((به نام خداوند بخشنده و مهربان))

نام و نام خانوادگی: ساناز گرامی

شماره دانشجویی: 9929873

درس: مبانی سیستم‌های هوشمند

استاد: دکتر مهدی علیاری

مینی‌پروژه2

سوال اول

مجموعه داده مربوط به این سوال را از طریق این پیوند دانلود کنید و در مراحل بعدی از آن استفاده کنید. ستون اول و دوم فایل CSV مربوط به این مجموعه داده، مربوط به ویژگی‌ها و ستون سوم آن مربوط به کلاس هر داده است.

1. داده‌ها را با نسبت 80 به 20 درصد به دو قسمت آموزش و آزمون تقسیم کنید. سپس با استفاده از قاعده پرسپترون، یک نورون روی داده‌های مجموعه آموزشی، آموزش دهید (آستانه را دلخواه در نظر بگیرید).

ابتدا مطابق شکل1 فایل Perceptron.csv (که در گوگل درایو آپلود شده است) را در محیط گوگل‌کولب بارگذاری می‌کنیم. سپس دو ستون اول فایل را با دستور X = df.values[:, : -1] در بخش ویژگی‌ها و ستون آخر را با دستور y = df.values[:, -1] در بخش تارگت قرار می‌دهیم. برای اینکه y از حالت برداری، به حالت ماتریسی با یک ستون تبدیل شود، از دستور y.reshape((-1, 1)) استفاده می‌کنیم. در قدم بعد با فراخوانی تابع train_test_split از کتابخانه sklearn.model_selection، داده‌ها را به دو بخش trian و test تقسیم می‌کنیم (شکل2).

```
#1.1

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

!pip install gdown
!gdown 1rQX8YUre3q2qXS3El3w2aNFGThCPiRgA

df = pd.read_csv("Perceptron.csv")
```

شکل1: بارگذاری فایل Perceptron.csv

```
from sklearn.model_selection import train_test_split
X = df.values[:, :-1]
y = df.values[:, -1]
y = y.reshape((-1, 1))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

شکل2: قرار دادن داده در دو بخش train و test

برای آموزش یک نورون به روش پرسپترون، یک کلاس با نام Neuron با پارامترهای in_features (تعداد ویژگی‌ها)، activation_function (تابع فعال‌سازی)، loss_fn (تابع اتلاف)، n_itter (تعداد دوره‌های آموزش) و eta (تعداد گام آموزش) تعریف می‌کنیم. سپس باید هر کدام از این پارامترها را در بخش def __init__ رجیستر کنیم. در بخش def __init__ علاوه بر رجیستر کردن پارامترهای تعریف شده، متغیرهای w (وزن) و b (بایاس) را نیز تعریف کرده و آنها را به صورت رندوم مقداردهی اولیه می‌کنیم. سپس یک لیست loss_hist = [] برای ذخیره‌سازی مقدار تابع اتلاف در هر مرحله و متغیرهای w_grad و b_grad (موردنیاز در بخش گرادیان) را نیز ذخیره می‌کنیم (شکل3).

```
class Neuron:

  def __init__(self, in_features, activation_function = None, loss_fn = None, n_itter = 100, eta = 0.1):
    self.in_features = in_features
    self.activation_function = activation_function
    self.loss_fn = loss_fn
    self.n_itter = n_itter
    self.eta = eta
    self.w = np.random.randn(in_features, 1)
    self.b = np.random.randn()
    self.loss_hist = []
    self.w_grad, self.b_grad = None, None
```

شکل3: تعریف کلاس نورون و پارامترهای آن

```
def predict(self, x):
  y_hat = x @ self.w + self.b
  y_hat = y_hat if self.activation_function is None else self.activation_function(y_hat)
  return y_hat
```

شکل4: تعریف تابع predict

در قدم بعد شروع به تعریف توابع موردنیاز در داخل کلاس نورون می‌کنیم.

تابع predict:

همانطور که از اسم تابع مشخص است، وظیفه این تابع دریافت مقادیر X_test و پیش‌بینی تارگت آن (y_hat) است. پس این تابع آرگومان x را به عنوان ورودی می‌پذیرد. براساس قاعده پرسپترون، برای بدست آوردن y_hat، ماتریس x باید در ماتریس w ضرب داخلی شده و بعد با مقدار بایاس b جمع شود؛ سپس نتیجه از یک تابع فعال‌سازی (activation_function) عبور داده شده و مقدار y_hat برگردانده می‌شود. اگر مقدار

activation_function برابر None بود، رگرسیون به صورت خطی است و اگر برای activation_function تابع خاصی (مانند sigmoid) معرفی شده بود، y_hat محاسبه شده از تابع عبور کرده و مقدار جدید آن برگردانده می‌شود. (شکل4).

تابع fit:

این تابع برای آموزش داده‌های تست استفاده می‌شود؛ بنابراین دو آرگومان x و y را به عنوان X_train و y_train می‌پذیرد. در این تابع از یک حلقه for استفاده شده است که به تعداد دوره‌های آموزش (n_itter)، داده‌ها را آموزش می‌دهد. سپس با استفاده از دستور loss = self.loss_fn(y, y_hat) مقدار محاسبه شده y_hat و مقدار تارگت اصلی y را به تابع loss_fn داده و مقدار تابع اتلاف را محاسبه می‌کند. سپس با استفاده از دستور self.loss_hist.append(loss)، مقدار تابع اتلاف را در لیست loss_hist می‌ریزد. در مرحله بعد به منظور آپدیت مقدار w، از تکنیک گرادیان نزولی استفاده می‌شود (شکل5).

تابع gradient:

با توجه به مفهوم گرادیان، ماتریس ترانهاده داده‌های آموزش X_train باید در اختلاف y_hat و y_train ضرب داخلی شده و سپس نتیجه بر طول y_train تقسیم شود تا مقدار w_grad حاصل شود. همچنین برای بدست آوردن مقدار b_grad از اختلاف میان y و y_hat میانگین گرفته می‌شود. لذا این تابع سه آرگومان x، y و y_hat را به ترتیب به عنوان X_train، y_train و y_hat می‌پذیرد. شکل6 پیاده‌سازی تابع gradient را نشان می‌دهد.

تابع gradient_decent:

این تابع با گام آموزشی eta که در کلاس نورون به صورت پیش‌فرض 0.1 در نظر گرفته شده، مقادیر w و b را آپدیت می‌کند (شکل7).

تابع parameters:

این تابع با هربار فراخوانی شدن، مقدار پارامترهای w و b را در قالب دیکشنری نشان می‌دهد (شکل8).

```python
def fit(self, x, y):
  for i in range(self.n_itter):
    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_decent
```

شکل5: تعریف تابع fit

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

شکل6: تعریف تابع gradient

```python
def gradient_decent(self, ):
  self.w -= self.eta * self.w_grad
  self.b -= self.eta * self.b_grad
```

شکل7: تعریف تابع gradient_decent

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

شکل8: تعریف تابع parameters

```python
neuron1 = Neuron(in_features = 2, activation_function = tanh, loss_fn = bce, n_itter = 10, eta = 0.1)
neuron1.fit(X_train, y_train)
```

شکل10: آموزش نورون

```python
#activation_function

def relu(x):
  return np.maximum(0, x)

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

import math
def tanh(x):
  return np.tanh(x)

#loss_fn

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)
```

شکل9: تعریف توابع activation_function و loss_fn

تعریف توابع موردنیاز برای کلاس نورون به پایان رسید. اما همانطور که پیش‌تر اشاره شد، ما به دو تابع activation_function و loss_fn نیز نیاز داریم؛ پس کمی بالاتر از محل تعریف کلاس، سه مدل تابع (relu, sigmoid, tanh) برای activation_function و دو مدل تابع (bce, mse) برای loss_fn تعریف می‌کنیم تا در پروسه کلاس‌بندی بتوان از آنها استفاده کرد (شکل9).

یک نورون به نام neuron1 از کلاس Neuron تعریف کرده و پارامترهای آن را مقداردهی می‌کنیم. سپس با استفاده از دستور neuron1.fit، نورون را روی داده‌های آموزشی، آموزش می‌دهیم (شکل10).

2. نتیجه را روی داده‌های مجموعه آزمون نشان دهید و دقت را بدست آورید. برای داده‌های تست دو خط موازی جداکننده بدست آمده از قاعده پرسپترون را نمایش دهید و داده‌های تفکیک شده دو کلاس را با رنگ مجزا در Scatter Plot مشخص کنید.

برای نمایش دقت نیاز به تعریف تابع Accuracy است که در شکل11 پیاده‌سازی آن نشان داده شده است. این تابع آرگومان‌های y و y_hat را به عنوان ورودی می‌پذیرد و مقدار آستانه آن t = 0 در نظر گرفته شده است؛ زیرا مقدار تارگت‌های داده‌ها 1 یا 1- است.

با استفاده از دستور neuron1.predict(X_test)، تارگت داده‌های تست پیش‌بینی شده و با استفاده از دستور accuracy(y_test, y) دقت ارزیابی نشان داده می‌شود(شکل‌های12 و 13).

```python
#accuracy

def accuracy(y, y_hat, t = 0):
  y_hat = np.where(y_hat < t, -1, 1)
  acc = np.sum(y == y_hat) / len(y)
  return acc
```

شکل11: تعریف تابع accuracy

```python
y_hat = neuron1.predict(X_test)
acc = accuracy(y_test, y_hat)
print("The accuracy is: " + str(acc))
```

شکل12: دادن داده‌های تست به نورون آموزش داده شده

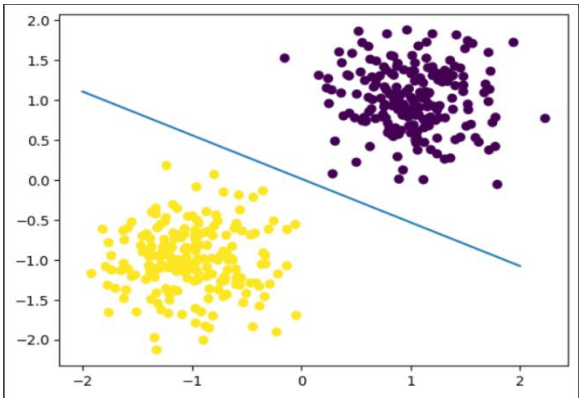
```
The accuracy is: 0.9625
```

شکل13: نمایش دقت نورون

به منظور رسم نواحی تصمیم‌گیری، می‌دانیم که قاعده پرسپترون به صورت $w_1x_1 + w_2x_2 + b = 0$ یا $w_1x + w_2y + b = 0$ است که می‌توان y را برحسب x به صورت $y = -\frac{w_1}{w_2}x - \frac{b}{w_2}$ نوشت. سپس با استفاده از دستور x = np.linspace(-2, 2, 30)، مقادیری در آرایه x قرار داده، آرایه y را ساخته و آن را نمایش می‌دهیم (شکل14 و 15).

```python
x = np.linspace(-2, 2, 30)
plt.scatter(X[:, 0], X[:, 1], c = y)
y1 = (-(neuron1.b / neuron1.w[1]) / (neuron1.b / neuron1.w[0]))*x + (-neuron1.b / neuron1.w[1])
plt.plot(x, y1)
```

شکل15: کدنویسی برای رسم نواحی تصمیم‌گیری



شکل14: رسم ناحیه تصمیم‌گیری

3. قسمت‌های 1 و 2 را با آستانه دیگری انجام داده و نتایج را با حالت قبل مقایسه کنید. تحلیل کنید که انتخاب آستانه پرسپترون چه تاثیری روی نتایج طبقه‌بندی دارد. ضمن پیاده‌سازی تحلیل کنید که حذف بایاس چه تاثیری بر نتایج خواهد گذاشت.

اینبار آستانه t را از 0 به 0.8- تغییر داده (شکل16) و دقت(شکل17) و نواحی تصمیم‌گیری(شکل18) را نمایش می‌دهیم. همانطور که مشاهده می‌شود، مقدار آستانه تاثیر بسزایی در دقت طبقه‌بندی دارد. لذا باید قبل از نوشتن تابع accuracy، به بازه تارگت توجه شود و مقدار آستانه تا حد امکان در وسط بازه قرار بگیرد. زیرا در غیر این صورت، اگر مقدار آستانه به مقدار یک تارگت نزدیک‌تر باشد، ممکن است کلاس‌بندی اشتباه صورت بگیرد و دقت کاهش یابد. مانند این سوال که دقت از مقدار 0.9625 به مقدار 0.65 رسیده است.

برای یافتن تاثیر بایاس، تمام بخش‌هایی از کد که از b استفاده شده است را کامنت کرده و کد را یکبار دیگر اجرا می‌کنیم.

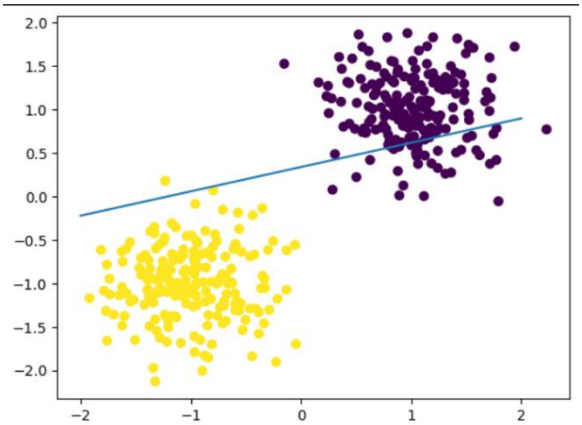
بایاس در واقع مقداری ثابت است که به ورودی‌های پرسپترون اضافه می‌شود. این ثوابت به عنوان یک انتقال سراسری برای تصمیم‌گیری و درک الگوها با اهمیت است. حذف بایاس می‌تواند منجر به یک تراز جدید افقی برای خط تصمیم در پرسپترون شود و باعث تغییر و شیفت کردن خط تصمیم می‌شود. این امر می‌تواند باعث ضرر در عملکرد طبقه‌بندی شود. همچنین بایاس به پرسپترون اجازه می‌دهد مدلسازی الگوها و انحراف‌هایی که در داده‌ها وجود دارند را بیاموزد. حذف بایاس می‌تواند منجر به ناتوانی در مدل‌سازی این الگوها و به عبارت دیگر عدم توانایی در تطبیق با داده‌ها شود.

```
#accuracy

def accuracy(y, y_hat, t = -0.8):
  y_hat = np.where(y_hat < t, -1, 1)
  acc = np.sum(y == y_hat) / len(y)
  return acc
```

شکل16: تغییر مقدار آستانه t در تابع accuracy

```
The accuracy is: 0.65
```

شکل17: نمایش دقت طبقه‌بندی با t = -0.8



شکل18: نواحی تصمیم‌گیری به ازای t = -0.8

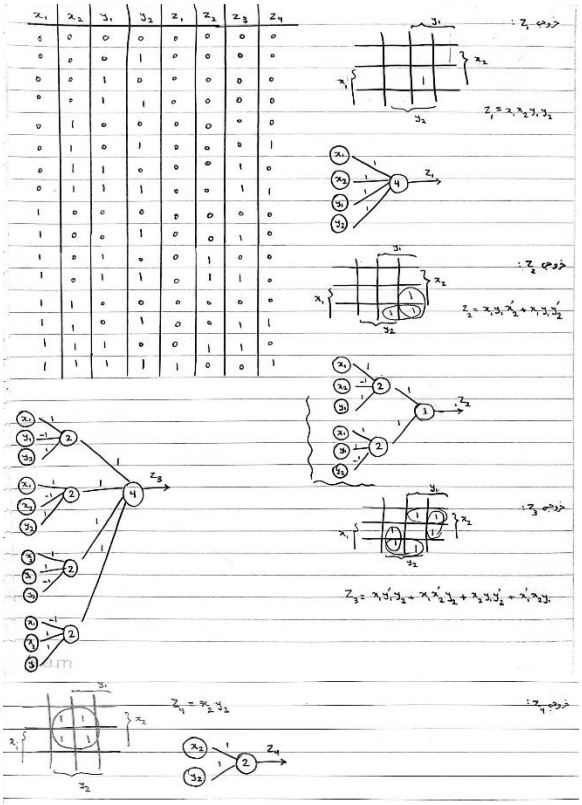
سوال دوم

1. به کمک نورون McCulloch-Pitts توسعه یافته، یک ضرب‌کننده باینری بسازید که دو ورودی دوبیتی را گرفته و آنها را ضرب می‌کند. برای این‌کار به دو ورودی دوبیتی (در واقع چهار نورون برای همه ورودی‌ها) نیاز داریم. همچنین چهار بیت خروجی (چهار نورون) موردنیاز است. توجه شود که تمامی نورون‌های ورودی و خروجی باینری هستند (صفر و یک). ترتیب زمانی انجام عملیات در این سوال مهم نیست؛ بنابراین، نیازی به در نظر گرفتن تأخیر برای انجام عملیات نیست. ضمن رسم جدول ورودی-خروجی، شبکه هر خروجی را به همراه توضیحات مختصری رسم کنید (نیازی به کدنویسی در این قسمت نیست). دقت داشته باشید که شبکه‌ای که برای هر خروجی رسم می‌کنید تا حد ممکن دارای کمترین تعداد نورون و کمترین آستانه باشد (تعداد نورون کمتر دارای اهمیت بالاتری نسبت به آستانه کوچکتر است). همچنین توجه کنید که تمام شبکه برای یک خروجی دارای آستانه یکسان باشد.

ابتدا جدول درستی را برای ورودی‌های x1، x2، y1 و y2 که به ترتیب دو بیت ورودی اول و دو بیت ورودی دوم هستند را رسم می‌کنیم. پس از رسم جدول و بدست آمدن چهار خروجی z1، z2، z3 و z4، برای بدست آوردن شبکه هر خروجی از جدول کارنو کمک گرفته و با توجه به نتایج بدست آمده از جدول کارنو، شبکه خروجی را به همراه وزن‌ها و مقدار آستانه رسم می‌کنیم. شکل‌های19 شبکه‌های خروجی بدست آمده را نشان می‌دهد.

2. با استفاده از زبان پایتون شبکه‌های طراحی شده در قسمت1 را پیاده‌سازی کرده و تمامی حالات ممکن را به صورت مناسبی نشان دهید.

در پایتون کلاسی به نام McColluch_Pitts تعریف می‌کنیم که دو آرگومان وزن و مقدار آستانه را می‌پذیرد و با استفاده از تابع model که در آن تعریف شده است، بررسی می‌کند که آیا حاصل‌ضرب ورودی در ماتریس وزن بزرگتر از مقدار آستانه است یا خیر. اگر این حاصل‌ضرب بزرگتر بود، مقدار 1 و در غیر این صورت مقدار 0 را برمی‌گرداند. سپس خارج از کلاس، یک تابع به نام multiplier تعریف می‌شود که برای هر خروجی، نورون‌های مجزایی ایجاد می‌کند تا خروجی موردنظر بدست آید. کدنویسی این بخش در شکل‌های20 و 21 و نتایج در شکل22 نشان داده شده است.



شکل19: بدست آوردن شبکه‌های خروجی

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

class McCulloch_Pitts():

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

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

def multiplier(input):
  neur1 = McCulloch_Pitts([1, 1, 1, 1], 4)
  z1 = neur1.model(np.array([input[0], input[1], input[2], input[3]]))

  neru21 = McCulloch_Pitts([1, -1, 1], 2)
  z21 = neru21.model(np.array([input[0], input[1], input[2]]))
  neru22 = McCulloch_Pitts([1, 1, -1], 2)
  z22 = neru22.model(np.array([input[0], input[2], input[3]]))
  neru2 = McCulloch_Pitts([1, 1], 1)
  z2 = neru2.model(np.array([z21, z22]))

  neru31 = McCulloch_Pitts([1, -1, 1], 2)
  z31 = neru31.model(np.array([input[0], input[2], input[3]]))
  neru32 = McCulloch_Pitts([1, -1, 1], 2)
  z32 = neru32.model(np.array([input[0], input[1], input[3]]))
  neru33 = McCulloch_Pitts([1, 1, -1], 2)
  z33 = neru33.model(np.array([input[1], input[2], input[3]]))
  neru34 = McCulloch_Pitts([-1, 1, 1], 2)
  z34 = neru34.model(np.array([input[0], input[1], input[2]]))
  neru3 = McCulloch_Pitts([1, 1, 1, 1], 1)
  z3 = neru3.model(np.array([z31, z32, z33, z34]))
```

شکل20: ضرب‌کننده باینری با استفاده از McColluch-Pitts

```
  neru4 = McCulloch_Pitts([1, 1], 2)
  z4 = neru4.model(np.array([input[1], input[3]]))

  y = [z1, z2, z3, z4]

  return y


a = [0, 1, 0, 1]
X = list(itertools.product(a, a, a, a))
input = []
for i in range(0, len(X)):
  n = input.count(X[i])
  if n == 0:
    input.append(X[i])

for i in range(0, len(input)):
  z = multiplier(input[i])
  print(str(input[i]) + ': result of the multiplication ' + str(z))
```

شکل21: ضرب‌کننده باینری با استفاده از McColluch-Pitts – ادامه

```
(0, 0, 0, 0): result of the multiplication [0, 0, 0, 0]
(0, 0, 0, 1): result of the multiplication [0, 0, 0, 0]
(0, 0, 1, 0): result of the multiplication [0, 0, 0, 0]
(0, 0, 1, 1): result of the multiplication [0, 0, 0, 0]
(0, 1, 0, 0): result of the multiplication [0, 0, 0, 0]
(0, 1, 0, 1): result of the multiplication [0, 0, 0, 1]
(0, 1, 1, 0): result of the multiplication [0, 0, 1, 0]
(0, 1, 1, 1): result of the multiplication [0, 0, 1, 1]
(1, 0, 0, 0): result of the multiplication [0, 0, 0, 0]
(1, 0, 0, 1): result of the multiplication [0, 0, 1, 0]
(1, 0, 1, 0): result of the multiplication [0, 1, 0, 0]
(1, 0, 1, 1): result of the multiplication [0, 1, 1, 0]
(1, 1, 0, 0): result of the multiplication [0, 0, 0, 0]
(1, 1, 0, 1): result of the multiplication [0, 0, 1, 1]
(1, 1, 1, 0): result of the multiplication [0, 1, 1, 0]
(1, 1, 1, 1): result of the multiplication [1, 0, 0, 1]
```

شکل22: نمایش نتایج ضرب‌کننده باینری

سوال سوم

به این دفترچه کد مراجعه کنید و با اجرای سلول اول، 5 داده تصویری مربوط به حروف الفبای فارسی را دریافت کنید و سپس به سوالات زیر پاسخ دهید. دقت داشته باشید که در هر مرحله ارائه توضیحات متنی و دیداری مناسب لازم است. مثلا می‌توانید ورودی نویزی و خروجی پیش‌بینی شده را در یک تصویر در کنار هم قرار دهید.

1. دو تابع پایتونی در سلول‌های دوم و سوم این دفترچه کد نوشته شده‌اند. اولین تابع تصویر را در ورودی خود دریافت و به صورت نمایش باینری درمی‌آورد و دومین تابع با افزودن نویز به داده‌ها، داده‌های جدید نویزی تولید می‌کند. در مورد نحوه عملکرد هر تابع توضیح دهید. همچنین، می‌توانید این دستورات را به صورتی بهتر و کارآمدتر بازنویسی کنید.

تابع convertImageToBinary:

هدف این تابع، دریافت یک تصویر و نمایش آن به صورت باینری است و یک ورودی path را به عنوان آدرس تصویر می‌پذیرد.

دستور image = Image.open(path)، با دریافت آدرس تصویر، آن را در محیط پایتون باز می‌کند.

دستور draw = ImageDraw.Draw(image)، برای ایجاد یک شی طراحی استفاده می‌شود که این امکان را می‌دهد که با استفاده از کتابخانه PIL روی یک تصویر بکشیم.

دستورهای width = image.size[0] و height = image.size[1] برای یافتن ابعاد (عرض و طول) یک تصویر به کار می‌روند.

دستور pix = image.load() برای بارگذاری داده‌های پیکسلی یک تصویر در یک شی دسترسی پیکسلی دو بعدی استفاده می‌شود. با فراخوانی image.load()، یک شی دسترسی به پیکسل برگردانده می‌شود که به متغیر pix اختصاص داده می‌شود. این شی دسترسی به پیکسل راه مناسبی را برای بازیابی و تغییر مقادیر پیکسل‌های فردی در تصویر فراهم می‌کند. هنگامی که داده‌های پیکسل در pix بارگذاری شوند، می‌توان با استفاده از نمایه‌سازی دوبعدی به پیکسل‌های جداگانه دسترسی پیدا کرده و آن‌ها را تغییر داد. به عنوان مثال، pix[x, y] این امکان را می‌دهد که به مقدار پیکسل در مختصات (x, y) در تصویر دسترسی داشته باشیم.

در داخل حلقه for (که برای هر دو بعد تصویر نوشته شده است) مقادیر RGB هر پیکسل محاسبه شده و با جمع مقادیر آنها، مقدار total_intensity محاسبه می‌شود. سپس این مقدار با یک مقدار معین (که در تابع به صورت (3 * (255 + factor) // 2)) تعریف شده است و در آن factor مقدار آستانه است) مقایسه می‌شود؛ اگر total_intensity بزرگتر بود، تصویر سفید و اگر کوچکتر بود، تصویر سیاه تشخیص داده شده و به ترتیب به آن مقادیر باینری 1- و 1 نسبت داده می‌شود.

تابع generateNoisyImages:

این تابع تصاویر اصلی را دریافت کرده و از روی آنها یک تصویر دارای نویز تولید می‌کند.

تابع getNoisyBinaryImage:

بخشی از دستورهای این تابع مشابه تابع convertImageToBinary است.

در حلقه for با استفاده از تابع rand یک نویز به صورت رندوم تولید شده و سپس این مقدار به هریک از پیکسل‌ها اضافه می‌شود. برای اینکه اطمینان حاصل شود که مقادیر RGB در بازه 0 تا 255 قرار دارند، مقادیر منفی پیکسل‌ها را برابر 0 و مقادیر بزرگتر از 255 پیکسل‌ها را برابر 255 قرار می‌دهیم.

سپس با استفاده از دستور image.save، تصویر دارای نویز ذخیره می‌شود.

---

2. یک شبکه عصبی (همینگ یا هاپفیلد) طراحی کنید که با اعمال ورودی دارای میزان مشخصی نویز برای هر یک از داده‌ها، خروجی متناسب با آن داده نویزی را بیابد. میزان نویز را تا زمانی که شبکه شما موفق عمل کند افزایش دهید و نتایج را مقایسه و تحلیل کنید.

در این بخش از شبکه عصبی همینگ استفاده می‌شود. همچنین برای پیاده‌سازی این شبکه عصبی، نیاز به تعریف تعدادی تابع داریم که به شرح زیر می‌باشند:

تابع change:

این تابع، یک بردار vector را به همراه دو مقدار a و b به عنوان ورودی می‌پذیرد و بردار vector را با استفاده از دستور vector.reshape((a, b))، به یک ماتریس با ابعاد a و b تبدیل می‌کند (شکل23).

```
def change(vector, a, b):
  vector = np.array(vector)
  matrix = vector.reshape((a, b))
  return matrix
```

شکل23: تعریف تابع change

تابع product:

این تابع یک ماتریس، یک بردار و یک مقدار آستانه T را به عنوان ورودی می‌پذیرد. سپس ماتریس را در بردار ضرب کرده، نتیجه را با مقدار آستانه جمع کرده و آن را در یک لیست به نام result_vector ذخیره می‌کند (شکل24).

```
def product(matrix, vector, T):
  result_vector = []
  for i in range(len(matrix)):
    sum = 0
    for j in range(len(vector)):
      sum = sum + matrix[i][j] * vector[j]
    result_vector.append((sum + T))
  return result_vector
```

شکل24: تعریف تابع product

تابع action:

این تابع یک بردار، یک مقدار آستانه T و آرگومان Emax (بیشینه مقدار مجاز برای تفاوت مقدار خروجی‌های متوالی) را به عنوان ورودی می‌پذیرد. سپس هریک از مقادیر بردار را با مقدار آستانه مقایسه کرده و برای هر حالت، خروجی مناسب را در لیست result_vector اضافه می‌کند (شکل25).

```
def action(vector, T, Emax):
  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
```

شکل25: تعریف تابع action

تابع sum:

این تابع یک بردار و یک مقدار j را به عنوان ورودی می‌پذیرد. سپس تمام درایه‌های بردار (به جز درایه‌ای که اندیس آن برابر j است) را با هم جمع کرده و حاصل جمع را به عنوان خروجی برمی‌گرداند (شکل26).

```
def sum(vector, j):
  total_sum = 0
  for i in range(0, len(vector)):
    if i != j:
      total_sum = total_sum + vector[i]
  return total_sum
```

شکل26: تعریف تابع sum

تابع norm:

این تابع دو بردار را به عنوان ورودی دریافت کرده و نرم آنها را محاسبه می‌کند (شکل27).

```
def norm(vector, p):
  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)
```

شکل27: تعریف تابع norm

```
path = [
    '/content/1.jpg',
    '/content/2.jpg',
    '/content/3.jpg',
    '/content/4.jpg',
    '/content/5.jpg',
]

x = []
for i in path:
  x.append(convertImageToBinary(i))

image_path = "/content/noisy1.jpg"
y = convertImageToBinary(image_path)
```

شکل28: لیست تصاویر اصلی و آرایه تصویر نویزی

```
k = len(x)
a = 96
b = 96
q = change(y, a, b)
plt.matshow(q)
m = len(x[0])
T = m / 2
Emax = 0.000001
U = 1 / Emax
```

شکل29: تعریف متغیرهای موردنیاز

```
w = [[(x[i][j]) / 2 for j in range(m)] for i in range(k)]
e = round(1 / len(x), 1)
E = [[0 for j in range(k)] for i in range(k)]
```

شکل30: تعریف ماتریس‌های w و E

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

شکل31: مشخص کردن مقادیر ماتریس E

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

شکل32: مقدار اولیه بردار خروجی

```
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*sum(y[i], j)
  y.append((action(s[i + 1], U, Emax)))
  i += 1
  p = y[i - 1]
```

شکل33: آموزش شبکه عصبی

```
result_index = y[len(y) - 1].index(max(y[len(y) - 1])) + 1
q = change(x[result_index - 1], a, b)

matplotlib.image.imsave('output.jpg', q)
output_img = Image.open('output.jpg')
output_img = output_img.transpose(Image.FLIP_TOP_BOTTOM)
output_img = output_img.transpose(Image.ROTATE_270)
output_img.save('output.jpg')

from matplotlib import pyplot as plt
from matplotlib import image as img

plt.show()
image = img.imread('output.jpg')
plt.imshow(image)
plt.show()
```

شکل34: نمایش تصویر بدست آمده از تصویر نویزی

در شکل28، یک لیست به نام path تعریف شده است که تمام تصاویر را در خود ذخیره می‌کند. سپس یک آرایه x تعریف می‌شود که مقدار باینری تصاویر ذخیره شده در لیست path را به کمک تابع convertImageToBianry در خود ذخیره می‌کند. سپس یکی از تصاویر نویزی را به عنوان بردار y انتخاب می‌کنیم تا ببینیم شبکه عصبی می‌تواند تصویر اصلی آن را درست تشخیص دهد یا خیر.

در شکل29، تعدادی متغیر تعریف شده است. متغیر k اندازه لیست x که لیست باینری تصاویر است را در خود ذخیره می‌کند. متغیرهای a و b که ابعاد ماتریس را مشخص می‌کنند، هر دو برابر 96 انتخاب شده‌اند. بردار y (بردار تصویر نویزی) با استفاده از تابع change به ماتریس q با ابعاد a و b تبدیل می‌شود و سپس با استفاده از تابع plt.matshow(q) نمایش داده می‌شود. متغیر m طول یکی از تصاویر ذخیره شده در ماتریس x (ماتریس تصاویر اصلی) را در خود ذخیره می‌کند و مقدار متغیر T (مقدار آستانه تابع action) برابر m/2 تعریف می‌شود. متغیر Emax بیشینه مقدار مجاز برای اختلاف نرم مقادیر خروجی متوالی را مشخص می‌کند و متغیر U برابر عکس آن تعریف می‌شود.

در شکل30، دو ماتریس w (ماتریس وزن‌ها) و ماتریس E (ماتریس سیناپسی شبکه عصبی) تعریف شده است.

در شکل31 مقادیر ماتریس E مشخص می‌شود. به این صورت که مقادیر عناصر قطری ماتریس برابر 1 و مقادیر آرایه‌های دیگر برابر e- (تعریف شده در شکل30) قرار داده می‌شود.

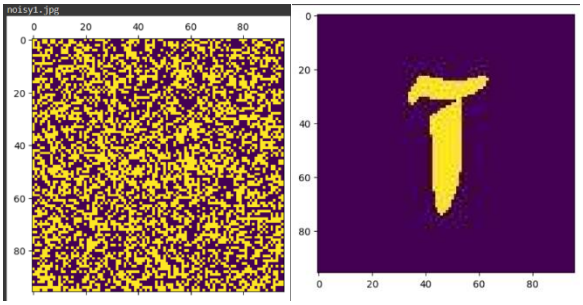
در شکل32، مقدار اولیه بردار خروجی با استفاده از تابع product بدست می‌آید. سپس این بردار از تابع action عبور داده شده و در بردار p ذخیره می‌شود.

در شکل33 آموزش شبکه عصبی نشان داده شده است. این آموزش تا زمانی که نرم میان بردارهای y و p بزرگتر از مقدار Emax باشد، ادامه خواهد داشت.

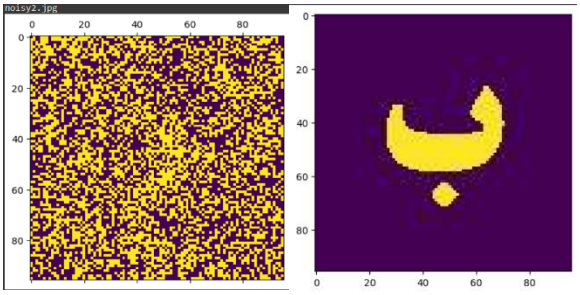
شکل34 نیز تصویر تشخیص داده شده توسط شبکه عصبی را نشان می‌دهد. چون تابع change تصویر را وارونه می‌کند و آن را می‌چرخاند، لذا به ترتیب از دستورهای output_image.transpose(FLIP_TOP_BUTTOM) و output_image.transpose(ROTATE_270) استفاده شده تا تصویر به درستی نشان داده شود.

مقدار نویز اضافه شده در مرحله ابتدایی noise_factor = 1000 در نظر گرفته شده است.

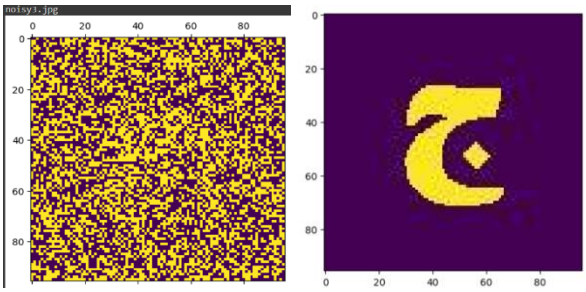
در ادامه به ترتیب هریک از تصاویر نویزی را به شبکه عصبی می‌دهیم و نتایج را در شکل‌های 35 تا 39 نشان می‌دهیم.



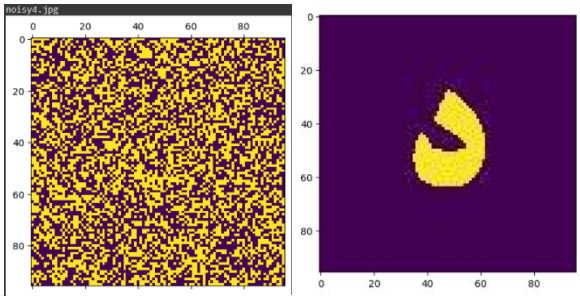
شکل35: تشخیص درست حرف "آ"



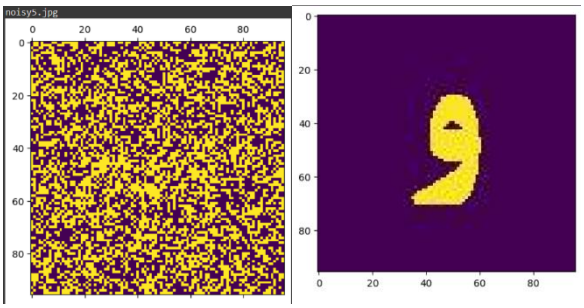
شکل36: تشخیص درست حرف "ب"



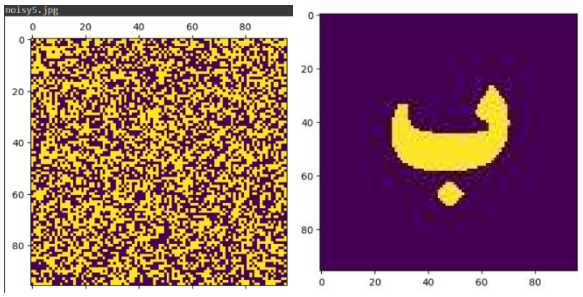
شکل37: تشخیص درست حرف "ج"



شکل38: تشخیص درست حرف "د"



شکل39: تشخیص درست حرف "و"



شکل40: تشخیص اشتباه حرف "ب" به جای حرف "و"

اینک مقدار نویز تصاویر (noise_fator) را افزایش می‌دهیم تا ببینیم شبکه عصبی تا چه میزان نویز به خوبی عمل می‌کند.

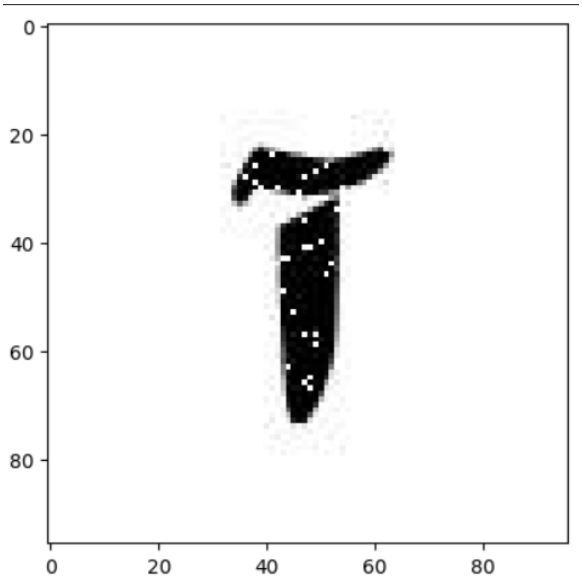
شبکه عصبی طراحی شده به ازای noise_factor <= 4000 عملکرد خوبی دارد؛ ولی به ازای مقادیر بیشتر نویز، دچار اشتباه می‌شود. برای مثال عملکرد شبکه عصبی به ازای مقدار noise_factor = 4100 در شکل40 نشان داده شده است.

3. با الهام گرفتن از تابع نوشته شده برای تولید داده‌های نویزی، یک تابع بنویسید که از داده‌های ورودی، خروجی‌های دارای Missing Point تولید کند. سپس عملکرد شبکه خود را با مقدار مشخصی Missing Point آزمایش و تحلیل کنید. اگر میزان Missing Point از چه حدی بیشتر شود عملکرد شبکه طراحی شده شما دچار اختلال می‌شود؟ راه‌حل چیست؟

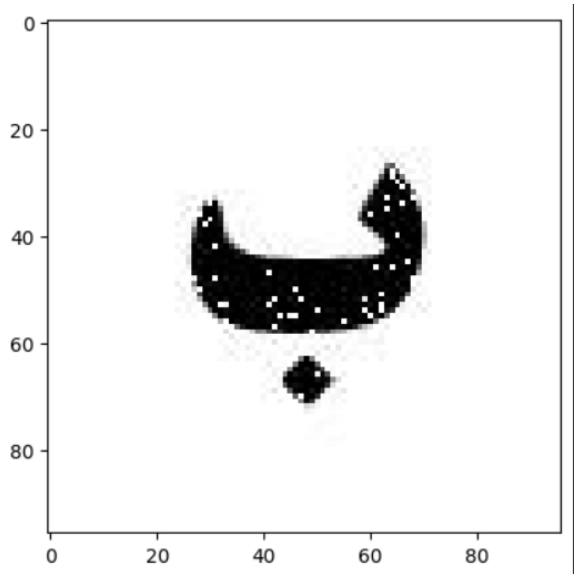
در این قسمت از تابع getNoisyBinaryImage استفاده کرده و با کمی تغییر در آن، نقاط Missing Points را ایجاد می‌کنیم. ابتدا برای این تابع یک آرگومان دیگر با نام num_missing_points به عنوان ورودی تعریف کرده و در بدنه تابع، یک حلقه for به صورت شکل 41 می‌نویسیم که به تعداد num_missing_points در بعد x و y نقاطی را ایجاد می‌کند. تصاویر ایجاد شده دارای Missing Point در نهایت توسط تابع generateNoisyImages به نام noisy_mp{i}.jpg ذخیره می‌شوند. شکل‌های 42 تا 46 این تصاویر ساخته شده را نمایش می‌دهند.

```
for i in range(num_missing_points):
  x = random.randint(0, width - 1)
  y = random.randint(0, height - 1)
  draw.point((x, y), (255, 255, 255))
```

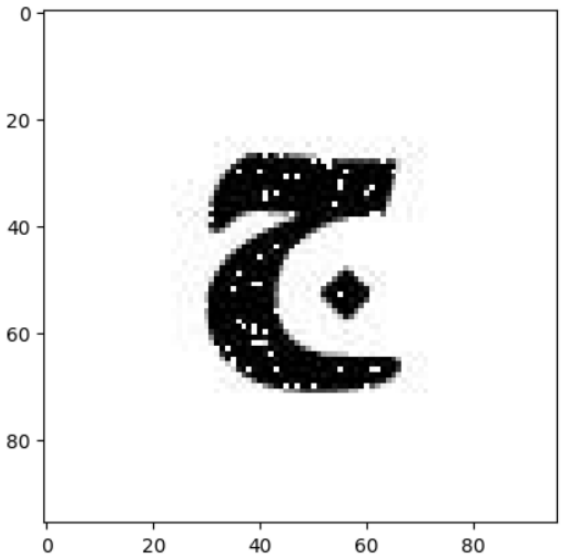
شکل41: ایجاد نقاط Missing Points



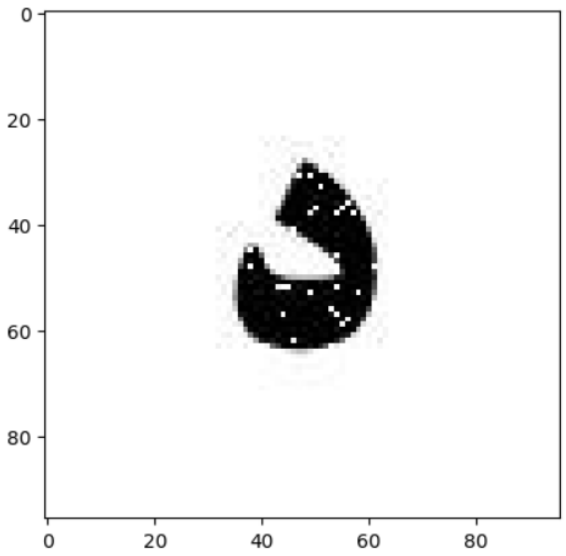
شکل42: حرف "آ" دارای Missing Points



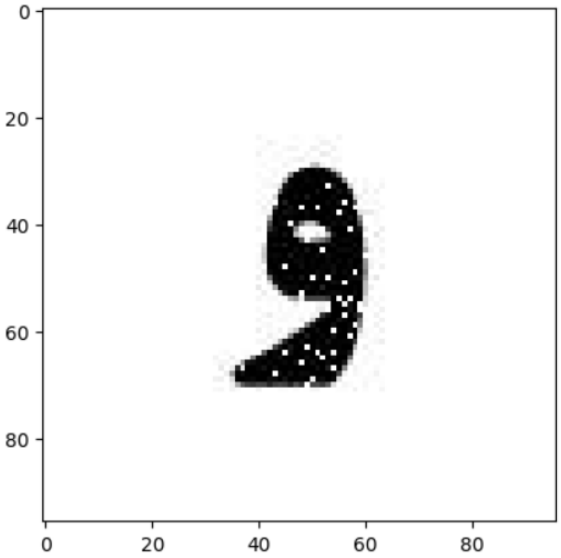
شکل43: حرف "ب" دارای Missing Points



شکل44: حرف "ج" دارای Missing Points



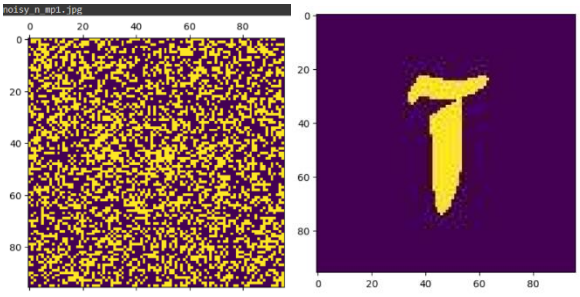
شکل45: حرف "د" دارای Missing Points



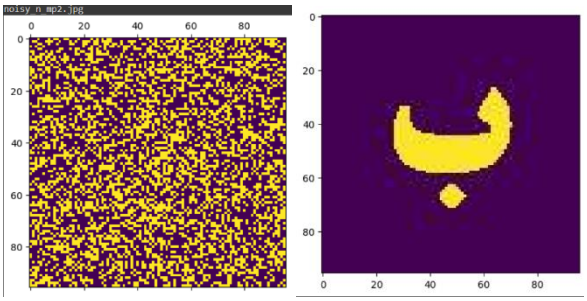
شکل46: حرف "و" دارای Missing Points

اینک عملکرد شبکه عصبی را هم با مقداری نویز و هم با مقداری Missing Points ارزیابی می‌کنیم. اینبار مقدار noise_factor را ثابت و برابر 1000 قرار می‌دهیم و با افزایش تدریجی تعداد Missing Points، عملکرد شبکه عصبی را ارزیابی می‌کنیم.

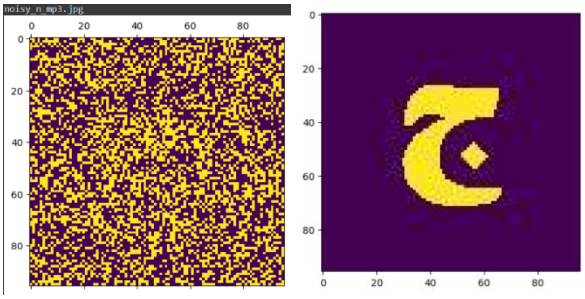
تا زمانی که num_missing_points <= 1500 باشد، شبکه عصبی تصاویر را به خوبی تشخیص می‌دهد (شکل‌های 47 تا 51)؛ اما به ازای مقادیر بیشتر Missing Points، شبکه عصبی دچار اشتباه می‌شود (برای مثال، شکل52).



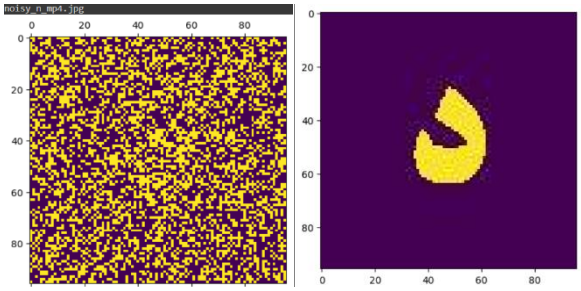
شکل47: تشخیص درست حرف "آ"



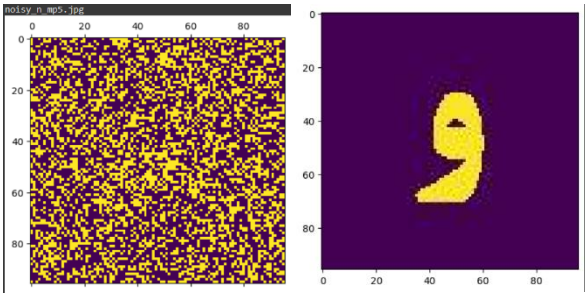
شکل48: تشخیص درست حرف "ب"



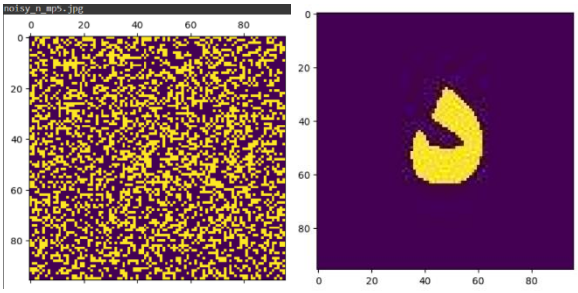
شکل49: تشخیص درست حرف "ج"



شکل50: تشخیص درست حرف "د"



شکل51: تشخیص درست حرف "و"



شکل52: تشخیص اشتباه حرف "و" به جای حرف "د"

سوال چهارم

یک مجموعه داده برای پیش‌بینی قیمت خانه‌ها را از طریق این پیوند دانلود کنید و مراحل ذکر شده در سوالات بعدی را برای فایل data.csv آن انجام دهید. لازم است که هر قسمت و مورد خواسته شده را با استفاده از دستورات پایتون انجام دهید و در جایی که نیاز است، نتایج را به صورت دقیق و کامل نمایش داده و تحلیل کنید.

1. فایل csv مربوط به این سوال را خوانده و سپس تابع Info. را از Pandas فراخوانی کنید. تعداد داده‌هایی که Nan هستند را برحسب هر ستون نمایش دهید و اگر نیاز است دستوراتی برای رفع این مشکل بنویسید.

پس از بارگذاری فایل data.csv از طریق گوگل درایو در محیط گوگل‌کولب، با استفاده از دستور df.info اطلاعات دیتافریم را نمایش می‌دهیم (شکل53 و 54). برای نمایش تعداد داده‌های Nan در هر ستون، دستور df[‘column_name’].isnull().sum() را به کار برده و نتایج را نمایش می‌دهیم (شکل55). کدنویسی این بخش در شکل56 نشان داده شده است.

```
<bound method DataFrame.info of                  date        price  bedrooms  bathrooms  sqft_living  \
0     2014-05-02 00:00:00  3.130000e+05       3.0       1.50         1340
1     2014-05-02 00:00:00  2.384000e+06       5.0       2.50         3650
2     2014-05-02 00:00:00  3.420000e+05       3.0       2.00         1930
3     2014-05-02 00:00:00  4.200000e+05       3.0       2.25         2000
4     2014-05-02 00:00:00  5.500000e+05       4.0       2.50         1940
...                   ...           ...       ...        ...          ...
4595  2014-07-09 00:00:00  3.081667e+05       3.0       1.75         1510
4596  2014-07-09 00:00:00  5.343333e+05       3.0       2.50         1460
4597  2014-07-09 00:00:00  4.169042e+05       3.0       2.50         3010
4598  2014-07-10 00:00:00  2.034000e+05       4.0       2.00         2090
4599  2014-07-10 00:00:00  2.206000e+05       3.0       2.50         1490

      sqft_lot  floors  waterfront  view  condition  sqft_above  \
0         7912     1.5           0     0          3        1340
1         9050     2.0           0     4          5        3370
2        11947     1.0           0     0          4        1930
3         8030     1.0           0     0          4        1000
4        10500     1.0           0     0          4        1140
...        ...     ...         ...   ...        ...         ...
4595      6360     1.0           0     0          4        1510
4596      7573     2.0           0     0          3        1460
4597      7014     2.0           0     0          3        3010
4598      6630     1.0           0     0          3        1070
4599      8102     2.0           0     0          4        1490
```

شکل53: نمایش اطلاعات دیتافریم

```
      sqft_basement  yr_built  yr_renovated                    street  \
0                 0      1955          2005       18810 Densmore Ave N
1               280      1921             0         709 W Blaine St
2                 0      1966             0  26206-26214 143rd Ave SE
3              1000      1963             0         857 170th Pl NE
4               800      1976          1992        9105 170th Ave NE
...             ...       ...           ...                      ...
4595              0      1954          1979         501 N 143rd St
4596              0      1983          2009        14855 SE 10th Pl
4597              0      2009             0         759 Ilwaco Pl NE
4598           1020      1974             0        5148 S Creston St
4599              0      1990             0       18717 SE 258th St

           city  statezip country
0     Shoreline  WA 98133     USA
1       Seattle  WA 98119     USA
2          Kent  WA 98042     USA
3      Bellevue  WA 98008     USA
4      Redmond   WA 98052     USA
...         ...       ...     ...
4595    Seattle  WA 98133     USA
4596   Bellevue  WA 98007     USA
4597     Renton  WA 98059     USA
4598    Seattle  WA 98178     USA
4599  Covington  WA 98042     USA
```

شکل54: نمایش اطلاعات دیتافریم - ادامه

```
The number of Nan data under the 'date' column: 0
The number of Nan data under the 'price' column: 0
The number of Nan data under the 'bedroos' column: 0
The number of Nan data under the 'bathrooms' column: 0
The number of Nan data under the 'sqft_living' column: 0
The number of Nan data under the 'floors' column: 0
The number of Nan data under the 'waterfront' column: 0
The number of Nan data under the 'view' column: 0
The number of Nan data under the 'condition' column: 0
The number of Nan data under the 'sqft_above' column: 0
The number of Nan data under the 'sqft_basement' column: 0
The number of Nan data under the 'yr_built' column: 0
The number of Nan data under the 'yr_renovated' column: 0
The number of Nan data under the 'street' column: 0
The number of Nan data under the 'city' column: 0
The number of Nan data under the 'statezip' column: 0
The number of Nan data under the 'country' column: 0
```

شکل55: نمایش تعداد داده‌های Nan در هر ستون

```
#4.1

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

!pip install gdown
!gdown 1VkyRxeJu04QKglgjUruI634cUy1-I1Lc

df = pd.read_csv("data.csv")
print(df.info)
print("The number of Nan data under the 'date' column: " + str(df['date'].isnull().sum()))
print("The number of Nan data under the 'price' column: " + str(df['price'].isnull().sum()))
print("The number of Nan data under the 'bedroos' column: " + str(df['bedrooms'].isnull().sum()))
print("The number of Nan data under the 'bathrooms' column: " + str(df['bathrooms'].isnull().sum()))
print("The number of Nan data under the 'sqft_living' column: " + str(df['sqft_living'].isnull().sum()))
print("The number of Nan data under the 'floors' column: " + str(df['floors'].isnull().sum()))
print("The number of Nan data under the 'waterfront' column: " + str(df['waterfront'].isnull().sum()))
print("The number of Nan data under the 'view' column: " + str(df['view'].isnull().sum()))
print("The number of Nan data under the 'condition' column: " + str(df['condition'].isnull().sum()))
print("The number of Nan data under the 'sqft_above' column: " + str(df['sqft_above'].isnull().sum()))
print("The number of Nan data under the 'sqft_basement' column: " + str(df['sqft_basement'].isnull().sum()))
print("The number of Nan data under the 'yr_built' column: " + str(df['yr_built'].isnull().sum()))
print("The number of Nan data under the 'yr_renovated' column: " + str(df['yr_renovated'].isnull().sum()))
print("The number of Nan data under the 'street' column: " + str(df['street'].isnull().sum()))
print("The number of Nan data under the 'city' column: " + str(df['city'].isnull().sum()))
print("The number of Nan data under the 'statezip' column: " + str(df['statezip'].isnull().sum()))
print("The number of Nan data under the 'country' column: " + str(df['country'].isnull().sum()))
```

شکل56: کدنویسی نمایش دیتافریم و نمایش تعداد داده‌های Nan هر ستون

---

2. ماتریس همبستگی را رسم کنید. چه ویژگی‌ای با قیمت همبستگی بیشتری دارد؟

برای رسم ماتریس همبستگی از دستور df.corr() استفاده می‌کنیم (شکل57). ماتریس همبستگی در شکل58 و 59 نشان داده شده است. همانطور که دیده می‌شود، ضریب همبستگی ویژگی sqft_living با ویژگی قیمت از سایر ویژگی‌ها بیشتر است.

```
#4.2

corr_matrix = df.corr()
print("\n")
print(corr_matrix)
```

شکل57: استفاده از دستور df.corr() برای رسم ماتریس همبستگی

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors
price	1.000000	0.200336	0.327110	0.430410	0.050451	0.151461
bedrooms	0.200336	1.000000	0.545920	0.594884	0.068819	0.177895
bathrooms	0.327110	0.545920	1.000000	0.761154	0.107837	0.486428
sqft_living	0.430410	0.594884	0.761154	1.000000	0.210538	0.344850
sqft_lot	0.050451	0.068819	0.107837	0.210538	1.000000	0.003750
floors	0.151461	0.177895	0.486428	0.344850	0.003750	1.000000
waterfront	0.135648	-0.003483	0.076232	0.117616	0.017241	0.022024
view	0.228504	0.111028	0.211960	0.311009	0.073907	0.031211
condition	0.034915	0.025080	-0.119994	-0.062826	0.000558	-0.275013
sqft_above	0.367570	0.484705	0.689918	0.876443	0.216455	0.522814
sqft_basement	0.210427	0.334165	0.298020	0.447206	0.034842	-0.255510
yr_built	0.021857	0.142461	0.463498	0.287775	0.050706	0.467481
yr_renovated	-0.028774	-0.061082	-0.215886	-0.122817	-0.022730	-0.233996

	waterfront	view	condition	sqft_above	sqft_basement
price	0.135648	0.228504	0.034915	0.367570	0.210427
bedrooms	-0.003483	0.111028	0.025080	0.484705	0.334165
bathrooms	0.076232	0.211960	-0.119994	0.689918	0.298020
sqft_living	0.117616	0.311009	-0.062826	0.876443	0.447206
sqft_lot	0.017241	0.073907	0.000558	0.216455	0.034842
floors	0.022024	0.031211	-0.275013	0.522814	-0.255510
waterfront	1.000000	0.360935	0.000352	0.078911	0.097501
view	0.360935	1.000000	0.063077	0.174327	0.321602
condition	0.000352	0.063077	1.000000	-0.178196	0.200632
sqft_above	0.078911	0.174327	-0.178196	1.000000	-0.038723
sqft_basement	0.097501	0.321602	0.200632	-0.038723	1.000000
yr_built	-0.023563	-0.064465	-0.399698	0.408535	-0.161675
yr_renovated	0.008625	0.022967	-0.186818	-0.160426	0.043125

شکل58: ماتریس همبستگی

	yr_built	yr_renovated
price	0.021857	-0.028774
bedrooms	0.142461	-0.061082
bathrooms	0.463498	-0.215886
sqft_living	0.287775	-0.122817
sqft_lot	0.050706	-0.022730
floors	0.467481	-0.233996
waterfront	-0.023563	0.008625
view	-0.064465	0.022967
condition	-0.399698	-0.186818
sqft_above	0.408535	-0.160426
sqft_basement	-0.161675	0.043125
yr_built	1.000000	-0.321342
yr_renovated	-0.321342	1.000000

شکل59: ماتریس همبستگی - ادامه

---

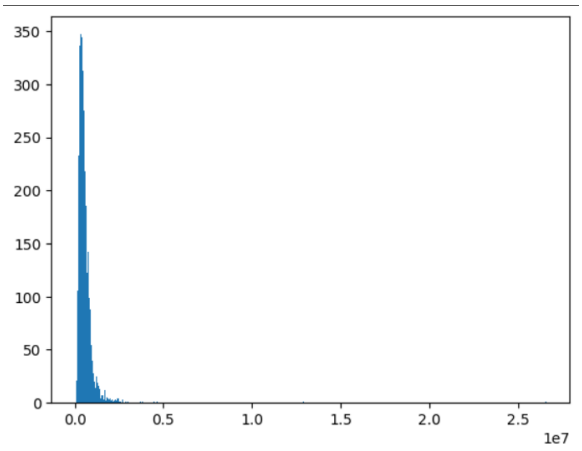
3. نمودار توزیع قیمت و نمودار قیمت و ویژگی‌ای که همبستگی زیادی با قیمت دارد را رسم کنید.

برای رسم نمودار توزیع قیمت، از دستور plt.hist(df['price'], bins = 'auto') استفاده شده که تعداد بازه‌ها (bins) به طور خودکار مقداردهی می‌شود (شکل‌های 60 و 61).

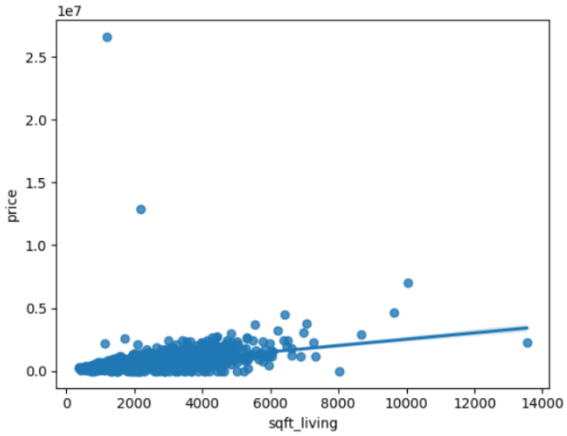
برای رسم نمودار قیمت برحسب ویژگی sqft_living (که بیشترین همبستگی را با price دارد)، ابتدا کتابخانه seaborn را به عنوان sns فراخوانی کرده و سپس از دستور sns.regplot استفاده می‌کنیم (شکل62).

```
#4.3

plt.hist(df['price'], bins = 'auto')
plt.show()

import seaborn as sns
sns.regplot(x = df['sqft_living'], y = df['price'])
```

شکل60: استفاده از دستورهای plt.hist و sns.regplot



شکل61: نمودار توزیع قیمت



شکل62: رسم نمودار قیمت برحسب ویژگی sqft_living

---

4. ستون Date را به دو ستون ماه و سال تبدیل کنید و این ستون را از دیتافریم حذف کنید.

به منظور تبدیل ستون Date به ستون‌های مجزا، از دستور df[‘date’].str.split(‘-‘, expand = True) استفاده می‌کنیم. این دستور، شروع به جدا کردن محتویات ستون Date می‌کند و هر زمان به علامت ‘-‘ برسد، متوقف شده و محتویات جدا شده را در یک ستون قرار می‌دهد. سپس دوباره شروع به کار کرده تا مجددا به علامت ‘-‘ برسد. بدین ترتیب این دستور ستون Date را به سه ستون مجزای year، month و day تبدیل می‌کند؛ اما چون فقط باید ستون‌های year و month باقی بمانند و ستون‌های day و Date حذف شوند، از دستور df.drop استفاده کرده و آنها را حذف می‌کنیم (شکل63). دیتافریم جدید در شکل64 و 65 نشان داده شده است.

```python
#4.4

df[['year', 'month', 'day']] = df['date'].str.split('-', expand = True)
df = df.drop('date', axis = 1)
df = df.drop('day', axis = 1)
print(df)
```

شکل63: تبدیل ستون date به دو ستون year و month

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	\
0	3.130000e+05	3.0	1.50	1340	7912	1.5	
1	2.384000e+06	5.0	2.50	3650	9050	2.0	
2	3.420000e+05	3.0	2.00	1930	11947	1.0	
3	4.200000e+05	3.0	2.25	2000	8030	1.0	
4	5.500000e+05	4.0	2.50	1940	10500	1.0	
...	...	...	...	...	...	...	
4595	3.081667e+05	3.0	1.75	1510	6360	1.0	
4596	5.343333e+05	3.0	2.50	1460	7573	2.0	
4597	4.169042e+05	3.0	2.50	3010	7014	2.0	
4598	2.034000e+05	4.0	2.00	2090	6630	1.0	
4599	2.206000e+05	3.0	2.50	1490	8102	2.0	

	waterfront	view	condition	sqft_above	sqft_basement	yr_built	\
0	0	0	3	1340	0	1955	
1	0	4	5	3370	280	1921	
2	0	0	4	1930	0	1966	
3	0	0	4	1000	1000	1963	
4	0	0	4	1140	800	1976	
...	...	...	...	...	...	...	
4595	0	0	4	1510	0	1954	
4596	0	0	3	1460	0	1983	
4597	0	0	3	3010	0	2009	
4598	0	0	3	1070	1020	1974	
4599	0	0	4	1490	0	1990	

شکل64: دیتافریم جدید

```
      yr_renovated                  street       city  statezip country  \
0             2005     18810 Densmore Ave N  Shoreline  WA 98133     USA
1                0           709 W Blaine St    Seattle  WA 98119     USA
2                0  26206-26214 143rd Ave SE       Kent  WA 98042     USA
3                0           857 170th Pl NE  Bellevue  WA 98008     USA
4             1992         9105 170th Ave NE   Redmond  WA 98052     USA
...            ...                       ...       ...       ...     ...
4595          1979           501 N 143rd St    Seattle  WA 98133     USA
4596          2009          14855 SE 10th Pl  Bellevue  WA 98007     USA
4597             0          759 Ilwaco Pl NE    Renton  WA 98059     USA
4598             0          5148 S Creston St   Seattle  WA 98178     USA
4599             0         18717 SE 258th St  Covington  WA 98042     USA

      year month
0     2014    05
1     2014    05
2     2014    05
3     2014    05
4     2014    05
...    ...   ...
4595  2014    07
4596  2014    07
4597  2014    07
4598  2014    07
4599  2014    07
```

شکل65: دیتافریم جدید - ادامه

---

5. داده‌ها را با نسبت 80 به 20 درصد به مجموعه‌های آموزش و آزمون تقسیم کنید و داده‌های آموزشی و آزمون را با استفاده از MinMaxScaler مقیاس کنید.

در این سوال، قیمت به عنوان تارگت در نظر گرفته شده و سایر ستون‌ها، ویژگی‌ها را تشکیل می‌دهند. پس با استفاده از دستورهای X = df.values[:, 1 : 13] و y = df.values[:, 0] ویژگی‌ها و تارگت را از هم جدا کرده و سپس با فراخوانی تابع train_test_split از کتابخانه sklearn.model_selection، داده‌ها را به دو بخش train و test با test_size = 0.2 تقسیم می‌کنیم (شکل66).

تابع MinMaxScaler مقادیر داده‌ها را به مقداری از 0 تا 1 مقیاس می‌کند. برای استفاده از این تابع، آن را از کتابخانه sklearn.preprocessing فراخوانی کرده و یک object با نام scaler = MinMaxScaler() را تعریف می‌کنیم. سپس با استفاده از دستور scaler.fit(X_train)، پارامترهای موردنیاز برای نرمالیزه کردن داده‌ها را از روی داده‌های X_train بدست آورده و با استفاده از دستورهای scaler.transform(X_train) و scaler.transform(X_test)، داده‌های آموزش و تست را نرمالیزه می‌کنیم (شکل67).

```python
#4.5

from sklearn.model_selection import train_test_split
X = df.values[:, 1 : 13]
print(X)
y = df.values[:, 0]
y = y.reshape((-1, 1))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

شکل66: تعریف ویژگی و تارگت و تقسیم آنها به دو بخش train و test

```python
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

شکل67: نرمالیزه کردن داده‌ها با استفاده از MinMaxScaler

6. یک مدل Multi-Layer Perceptron(MLP) ساده با 2 لایه پنهان یا بیشتر بسازید. بخشی از داده‌های آموزش را برای اعتبارسنجی کنار بگذارید و با انتخاب بهینه‌ساز و تابع اتلاف مناسب، مدل را آموزش دهید. نمودارهای اتلاف و R2 Score مربوط به آموزش و اعتبارسنجی را رسم و تحلیل کنید.

ابتدا کتابخانه‌های موردنیاز برای حل این سوال را اضافه می‌کنیم (شکل68). علاوه بر کتابخانه‌های numpy، pandas و matplotlib.pyplot، کتابخانه جدیدی به نام torch نیز اضافه شده است. این کتابخانه متشکل از الگوریتم‌های یادگیری عمیق است و در مورد شبکه عصبی MLP به ما کمک می‌کند. کتابخانه دیگر، کتابخانه torch.nn است که امکانات لازم برای تعریف یک شبکه عصبی را فراهم می‌کند. کتابخانه torch.optim نیز فراخوانی می‌شود که شامل مجموعه‌ای از الگوریتم‌های بهینه‌سازی است. درنهایت با توجه به صورت سوال که رسم نمودار R2 Score را می‌خواهد، از کتابخانه sklearn.metrics تابع r2_score را فراخوانی می‌کنیم. این تابع با دریافت تارگت اصلی و تارگت پیش‌بینی شده، مشخص می‌کند که عملکرد شبکه عصبی تا چه حد خوب بوده است.

پس از اضافه کردن کتابخانه‌ها، یک کلاس MLP تعریف می‌کنیم که شامل آرگومان‌های num_input (تعداد نورون‌های ورودی)، num_hidden1، num_hidden2 و num_hidden3 (تعداد نورون‌های لایه‌های پنهان) و num_output (تعداد نورون‌های خروجی) است. داخل کلاس از دستور super(MLP, self).__init__() استفاده می‌شود. این دستور تضمین می‌کند که کلاس، بتواند از متدها و امکانات کلاس‌های دیگر ارث ببرد. سه متغیر به صورت self.fc{i} = nn.Linear(in_features, out_features) تعریف می‌شود که هرکدام از آنها، لایه ورودی و لایه خروجی مخصوص به خود را دارند. پس از تعریف متغیرها، شروع به کدنویسی forward_propagation کنیم. در این بخش، تابع forward_propagation یک آرگومان به نام x را می‌پذیرد. سپس آن آرگومان از یک تابع فعال (در اینجا relu) با استفاده از دستور nn.relu عبور داده شده و مقدار آن برگردانده می‌شود. شکل69 کدنویسی کلاس MLP را نشان می‌دهد.

پس از تعریف کلاس MLP، دیتاست را بارگذاری کرده و با استفاده از دستور train_test_split آن را به سه بخش آموزش، اعتبارسنجی و ارزیابی تقسیم می‌کنیم. سپس هریک از داده‌های X_train، X_val و X_test را با استفاده از MinMaxScaler نرمالیزه می‌کنیم (شکل70).

برای آموزش داده‌ها، تابع train را تعریف می‌کنیم که دارای آرگومان‌های model، optimizer، criterion، X_train، y_train، X_val، y_val و num_epock است.

فرآیند آموزش:

در داخل حلقه for که برای آموزش داده‌ها نوشته می‌شود، با استفاده از model.train()، آموزش شروع می‌شود.

دستور optimizer.zero_grad، پارامترهای محاسبه شده برای گرادیان در هر epock را دوباره به صفر برمی‌گرداند تا عملیات آموزش بتواند در هر epock مستقل از epock قبلی اتفاق بیفتد.

دستور yhat_train = model(X_train)، خروجی داده‌های آموزش داده شده را در متغیر yhat_train قرار می‌دهد.

دستور train_loss = criterion(yhat_train, y_train) با مقایسه مقدار تارگت پیش‌بینی شده (yhat_train) و تارگت اصلی (y_train)، مقدار اتلاف را محاسبه می‌کند.

دستور train_loss.backward() عملیات back propagation را برای محاسبه گرادیان train_loss با توجه به پارامترهای مدل انجام می‌دهد و دستور optimizer.step() پارامترهای مدل را براساس گرادیان‌های محاسبه شده و الگوریتم بهینه‌سازی انتخاب شده، آپدیت می‌کند.

دستور train_loss_list.append(train_loss.item()) مقدار هر اتلاف را در لیست train_loss_list قرار می‌دهد.

دستور train_r2score = r2_score(y_train, yhat_train.squeeze().detach().numpy()) با دریافت تارگت اصلی و تارگت پیش‌بینی شده، مقدار r2 score را محاسبه می‌کند. دستور squeeze() برای کاهش تعداد ابعاد به کار می‌رود و ورودی‌های تک‌بعدی را از ابعاد ماتریس حذف می‌کند. دستور detack() یک کپی از tensor را ایجاد می‌کند، بدون اینکه اطلاعاتی از نحوه محاسبه آن داشته باشد. این دستور تضمین می‌کند که پارامترهای مدل در این مرحله آپدیت نشود و مقادیر گرادیان تغییر نکند. دستور numpy نیز برای دسترسی ما به امکانات کتابخانه numpy اضافه شده است. درنهایت مقدار train_r2score با استفاده از دستور train_r2score_list.append(r2_score) در یک لیست قرار می‌گیرد.

فرآیند آموزش در شکل 71 نشان داده شده است.

فرآیند اعتبارسنجی:

با استفاده از دستور model.eval() فرآیند اعتبارسنجی آغاز می‌شود.

دستور with torch.no_grad() تضمین می‌کند که در فرآیند اعتبارسنجی، هیچگونه محاسباتی برای گرادیان انجام نشود و مقدار آن تغییر نکند. سپس همان دستورات مرحله قبل را در اینجا برای داده‌های X_val به کار می‌بریم (توضیح کدها مشابه قبل است).

شکل72 فرآیند اعتبارسنجی را نشان می‌دهد.

پس از پایان بخش‌های آموزش و اعتبارسنجی، نمودارهای اتلاف و r2 score را برای داده‌های آموزش و اعتبارسنجی رسم می‌کنیم (شکل73).

```
class MLP(nn.Module):
  def __init__(self, num_input, num_hidden1, num_hidden2, num_hidden3, num_output):
    super(MLP, self).__init__()
    self.fc1 = nn.Linear(num_input, num_hidden1)
    self.fc2 = nn.Linear(num_hidden1, num_hidden2)
    self.fc3 = nn.Linear(num_hidden2, num_hidden3)
    self.fc4 = nn.Linear(num_hidden3, num_output)

  def forward(self, x):
    x = torch.relu(self.fc1(x))
    x = torch.relu(self.fc2(x))
    x = torch.relu(self.fc3(x))
    x = self.fc4(x)
    return x
```

شکل69: کلاس MLP

```
#4.6

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import r2_score
```

شکل68: اضافه کردن کتابخانه‌های موردنیاز

```
def train(model, optimizer, criterion, X_train, y_train, X_val, y_val, num_epoch = 1000):
  train_loss_list = []
  validation_loss_list = []
  train_r2score_list = []
  r2score_list = []

  for i in range(num_epoch):
    model.train()
    optimizer.zero_grad()
    yhat_train = model(X_train)
    train_loss = criterion(yhat_train.squeeze(), y_train)
    train_loss.backward()
    optimizer.step()
    train_loss_list.append(train_loss.item())
    train_r2score = r2_score(y_train, yhat_train.squeeze().detach().numpy())
    train_r2score_list.append(train_r2score)
```

شکل71: فرآیند آموزش

```
from sklearn.model_selection import train_test_split
X = df.values[:, 1 : 13]
y = df.values[:, 0]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size = 0.2)

from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X_train)
```

شکل70: تقسیم دیتاست به سه بخش آموزش، اعتبارسنجی و ارزیابی و نرمال‌سازی آنها

```
model.eval()
with torch.no_grad():
  yhat_val = model(X_val)
  val_loss = criterion(yhat_val.squeeze(), y_val)
  validation_loss_list.append(val_loss.item())
  r2score = r2_score(y_val, yhat_val.squeeze().detach().numpy())
  r2score_list.append(r2score)
```

شکل72: فرآیند اعتبارسنجی

```
plt.plot(train_loss_list, color = 'red', label = 'train')
plt.plot(validation_loss_list, color = 'blue', label = 'validation')
plt.xlabel('epock')
plt.ylabel('Loss')
plt.legend()
plt.show()
plt.plot(train_r2score_list, color = 'red', label = 'train')
plt.plot(r2score_list, color = 'blue', label = 'validation')
plt.xlabel('epock')
plt.ylabel('r2score')
plt.legend()
plt.show()
```

شکل73: کدنویسی برای رسم نمودارهای Loss و r2 score

اینک نوشتن تابع train به پایان رسیده است و باید با استفاده از دستور torch.Tensor، آرایه دیتاها به Pytorch tensor تبدیل شود (شکل74).

در ادامه یک نمونه mlp از کلاس MLP با آرگومان‌های num_input = 12 (تعداد ویژگی‌ها)، num_hidden1 = 50، num_hidden2 = 100، num_hidden3 = 150 و num_output = 1 (تعداد تارگت‌ها) ایجاد می‌کنیم. سپس با استفاده از دستور optim.Adam الگوریتم بهینه‌سازی Adam و به کمک دستور nn.MSELoss() تابع اتلاف MSE را فراخوانی می‌کنیم و آموزش و ارزیابی را انجام می‌دهیم (شکل75).

نمودارهای اتلاف و r2 score به ترتیب در شکل‌های 76 و 77 (برای داده‌های train و validation) نمایش داده شده‌اند. همچنین نتیجه عملکرد شبکه عصبی برای داده‌های ارزیابی در شکل78 نشان داده شده است. همانطور که از شکل‌ها استنتاج می‌شود، دقت ارزیابی شبکه عصبی چندان بالا نیست (r2 score ≈ 0.33). برای افزایش عملکرد شبکه عصبی چندین راهکار وجود دارد که در ادامه آمده است:

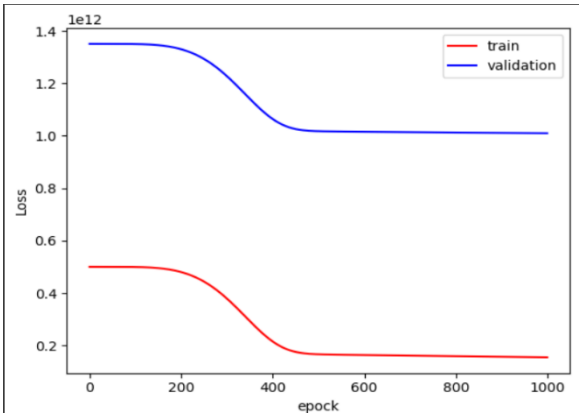
1. می‌توان تعداد epock را افزایش داد.
2. می‌توان lr = 0.001 (learning rate) را کاهش داد.
3. می‌توان تعداد لایه‌های پنهان شبکه عصبی را افزایش داد.
4. می‌توان تعداد نورون‌های لایه‌های پنهان را افزایش داد.
5. می‌توان از توابع اتلاف و توابع بهینه‌ساز متفاوتی استفاده کرد.

```
X_train = torch.Tensor(X_train)
y_train = y_train.astype(float)
y_train = torch.Tensor(y_train)
X_val = torch.Tensor(X_val)
y_val = y_val.astype(float)
y_val = torch.Tensor(y_val)
X_test = torch.Tensor(X_test)
y_test = y_test.astype(float)
y_test = torch.Tensor(y_test)
```

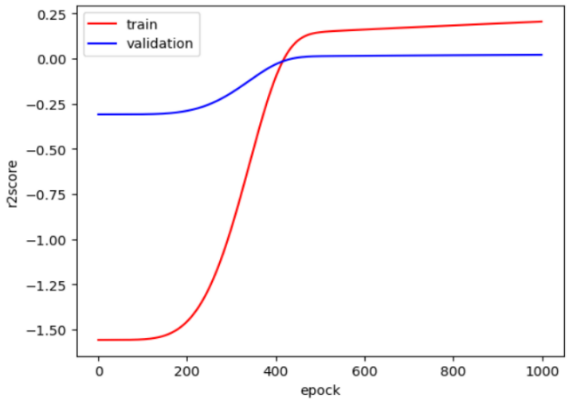
شکل74: استفاده از دستور torch.Tensor

```
mlp = MLP(12, 50, 100, 150, 1)
optimizer = optim.Adam(mlp.parameters(), lr = 0.001)
criterian = nn.MSELoss()
mlp_ = train(mlp, optimizer, criterian, X_train, y_train, X_val, y_val, num_epock = 1000)

mlp.eval()
with torch.no_grad():
  yhat = mlp(X_test)
  test_loss = criterian(yhat.squeeze(), y_test)
  test_r2score = r2_score(y_test, yhat.squeeze().detach().numpy())

print("Loss: " + str(test_loss) + "R2 Score: " + str(test_r2score))
```

شکل75: ایجاد یک نمونه کلاس mlp و آموزش مدل با دیتاست موردنظر



شکل76: نمودار Loss



شکل77: نمودار r2 score

```
Loss: tensor(9.2773e+10)R2 Score: 0.326733937028881
```

شکل78: نمایش عملکرد شبکه عصبی MLP

---

7. فرآیند سوال قبل با یک بهینه‌ساز و تابع اتلاف جدید انجام داده و نتایج را مقایسه و تحلیل کنید.

تغییر تابع بهینه‌ساز:

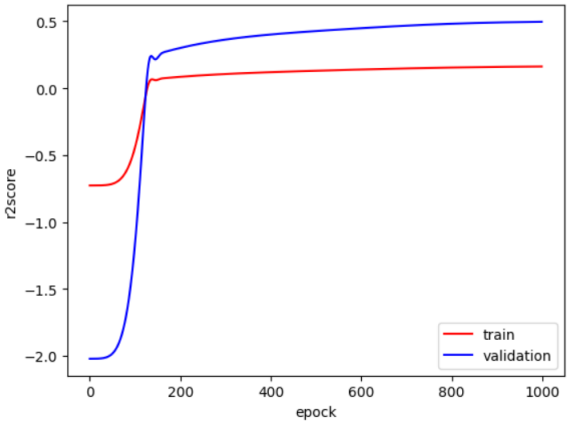
در این مرحله تابع بهینه‌ساز را از Adam به Adamax تغییر می‌دهیم (شکل79).

نمودارهای اتلاف و r2 score به ترتیب در شکل‌های 80 و 81 (برای داده‌های train و validation) نمایش داده شده‌اند. همچنین نتیجه عملکرد شبکه عصبی برای داده‌های ارزیابی در شکل82 نشان داده شده است.

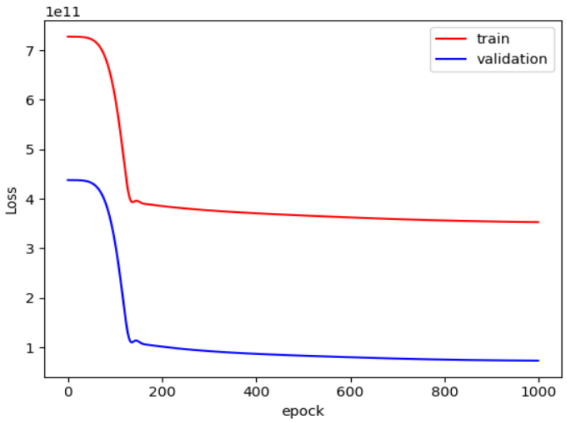
همانطور که دیده می‌شود، دقت شبکه عصبی نسبت به قبل افزایش یافته است (r2 score ≈ 0.5).

```
mlp2 = MLP(12, 50, 100, 150, 1)
optimizer = optim.Adamax(mlp2.parameters(), lr = 0.01)
criterian = nn.MSELoss()
mlp_ = train(mlp2, optimizer, criterian, X_train, y_train, X_val, y_val, num_epock = 1000)

mlp2.eval()
with torch.no_grad():
  yhat = mlp2(X_test)
  test_loss = criterian(yhat.squeeze(), y_test)
  test_r2score = r2_score(y_test, yhat.squeeze().detach().numpy())

print("Loss: " + str(test_loss) + "R2 Score: " + str(test_r2score))
```

شکل79: ایجاد کلاس mlp2 و تغییر تابع بهینه‌ساز به Adamax



شکل81: نمودار r2 score



شکل80: نمودار Loss

```
Loss: tensor(6.1958e+10)R2 Score: 0.5058299758424991
```

شکل82: نمایش عملکرد شبکه عصبی MLP

تغییر تابع اتلاف:

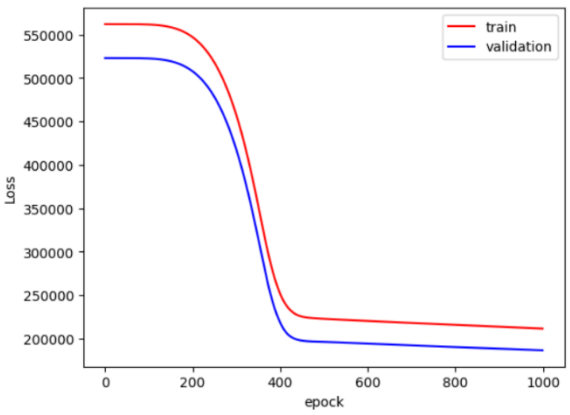
در این مرحله تابع اتلاف را از MSELoss به HuberLoss تغییر می‌دهیم (شکل82).

نمودارهای اتلاف و r2 score به ترتیب در شکل‌های 83 و 84 (برای داده‌های train و validation) نمایش داده شده‌اند. همچنین نتیجه عملکرد شبکه عصبی برای داده‌های ارزیابی در شکل85 نشان داده شده است.

همانطور که دیده می‌شود، دقت شبکه عصبی با تابع اتلاف HuberLoss نسبت به قبل کاهش پیدا کرده است (r2 score ≈ 0.245).

```python
mlp3 = MLP(12, 50, 100, 150, 1)
optimizer = optim.Adam(mlp3.parameters(), lr = 0.001)
criterian = nn.HuberLoss()
mlp_ = train(mlp3, optimizer, criterian, X_train, y_train, X_val, y_val, num_epock = 1000)

mlp2.eval()
with torch.no_grad():
  yhat = mlp3(X_test)
  test_loss = criterian(yhat.squeeze(), y_test)
  test_r2score = r2_score(y_test, yhat.squeeze().detach().numpy())

print("Loss: " + str(test_loss) + "R2 Score: " + str(test_r2score))
```

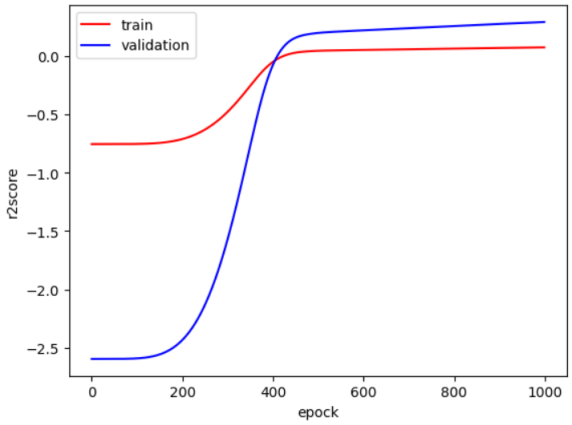
شکل82: ایجاد کلاس mlp3 و تغییر تابع اتلاف به HuberLoss



شکل83: نمودار Loss

```
Loss: tensor(190513.2969)R2 Score: 0.24518736579748313
```

شکل85: نمایش عملکرد شبکه عصبی MLP



شکل84: نمودار r2 score

بنابراین بهترین نتیجه زمانی حاصل می‌شود که تابع اتلاف MSELoss و تابع بهینه‌ساز Adamax باشد.

---

8. پنج داده را به صورت تصادفی از مجموعه ارزیابی انتخاب کرده و قیمت پیش‌بینی شده را به همراه قیمت واقعی نشان دهید. قیمت پیش‌بینی شده با قیمت واقعی چقدر تفاوت دارد؟ آیا این عملکرد مناسب است؟ برای بهبود آن چه پیشنهادی دارید؟

ابتدا برای اینکه تنسورهای y_test (تارگت اصلی) و y_hat (تارگت پیش‌بینی شده) دارای ابعاد یکسانی باشند، کتابخانه tensorflow را فراخوانی کرده و با اعمال دستور y_test.reshape([1, 920]) و yhat.reshape([1, 920]) برای آنها، ابعاد آنها را تغییر داده و با هم برابر می‌کنیم. سپس با استفاده از حلقه for، پنج ایندکس تصادفی در بازه 0 تا تعداد داده‌های ارزیابی تولید کرده و مقدار y_test و yhat متناظر آنها را نمایش می‌دهیم (شکل86).

همانطور که در شکل87 دیده می‌شود، قیمت اصلی و قیمت پیش‌بینی شده با دقت خوبی به هم نزدیک هستند؛ اما برای بهبود عملکرد شبکه عصبی، همانطور که پیش‌تر توضیح داده شد، می‌توان از راهکارهای زیر استفاده کرد:

1. می‌توان تعداد epock را افزایش داد.

2. می‌توان lr = 0.001 (learning rate) را کاهش داد.

3. می‌توان تعداد لایه‌های پنهان شبکه عصبی را افزایش داد.

4. می‌توان تعداد نورون‌های لایه‌های پنهان را افزایش داد.

5. می‌توان از توابع اتلاف و توابع بهینه‌ساز متفاوتی استفاده کرد.

```
#4.8

import tensorflow as tf

mlp = MLP(12, 50, 100, 150, 1)
optimizer = optim.Adamax(mlp.parameters(), lr = 0.01)
criterian = nn.MSELoss()
mlp_ = train(mlp, optimizer, criterian, X_train, y_train, X_val, y_val, num_epoch = 1000)

mlp.eval()
with torch.no_grad():
  yhat = mlp(X_test)
  test_loss = criterian(yhat.squeeze(), y_test)
  test_r2score = r2_score(y_test, yhat.squeeze().detach().numpy())

print("Loss: " + str(test_loss) + "R2 Score: " + str(test_r2score))

y_test.reshape([1, 920])
yhat.reshape([1, 920])

for i in range(0, 5):
  n = np.random.randint(0, int(len(y)* 0.2))
  print("y_test: " + str(y_test[n]))
  print("predicted y: " + str(yhat[n]))
  print('\n')
```

شکل86: کدنویسی برای نماش قیمت اصلی و پیش‌بینی شده پنج داده ارزیابی تصادفی

```
y_test: tensor(314500.)
predicted y: tensor([481990.1875])


y_test: tensor(370000.)
predicted y: tensor([320040.4688])


y_test: tensor(550000.)
predicted y: tensor([499390.])


y_test: tensor(445000.)
predicted y: tensor([636474.5000])


y_test: tensor(620000.)
predicted y: tensor([601716.3750])
```

شکل87: نماش قیمت اصلی و پیش‌بینی شده پنج داده ارزیابی تصادفی

سوال پنجم

1. مجموعه داده Iris را فراخوانی کنید و روش‌های تحلیل داده‌ای که آموخته‌اید را روی آن به کار ببندید. داده‌ها را با نسبتی دلخواه و مناسب به مجموعه‌های آموزش و ارزیابی تقسیم کنید.

دیتاست Iris شامل اطلاعات سه نوع متفاوت گیاه (Setosa، Versicolour و Virginica) نظیر طول کاسبرگ، عرض کاسبرگ، طول گلبرگ و عرض گلبرگ است (4 ویژگی). تعداد کل نمونه‌ها برابر 150، 50 نمونه برای هر گیاه، است؛ به طوریکه پنجاه نمونه اول برای Setosa (کلاس0)، پنجاه نمونه دوم برای Versicolour(کلاس1) و پنجاه نمونه آخر برای Virginica(کلاس2) است. ماتریس‌ها ویژگی (X) و تارگت (y) را از دیتاست جدا کرده و با استفاده از دستور train_test_split آنها را به دو بخش آموزش و ارزیابی با test_size = 0.3 تقسیم می‌کنیم.

```
#5.1

import pandas as ps
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split

df = datasets.load_iris()
X = df['data']
y = df['target']
y = y.reshape((-1, 1))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

شکل88: فراخوانی دیتاست Iris، ایجاد ماتریس ویژگی و تارگت و ایجاد مجموعه‌های آموزش و ارزیابی

---

2. با استفاده از روش‌های آماده پایتون، سه مدل بر مبنای رگرسیون لجستیگ، MLP و شبکه‌های عصبی پایه شعاعی (RBF) را تعریف کرده و روی داده‌ها آموزش دهید. نتایج روی داده‌های ارزیابی را حداقل با چهار شاخص و ماتریس درهم‌ریختگی نشان داده و تحلیل کنید. در انتخاب فراپارامترها آزاد هستید؛ اما لازم است که نتایج را به صورت کامل مقایسه و تحلیل کنید.

در مرحله اول شاخص‌های ارزیابی موردنظر را به صورت توابع جداگانه تعریف می‌کنیم.

شاخص ارزیابی accuracy:

این شاخص نسبت تعداد پیش‌بینی‌های صحیح به کل پیش‌بینی‌ها را نشان می‌دهد (شکل89 و 90). اگر معیار accuracy برای یک کلاس‌بندی 90% باشد، بدین معناست که مدل از 100 نمونه، 90 نمونه را به درستی پیش‌بینی کرده است.

شاخص ارزیابی precision:

این شاخص نسبت تعداد پیش‌بینی‌های درست مثبت را به تعداد پیش‌بینی‌های مثبت نشان می‌دهد (شکل91). اگر معیار precision برای یک کلاس‌بندی 30% = 1/3 باشد، بدین معناست که مدل 3 بار یک کلاس مشخص را تشخیص داده است، اما تشخیص آن فقط یک بار درست بوده است. برای پیاده‌سازی تابع آن در پایتون، ابتدا معیار precision را برای هر کلاس به طور جداگانه محاسبه کرده و سپس با توجه به شکل92، معیارهای محاسبه شده را با هم جمع و تقسیم بر تعداد معیارها می‌کنیم تا معیار اصلی (برای کل فرآیند کلاس‌بندی) بدست آید (شکل93).

شاخص ارزیابی recall:

این شاخص نشان می‌دهد که چه تعداد از موارد مثبت واقعی را توانستیم با مدل خود پیش‌بینی کنیم (شکل94). اگر معیار recall برای یک کلاس‌بندی 20% باشد، بدین معناست که مدل تنها 1 نمونه را درست پیش‌بینی کرده است. برای پیاده‌سازی تابع آن در پایتون، ابتدا معیار recall را برای هر کلاس به طور جداگانه محاسبه کرده و سپس با توجه به شکل95، معیارهای محاسبه شده را با هم جمع و تقسیم بر تعداد معیارها می‌کنیم تا معیار اصلی (برای کل فرآیند کلاس‌بندی) بدست آید (شکل96).

شاخص ارزیابی F1 score:

این شاخص یک ایده ترکیبی در مورد معیارهای precision و recall می دهد و زمانی که Precision برابر با Recall باشد، حداکثر است (شکل97 و 98).

ماتریس درهم‌ریختگی:

برای تعریف تابع ماتریس درهم‌ریختگی، از کتابخانه sklearn.metrics تابع confusion_matrix را فراخوانی کرده و از آن استفاده می‌کنیم (شکل99).

$$\text{Accuracy} = \frac{\text{Correct predictions}}{\text{All predictions}}$$

شکل89: فرمول accuracy

```
def accuracy(yhat, y):
  n = 0
  for i in range(0, len(y)):
    if (y[i] == yhat[i]):
      n = n + 1
  a = n/len(y)
  return a
```

شکل90: پیاده‌سازی تابع accuracy

$$\text{Precision}_{\text{Class A}} = \frac{TP_{\text{Class A}}}{TP_{\text{Class A}} + FP_{\text{Class A}}}$$

شکل91: فرمول precision

$$\text{Precision}_{\text{Macro-average}} = \frac{\text{Precision}_{\text{Class A}} + \text{Precision}_{\text{Class B}} + \ldots \text{Precision}_{\text{Class N}}}{N}$$

شکل92: معیار precision میانگین

$$\text{Recall}_{\text{Class A}} = \frac{TP_{\text{Class A}}}{TP_{\text{Class A}} + FN_{\text{Class A}}}$$

شکل94: فرمول recall

$$\text{Recall}_{\text{Macro-average}} = \frac{\text{Recall}_{\text{Class A}} + \text{Recall}_{\text{Class B}} + \ldots \text{Recall}_{\text{Class N}}}{N}$$

شکل95: معیار recall میانگین

$$F1 = 2.\frac{Precision \times Recall}{Precision + Recall}$$

شکل97: فرمول F1 score

```
def F1score(p, r):
  a = (2*p*r) / (p + r)
  return a
```

شکل98: پیاده‌سازی تابع F1score

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
def confusion_m(y, yhat):
  cm = confusion_matrix(y, yhat)
  sns.heatmap(cm, annot = True, xticklabels = ['Setosa', 'Versicolour', 'Virginica'], yticklabels = ['Setosa', 'Versicolour', 'Virginica'])
  plt.ylabel('Prediction')
  plt.xlabel('Actual')
  plt.title('Confusion Matrix')
  plt.show()
```

شکل99: پیاده‌سازی تابع confusion_m

```python
def precision(yhat, y):
  TP_0 = 0
  FP_0 = 0
  TP_1 = 0
  FP_1 = 0
  TP_2 = 0
  FP_2 = 0
  for i in range(len(y)):
    if (y[i] == 0):
      if (yhat[i] == 0):
        TP_0 = TP_0 + 1
    if (yhat[i] == 0):
      if (y[i] != 0):
        FP_0 = FP_0 + 1
    if (y[i] == 1):
      if (yhat[i] == 1):
        TP_1 = TP_1 + 1
    if (yhat[i] == 1):
       if (y[i] != 1):
        FP_1 = FP_1 + 1
    if (y[i] == 2):
      if (yhat[i] == 2):
        TP_2 = TP_2 + 1
    if (yhat[i] == 2):
      if (y[i] != 2):
        FP_2 = FP_2 + 1
  a_0 = TP_0 / (TP_0 + FP_0)
  a_1 = TP_1 / (TP_1 + FP_1)
  a_2 = TP_2 / (TP_2 + FP_2)
  a = (a_0 + a_1 + a_2) / 3
  return a
```

شکل93: پیاده‌سازی تابع precision

```python
def recall(yhat, y):
  TP_0 = 0
  FN_0 = 0
  TP_1 = 0
  FN_1 = 0
  TP_2 = 0
  FN_2 = 0
  for i in range(len(y)):
    if (y[i] == 0):
      if (yhat[i] == 0):
        TP_0 = TP_0 + 1
      else:
        FN_0 = FN_0 + 1
    if (y[i] == 1):
      if (yhat[i] == 1):
        TP_1 = TP_1 + 1
      else:
        FN_1 = FN_1 + 1
    if (y[i] == 2):
      if (yhat[i] == 2):
        TP_2 = TP_2 + 1
      else:
        FN_2 = FN_2 + 1
  a_0 = TP_0 / (TP_0 + FN_0)
  a_1 = TP_1 / (TP_1 + FN_1)
  a_2 = TP_2 / (TP_2 + FN_2)
  a = (a_0 + a_1 + a_2) / 3
  return a
```

شکل96: پیاده‌سازی تابع recall

پس از تعریف شاخص‌های ارزیابی، به سراغ کلاس‌بندی می‌رویم.

کلاس‌بندی LogisticRegression (استفاده از کتابخانه پایتون):

از کتابخانه sklearn.linear_model تابع LogisticRegression را فراخوانی می‌کنیم و یک مدل با آن طبقه‌بندی می‌سازیم. سپس با استفاده از دستور model.fit(X_train, y_train) مدل را آموزش داده و با استفاده از دستور yhat = model.predict(X_test)، داده‌های ارزیابی را پیش‌بینی می‌کنیم. نتایج yhat را به همراه y_test به شاخص‌های ارزیابی داده و نتایج را نمایش می‌دهیم (شکل100).

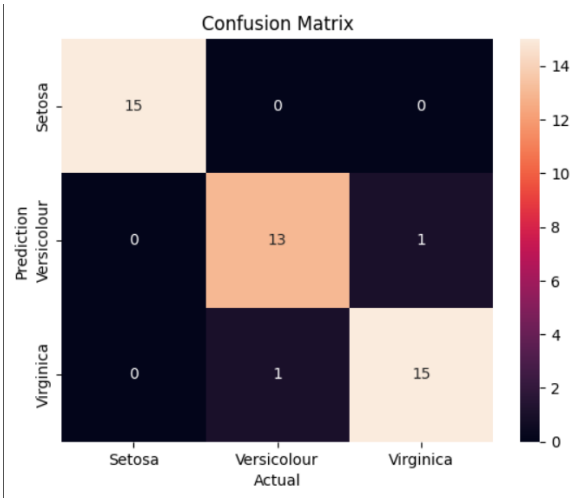
شکل101 نتایج ارزیابی و شکل102 ماتریس درهم‌ریختگی را نشان می‌دهد. همانطور که در شکل101 دیده می‌شود، تمام شاخص‌های ارزیابی دقت بالای 0.95 را نشان می‌دهند که نشانه عملکرد بسیار خوب LogisticRegression است. همچنین با توجه به ماتریس درهم‌ریختگی، مدل فقط در دو مورد پیش‌بینی اشتباه داشته است.

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(random_state = 73)
model.fit(X_train, y_train)
yhat = model.predict(X_test)
a1 = accuracy(yhat, y_test)
p = precision(yhat, y_test)
r = recall(yhat, y_test)
F = F1score(p, r)
cm = confusion_m(yhat, y_test)
print('\n' + '\n')
print("The 'accuracy' of 'LogisticRegression': " + str(a1))
print("The 'precision' of 'LogisticRegression': " + str(p))
print("The 'recall' of 'LogisticRegression': " + str(r))
print("The 'F1score' of 'LogisticRegression': " + str(F))
```

شکل100: استفاده از کتابخانه آماده LogisticRegression

```
The 'accuracy' of 'LogisticRegression': 0.9555555555555556
The 'precision' of 'LogisticRegression': 0.9553571428571429
The 'recall' of 'LogisticRegression': 0.9553571428571429
The 'F1score' of 'LogisticRegression': 0.9553571428571429
```

شکل101: نمایش نتایج ارزیابی کتابخانه آماده LogisticRegression



شکل102: ماتریس درهم‌ریختگی کتابخانه آماده LogisticRegression

کلاس‌بندی LogisticRegression (بدون استفاده از کتابخانه پایتون):

در این مرحله، یک تابع predict، یک تابع gradient و یک تابع gradient_descent تعریف می‌شود. سپس ماتریسی از w به صورت تصادفی انتخاب شده و با گام آموزش eta = 0.001 و تعداد دوره آموزش epock = 500، فرآیند آموزش انجام می‌شود. پس از آموزش مدل، داده‌های ارزیابی به تابع predict داده می‌شوند تا تارگت آنها پیش‌بینی شود (شکل103).

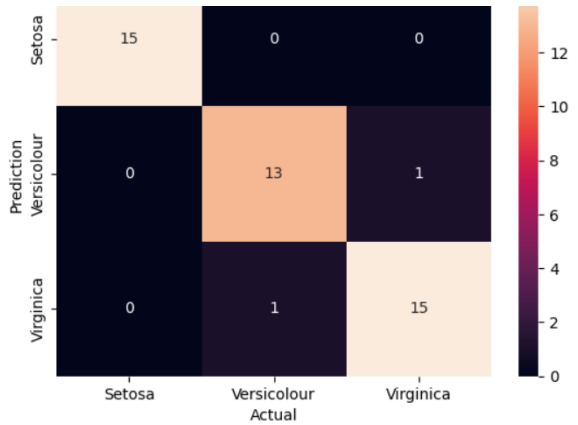
شکل104 نتایج ارزیابی و شکل105 ماتریس درهم‌ریختگی را نشان می‌دهد. نتایج بدست آمده در این مرحله دقیقا مشابه نتایج بدست آمده از کتابخانه آماده LogisticRegression است.

```python
def predict(x, w):
  y_ = x @ w
  y_hat = 1 / (1 + np.exp(-(y_)))
  return y_hat

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

def gradient_descent(w, eta, g):
  w -= eta*g
  return w

w = np.random.randn(4, 1)

eta = 0.001
epock = 500

for i in range(0, epock):
  y_hat = predict(X_train, w)
  g = gradient(X_train, y_train, y_hat)
  w = gradient_descent(w, eta, g)

y_hat = predict(X_test, w)
cm = confusion_m(y_test, yhat)
print('\n' + '\n')
print("The 'accuracy' of 'LogisticRegression': " + str(a1))
print("The 'precision' of 'LogisticRegression': " + str(p))
print("The 'recall' of 'LogisticRegression': " + str(r))
print("The 'F1score' of 'LogisticRegression': " + str(F))
```

شکل103: پیاده‌سازی LogisticRegression به صورت scratch

```
The 'accuracy' of 'LogisticRegression': 0.9555555555555556
The 'precision' of 'LogisticRegression': 0.9553571428571429
The 'recall' of 'LogisticRegression': 0.9553571428571429
The 'F1score' of 'LogisticRegression': 0.9553571428571429
```

شکل104: نمایش نتایج ارزیابی LogisticRegression به صورت scratch



شکل105: ماتریس درهم‌ریختگی LogisticRegression به صورت scratch

کلاس‌بندی MLP (استفاده از کتابخانه پایتون):

ابتدا با استفاده از کتابخانه آماده sklearn.neural_network تابع MLPClassifier را فراخوانی کرده و سپس یک مدل از آن را می‌سازیم. تعداد لایه‌های پنهان را برابر 3 و تعداد نورون‌های هر لایه پنهان را به ترتیب 50، 20 و 5 انتخاب می‌کنیم. همچنین تعداد دوره آموزشی را برابر max_iter = 500 قرار می‌دهیم. پس از ایجاد مدل، با استفاده از دستور model.fit(X_train, y_train) مدل را آموزش داده و با استفاده از دستور yhat = model.predict(X_test)، داده‌های ارزیابی را پیش‌بینی می‌کنیم. درنهایت نتایج yhat را به همراه y_test به شاخص‌های ارزیابی داده و نتایج را نمایش می‌دهیم (شکل106).

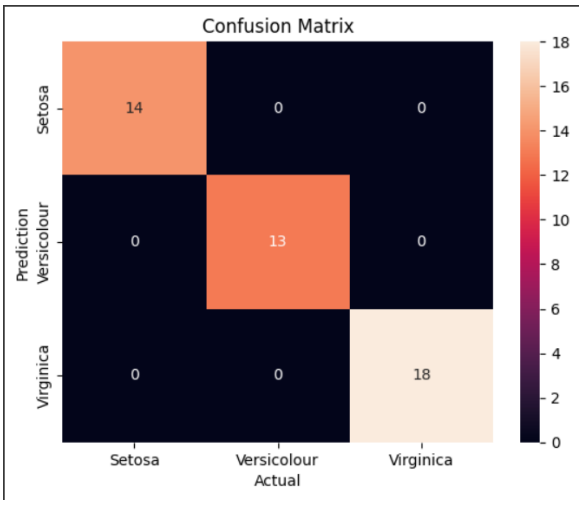
شکل107 نتایج ارزیابی و شکل108 ماتریس درهم‌ریختگی را نشان می‌دهد. همانطور که در شکل107 دیده می‌شود، تمام شاخص‌های ارزیابی دقت 1 را نشان می‌دهند که بسیار ایده‌آل است. همچنین با توجه به ماتریس درهم‌ریختگی، مدل همه نمونه‌ها را درست پیش‌بینی کرده است.

```
from sklearn.neural_network import MLPClassifier
model = MLPClassifier(hidden_layer_sizes = (50, 20, 5), max_iter = 500, random_state = 73)
model.fit(X_train, y_train)
yhat = model.predict(X_test)
a = accuracy(yhat, y_test)
p = precision(yhat, y_test)
r = recall(yhat, y_test)
F = F1score(p, r)
cm = confusion_m(y_test, yhat)
print('\n' + '\n')
print("The 'accuracy' of 'MLP': " + str(a))
print("The 'precision' of 'MLP': " + str(p))
print("The 'recall' of 'MLP': " + str(r))
print("The 'F1score' of 'MLP': " + str(F))
```

شکل106: استفاده از کتابخانه آماده MLPClassifier

```
The 'accuracy' of 'MLP': 1.0
The 'precision' of 'MLP': 1.0
The 'recall' of 'MLP': 1.0
The 'F1score' of 'MLP': 1.0
```

شکل107: نمایش نتایج ارزیابی کتابخانه آماده MLPClassifier



شکل108: ماتریس درهم‌ریختگی کتابخانه آماده MLPClassifier

کلاس‌بندی RBF (استفاده از کتابخانه پایتون):

از کتابخانه sklearn.svm تابع SVC را فراخوانی کرده و با استفاده از دستور model = SVC(kernel = ‘rbf’) یک مدل RBF ایجاد می‌کنیم. پس از ایجاد مدل، با استفاده از دستور model.fit(X_train, y_train) مدل را آموزش داده و با استفاده از دستور yhat = model.predict(X_test)، داده‌های ارزیابی را پیش‌بینی می‌کنیم. درنهایت نتایج yhat را به همراه y_test به شاخص‌های ارزیابی داده و نتایج را نمایش می‌دهیم (شکل109).

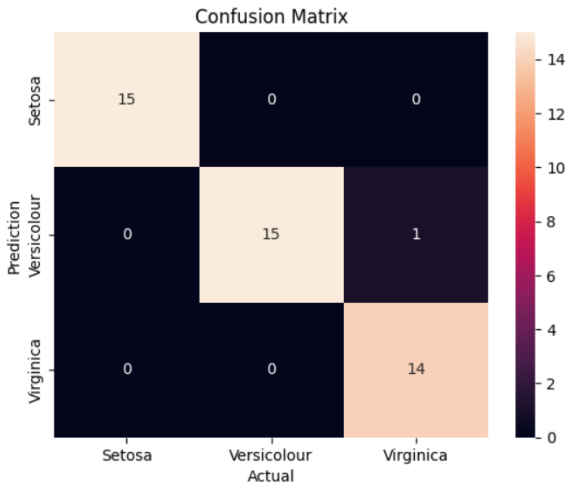
شکل110 نتایج ارزیابی و شکل111 ماتریس درهم‌ریختگی را نشان می‌دهد. همانطور که در شکل110 دیده می‌شود، شاخص ارزیابی بیشتر از 0.97 است که نشان‌دهنده عملکرد بسیار خوب RBF است. همچنین با توجه به ماتریس درهم‌ریختگی، مدل فقط در یک مورد پیش‌بینی اشتباه داشته است.

```
from sklearn.svm import SVC
model = SVC(kernel = 'rbf')
model.fit(X_train, y_train)
yhat = model.predict(X_test)
a = accuracy(yhat, y_test)
p = precision(yhat, y_test)
r = recall(yhat, y_test)
F = F1score(p, r)
cm = confusion_m(y_test, yhat)
print('\n' + '\n')
print("The 'accuracy' of 'MLP': " + str(a))
print("The 'precision' of 'MLP': " + str(p))
print("The 'recall' of 'MLP': " + str(r))
print("The 'F1score' of 'MLP': " + str(F))
```

شکل109: استفاده از کتابخانه آماده SVC

```
The 'accuracy' of 'MLP': 0.9777777777777777
The 'precision' of 'MLP': 0.9777777777777779
The 'recall' of 'MLP': 0.9791666666666666
```

شکل110: نمایش نتایج ارزیابی کتابخانه آماده SVC



شکل111: ماتریس درهم‌ریختگی کتابخانه آماده SVC

به عنوان نتیجه‌گیری این بخش، عملکرد شبکه عصبی MLP از دو شبکه RBF و LogisticRegression بهتر است. همچنین در مقایسه بین RBF و LogisticRegression، RBF از دقت بالاتری برخوردار است.