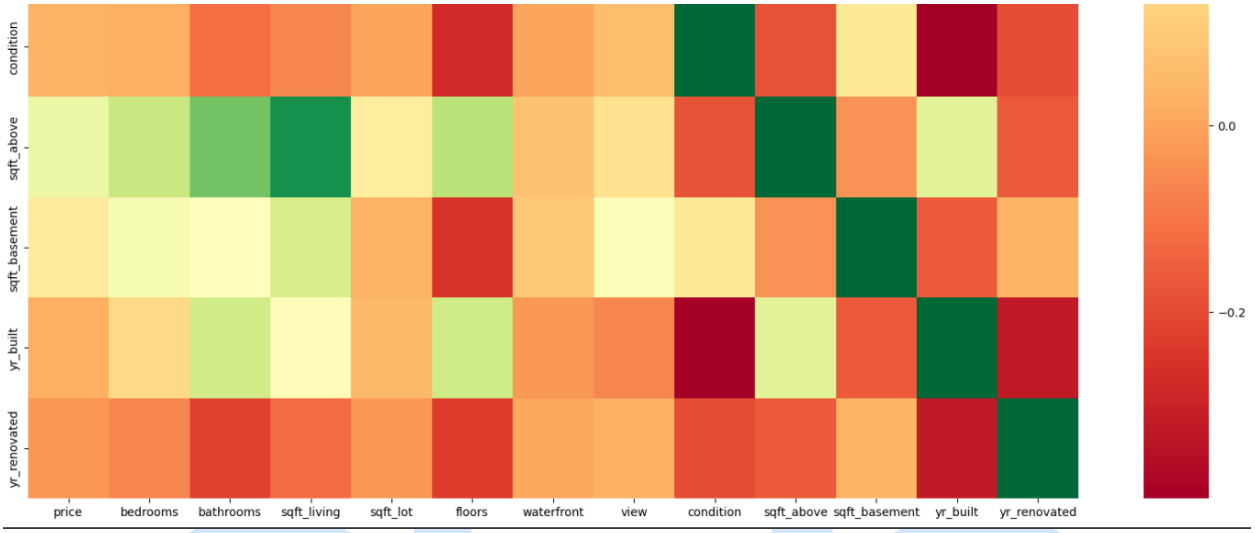

```
# Calculate the correlation between columns and 'price', then sort them in
descending order
correlation_matrix = data.corr()['price'].sort_values(ascending=False)
correlation_matrix
```

این کد برای محاسبه همبستگی بین ستون‌های مختلف و ستون 'price' می‌باشد. نتیجه ماتریس همبستگی بین هر ستون و ستون price به ترتیب نزولی مرتب شده و در متغیر correlation_matrix ذخیره می‌شود. این اطلاعات معمولاً برای درک ارتباطات بین متغیرها و تاثیر آنها بر متغیر price مفید است.

```
# Plot a heatmap to visualize the correlation matrix
plt.figure(figsize=(20, 20))
sns.heatmap(data.corr(), cmap="RdYlGn")
plt.show()
```

در این بخش از کد، یک نمودار حرارت heatmap ایجاد می‌شود تا ماتریس همبستگی بین تمام ستون‌های داده را به صورت گرافیکی نمایش دهد. از نقشه رنگی RdYlGn برای نمایش مقادیر همبستگی استفاده شده است.

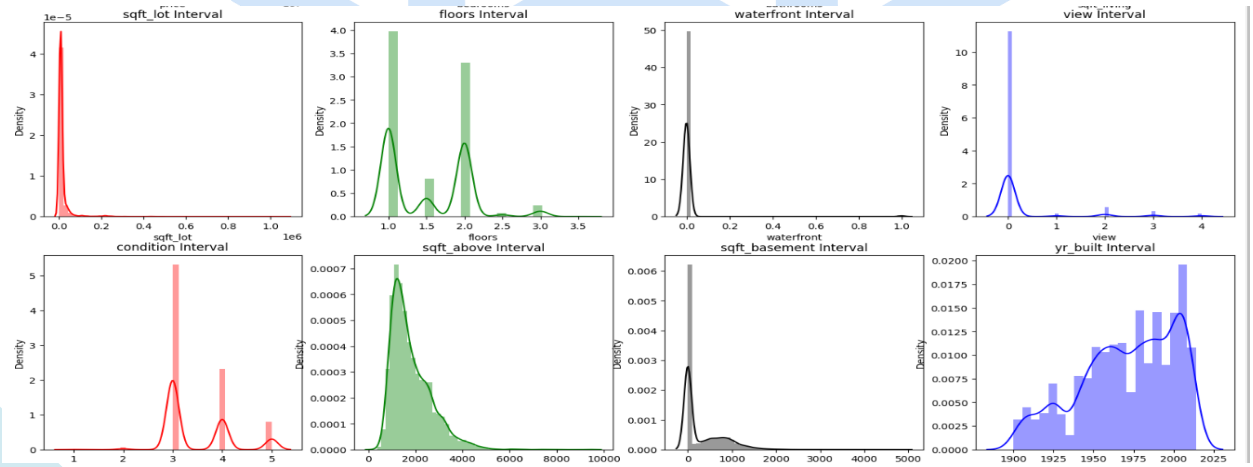
در نهایت، نمودار حرارت با استفاده از دستور plt.show نمایش داده می‌شود. این گونه نمودارها به تحلیل ارتباطات میان متغیرها در داده‌ها کمک می‌کنند.




```
# Create a 4x4 grid of subplots for various numerical variables
plt.figure(figsize=(20, 20))

plt.subplot(4,4,1)
sns.distplot(data['price'], color="red").set_title('price Interval')

plt.subplot(4,4,2)
sns.distplot(data['bedrooms'], color="green").set_title('bedrooms
Interval')

plt.subplot(4,4,3)
sns.distplot(data['bathrooms'], color="black").set_title('bathrooms
Interval')

plt.subplot(4,4,4)
sns.distplot(data['sqft_living'], color="blue").set_title('sqft_living
Interval')

plt.subplot(4,4,5)
sns.distplot(data['sqft_lot'], color="red").set_title('sqft_lot Interval')

plt.subplot(4,4,6)
sns.distplot(data['floors'], color="green").set_title('floors Interval')

plt.subplot(4,4,7)
sns.distplot(data['waterfront'], color="black").set_title('waterfront
Interval')
```

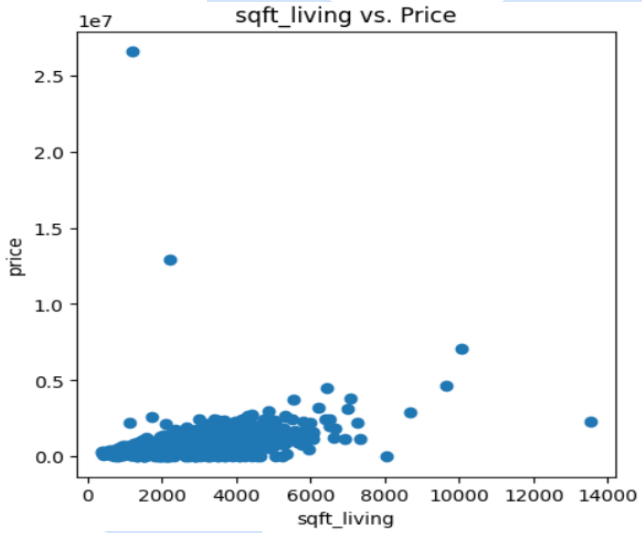
```
plt.subplot(4,4,8)
sns.distplot(data['view'], color="blue").set_title('view Interval')

plt.subplot(4,4,9)
sns.distplot(data['condition'], color="red").set_title('condition
Interval')

plt.subplot(4,4,10)
sns.distplot(data['sqft_above'], color="green").set_title('sqft_above
Interval')

plt.subplot(4,4,11)
sns.distplot(data['sqft_basement'],
color="black").set_title('sqft_basement Interval')

plt.subplot(4,4,12)
sns.distplot(data['yr_built'], color="blue").set_title('yr_built
Interval')

plt.subplot(4,4,13)
sns.distplot(data['yr_renovated'], color="red").set_title('yr_renovated
Interval')
```

در این قسمت از کد، یک شبکه ۴x۴ از نمودارهای توزیع فراهم شده است تا توزیع متغیرهای عددی مختلف را نشان دهد. هر سطر از شبکه به چهار متغیر مختلف اختصاص دارد. از تابع sns.distplot برای رسم نمودارهای توزیع استفاده شده است. هر subplot به یک متغیر عددی از داده اختصاص داده شده و رنگ متناسب با هر subplot تعیین شده است. عنوان هر subplot نیز نام متغیر متناظر با آن است. از set_title برای افزودن عنوان به هر subplot استفاده شده است. این کد به تحلیل توزیع متغیرهای عددی مختلف در مجموعه داده کمک می‌کند.



```
# Create a scatter plot of enginesize against price
plt.figure(figsize=(5, 5))
plt.scatter(x='sqft_living', y='price', data=data)
plt.xlabel('sqft_living')
plt.title('sqft_living vs. Price')
plt.ylabel('price')
plt.show()
```

این بخش از کد یک نمودار scatter ایجاد می‌کند که رابطه بین متغیرهای sqft_living و price را نشان می‌دهد. این نمودار برای بررسی احتمال وجود رابطه خطی بین مساحت زندگی sqft_living و قیمت ملک price استفاده می‌شود.




```
# Extract 'year', 'month', and 'day' from the 'date' column
data['year'] = pd.to_datetime(data['date']).dt.year
data['month'] = pd.to_datetime(data['date']).dt.month

# Show the DataFrame with the separate 'year' and 'month' columns
data = data[['year', 'month'] + [col for col in data.columns if col not in
['year', 'month']]]

# Drop the specified columns from the DataFrame
data = data.drop(['date'], axis=1)
data
```

این بخش از کد، اطلاعات سال و ماه را از ستون date استخراج می‌کند و این اطلاعات را به دیتافریم اضافه می‌کند. سپس، ستون‌های year و month به ابتدای دیتافریم منتقل می‌شوند و ستون date حذف می‌شود. این کار برای افزایش قابلیت تحلیل زمانی داده‌ها انجام شده است.

```
# Drop year columns from the DataFrame
```

```
data = data.drop(['year'], axis=1)
data
```

این بخش از کد، ستون year را از DataFrame حذف می‌کند. این کار ممکن است به منظور حذف اطلاعات غیرضروری یا تغییر در ساختار داده‌ها باشد.

```
# List of specified categorical columns
dummy = ['city']
# Convert categorical columns to numerical using one-hot encoding
df2 = pd.get_dummies(data, columns=dummy, drop_first=True)

# Display the first few rows of the modified DataFrame
df2.head()
```

در این بخش از کد، ستون دسته‌ای city با استفاده از one-hot encoding به چندین ستون جدید تبدیل شده است. این اقدام باعث ایجاد متغیرهای دودویی جدید متناظر با دسته‌های مختلف در city می‌شود.

```
# PART5
x = df2.drop(["price"], axis=1)  # features
y = df2["price"]                  # Output data
# Perform train-test split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=83, shuffle=True)
# Print the shapes of the datasets
print("X Train Scaler : ", x_train.shape) # Print shape of x_train
print("X Test Scaler : ",  x_test.shape)  # Print shape of x_test
print("Y Train Scaler : ", y_train.shape) # Print shape of y_train
print("Y Test Scaler : ",  y_test.shape)  # Print shape of y_test
```

این بخش از کد داده‌ها را برای آموزش و آزمون مدل آماده می‌کند و ابعاد مجموعه‌های آموزش و آزمون را نمایش می‌دهد.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

```
# Initialize Min-Max Scaler
scaler_1 = MinMaxScaler()

# Normalize the training input data
x_train = scaler_1.fit_transform(x_train)

# Normalize the test input data
x_test = scaler_1.transform(x_test)
# Convert y_train and y_test type to DataFrame
```

```
y_train = pd.DataFrame(y_train)
y_test = pd.DataFrame(y_test)

scaler_2 = MinMaxScaler()

# Normalize outputs
y_train = scaler_2.fit_transform(y_train)
y_test = scaler_2.transform(y_test)
```

در این بخش از MinMaxScaler برای نرمال‌سازی داده‌های ورودی و خروجی استفاده شده است. داده‌های ورودی و خروجی آموزش و آزمون به صورت جداگانه نرمال‌سازی شده‌اند. بردارهای وابسته به فرمت DataFrame تبدیل شده‌اند. این مراحل به منظور آماده‌سازی داده‌ها برای مدلسازی می‌باشند.

```
model_3 = Sequential()

# Add the first hidden layer with 10 neurons and ReLU activation function
model_3.add(Dense(10, activation='relu', input_shape=(x_train.shape[1],)))

# Add the second hidden layer with 10 neurons and ReLU activation function
model_3.add(Dense(10, activation='relu'))

# Add the third hidden layer with 10 neurons and ReLU activation function
model_3.add(Dense(10, activation='relu'))

# Add an output layer with 1 neuron and linear activation function
model_3.add(Dense(1, activation='linear'))

model_3.summary()
```

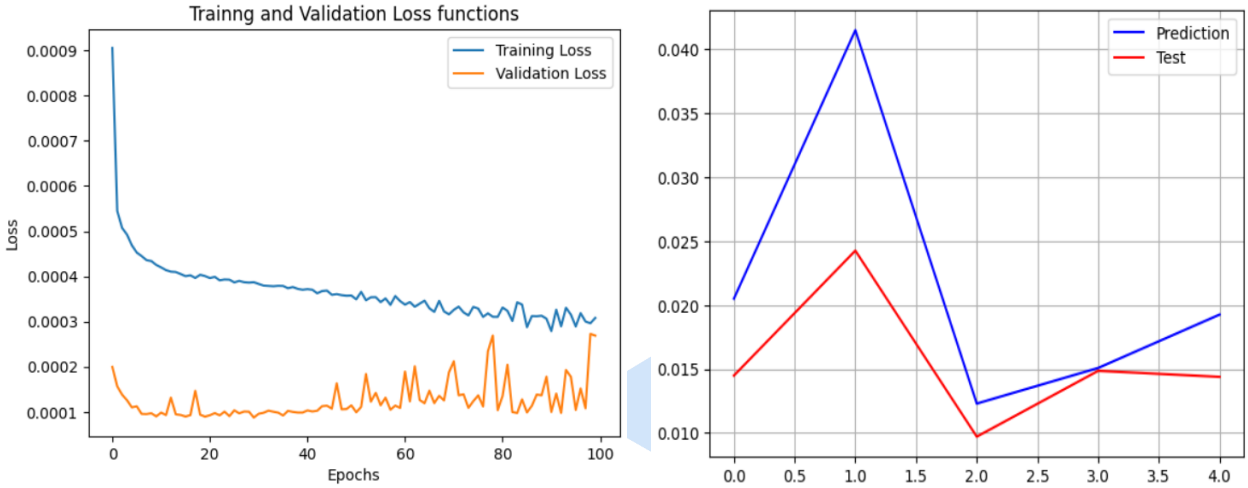
این بخش از کد یک مدل شبکه عصبی با سه لایه پنهان از نوع Dense ایجاد می‌کند:

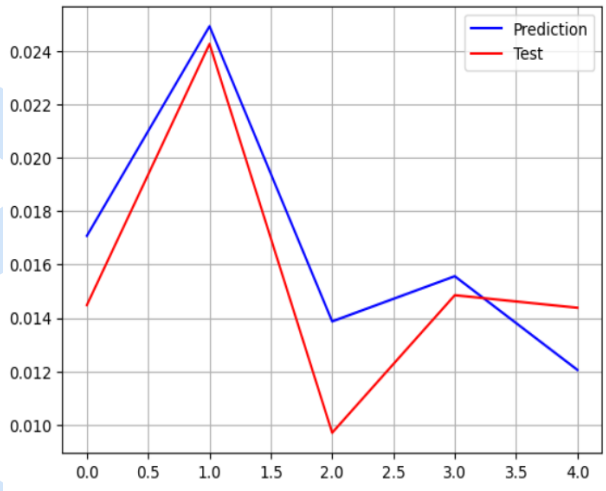
- لایه اول با ۱۰ نورون و فعال‌ساز ReLU.
- لایه دوم با ۱۰ نورون و فعال‌ساز ReLU.
- لایه سوم با ۱۰ نورون و فعال‌ساز ReLU.

در ادامه نیز دقت عوامل موثر و مدل مورد نظر ارائه شده بررسی می شود :

```
92/92 [==============================] - 0s 2ms/step
23/23 [==============================] - 0s 3ms/step
29/29 [==============================] - 0s 3ms/step
Test R2score: 0.23508013075766343
Train R2score: 0.40149585006740107
Validation R2score: 0.47556535537963096
```

در ادامه نمودار مربوط به تابع اتلاف و r2score را برای این حالت رسم می کنیم :



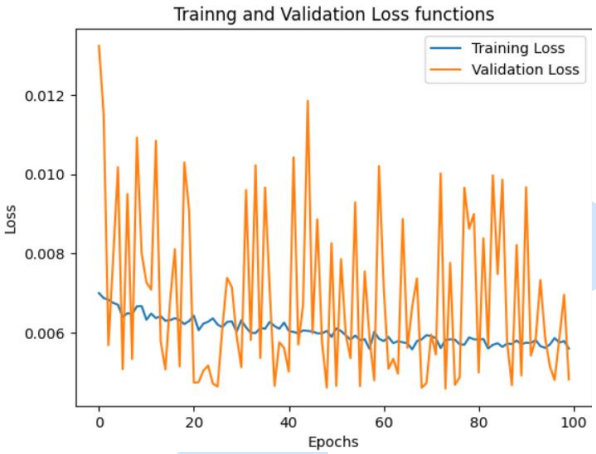


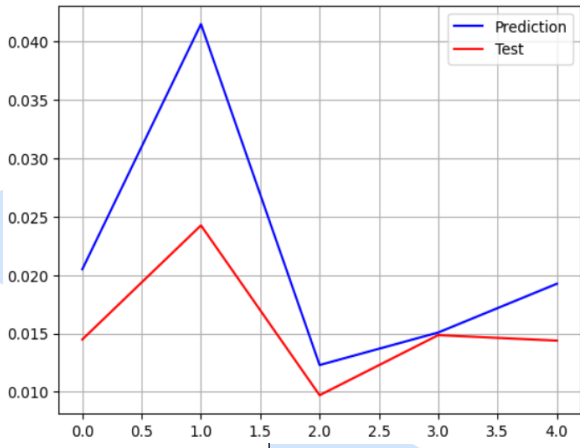
حال می خواهیم تمامی این مراحل را برای یک بهینه ساز و تابع اتلاف جدید بررسی کنیم :

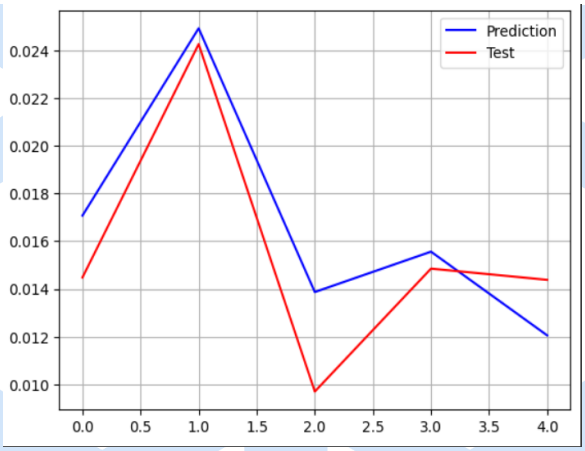
```
# PART7
# Compile model with stochastic gradient descent optimizer and mean
absolute error loss
model_3.compile(optimizer = 'sgd',loss = 'mae')

# Split the data into training and validation sets
x_train1, x_val, y_train1, y_val = train_test_split(x_train, y_train,
test_size=0.2, random_state=83, shuffle=True)

history = model_3.fit(x_train1, y_train1, validation_data=(x_val, y_val),
epochs=100, batch_size=10, verbose=0)

# Evaluate the model
loss = model_3.evaluate(x_test , y_test)
```

```
92/92 [==============================] - 0s 2ms/step
23/23 [==============================] - 0s 2ms/step
29/29 [==============================] - 0s 1ms/step
Test R2score: 0.23508013075766343
Train R2score: 0.40149585006740107
Validation R2score: 0.47556535537963096
```

در این حالت نیز نمودار های اتلاف و r2score را رسم می کنیم تا با حالت قبل مقایسه کنیم :







استفاده از یک بهینه‌ساز و تابع اتلاف خاص در آموزش مدل عصبی می‌تواند تأثیر مهمی بر عملکرد و کارایی مدل داشته باشد. در ادامه توضیح داده‌ام:

- **انتخاب بهینه‌ساز**: انتخاب بهینه‌ساز می‌تواند تأثیر زیادی در سرعت و کیفیت آموزش مدل داشته باشد. بهینه‌سازها مسئول به‌روزرسانی وزن‌ها در هر مرحله از آموزش هستند. بهینه‌سازهای مختلف مثل Adam، SGD، و RMSprop، خصوصیات و الگوریتم‌های مختلفی دارند که بر اساس مشخصه داده و مسئله مورد استفاده متفاوت هستند.

- **انتخاب تابع اتلاف**: تابع اتلاف مشخص می‌کند که مدل چقدر از مقدار واقعی فاصله دارد. انتخاب صحیح تابع اتلاف بر اساس نوع مسئله موجود مهم است. برای مسائل رگرسیون معمولاً از MSE میانگین مربعات خطا و برای مسائل طبقه‌بندی از توابعی مانند Cross-Entropy استفاده می‌شود.

تأثیر این انتخابات:

۱. **سرعت آموزش**: بهینه‌سازها ممکن است به سرعت و یا کندی فرآیند آموزش مدل تأثیر بگذارند. بهینه‌سازهای مبتنی بر مفاهیم گرادیان، معمولاً در جستجوی مسیرهای کمینه سریعتر عمل می‌کنند.

۲. **پایداری**: انتخاب تابع اتلاف صحیح می‌تواند در پایداری آموزش مدل تأثیر داشته باشد. توابع اتلاف مختلف می‌توانند بر اساس خصوصیات داده مثبت یا منفی بر این پایداری تأثیر بگذارند.

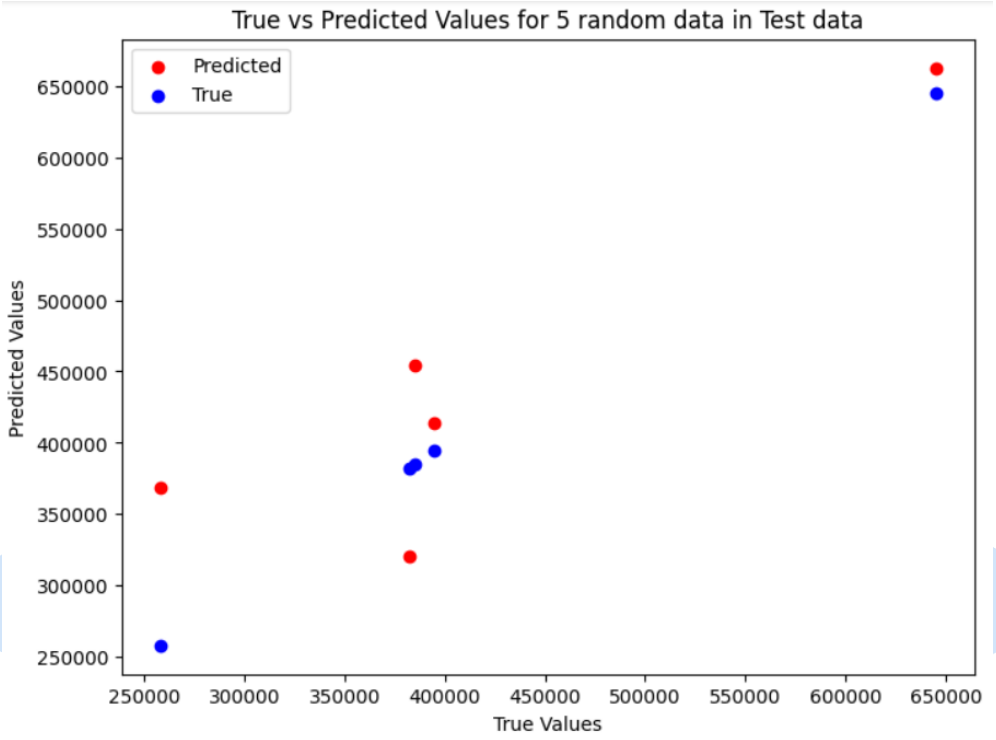
۳. **مقاومت به داده‌های نویزی**: برخی توابع اتلاف مقاومت بیشتری در مقابل داده‌های نویزی دارند و می‌توانند مدل را بهتر در برابر داده‌های ناهمگن سازند.

۴. **عملکرد نهایی مدل**: این تنظیمات به صورت مشترک می‌توانند به عملکرد نهایی مدل در مقابل داده‌های تست تأثیر بگذارند. این انتخابات ممکن است مدل را به سمت یک جواب بهینه‌تر و کارآمدتر هدایت کنند.

```
# PART8
# Inverse transform the scaled test data and predictions
y_test_unscaled = scaler_2.inverse_transform(y_test)
y_pred_unscaled = scaler_2.inverse_transform(y_pred_3_test)

random_pred = []
random_test = []

for i in range(5):
  j = random.randint(0, len(y_test_unscaled))
  random_pred.append(y_pred_unscaled[i])
  random_test.append(y_test_unscaled[i])

# Plotting the unscaled true test data against predictions with different
colors
plt.figure(figsize=(8, 6))
plt.scatter(random_test, random_pred, color='red', label='Predicted')
plt.scatter(random_test, random_test, color='blue', label='True')
plt.title('True vs Predicted Values for 5 random data in Test data')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.legend()
```

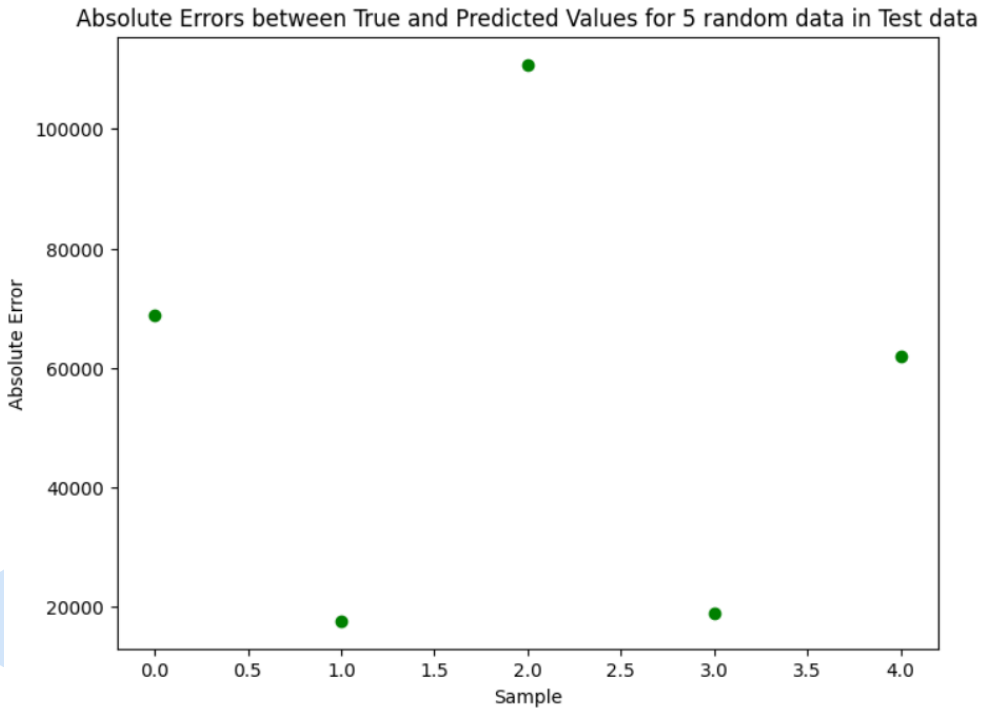
```
plt.show()
```

این بخش از کد، مقادیر پیش‌بینی شده و واقعی برای پنج نمونه تصادفی از داده‌های آزمون را در یک نمودار مقایسه نشان می‌دهد. این کار به بررسی تطابق یا عدم تطابق مقادیر پیش‌بینی با واقعی کمک می‌کند.




```
# Assuming random_test and random_pred are lists
random_test = np.array(random_test)
random_pred = np.array(random_pred)

# Calculate errors between true and predicted values
errors = np.abs(random_test - random_pred)

# Plotting the errors
plt.figure(figsize=(8, 6))
plt.plot(errors, marker='o', linestyle='', color='green')
plt.title('Absolute Errors between True and Predicted Values for 5 random
data in Test data')
plt.xlabel('Sample')
plt.ylabel('Absolute Error')
plt.show()
```

این بخش از کد، اختلاف مطلق بین مقادیر واقعی و پیش‌بینی شده را برای پنج نمونه تصادفی نشان می‌دهد. این اختلافات در یک نمودار با استفاده از خطاهای مطلق نمایش داده شده‌اند.



برای بهبود عملکرد و کاهش اختلاف مطلق بین مقادیر واقعی و پیش‌بینی شده، می‌توانید اقدامات زیر را انجام دهید:

۱. **تنظیم پارامترها**: بهینه‌سازی پارامترهای مدل و شبکه عصبی می‌تواند بهبود عملکرد را به همراه داشته باشد. از ابزارهایی مانند جستجوی هیپرپارامتر، شبکه‌های عصبی با پیچیدگی مناسب، و تنظیم مناسب تعداد لایه‌ها و نورون‌ها بهره ببرید.

۲. **استفاده از لایه‌های با پیچیدگی بالا**: افزایش پیچیدگی مدل می‌تواند بهبود عملکرد آن را داشته باشد، اما باید از اورفیتینگ overfitting جلوگیری کرد.

۳. **تعویض تابع اتلاف**: استفاده از توابع اتلاف مختلف و انتخاب یک تابع مناسب با توجه به مسئله ممکن است تأثیر بسزایی داشته باشد.

۴. **افزایش حجم داده آموزشی**: افزایش تعداد نمونه‌های آموزشی می‌تواند به مدل کمک کند تا الگوهای بیشتری را فراگیری کند و به دقت بیشتری دست یابد.

۵. **کاهش ابعاد ویژگی‌ها**: اگر تعداد ویژگی‌ها زیاد است، امکان دارد با کاهش ابعاد dimensionality reduction یا حتی انتخاب ویژگی feature selection دقت مدل افزایش یابد.

۶. **نرمال‌سازی و وزن‌دهی**: اطمینان حاصل کنید که داده‌ها نرمال‌سازی شده باشند و وزن‌دهی به مناسبی انجام شود.

۷. **آگمنتیشن داده**: با اعمال تغییرات جزئی یا ایجاد داده‌های مصنوعی، می‌توانید تنوع بیشتری به داده‌های آموزشی اضافه کنید.

۸. **رگولاریزاسیون**: استفاده از تکنیک‌های رگولاریزاسیون مانند dropout یا L1/L2 regularization می‌تواند از اورفیتینگ جلوگیری کرده و عملکرد مدل را بهبود بخشد.

بخش۱: سوالات تحلیلی

سوال پنجم :

۱-مجموعه دادۀ Iris را فراخوانی کنید و روش های تحلیل داده ای که آموخته اید را روی آن به کار ببنیدید. داده ها را با نسبتی دلخواه و مناسب به مجموعه های آموزش و ارزیابی تقسیم کنید.

۲- با استفاده از روش های آمادۀ پایتون، سه مدل بر مبنای رگرسیون لجستیگ، MLPو شبکه های عصبی پایه شعاعی RBF را تعریف کرده و روی داده ها آموزش دهید. نتایج روی داده های ارزیابی را حداقل با چهار شاخص و ماتریس درهم ریختگی نشان داده و تحلیل کنید. در انتخاب فراپارامترها آزاد هستید؛ اما لازم است که نتایج را به صورت کامل مقایسه و تحلیل کنید. به دانشجویانی که این سوال را بدون استفاده از کتابخانه ها و مدل های آمادۀ پایتونی انجام دهند، تا ۲۰ درصد نمرۀ امتیازی تعلق خواهد گرفت.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix
from sklearn.cluster import KMeans
import numpy as np
```

این کد از کتابخانه‌های scikit-learn استفاده می‌کند تا یک مجموعه داده Iris را بخواند و سپس با استفاده از مدل‌های مختلف، اعم از رگرسیون لجستیک Logistic Regression شبکه عصبی MLPClassifier و کاهش بعد با الگوریتم خوشه‌بندی KMeans، مسائل مختلف را حل کند. سپس معیارهای ارزیابی مانند دقت accuracy دقت پیش‌بینی precision حساسیت recall اسکور F1 و ماتریس اشتباهات confusion matrix برای ارزیابی عملکرد مدل‌ها استفاده شده‌اند. داده‌ها به دو بخش آموزش و آزمون تقسیم شده و سپس مدل‌ها روی داده‌های آموزش آموزش داده شده و عملکرد آنها روی داده‌های آزمون ارزیابی شده است.

```
iris = load_iris()
data = iris.data
target = iris.target
feature_names = iris.feature_names
```

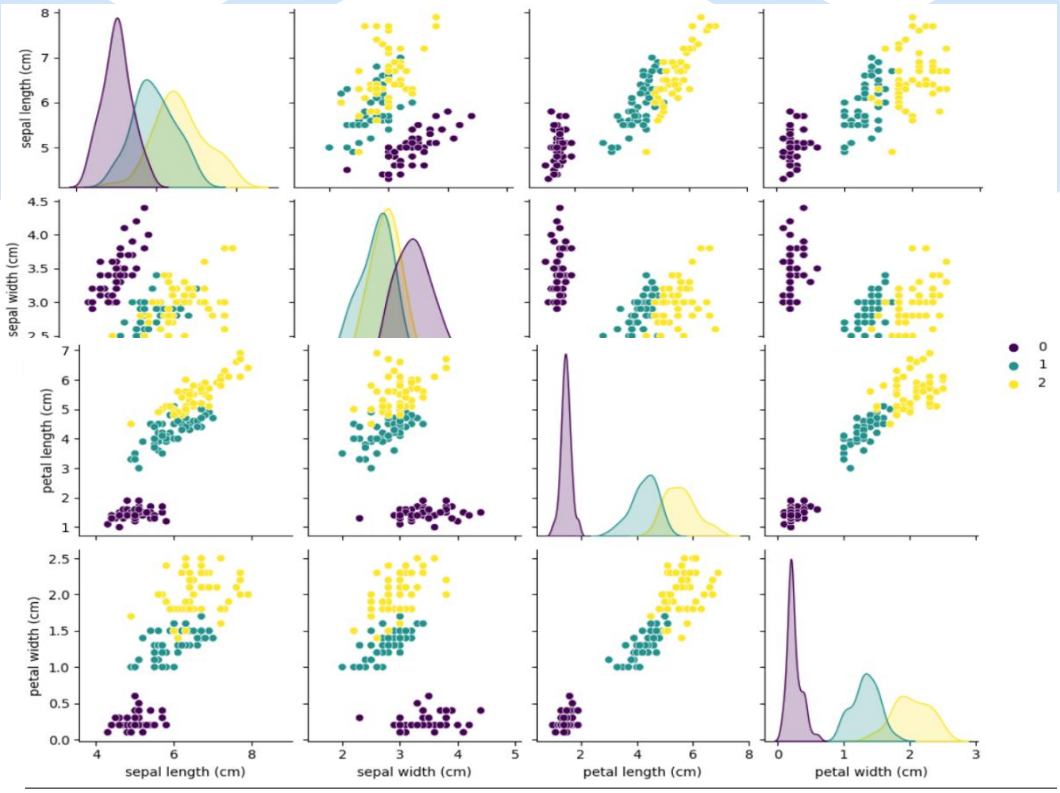
این قسمت از کد از دیتاست Iris استفاده می‌کند که یک مجموعه داده معروف در زمینه یادگیری ماشین است. داده‌های ویژگی features در متغیر data و برچسب‌های کلاس target در متغیر target قرار دارند. علاوه بر این، نام ویژگی‌ها در feature_names ذخیره شده است.

```
iris_df = pd.DataFrame(data, columns=feature_names)
iris_df['target'] = target
```

این بخش از کد از کتابخانه pandas برای ساخت یک DataFrame از داده‌های Iris استفاده می‌کند. ابتدا داده‌های ویژگی‌ها features به عنوان ستون‌های DataFrame با نام‌های مشخص شده در feature_names افزوده شده‌اند. سپس ستون جدیدی به نام target با افزودن مقادیر کلاس‌ها target به DataFrame ایجاد شده است. در نهایت، DataFrame نهایی به نام iris_df ذخیره شده است.

```
# Explore the dataset
sns.pairplot(iris_df, hue='target', palette='viridis')
plt.show()
```

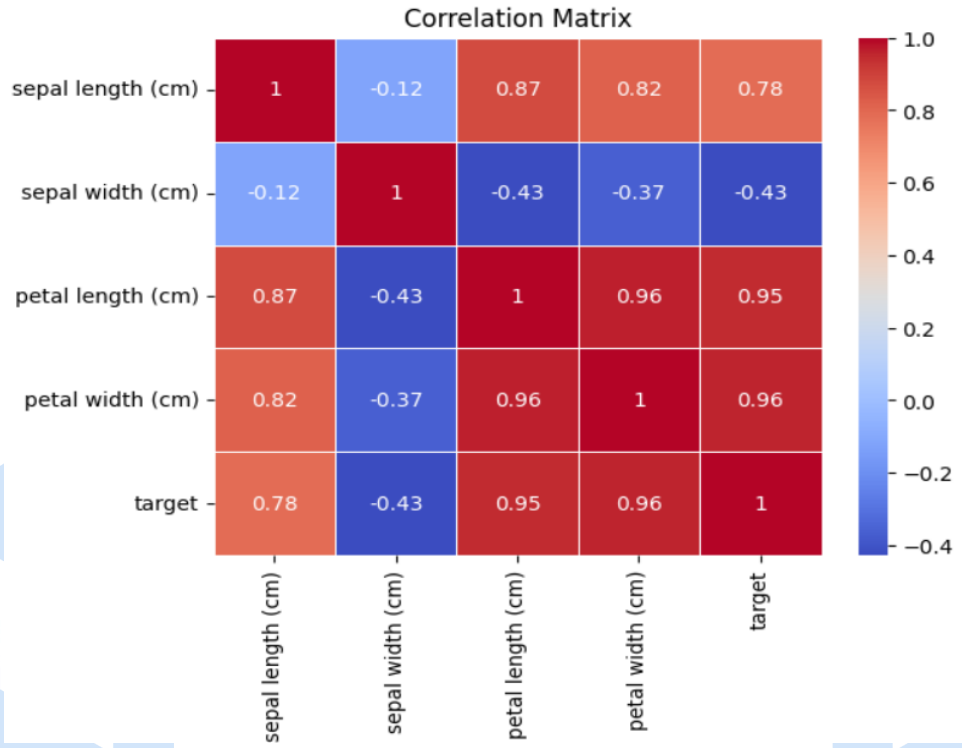
این بخش از کد از کتابخانه seaborn برای تصویرسازی اطلاعات داده‌های Iris با استفاده از نمودارهای جفت Pair Plot استفاده می‌کند.




```
correlation_matrix = iris_df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
linewidths=0.5)
```

```
plt.title('Correlation Matrix')
plt.show()
```

این بخش از کد یک نمودار حرارتی از ماتریس همبستگی بین ویژگی‌های داده‌های Iris ایجاد می‌کند. این نمودار حرارتی از تفاوت همبستگی بین هر دو ویژگی استفاده می‌کند و مقادیر همبستگی در هر خانه نمایش داده می‌شوند.



حال به سراغ تقسیم بندی داده ها به train و test می رویم :

```
# Split the data into training and evaluation sets
X_train, X_test, y_train, y_test = train_test_split(data, target,
test_size=0.2, random_state=83)
```

این بخش از کد داده‌ها را به دو مجموعه آموزش و ارزیابی تقسیم می‌کند. train_test_split از scikit-learn برای این کار استفاده شده است. داده‌های ویژگی data و برچسب‌های کلاس target به ترتیب به متغیرهای X_train و y_train برای مجموعه آموزش و X_test و y_test برای مجموعه ارزیابی اختصاص می‌یابند. میزان اختصاص داده به مجموعه آزمون با test_size=0.2 مشخص شده است و random_state برای اعمال تصادف یکسان در تقسیم داده‌ها به کار می‌رود.

```
# Add bias term to the features
X_train = np.hstack((X_train, np.ones((X_train.shape[0], 1))))  # Adding
bias term
```

```
X_eval = np.hstack((X_eval, np.ones((X_eval.shape[0], 1))))  # Adding bias
term

# Logistic Regression
class LogisticRegression:
    def __init__(self, learning_rate=0.01, epochs=1000):
        self.learning_rate = learning_rate
        self.epochs = epochs

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

    def train(self, X, y):
        self.weights = np.zeros(X.shape[1])

        for epoch in range(self.epochs):
            z = np.dot(X, self.weights)
            predictions = self.sigmoid(z)
            error = y - predictions

            gradient = np.dot(X.T, error)
            self.weights += self.learning_rate * gradient

    def predict(self, X):
        z = np.dot(X, self.weights)
        predictions = self.sigmoid(z)
        return np.round(predictions)

# Multi-Layer Perceptron (MLP)
class MLP:
    def __init__(self, input_size, hidden_size, output_size,
learning_rate=0.01, epochs=1000):
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.learning_rate = learning_rate
        self.epochs = epochs

        self.weights_input_hidden = np.random.rand(self.input_size,
self.hidden_size)
        self.weights_hidden_output = np.random.rand(self.hidden_size,
self.output_size)

    def sigmoid(self, x):
        return 1 / (1 + np.exp(-x))
```

```
    def softmax(self, x):
        exp_values = np.exp(x - np.max(x, axis=1, keepdims=True))
        return exp_values / np.sum(exp_values, axis=1, keepdims=True)

    def train(self, X, y):
        for epoch in range(self.epochs):
            # Forward pass
            hidden_layer_input = np.dot(X, self.weights_input_hidden)
            hidden_layer_output = self.sigmoid(hidden_layer_input)

            output_layer_input = np.dot(hidden_layer_output,
self.weights_hidden_output)
            output_layer_output = self.softmax(output_layer_input)

            # Backward pass
            output_error = y - output_layer_output
            output_delta = output_error

            hidden_layer_error =
output_delta.dot(self.weights_hidden_output.T)
            hidden_layer_delta = hidden_layer_error * (hidden_layer_output
* (1 - hidden_layer_output))

            # Update weights
            self.weights_hidden_output += self.learning_rate *
hidden_layer_output.T.dot(output_delta)
            self.weights_input_hidden += self.learning_rate *
X.T.dot(hidden_layer_delta)

    def predict(self, X):
        hidden_layer_input = np.dot(X, self.weights_input_hidden)
        hidden_layer_output = self.sigmoid(hidden_layer_input)

        output_layer_input = np.dot(hidden_layer_output,
self.weights_hidden_output)
        output_layer_output = self.softmax(output_layer_input)

        return np.argmax(output_layer_output, axis=1)

# Radial Basis Function (RBF) Neural Network
class RBFNN:
    def __init__(self, num_centers, learning_rate=0.01, epochs=1000):
        self.num_centers = num_centers
        self.learning_rate = learning_rate
```

```
        self.epochs = epochs

    def gaussian_rbf(self, x, c, sigma):
        return np.exp(-np.linalg.norm(x - c) / (2 * sigma**2))

    def train(self, X, y):
        self.centers = X[np.random.choice(X.shape[0], self.num_centers,
replace=False)]
        self.sigma = np.std(X)

        self.weights = np.random.rand(self.num_centers)

        for epoch in range(self.epochs):
            for i in range(X.shape[0]):
                phi = np.array([self.gaussian_rbf(X[i], c, self.sigma) for
c in self.centers])
                prediction = np.dot(phi, self.weights)
                error = y[i] - prediction

                # Update weights
                self.weights += self.learning_rate * error * phi

    def predict(self, X):
        predictions = []
        for i in range(X.shape[0]):
            phi = np.array([self.gaussian_rbf(X[i], c, self.sigma) for c
in self.centers])
            prediction = np.dot(phi, self.weights)
            predictions.append(prediction)

        return np.round(predictions)

# Convert target labels to one-hot encoding for MLP
def one_hot_encode(labels, num_classes):
    one_hot = np.zeros((len(labels), num_classes))
    one_hot[np.arange(len(labels)), labels] = 1
    return one_hot

# Convert target labels to integers for RBFNN
def convert_to_integer_labels(labels):
    label_map = {label: i for i, label in enumerate(np.unique(labels))}
    return np.array([label_map[label] for label in labels])

# One-hot encode target labels for MLP
y_train_one_hot = one_hot_encode(y_train, len(np.unique(y_train)))
```

```
y_eval_one_hot = one_hot_encode(y_eval, len(np.unique(y_eval)))

# Logistic Regression
lr_model = LogisticRegression()
lr_model.train(X_train, y_train)
lr_pred = lr_model.predict(X_eval)

# MLP
mlp_model = MLP(input_size=X_train.shape[1], hidden_size=10,
output_size=len(np.unique(y_train)), epochs=1000)
mlp_model.train(X_train, y_train_one_hot)
mlp_pred = mlp_model.predict(X_eval)

# RBF Neural Network
num_rbf_centers = 10
rbf_model = RBFNN(num_centers=num_rbf_centers, epochs=1000)
rbf_model.train(X_train, convert_to_integer_labels(y_train))
rbf_pred = rbf_model.predict(X_eval)

# Evaluate Logistic Regression
lr_accuracy = accuracy_score(y_eval, lr_pred)
lr_precision = precision_score(y_eval, lr_pred, average='weighted')
lr_recall = recall_score(y_eval, lr_pred, average='weighted')
lr_f1 = f1_score(y_eval, lr_pred, average='weighted')

print("Logistic Regression Metrics:")
print(f"Accuracy: {lr_accuracy:.4f}")
print(f"Precision: {lr_precision:.4f}")
print(f"Recall: {lr_recall:.4f}")
print(f"F1 Score: {lr_f1:.4f}")

# Confusion matrix for Logistic Regression
lr_cm = confusion_matrix(y_eval, lr_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(lr_cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix - Logistic Regression")
plt.show()

# Evaluate MLP
mlp_accuracy = accuracy_score(y_eval, mlp_pred)
mlp_precision = precision_score(y_eval, mlp_pred, average='weighted')
mlp_recall = recall_score(y_eval, mlp_pred, average='weighted')
mlp_f1 = f1_score(y_eval, mlp_pred, average='weighted')

print("\nMLP Metrics:")
```

```
print(f"Accuracy: {mlp_accuracy:.4f}")
print(f"Precision: {mlp_precision:.4f}")
print(f"Recall: {mlp_recall:.4f}")
print(f"F1 Score: {mlp_f1:.4f}")

# Confusion matrix for MLP
mlp_cm = confusion_matrix(y_eval, mlp_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(mlp_cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix - MLP")
plt.show()

# Evaluate RBF Neural Network
rbf_accuracy = accuracy_score(y_eval, rbf_pred)
rbf_precision = precision_score(y_eval, rbf_pred, average='weighted')
rbf_recall = recall_score(y_eval, rbf_pred, average='weighted')
rbf_f1 = f1_score(y_eval, rbf_pred, average='weighted')

print("\nRBF Neural Network Metrics:")
print(f"Accuracy: {rbf_accuracy:.4f}")
print(f"Precision: {rbf_precision:.4f}")
print(f"Recall: {rbf_recall:.4f}")
print(f"F1 Score: {rbf_f1:.4f}")

# Confusion matrix for RBF Neural Network
rbf_cm = confusion_matrix(y_eval, rbf_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(rbf_cm, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.title("Confusion Matrix - RBF Neural Network")
plt.show()
```

این بخش از کد مدل‌های مختلف را برای آموزش و ارزیابی روی داده‌های Iris اجرا می‌کند:

۱. Logistic Regression

- یک مدل رگرسیون لجستیک ایجاد می‌شود و روی داده‌های آموزش X_train , y_train آموزش داده می‌شود.

- پیش‌بینی‌های مدل بر روی داده‌های ارزیابی X_eval انجام می‌شود.

- معیارهای ارزیابی مانند دقت Accuracy دقت پیش‌بینی Precision حساسیت Recall و اسکور F1 محاسبه می‌شود.

- یک ماتریس اشتباهات برای ارزیابی دقیق‌تر نمایش داده می‌شود.

۲. Multi-Layer Perceptron (MLP)

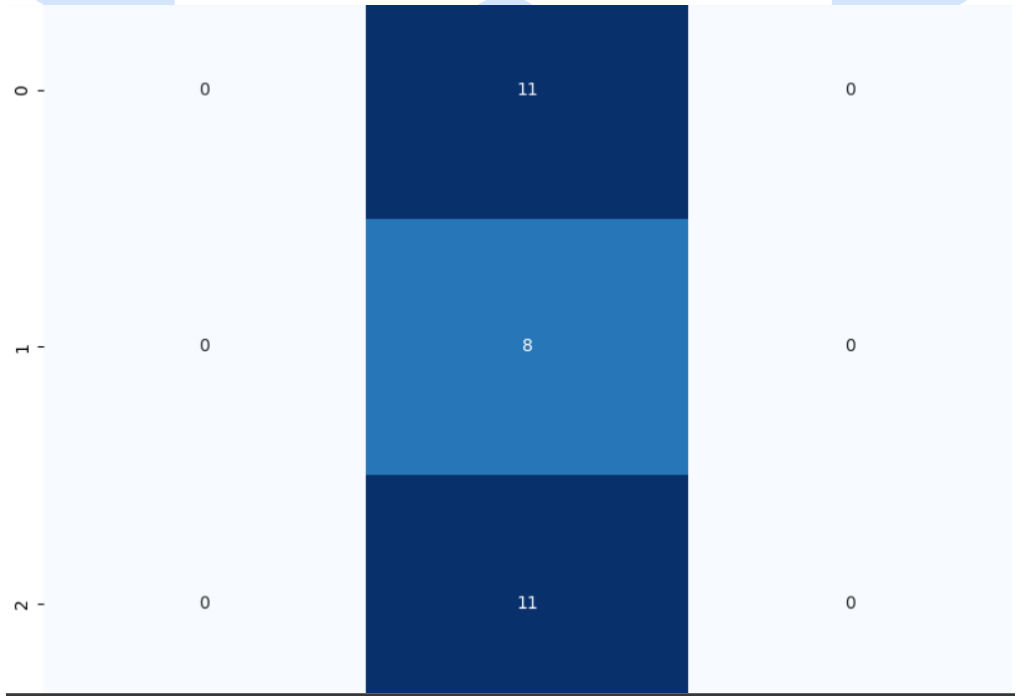
- یک مدل شبکه عصبی چندلایه ((MLPبا تعداد لایه‌ها و اندازه‌های مخفی مشخص شده ایجاد می‌شود.
- مدل روی داده‌های آموزش آموزش داده می‌شود.
- پیش‌بینی‌های مدل بر روی داده‌های ارزیابی انجام می‌شود.
- معیارهای ارزیابی محاسبه می‌شود و ماتریس اشتباهات نمایش داده می‌شود.

۳. Radial Basis Function (RBF) Neural Network

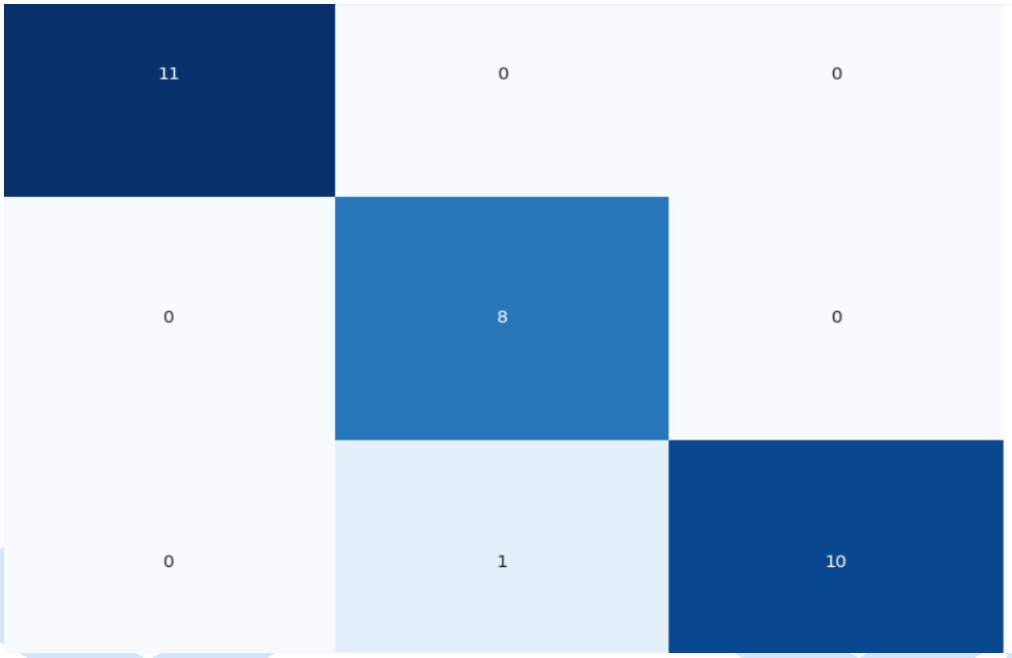
- یک مدل شبکه عصبی با توابع پایه شعاعی ((RBFایجاد می‌شود.
- مدل روی داده‌های آموزش آموزش داده می‌شود.
- پیش‌بینی‌های مدل بر روی داده‌های ارزیابی انجام می‌شود.
- معیارهای ارزیابی محاسبه می‌شود و ماتریس اشتباهات نمایش داده می‌شود.

برای هر مدل، دقت، دقت پیش‌بینی، حساسیت، و اسکور F1 ارائه شده و ماتریس اشتباهات با استفاده از نمودار حرارتی نمایش داده می‌شود.

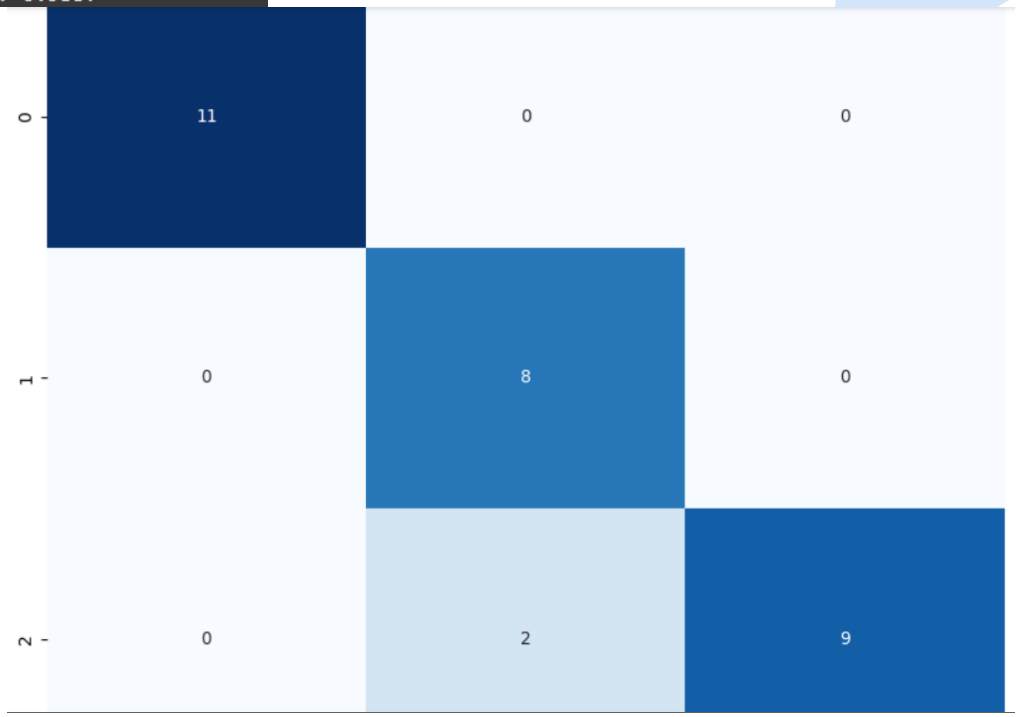
```
Logistic Regression Metrics:
Accuracy: 0.2667
Precision: 0.0711
Recall: 0.2667
F1 Score: 0.1123
```



```
MLP Metrics:
Accuracy: 0.9667
Precision: 0.9704
Recall: 0.9667
F1 Score: 0.9669
```




```
RBF Neural Network Metrics:
Accuracy: 0.9333
Precision: 0.9467
Recall: 0.9333
F1 Score: 0.9337
```



به دلیل دقت پایین روش رگرسیون لجستیگ می توانیم برای این مدل بجای تعریف کلاس از کتابخانه های آماده پایتونی نیز استفاده کنیم بنابراین به صورت زیر عمل می کنیم :

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Define the logistic regression model
logistic_regression_model = LogisticRegression()

# Train the model on the training data
logistic_regression_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = logistic_regression_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')
```

این کد از داده‌های iris استفاده می‌کند، یک مدل رگرسیون لجستیک را با داده‌های آموزشی آموزش داده و سپس بر روی داده‌های آزمایشی ارزیابی کرده و دقت مدل را چاپ می‌کند.

نتیجه دقت این مدل :

```
Accuracy: 0.90
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

مراجع

https://drive.google.com/drive/folders/1HvH5E9cI7d1Z8aYc26IhEJiux-HCNUQQ?usp=sharing