|  |  |
| --- | --- |
| نام و نام خانوادگی | **محمد پویا افشاری – علیرضا اسمعیل زاده** |
| شماره دانشجویی | **810198351-810198577** |
| تاریخ ارسال گزارش | **۱۴۰۱.۰۷.۰۱** |

|  |  |  |
| --- | --- | --- |
|  | **به نام خدا**  **دانشگاه تهران**  **دانشکده‌ مهندسی برق و کامپیوتر** |  |
| **درس شبکه‌های عصبی و یادگیری عمیق**  **تمرین اول** | | |

**فهرست**

[**پاسخ 1**. **شبکه عصبی Mcculloch-Pitts** 4](#_Toc150368706)

[۱-۱. نمایشگر 7-segment 4](#_Toc150368707)

[۱-۱-۱. وظیفه 4](#_Toc150368708)

[۱-۱-۲. معماری مدل 4](#_Toc150368709)

[۱-۱-۳. اضافه کردن کتابخانه های مورد نیاز 4](#_Toc150368712)

[۱-۲. شبکه عصبی یک لایه 4](#_Toc150368713)

[۱-۲-۱. معماری 5](#_Toc150368714)

[۱-۲-۲. Forward Propagation 5](#_Toc150368715)

[۱-۲-۳. 5](#_Toc150368716)

[۱-۳. شبکه عصبی دو لایه 6](#_Toc150368721)

[۱-۳-۱. معماری 6](#_Toc150368722)

[۱-۳-۲. کد های این بخش 6](#_Toc150368724)

[**پاسخ ۲** **– آموزش شبکه های Adaline و Madaline** 7](#_Toc150368727)

[۱-۲. **Adaline** 7](#_Toc150368728)

[۱-۲-۱. **رسم داده ها و آموزش مدل** 7](#_Toc150368729)

[۱-۲-۲. **عنبیه Versicolour** 12](#_Toc150368730)

[۲-۲. **Madaline** 13](#_Toc150368731)

[۲-۲-۱. **توضیح الگورتیم های MRI و MRII** 13](#_Toc150368732)

[۲-۲-۲. **پیاده سازی Madaline** 16](#_Toc150368733)

[۲-۲-۳. **بررسی نمودار های حاصل شده بر اساس تعداد ایپاک های متفاوت** 23](#_Toc150368734)

[**پاسخ ۳** **– خوشه بندی با استفاده از Autoencoder** 24](#_Toc150368735)

[۱-۳-۱. **ایجاد شبکه عمیق اتوانکدر** 24](#_Toc150368736)

[۱-۳-۲. **آموزش اتوانکدر** 26](#_Toc150368737)

[۱-۳-۳. **خوشه بندی با استفاده از مدل K-Means** 28](#_Toc150368738)

[**پاسخ ۴** **–شبکه‌ی Multi-Layer Perceptron** 29](#_Toc150368739)

[۱-۴**-۱**. **وظیفه** 29](#_Toc150368740)

[۱-۴**-۲**. **پیش پردازش** 29](#_Toc150368741)

[. ۱-۴**-۲**-۱. **کد های مربوط به این بخش** 29](#_Toc150368743)

[۱-۴**-۳**. **شبکه Teacher** 33](#_Toc150368744)

[. ۱-۴**-۳**-۱. **کد های مربوط به این بخش** 33](#_Toc150368745)

[. ۱-۴**-۳**-**۲**. **Logit** 35](#_Toc150368746)

[۱-۴**-**۴. **شبکه Student** 35](#_Toc150368747)

[. ۱-۴**-**۴-۱. **کد های مربوط به این بخش** 35](#_Toc150368748)

[۱-۴**-**۵. Knowledge Distillation 38](#_Toc150368749)

[۱-۴**-**۵-۱. **کد های مربوط به این بخش** 38](#_Toc150368750)

[۱-۴**-**۵-۱. **کد های مربوط به این بخش** 38](#_Toc150368751)

[. ۱-۴**-**۵-**۲**. تعداد پیشبینی های غلط misclassifications مدل Student بخش قبل روی داده های آزمون را با مدل Student فعلی مقایسه کرده و نتیجه را گزارش کنید 39](#_Toc150368752)

[. ۱-۴**-**۵-**۳**. Accuracy هر دو مدل Student روی داده های train در طول اموزش را روی یک نمودار و در کنار هم رسم کرده و توزیح بدهید knowledge distillation چه تاثیراتی داشته است 39](#_Toc150368753)

**شکل‌ها**

شکل 1. عنوان تصویر نمونه 4

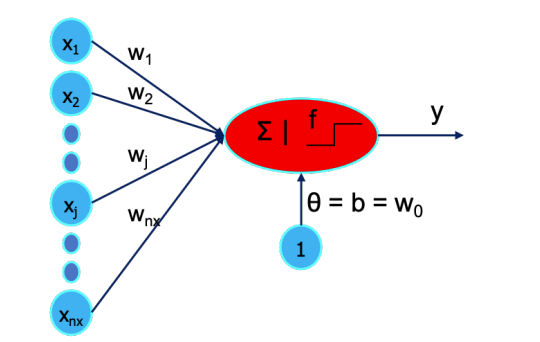
# **پاسخ 1**. **شبکه عصبی Mcculloch-Pitts**

۱-۱. نمایشگر 7-segment

۱-۱-۱. وظیفه

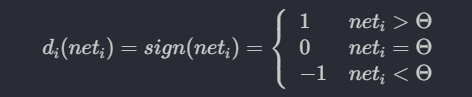
در این مساله باید به کمک نرون های مک کلانژ پیتز مساله تشخیص اعداد بین 6 تا 9 را حل کنیم. در بخش اول این مساله را بدون لایه مخفی و در بخش دوم با یک لایه مخفی انجام میدهیم.

۱-۱-۲. معماری مدل

****

شکل 1. معماری مدل Mcculloch pits

**درصورتی که در این حالت Theta = 0 برابر threshold در نظر گرفته بشود به ازای خروجی هر نرون خواهیم داشت:**

****

شکل 2. فرمول محاسبه فعال شدن نرون Mcculoch pits

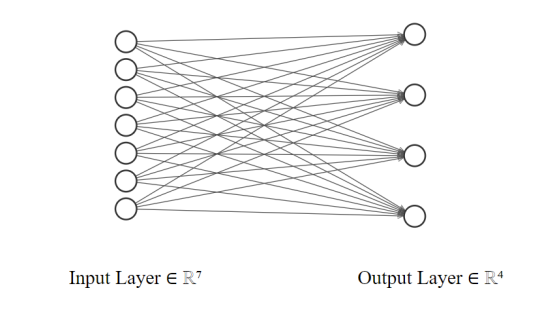
**وزن و بایاس هر نورون خروجی با توجه به ضریب آنها در معادله خط تنظیم می شود.**

۱-۱-۳. اضافه کردن کتابخانه های مورد نیاز

import numpy as np

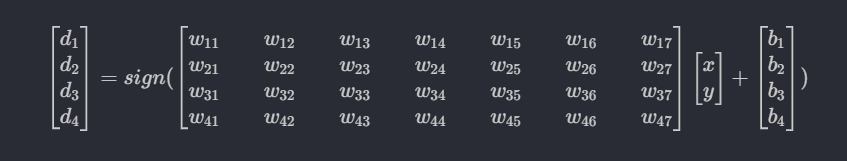
import pandas as pd

۱-۲. شبکه عصبی یک لایه

۱-۲-۱. معماری

شکل 3. مدل شبکه طراحی شده بدون لایه مخفی

۱-۲-۲. Forward Propagation

۱-۲-۳. کد های این بخش

شکل 4. Forward Propagation

**ایجاد شبکه عصبی**

class NeuralNetwork:

    def \_\_init\_\_(self, weights, threshold):

        self.weights = weights

        self.threshold = threshold

    def forward(self, inputs):

*# mult = np.sign(np.dot(self.weights, x) + self.biases)*

        mult = sum(w \* x for w, x in zip(self.weights, inputs))

        result = self.activate(mult)

        return result

    def activate(self, mult) -> bool:

        return 1 if mult >= self.threshold else 0

**تعیین وزن و بایاس و threshold:**

*# Define the weights and biases based on the settings you provided*

weights\_Q6 = [2, -9, 2, 2, 2, 2, 2]  *# Weight vector for O1 (6)*

weights\_Q7 = [1, 1, 1, -10, -10, -10, -10]  *# Weight vector for O2 (7)*

weights\_Q8 = [1, 1, 1, 1, 1, 1, 1]  *# Weight vector for O3 (8)*

weights\_Q9 = [1, 1, 1, 1, -10, 1, 1]   *# Weight vector for O4 (9)*

threshold\_Q6 = 7

threshold\_Q7 = 3

threshold\_Q8 = 7

threshold\_Q9 = 6

**ورودی:**

inputs = [[1, 0, 1, 1, 1, 1, 1],  *# Weight vector for O1 (6)*

          [1, 1, 1, 0, 0, 0, 0],  *# Weight vector for O1 (7)*

          [1, 1, 1, 1, 1, 1, 1],  *# Weight vector for O1 (8)*

          [1, 1, 1, 1, 0, 1, 1]  *# Weight vector for O1 (9)*

          ]

**تست خروجی:**

Neuron1 = NeuralNetwork(weights\_Q6, threshold\_Q6)

Neuron2 = NeuralNetwork(weights\_Q7, threshold\_Q7)

Neuron3 = NeuralNetwork(weights\_Q8, threshold\_Q8)

Neuron4 = NeuralNetwork(weights\_Q9, threshold\_Q9)

for i in range(4):

    print([Neuron1.forward(inputs[i]),

           Neuron2.forward(inputs[i]),

           Neuron3.forward(inputs[i]),

           Neuron4.forward(inputs[i])

           ]

          )

[1, 0, 0, 0]

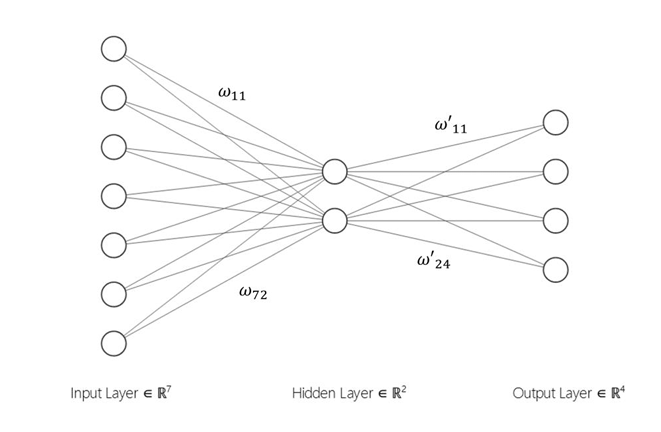
[0, 1, 0, 0]

[0, 0, 1, 0]

[0, 0, 0, 1]

۱-۳. شبکه عصبی دو لایه

۱-۳-۱. معماری

******با در نظر گرفتن دو نرون برای لایه مخفی مطابق شکل زیر داریم:**

شکل 5. مدل طراحی شده برای شبکه با یک لایه مخفی

۱-۳-۲. کد های این بخش

**استفاده از شبکه قبلی با اضافه کردن نود های لایه مخفی و تغییر مقادیر شبکه**

*# Threshold*

th\_layer1 = 4

*# Weights:*

weights\_hidden\_1 = [1, 1, 0, 0, 0, 1, 1]

weights\_hidden\_2 = [0, 0, 1, 1, 1, 0, 1]

Hidden\_neuron\_1 = NeuralNetwork(weights\_hidden\_1, th\_layer1)

Hidden\_neuron\_2 = NeuralNetwork(weights\_hidden\_2, th\_layer1)

*# Threshold*

final\_th1 = 1

final\_th2 = 0

final\_th3 = 2

final\_th4 = 1

*# Weights*

weights\_final\_1 = [-1, 1]

weights\_final\_2 = [-1, -1]

weights\_final\_3 = [1, 1]

weights\_final\_4 = [1, -1]

final\_neuron\_1 = NeuralNetwork(weights\_final\_1, final\_th1)

final\_neuron\_2 = NeuralNetwork(weights\_final\_2, final\_th2)

final\_neuron\_3 = NeuralNetwork(weights\_final\_3, final\_th3)

final\_neuron\_4 = NeuralNetwork(weights\_final\_4, final\_th4)

**بررسی و تست خروجی**

for i in range(4):

    output\_hidden = [Hidden\_neuron\_1.forward(inputs[i]),

                     Hidden\_neuron\_2.forward(inputs[i])

                     ]

    print([final\_neuron\_1.forward(output\_hidden),

           final\_neuron\_2.forward(output\_hidden),

           final\_neuron\_3.forward(output\_hidden),

           final\_neuron\_4.forward(output\_hidden)

           ])

[1, 0, 0, 0]

[0, 1, 0, 0]

[0, 0, 1, 0]

[0, 0, 0, 1]

# **پاسخ ۲** **– آموزش شبکه های Adaline و Madaline**

## ۱-۲. **Adaline**

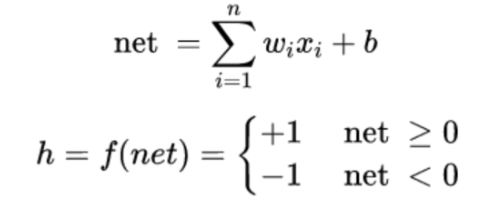
## ۱-۲-۱. **رسم داده ها و آموزش مدل**

مدل معماری:

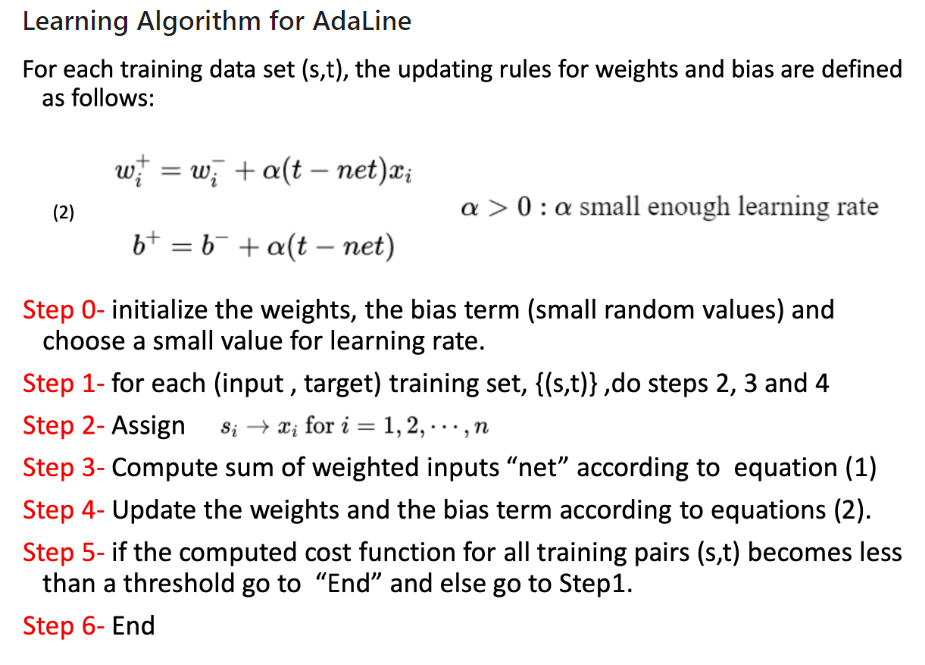


شکل 6. مدل طراحی شده برای Adaline

الگوریتم یادگیری Adaline:



شکل 7. فرمول محاسبه تابع فعال ساز و خروجی هر نورون Adaline



شکل 8. الگوریتم پیاده سازی Adaline

کد اضافه کردن کتابخانه های مورد نیاز

from sklearn import datasets

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

کد افزودن مجموعه داده های عنبیه

# Load the Iris dataset

iris = datasets.load\_iris()

# Access the features and target

X = iris.data # Features

y = iris.target # Target

کد ترسیم داده ها

# Create a scatter plot for Sepal length vs. Sepal width

plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1],

label="Setosa", c="r", marker="o")

plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1],

label="Versicolor", c="g", marker="x")

plt.scatter(X[y == 2][:, 0], X[y == 2][:, 1],

label="Virginica", c="b", marker="s")

# Add labels and a legend

plt.xlabel("Sepal Length (cm)")

plt.ylabel("Sepal Width (cm)")

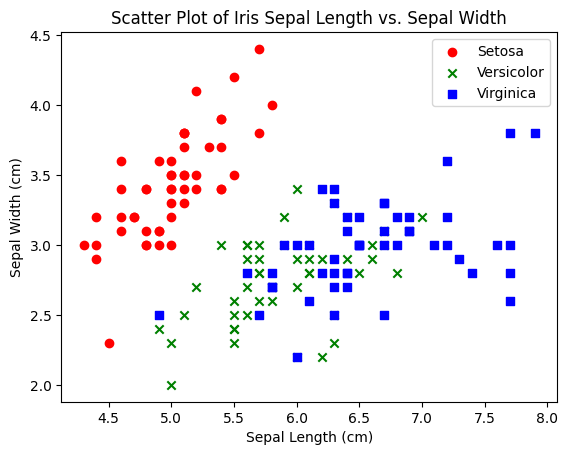
plt.legend(loc="upper right")

# Show the plot

plt.title("Scatter Plot of Iris Sepal Length vs. Sepal Width")

plt.show()

پراکندگی داده ها در دو بعد:



شکل 9. نمودار پراکندگی داده ها در دو بعد

کد استخراج دو ویژگی اول (Sepal-width Sepal-lenght) و تبدیل لیبل های 0 به 1-

# Using only the first 2 features (Sepal length and Sepal width)

X = iris.data[:100, [0, 1]]

y = iris.target[:100]

y = np.where(y == 0, -1, 1) # Setosa (-1) and Versicolor (1)

کد مدل Adaline براساس الگوریتم

# Adaline class

class Adaline:

def \_\_init\_\_(self, learning\_rate=0.01, n\_iterations=100):

self.learning\_rate = learning\_rate

self.n\_iterations = n\_iterations

def fit(self, X, y):

self.weights = np.zeros(1 + X.shape[1])

self.cost = []

for \_ in range(self.n\_iterations):

net = self.net\_input(X)

errors = (y - net)

self.weights[1:] += self.learning\_rate \* X.T.dot(errors)

self.weights[0] += self.learning\_rate \* errors.sum()

cost = (errors\*\*2).sum() / 2.0

self.cost.append(cost)

return self

def net\_input(self, X):

return np.dot(X, self.weights[1:]) + self.weights[0]

def activation(self, X):

# Identity activation function

return self.net\_input(X)

def predict(self, X):

return np.where(self.activation(self.net\_input(X)) >= 0.0, 1, -1)

کد آموزش مدل بر اساس ضریب یادگیری 0.0001 و تعداد ایپاک های 600 و ترسیم نمودار تغییرات خطا

# Initialize and train the Adaline model

learning\_rate = 0.0001

n\_iterations = 600

adaline = Adaline(learning\_rate, n\_iterations)

# Fit the model to the training data

adaline.fit(X, y)

# Plot the error after each epoch

plt.plot(range(1, len(adaline.cost) + 1), adaline.cost, marker='o')

plt.xlabel('Epochs')

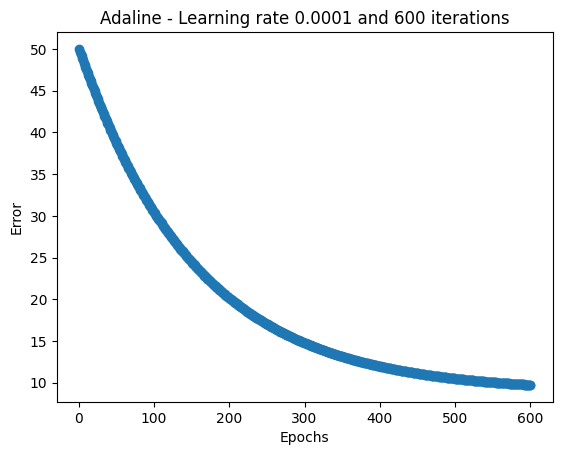
plt.ylabel('Error')

plt.title(

'Adaline - Learning rate {} and {} iterations'.format(learning\_rate, n\_iterations))

plt.show()

نمودار تغییرات خطا



شکل 10. نمودار تغییرات خطا برای جداسازی عنبیه Setosa

## ۱-۲-۲. **عنبیه Versicolour**

کد جداسازی عنبیه versicolour و non-versicolour لیبل عنبیه مورد نظر را -1 در نظر می گیرم و با همان ضریب یادگیری و تعداد ایپاک مدل را آموزش میدهیم

iris = datasets.load\_iris()

# Using only the first 2 features (Sepal length and Sepal width)

X = iris.data[50:150, [0, 1]]

y = iris.target[50:150]

y = np.where(y == 1, 1, -1) # Versicolor (1) and Non-Versicolor (-1)

# Initialize and train the Adaline model

learning\_rate = 0.0001

n\_iterations = 600

adaline = Adaline(learning\_rate, n\_iterations)

# Fit the model to the training data

adaline.fit(X, y)

# Plot the error after each epoch

plt.plot(range(1, len(adaline.cost) + 1), adaline.cost, marker='o')

plt.xlabel('Epochs')

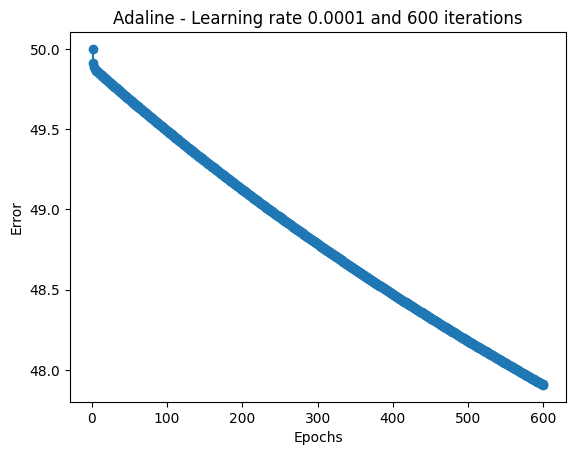
plt.ylabel('Error')

plt.title(

'Adaline - Learning rate {} and {} iterations'.format(learning\_rate, n\_iterations))

plt.show()

نمودار تغییرات خطا



شکل 11. نمودار تغییرات خطا برای جداسازی عنبیه Versicolour

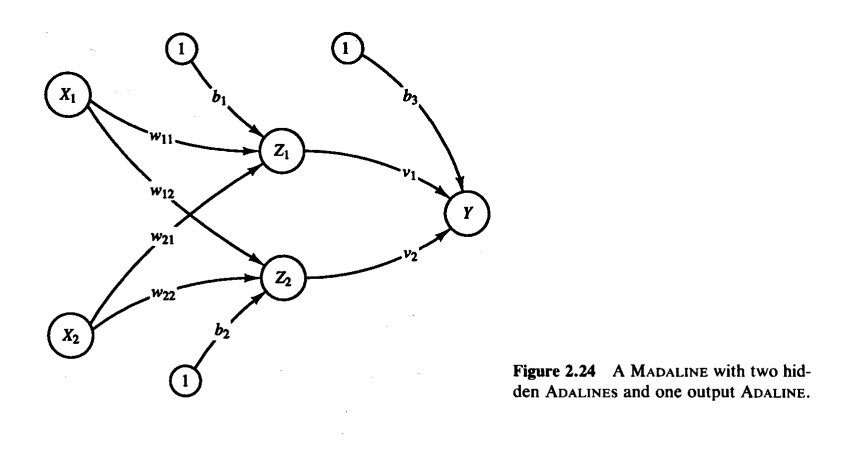
خیر، با توجه به نمودار تغییرات خطا بخش الف و ب همان طور که مشخص است خطا در بخش ب حدود ۵۰ درصد است که در مقایسه با بخش الف که زیر ۱۰ درصد است مقدار قابل توجهی است و نشان دهنده آن است که versicolour به دلیل همبستگی بسیار زیاد با verginica جداپذیری بسیار پایینی دارد.

## ۲-۲. **Madaline**

## ۲-۲-۱. **توضیح الگورتیم های MRI و MRII**

الگوریتم یادگیری MRI

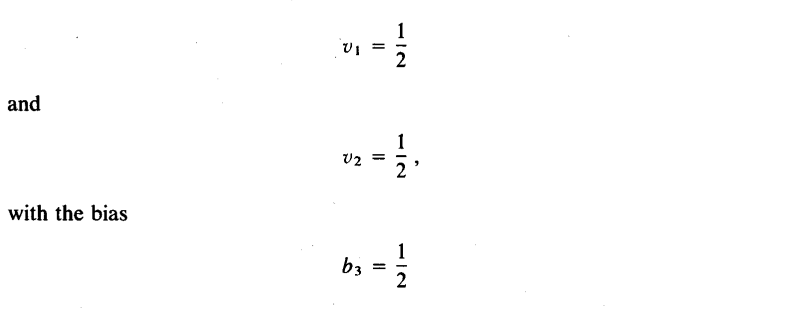
شکل اصلی آموزش Madaline که در سال 1960 توسط هاف و ویدو معرفی شد می گوید فقط وزن های Adaline تنظیم می شود و واحدهای خروجی ثابت هستند.



شکل ۱۲. مدل معماری Madaline

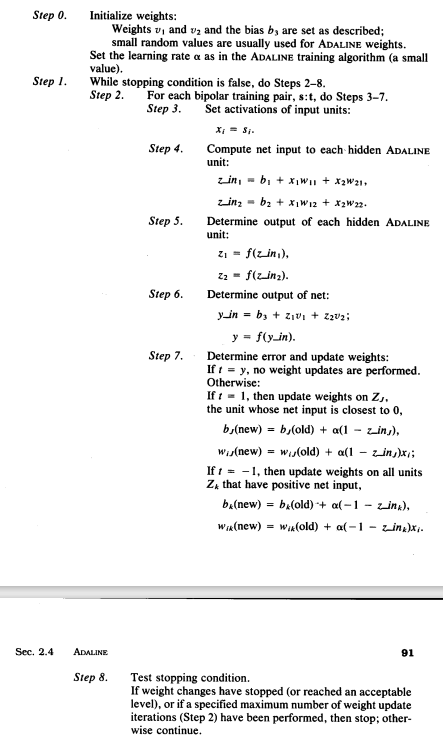
وزن‌های v1، v2 و بایاس b3 که وارد واحد خروجی Y می‌شوند، به‌گونه‌ای تعیین می‌شوند که پاسخ واحد Y1 باشد اگر سیگنالی که از Z1 یا Z2 یا هر دو دریافت می‌کند 1 باشد و زمانی که Z1 و Z2 -1 باشد. هر دو هستند.

وزن های Y عبارتند از:



شکل ۱۳. بایاس و وزن های نورون خروجی Madaline

وزن‌های آدالین اول (w11 و w21) و وزنه‌های لایه پنهان دوم آدالین (w12 و w22) طبق الگوریتم تنظیم می‌شوند.

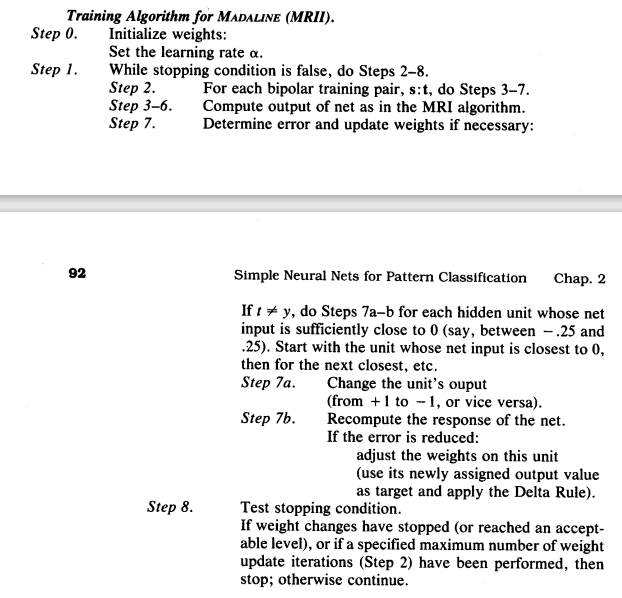


شکل ۱۴. الگوریتم به روزرسانی وزن ها MRI

الگوریتم یادگیری MRII

در این الگوریتم که در سال ۱۹۸۷ توسط Baxter, Winter, Widow معرفی شد همه ی وزن ها را تنظیم می کند.

الگوریتم زیر برای به روزرسانی وزن ها است:



شکل ۱۵. الگوریتم به روزرسانی وزن ها MRII

## ۲-۲-۲. **پیاده سازی Madaline**

کد افزودن کتابخانه ها

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import make\_moons

from sklearn.metrics import accuracy\_score

import copy

کد افزودن دیتاست و دسته بندی به دیتای آموزش و تست

X\_moons, Y\_moons = make\_moons(n\_samples=500, noise=0.1, random\_state=42)

X\_train\_moons, X\_test\_moons, Y\_train\_moons, Y\_test\_moons = train\_test\_split(X\_moons,

Y\_moons, test\_size=0.5, random\_state=3)

کد ترسیم نمودار داده ها

plt.scatter(X\_moons[Y\_moons == 0][:, 0], X\_moons[Y\_moons == 0][:, 1], color='red',

label='Class 0')

plt.scatter(X\_moons[Y\_moons == 1][:, 0], X\_moons[Y\_moons == 1][:, 1], color='blue',

label='Class 1')

plt.xlabel('Feature 1')

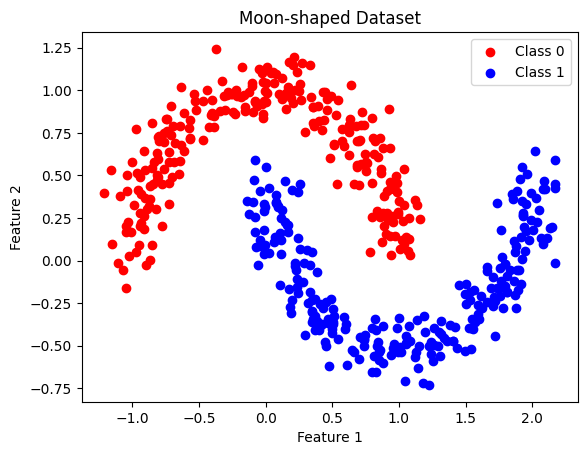
plt.ylabel('Feature 2')

plt.legend()

plt.title("Moon-shaped Dataset")

plt.show()

نمودار پراکندگی داده ها



شکل ۱۶. نمودار پراکندگی داده ها

کد پیاده سازی مدل Madaline

class Madaline:

def \_\_init\_\_(self, num\_l, n\_iter=500, learning\_rate=0.001):

self.n\_iter = n\_iter

self.learning\_rate = learning\_rate

np.random.seed(100)

self.weights = np.random.rand(num\_l, 2)

self.v = [1] \* num\_l

self.b = np.random.rand(num\_l, 1)

self.b2 = num\_l - 1

def fit(self, X, Y):

count = 0

self.cost\_ = []

for \_ in range(self.n\_iter):

old\_weights = copy.deepcopy(self.weights)

for x, label in zip(X, Y):

z\_in = np.array([np.matmul(x, self.weights.T)]).T + self.b

z = np.heaviside(z\_in, 1) \* 2 - 1

y\_in = np.dot(np.squeeze(z), np.squeeze(self.v)) + self.b2

y = np.heaviside(y\_in, 1) \* 2 - 1

if y != label:

if label == 1:

z\_j = max(z\_in)

ind = np.where(z\_in == z\_j)

self.weights[ind, :] = self.weights[ind, :] + \

self.learning\_rate \* (1 - z\_in[ind]) \* np.array(x)

self.b[ind] = self.b[ind] + \

self.learning\_rate \* (1 - z\_in[ind])

else:

ind = [i for i, x in enumerate(z\_in) if x > 0]

for indice in ind:

self.weights[indice, :] = self.weights[indice, :] + \

self.learning\_rate \* \

(-1 - z\_in[indice]) \* np.array(x)

self.b[indice] = self.b[indice] + \

self.learning\_rate \* (-1 - z\_in[indice])

self.cost\_.append(self.error(self.predict(X), Y))

if np.all((self.weights - old\_weights) < 0.0001) and \_ > 10:

count += 1

if count > 10:

print("Total iterations with",

len(self.b), "lines:", \_ + 1)

break

def predict(self, X):

y = []

for x in X:

z\_in = np.array([np.matmul(x, self.weights.T)]).T + self.b

z = np.heaviside(z\_in, 1) \* 2 - 1

y\_in = np.dot(np.squeeze(z), np.squeeze(self.v)) + self.b2

y.append(np.heaviside(y\_in, 1) \* 2 - 1)

return y

def error(self, net, t):

self.err = ((t - net) \*\* 2).sum() / 2.

return self.err

کد تغییر لیبل از 0 به -1 و تغییر داده های آموزش و تست

# Change label=0 to label=-1

Y\_moons[Y\_moons == 0] = -1

# Change X\_train\_moons, X\_test\_moons, Y\_train\_moons, Y\_test\_moons based on lable=-1

X\_train\_moons, X\_test\_moons, Y\_train\_moons, Y\_test\_moons = train\_test\_split(X\_moons,

Y\_moons, test\_size=0.5, random\_state=3)

کد آموزش و گرفتن خروجی از مدل Madaline براساس تعداد ایپاک های ۳، ۵ و ۸ و ترسیم نمودار تغییرات خطا و خطوط جداکننده Madaline

import matplotlib.pyplot as plt

neuron\_counts = [3, 5, 8]

for i in range(len(neuron\_counts)):

# Define Madaline with a different number of hidden layer neurons

madaline = Madaline(neuron\_counts[i], learning\_rate=0.001)

# Fit Madaline on X\_train and Y\_train

madaline.fit(X\_train\_moons, Y\_train\_moons)

#predict x\_test with madaline model

pred = madaline.predict(X\_test\_moons)

# Calculate Madaline model accuracy

from sklearn.metrics import accuracy\_score

acc = accuracy\_score(pred, Y\_test\_moons) # Assuming you have Y\_test\_moons

print("Accuracy of prediction for", neuron\_counts[i], "neurons is:", acc)

# Plot the moon shapes on the first plot

# Plot the lines produced by the Madaline model on the same plot

for i in range(neuron\_counts[i]):

plt.axline((0, float(-madaline.b[i] / madaline.weights[i][1])), slope=-madaline.weights[i][0] / madaline.weights[i][1], color="r", linestyle='dashed')

plt.scatter(X\_moons[:,0], X\_moons[:,1], c=['b' if y == 1 else 'r' for y in Y\_moons.reshape(-1)])

plt.title('Moon-shaped Dataset')

plt.xlim([-2, 2]) # Adjust the x-axis limits as needed

plt.ylim([-2, 2]) # Adjust the y-axis limits as needed

plt.show()

# Plot the loss (error) on the second plot

plt.plot(madaline.cost\_)

plt.title('Loss')

plt.xlabel('Epoch')

plt.ylabel('Error')

plt.grid()

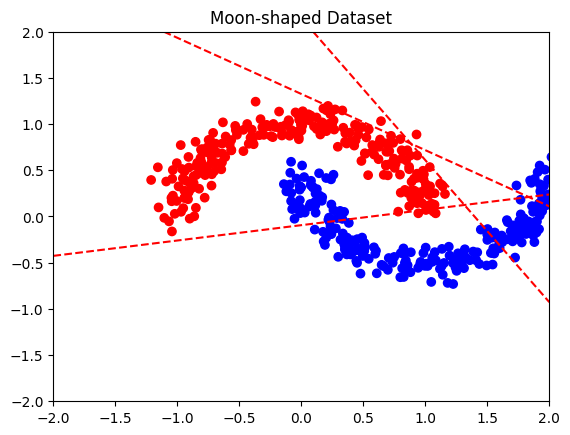
# Set the x and y limits for the second plot

plt.xlim([0, len(madaline.cost\_)]) # Adjust the x-axis limits for the loss plot

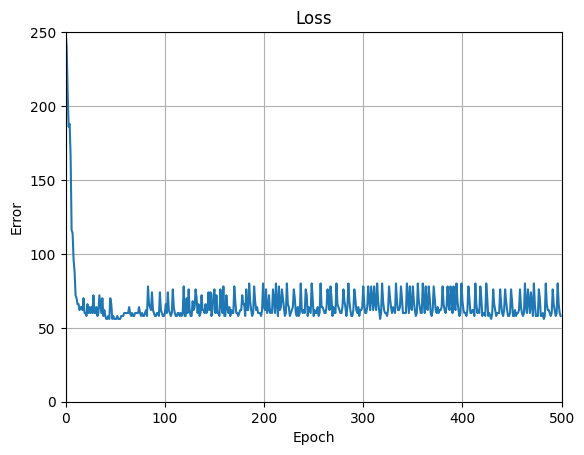
plt.ylim([0, max(madaline.cost\_)]) # Adjust the y-axis limits for the loss plot

plt.show()

Accuracy of prediction for 3 neurons is: 0.888

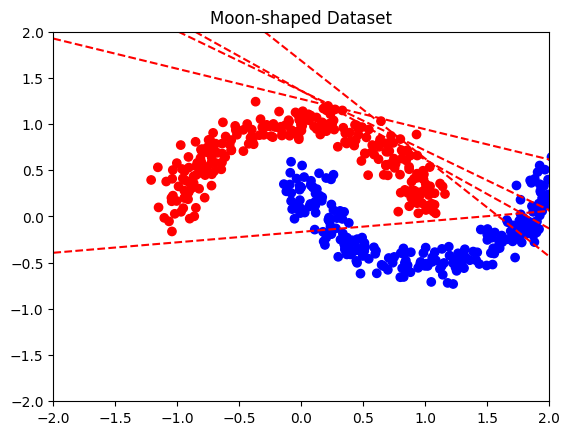


شکل ۱۷. خروجی مدل با ۳ نورون

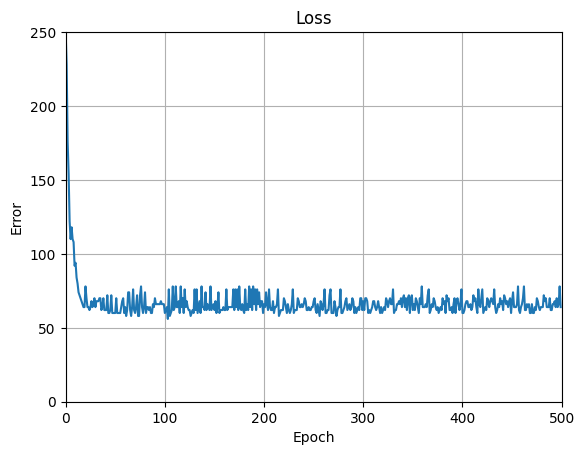


شکل ۱۸. نمودار تغییرات خطا با ۳ نورون

Accuracy of prediction for 5 neurons is: 0.888

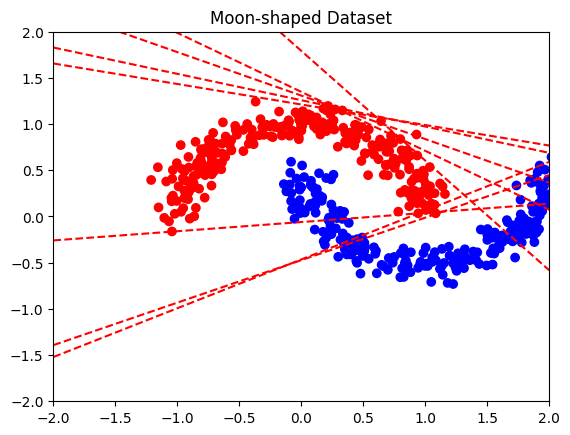


شکل ۱۹. خروجی مدل با ۵ نورون

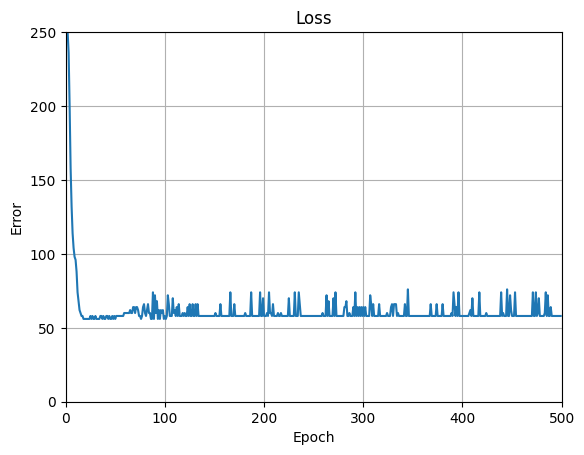


شکل ۲۰. نمودار تغییرات خطا با ۵ نورون

Accuracy of prediction for 8 neurons is: 0.892



شکل ۲۱. خروجی مدل با ۸ نورون



شکل ۲۲. نمودار تغییرات خطا با ۸ نورون

## ۲-۲-۳. **بررسی نمودار های حاصل شده بر اساس تعداد ایپاک های متفاوت**

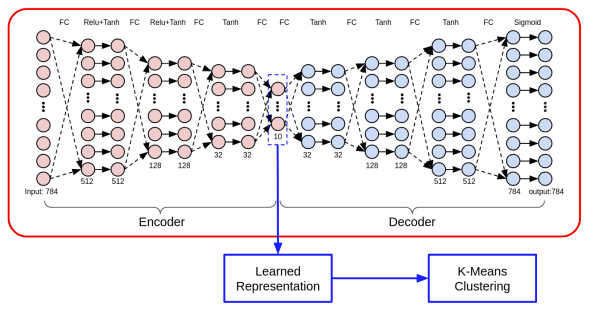
به دلیل شکل پراکندگی داده ها نمی توان آن را با یک چند ضلعی محدود کرد به همین دلیل نزدیک ترین شکل برای جداسازی ۳ خط است بر أساس خروجی نمودار های تغییرات خطا هر سه نمودار نوسان های بسیار زیادی دارند که نشان دهنده آن است که پراکندگی برای مدل Madaline ایده آل نیست و با افزایش تعداد نورون ها به تعداد ایپاک کمتری نیازمندیم و در صورت کاهش نیافتن ایپاک توان محاسباتی بیشتری مصرف می کنیم همچنین دچار overfit می شویم .

# **پاسخ ۳** **– خوشه بندی با استفاده از Autoencoder**

## ۱-۳-۱. **ایجاد شبکه عمیق اتوانکدر**

مدل پیشنهای:

شکل زیر شمای کلی شبکه عمیق اتوانکدر را نشان می دهد. در این روش دو مرحله وجود دارد آموزش مدل و سپس خوشه بندی، در مرحله آموزش اتوانکدر با دو بخش encoder و decoder با داده های آموزشی آموزش می بیند برای آموزش داده های دو بعدی به آرایه یک بعدی تبدیل می شوند و برای کاهش ابعاد به اتوانکدر داده میشوند اتوانکدر سعی می کند تا تصویر ورودی را بازسازی کند. در مرحله خوشه بندی داده های تست به encoder داده می شود تا ویژگی های آن استخراج شود سپس خروجی ان را به مدلK-Means می دهیم تا خوشه بندی را برای ما انجام دهد. در این روش با کاهش ابعاد به اطلاعات کلیدی باری خوشه بندی بهتر میرسیم.



شکل ۲۳. مدل پیشنهادی مقاله

کد افزودن کتابخانه ها

#import libraries

import keras

from keras.layers import Input, Dense

from keras.models import Model

from keras.datasets import mnist

import numpy as np

import matplotlib.pyplot as plt

from keras.optimizers import Adam

import keras.backend as K

کد دانلود و دسته بندی دیتای MNIST به دیتای آموزشی و تستی به همراه لیبل ها

#download dataset

(images\_train, label\_train),(images\_test,label\_test) = mnist.load\_data()

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>

11490434/11490434 [==============================] - 0s 0us/step

کد ترسیم ۵ داده از دیتاست

#show mnist dataset

import random

class\_names = ['0', '1', '2', '3', '4',

'5', '6', '7', '8', '9']

plt.figure(figsize=(10,10))

rand\_list=[]

for i in range(5):

rand\_list.append(random.randint(0,60000))

print(rand\_list)

j=0

for i in rand\_list:

plt.subplot(5,5,j+1)

plt.xticks([])

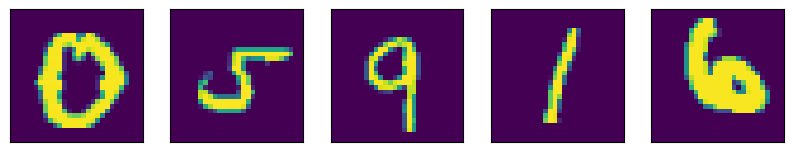
plt.yticks([])

plt.imshow(images\_train[i])

j=j+1

plt.show()

۵ داده با لیبل های متفاوت از دیتاست



شکل ۲۴. ۵ داده از دیتاست MNIST

کد اسکیل کردن داده ها بین ۰ و ۱

#data normalization

train\_images = images\_train.astype('float32')/255

test\_images = images\_test.astype('float32')/255

train\_images = train\_images.reshape((len(train\_images), np.prod(train\_images.shape[1:])))

test\_images = test\_images.reshape((len(test\_images), np.prod(test\_images.shape[1:])))

کد مدل Autoencoder

#auto\_encoder

input\_size = 784 #input size

hidden\_size1 = 512 #Number of neurons in the first layer

hidden\_size2 = 128 #Number of neurons in the second layer

hidden\_size3 = 32 #Number of neurons in the third layer

latent\_size = 10 #latent size

#encoder and decoder

input\_img = Input(shape = (input\_size,))

hidden\_layer\_1 = Dense(hidden\_size1, activation = "relu")(input\_img)

hidden\_layer\_2 = Dense(hidden\_size2, activation = "tanh")(hidden\_layer\_1)

hidden\_layer\_3 = Dense(hidden\_size3, activation = "tanh")(hidden\_layer\_2)

#latent space

latent\_layer = Dense(latent\_size)(hidden\_layer\_3)

hidden\_layer\_4 = Dense(hidden\_size3, activation = "tanh")(latent\_layer)

hidden\_layer\_5 = Dense(hidden\_size2, activation = "tanh")(hidden\_layer\_4)

hidden\_layer\_6 = Dense(hidden\_size1, activation = "tanh")(hidden\_layer\_5)

output\_layer = Dense(input\_size, activation = "sigmoid")(hidden\_layer\_6)

autoencoder = Model(input\_img, output\_layer)

#encoder model

encoder = Model(input\_img, latent\_layer)

## ۱-۳-۲. **آموزش اتوانکدر**

کد تعریف تابع خطا بر اساس فرمول ۶ مقاله و optimizer Adam با ضریب یادگیری 0.003

#compile the model

#fit the model with adam optimizer and binary cross entropy

def custom\_loss(y\_true, y\_pred):

# Calculate the Mean Squared Error (MSE)

mse = K.mean(K.square(y\_true - y\_pred))

# Add regularization term: 0.00001 \* MSE

custom\_loss = mse + 0.00001 \* mse

return custom\_loss

custom\_optimizer = Adam(learning\_rate=0.003)

autoencoder.compile(optimizer=custom\_optimizer, loss=custom\_loss)

کد آموزش با ۵۰ ایپاک

#fit the model with 50 epochs

autoencoder.fit(train\_images, train\_images, epochs =50, shuffle = True, validation\_data = (test\_images, test\_images))

Epoch 1/50

1875/1875 [==============================] - 23s 6ms/step - loss: 0.0355 - val\_loss: 0.0238

Epoch 2/50

1875/1875 [==============================] - 8s 4ms/step - loss: 0.0228 - val\_loss: 0.0212

Epoch 3/50

1875/1875 [==============================] - 9s 5ms/step - loss: 0.0205 - val\_loss: 0.0195

Epoch 4/50

1875/1875 [==============================] - 7s 4ms/step - loss: 0.0195 - val\_loss: 0.0185

Epoch 5/50

1875/1875 [==============================] - 8s 4ms/step - loss: 0.0189 - val\_loss: 0.0184

Epoch 6/50

1875/1875 [==============================] - 8s 4ms/step - loss: 0.0184 - val\_loss: 0.0182

Epoch 7/50

1875/1875 [==============================] - 8s 4ms/step - loss: 0.0181 - val\_loss: 0.0185

Epoch 8/50

1875/1875 [==============================] - 8s 4ms/step - loss: 0.0179 - val\_loss: 0.0176

Epoch 9/50

1875/1875 [==============================] - 8s 4ms/step - loss: 0.0178 - val\_loss: 0.0174

Epoch 10/50

1875/1875 [==============================] - 8s 4ms/step - loss: 0.0175 - val\_loss: 0.0173

Epoch 11/50

1875/1875 [==============================] - 8s 4ms/step - loss: 0.0173 - val\_loss: 0.0173

Epoch 12/50

1875/1875 [==============================] - 8s 4ms/step - loss: 0.0172 - val\_loss: 0.0173

Epoch 13/50

...

Epoch 49/50

1875/1875 [==============================] - 8s 4ms/step - loss: 0.0157 - val\_loss: 0.0158

Epoch 50/50

1875/1875 [==============================] - 8s 4ms/step - loss: 0.0156 - val\_loss: 0.0163

کد بررسی صحت عملکرد اتوانکدر

#predict test data with model

encoded\_img = autoencoder.predict(test\_images)

#Display actual test images and images made with model

n = 10

plt.figure(figsize=(20, 4))

for i in range(n):

# Original data

ax = plt.subplot(2, n, i + 1)

plt.imshow(test\_images[i].reshape(28, 28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

# Display reconstruction

ax = plt.subplot(2, n, i + 1 + n)

plt.imshow(encoded\_img[i].reshape(28, 28))

plt.gray()

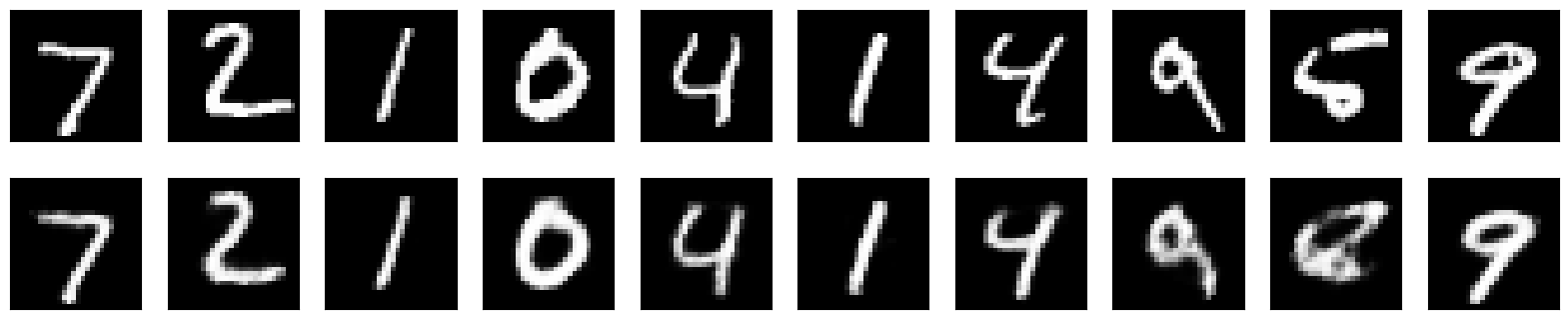
ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

plt.show()

313/313 [==============================] - 1s 2ms/step

مقایسه تصاویر بازسازی شده توسط اتوانکدر و مقایسه آن با تصاویر اصلی



شکل ۲۵. تصاویر بازسازی شده توسط اتوانکدر

## ۱-۳-۳. **خوشه بندی با استفاده از مدل K-Means**

کد افزودن کتابخانه ها

from sklearn.cluster import KMeans

کد خروجی گرفتن از encoder

encoded\_train = encoder.predict(train\_images)

encoded\_test = encoder.predict(test\_images)

1875/1875 [==============================] - 3s 2ms/step

313/313 [==============================] - 1s 2ms/step

کد خوشه بندی با استفاده از K-Means

import numpy as np

n\_clusters = 10

encoded\_train\_combined = np.column\_stack((encoded\_train, label\_train))

encoded\_test\_combined = np.column\_stack((encoded\_test, label\_test))

kmeans = KMeans(n\_clusters=n\_clusters, random\_state=0)

kmeans.fit(encoded\_train\_combined)

cluster\_labels\_test = kmeans.predict(encoded\_test\_combined)

کد محاسبه دقت مدل

cluster\_to\_label\_mapping = {}

for i in range(n\_clusters):

cluster\_indices = np.where(cluster\_labels\_test == i)[0]

true\_labels = encoded\_test\_combined[cluster\_indices, -1]

most\_common\_label = np.bincount(true\_labels.astype(int)).argmax()

cluster\_to\_label\_mapping[i] = most\_common\_label

predicted\_labels\_test = [cluster\_to\_label\_mapping[cluster] for cluster in cluster\_labels\_test]

accuracy\_test = accuracy\_score(label\_test, predicted\_labels\_test)

print(f"Test clustering accuracy: {accuracy\_test \* 100:.2f}%")

دقت مدل

Test clustering accuracy: 96.18%

کد محاسبه ARI (Adjusted Rand Index)

from sklearn.metrics import adjusted\_rand\_score

# Calculate the Adjusted Rand Index (ARI)

ari = adjusted\_rand\_score(label\_test, predicted\_labels\_test)

print(f"Adjusted Rand Index (ARI): {ari:.4f}")

شاخص به دست آمده

Adjusted Rand Index (ARI): 0.9189

# **پاسخ ۴** **–شبکه‌ی Multi-Layer Perceptron**

## ۱-۴**-۱**. **وظیفه**

در این سوال به کمک یافته های شبکه‌ی طراحی شده‌ی MLP بر روی MNIST می‌پردازیم. همچنین در بخش آخر سوال از مدل پیشرفته تر Teacher برای Knowledge Distillation مدل ساده تر Student استفاده کرده و نتایج را بررسی میکنیم.

## ۱-۴**-۲**. **پیش پردازش**

## 

شکل ۲۶. فرایند های طی شده در انجام مرحله پیش پردازش داده ها

## 

## . ۱-۴**-۲**-۱. **کد های مربوط به این بخش**

اضافه کردن کتابخانه های مورد نیاز

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

*# PyTorch for Deep Learning*

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

from torch.utils.data import DataLoader

*# TensorFlow and Keras for Deep Learning*

import tensorflow as tf

from tensorflow import keras

*# scikit-learn for clustering*

from sklearn import datasets

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score, adjusted\_rand\_score

import seaborn as sns

import torchvision

لود کردن دیتاست و خروجی dimention های test و train

transform = torchvision.transforms.Compose([torchvision.transforms.ToTensor()])

train\_dataset = torchvision.datasets.MNIST(

    root='./data', train=True, transform=transform, download=True)

train\_loader = torch.utils.data.DataLoader(

    train\_dataset, batch\_size=64, shuffle=True)

test\_dataset = torchvision.datasets.MNIST(

    root='./data', train=False, transform=transform, download=True)

test\_loader = torch.utils.data.DataLoader(

    test\_dataset, batch\_size=64, shuffle=False)

num\_train\_samples = len(train\_dataset)

train\_data\_dimensions = train\_dataset[0][0].shape

num\_test\_samples = len(test\_dataset)

test\_data\_dimensions = test\_dataset[0][0].shape

print("Training Data:")

print(f"Number of Samples: {num\_train\_samples}")

print(f"Data Dimensions: {train\_data\_dimensions}")

print("\nTest Data:")

print(f"Number of Samples: {num\_test\_samples}")

print(f"Data Dimensions: {test\_data\_dimensions}")

Training Data:

Number of Samples: 60000

Data Dimensions: torch.Size([1, 28, 28])

Test Data:

Number of Samples: 10000

Data Dimensions: torch.Size([1, 28, 28])

نمایش نمونه های هر کلاس

class\_shown = {}

fig, axs = plt.subplots(1, 10, figsize=(15, 2))

for i in range(len(train\_dataset)):

    image, label = train\_dataset[i]

    label = int(label)

    if label not in class\_shown:

        ax = axs[label]

        ax.imshow(image[0], cmap='gray')

        ax.set\_title(f'Class {label}')

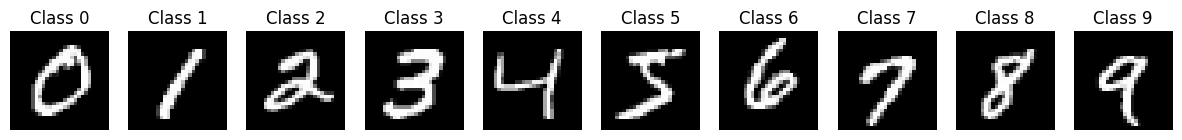
        ax.axis('off')

        class\_shown[label] = True

    if len(class\_shown) == 10:

        break

plt.show()



شکل ۲۷. نمونه از هر کلاس دیتاست MNIST

نمایش نمودار هیستوگرام داده های train و test

colors = ['blue', 'green', 'red', 'purple', 'orange',

          'brown', 'pink', 'gray', 'olive', 'cyan']

def plot\_class\_histogram(dataset, title):

    labels = [int(label) for \_, label in dataset]

    unique\_labels = list(set(labels))

    plt.figure(figsize=(10, 5))

    for i, label in enumerate(unique\_labels):

        color = colors[i % len(colors)]

        plt.hist(

            [label for label in labels if label == unique\_labels[i]],

            bins=[unique\_labels[i] - 0.5, unique\_labels[i] + 0.5],

            align='mid',

            rwidth=0.8,

            alpha=0.75,

            edgecolor='black',

            color=color,

            label=f'Class {unique\_labels[i]}'

        )

    plt.xticks(range(10))

    plt.xlabel("Class")

    plt.ylabel("Number of Samples")

    plt.title(title)

    plt.grid(axis='y', linestyle='--', alpha=0.7)

    plt.legend(loc='best')

*# Plot the histogram for the training data*

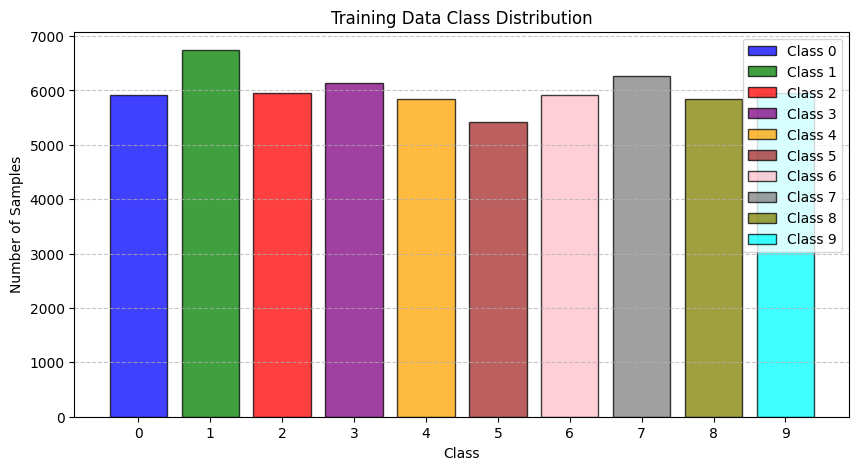
plot\_class\_histogram(train\_dataset, title="Training Data Class Distribution")

plt.show()

*# Plot the histogram for the test data*

plot\_class\_histogram(test\_dataset, title="Test Data Class Distribution")

plt.show()



شکل ۲۸. توزیع داده های کلاس train



شکل ۲۹. توزیع داده های کلاس تست

نرمال کردن داده های با وسیله min max

transform = torchvision.transforms.Compose([

    torchvision.transforms.ToTensor(),

    torchvision.transforms.Lambda(

        lambda x: (x - x.min()) / (x.max() - x.min()))

])

train\_dataset\_normalized = torchvision.datasets.MNIST(

    root='./data', train=True, transform=transform, download=True)

train\_loader\_normalized = torch.utils.data.DataLoader(

    train\_dataset\_normalized, batch\_size=64, shuffle=True)

test\_dataset\_normalized = torchvision.datasets.MNIST(

    root='./data', train=False, transform=transform, download=True)

test\_loader\_normalized = torch.utils.data.DataLoader(

    test\_dataset\_normalized, batch\_size=64, shuffle=False)

## ۱-۴**-۳**. **شبکه Teacher**

## . ۱-۴**-۳**-۱. **کد های مربوط به این بخش**

معماری شبکه

class Teacher(nn.Module):

    def \_\_init\_\_(self):

        super(Teacher, self).\_\_init\_\_()

        self.flatten = nn.Flatten()

        self.fc1 = nn.Linear(784, 1024)

        self.relu1 = nn.ReLU()

        self.fc2 = nn.Linear(1024, 512)

        self.relu2 = nn.ReLU()

        self.fc3 = nn.Linear(512, 10)

        self.relu3 = nn.ReLU()

    def forward(self, x):

        x = self.flatten(x)

        x = self.fc1(x)

        x = self.relu1(x)

        x = self.fc2(x)

        x = self.relu2(x)

        x = self.fc3(x)

        x = self.relu3(x)

        return x

تنظیم پارامتر های مطرح شده در صورت سوالات به ازای rate یادگیری و batch و تعداد epoch

*# Define loss function and optimizer*

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(teacher.parameters(), lr=0.01)

*# Training parameters*

batch\_size = 32

num\_epochs = 20

*# Data loaders*

train\_loader = torch.utils.data.DataLoader(

    train\_dataset\_normalized, batch\_size=batch\_size, shuffle=True)

*# Lists to store training history*

train\_loss\_history = []

train\_accuracy\_history = []

فرایند یادگیری

*# Training loop*

for epoch in range(num\_epochs):

    running\_loss = 0.0

    correct\_predictions = 0

    total\_samples = 0

    for inputs, labels in train\_loader:

        optimizer.zero\_grad()

        outputs = teacher(inputs)

        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

        running\_loss += loss.item()

        \_, predicted = torch.max(outputs.data, 1)

        total\_samples += labels.size(0)

        correct\_predictions += (predicted == labels).sum().item()

    train\_loss = running\_loss / len(train\_loader)

    train\_accuracy = (correct\_predictions / total\_samples) \* 100

    train\_loss\_history.append(train\_loss)

    train\_accuracy\_history.append(train\_accuracy)

    print(

        f'Epoch [{epoch + 1}/{num\_epochs}] - Loss: {train\_loss:.4f}, Accuracy: {train\_accuracy:.2f}%')

Epoch [1/20] - Loss: 1.0104, Accuracy: 78.57%

Epoch [2/20] - Loss: 0.3451, Accuracy: 90.20%

Epoch [3/20] - Loss: 0.2825, Accuracy: 91.93%

Epoch [4/20] - Loss: 0.2435, Accuracy: 93.08%

Epoch [5/20] - Loss: 0.2126, Accuracy: 93.94%

Epoch [6/20] - Loss: 0.1879, Accuracy: 94.66%

Epoch [7/20] - Loss: 0.1671, Accuracy: 95.23%

Epoch [8/20] - Loss: 0.1501, Accuracy: 95.74%

Epoch [9/20] - Loss: 0.1357, Accuracy: 96.16%

Epoch [10/20] - Loss: 0.1237, Accuracy: 96.52%

Epoch [11/20] - Loss: 0.1128, Accuracy: 96.87%

Epoch [12/20] - Loss: 0.1038, Accuracy: 97.10%

Epoch [13/20] - Loss: 0.0955, Accuracy: 97.32%

Epoch [14/20] - Loss: 0.0886, Accuracy: 97.51%

Epoch [15/20] - Loss: 0.0819, Accuracy: 97.72%

Epoch [16/20] - Loss: 0.0762, Accuracy: 97.90%

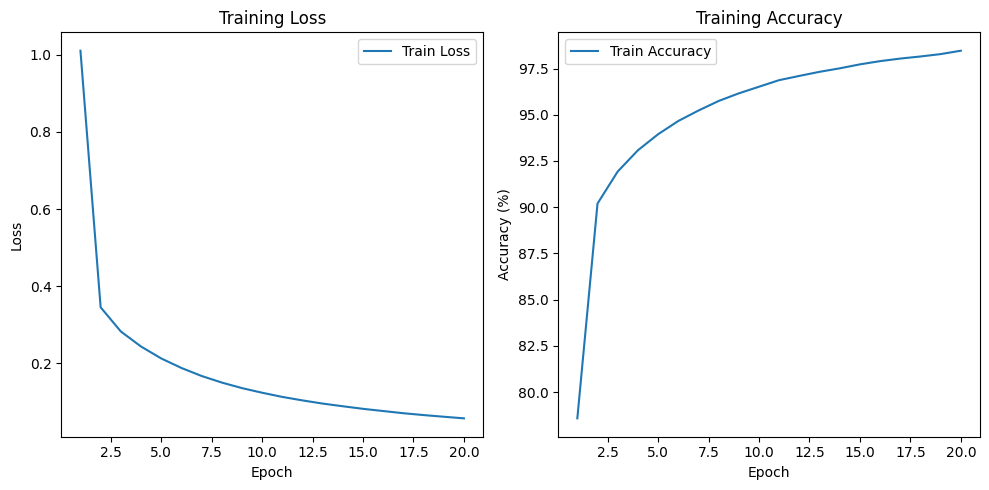
Epoch [17/20] - Loss: 0.0705, Accuracy: 98.04%

Epoch [18/20] - Loss: 0.0657, Accuracy: 98.15%

Epoch [19/20] - Loss: 0.0614, Accuracy: 98.28%

Epoch [20/20] - Loss: 0.0573, Accuracy: 98.46%

چاپ نمودار خطا و دقت



شکل ۳۰. نمودار دقت و خطای مدل Teacher

## . ۱-۴**-۳**-**۲**. **Logit**

برای پیدا کردن کلاس خروجی پیش بینی شده توسط شبکه در خروجی logit در PyTorch میتوانیم از تابع torch.max استفاده بکنیم تابع مذکور شاخص حداکثر مقدار را در بین امتیازات کلاس های مقادیر خروجی پیدا کرده و کلاس پیش بینی شده را مجددا تنظیم میکند.

دقت: تنظیم تعداد پاسخ های صحیح در کل نمونه است

Miss classification: تفاوت تعداد کل نمونه ها و پیش بینی های صحیح است

teacher.eval()

correct\_predictions = 0

total\_samples = 0

with torch.no\_grad():

    for inputs, labels in test\_loader\_normalized:

        outputs = teacher(inputs)

        \_, predicted = torch.max(outputs.data, 1)

        total\_samples += labels.size(0)

        correct\_predictions += (predicted == labels).sum().item()

test\_accuracy = (correct\_predictions / total\_samples) \* 100

print(f'Test Accuracy: {test\_accuracy:.2f}%')

*# Calculate the number of misclassifications*

misclassifications = total\_samples - correct\_predictions

print(f'Number of Misclassifications: {misclassifications}')

Test Accuracy: 97.65%

Number of Misclassifications: 235

## ۱-۴**-**۴. **شبکه Student**

## . ۱-۴**-**۴-۱. **کد های مربوط به این بخش**

به علت وجود تشابه با بخش قبل از نظر منطقی از تکرار دوباره کد های پرهیز شده

معماری مدل

class Student(nn.Module):

    def \_\_init\_\_(self):

        super(Student, self).\_\_init\_\_()

        self.flatten = nn.Flatten()

        self.fc1 = nn.Linear(784, 128)

        self.relu1 = nn.ReLU()

        self.fc2 = nn.Linear(128, 64)

        self.relu2 = nn.ReLU()

        self.fc3 = nn.Linear(64, 10)

        self.relu3 = nn.ReLU()

    def forward(self, x):

        x = self.flatten(x)

        x = self.fc1(x)

        x = self.relu1(x)

        x = self.fc2(x)

        x = self.relu2(x)

        x = self.fc3(x)

        x = self.relu3(x)

        return x

خروجی در ده ایپاک

*# Training loop*

for epoch in range(num\_epochs):

    running\_loss = 0.0

    correct\_predictions = 0

    total\_samples = 0

    for inputs, labels in train\_loader:

        optimizer.zero\_grad()

        outputs = student(inputs)

        loss = criterion(outputs, labels)

        loss.backward()

        optimizer.step()

        running\_loss += loss.item()

        \_, predicted = torch.max(outputs.data, 1)

        total\_samples += labels.size(0)

        correct\_predictions += (predicted == labels).sum().item()

    train\_loss = running\_loss / len(train\_loader)

    train\_accuracy = (correct\_predictions / total\_samples) \* 100

    train\_loss\_history.append(train\_loss)

    train\_accuracy\_history.append(train\_accuracy)

    print(

        f'Epoch [{epoch + 1}/{num\_epochs}] - Loss: {train\_loss:.4f}, Accuracy: {train\_accuracy:.2f}%')

Epoch [1/10] - Loss: 1.7266, Accuracy: 46.74%

Epoch [2/10] - Loss: 0.9774, Accuracy: 65.15%

Epoch [3/10] - Loss: 0.8982, Accuracy: 66.32%

Epoch [4/10] - Loss: 0.8694, Accuracy: 66.87%

Epoch [5/10] - Loss: 0.8519, Accuracy: 67.20%

Epoch [6/10] - Loss: 0.8376, Accuracy: 67.52%

Epoch [7/10] - Loss: 0.8260, Accuracy: 67.73%

Epoch [8/10] - Loss: 0.8154, Accuracy: 67.95%

Epoch [9/10] - Loss: 0.8058, Accuracy: 68.17%

Epoch [10/10] - Loss: 0.7972, Accuracy: 68.36%

بررسی نمودار های خطا و دقت

* در این حالت مشاهده میکنیم چون شبکه ضعیف تر هست تعداد نتایج مدل نسبت به حالت قبلی افت داشته (نظیر پارامتر epoch تعلیم مدل و...)

misclassifications = total\_samples - correct\_predictions

print(f'Number of Misclassifications on Test Data: {misclassifications}')

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.plot(range(1, num\_epochs + 1), train\_loss\_history, label='Train Loss')

plt.title('Training Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(range(1, num\_epochs + 1),

         train\_accuracy\_history, label='Train Accuracy')

plt.title('Training Accuracy')

plt.xlabel('Epoch')

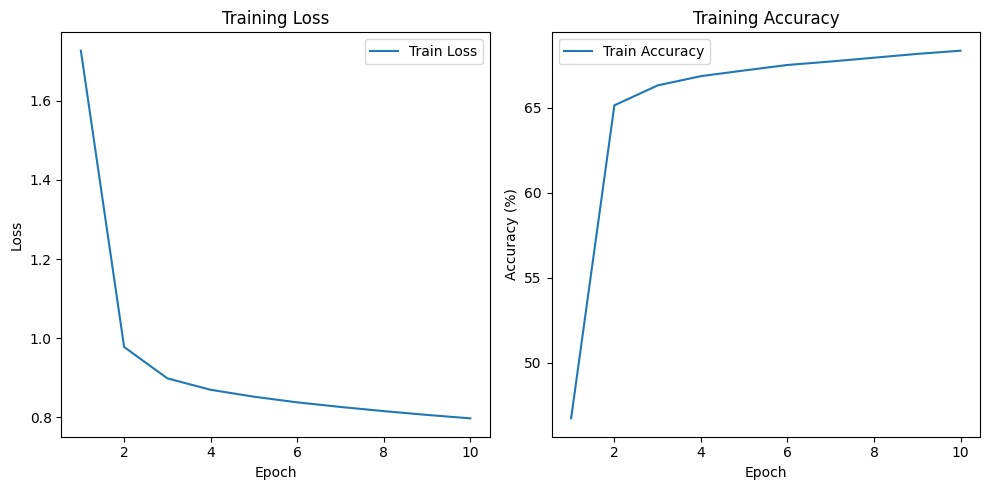
plt.ylabel('Accuracy (%)')

plt.legend()

plt.tight\_layout()

plt.show()

Number of Misclassifications on Test Data: 18983



شکل ۳۱. نمودار دقت خطای مدل Student

## ۱-۴**-**۵. Knowledge Distillation

## ۱-۴**-**۵-۱. **کد های مربوط به این بخش**

در این مدل از خروجی های مدل Teacher برای کمک به یادگیری مدل Student استفاده کرده ایم. این روش یادگیری که Knowledge Distillation نام دارد برای کاربرد مدل های بزرگ تر که ظرفیت یادگیری بیشتر دارند و سنگین تر هستند به کار میرود. به این صورت که دانش خود را به مدل های ضعیف و کوچکتر منتقل میکنند. در این فرایند مدل های بزرگ تر Teacher و مدل های کوچک تر Student نام دارند. یک راه برای انتقال داد ن Generalization از مدل Teacher به مدل Student این است که از توزیع احتمال خروجی مدل Teacher که آن را Target Soft می نامند برای آموزش مدل Student استفاده کنیم.

## ۱-۴**-**۵-۱. **کد های مربوط به این بخش**

student\_kd.train()  *# Set the Student model to training mode*

for epoch in range(num\_epochs):

    running\_loss = 0.0

    correct\_predictions = 0

    total\_samples = 0

    for inputs, labels in train\_loader:

        optimizer\_kd.zero\_grad()

        outputs\_kd = student\_kd(inputs)

*# Get Teacher's logits from the Teacher data loader*

        teacher\_logits = teacher(inputs)

        loss\_kd = criterion\_kd(outputs\_kd, teacher\_logits)

        loss\_kd.backward()

        optimizer\_kd.step()

        running\_loss += loss\_kd.item()

        \_, predicted = torch.max(outputs\_kd.data, 1)

        total\_samples += labels.size(0)

        correct\_predictions += (predicted == labels).sum().item()

    train\_loss\_kd = running\_loss / len(train\_loader)

    train\_accuracy\_kd = (correct\_predictions / total\_samples) \* 100

    train\_loss\_history\_kd.append(train\_loss\_kd)

    train\_accuracy\_history\_kd.append(train\_accuracy\_kd)

    print(f'Epoch [{epoch + 1}/{num\_epochs}] - Loss: {train\_loss\_kd:.4f}')

Epoch [1/10] - Loss: 4.4609

Epoch [2/10] - Loss: 0.5004

Epoch [3/10] - Loss: 0.2372

Epoch [4/10] - Loss: 0.1650

Epoch [5/10] - Loss: 0.1317

Epoch [6/10] - Loss: 0.1132

Epoch [7/10] - Loss: 0.1006

Epoch [8/10] - Loss: 0.0916

Epoch [9/10] - Loss: 0.0853

Epoch [10/10] - Loss: 0.0805

در این بخش از MSE برخلاف بخش های قبل برای تابع خطا استفاده شده. مدل Teacher یکبار اموزش داده شده و داده های لاجیت آن نگه داری میشود. سپس مدل Student استفاده شده این بار برای اموزش به جای لیبل های واقعی از لاجیت خروجی Teacher استفاده شده است.

correct\_predictions\_kd\_test = 0

total\_samples\_kd\_test = 0

for inputs, labels in test\_loader\_normalized:

    outputs\_kd = student\_kd(inputs)

    \_, predicted\_kd = torch.max(outputs\_kd.data, 1)

    total\_samples\_kd\_test += labels.size(0)

    correct\_predictions\_kd\_test += (predicted\_kd == labels).sum().item()

misclassifications\_kd\_test = total\_samples\_kd\_test - correct\_predictions\_kd\_test

print(

    f'Number of Misclassifications with Knowledge Distillation on Test Data: {misclassifications\_kd\_test}')

Number of Misclassifications with Knowledge Distillation on Test Data: 285

به وضوح مشخص است که نسبت Miss classification نسبت به حالت سوم کمتر شده است.

## . ۱-۴**-**۵-**۲**. تعداد پیشبینی های غلط misclassifications مدل Student بخش قبل روی داده های آزمون را با مدل Student فعلی مقایسه کرده و نتیجه را گزارش کنید

مقایسه بین تعداد طبقه‌بندی‌های اشتباه در داده‌های آزمون با استفاده از Distillation Knowledge و آموزش معمولی نشان می‌دهد که تقطیر دانش به طور قابل‌توجهی باعث کاهش طبقه‌بندی اشتباه شده است:

- با Distillation Knowledge: تعداد طبقه بندی اشتباه در داده های آزمون: 285

- بدون تقطیر دانش (آموزش عادی): تعداد طبقه بندی اشتباه در داده های آزمون: 18983

با استفاده از Distillation Knowledge ، مدل دانشجو یاد گرفته است که از خروجی های مدل معلم تقلید کند، که منجر به تعمیم بهتر و کاهش قابل توجهی در طبقه بندی اشتباه در داده های آزمون می شود. این نشان دهنده اثربخشی تقطیر دانش در انتقال دانش از یک مدل بزرگتر (معلم) به یک مدل کوچکتر (دانشجو) است.

## . ۱-۴**-**۵-**۳**. Accuracy هر دو مدل Student روی داده های train در طول اموزش را روی یک نمودار و در کنار هم رسم کرده و توزیح بدهید knowledge distillation چه تاثیراتی داشته است

num\_epochs = 10

plt.figure(figsize=(10, 5))

plt.plot(range(1, num\_epochs + 1),

         train\_accuracy\_history, label='Train Accuracy Student', color='blue')

plt.plot(range(1, num\_epochs + 1),

         train\_accuracy\_history\_kd, label='Train Accuracy Knowledge Distillation', color='red')

plt.title('Training Accuracy')

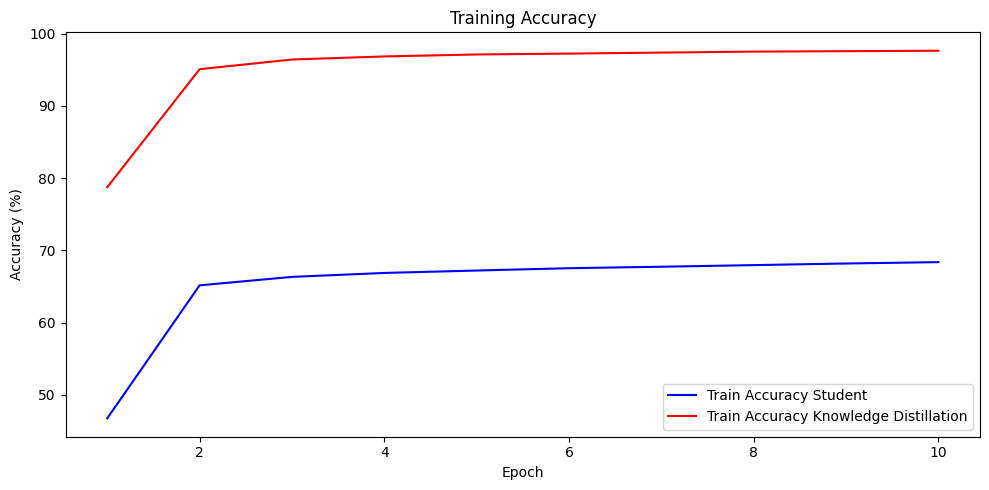
plt.xlabel('Epoch')

plt.ylabel('Accuracy (%)')

plt.legend()

plt.tight\_layout()

plt.show()



شکل ۳۲. نمودار مقایسه دقت مدل های student

با رسم نمودار دقت هر دو مدل مشاهده میشود اموزش مدل ضعیف تر به وسیله انتقال دانش از مدل پیچیده تر سبب بهبود دقت داده های اموزش شده است.