|  |  |
| --- | --- |
| نام و نام خانوادگی | **محمد پویا افشاری – علیرضا اسمعیل زاده** |
| شماره دانشجویی | **810198577 - 810000000** |
| 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** 6](#_Toc150013725)

[۱-۱. نمایشگر 7-segment 6](#_Toc150013726)

[۱-۱-۱. وظیفه 6](#_Toc150013727)

[۱-۱-۲. معماری مدل 6](#_Toc150013728)

[**درصورتی که در این حالت Theta = 0 برابر threshold در نظر گرفته بشود به ازای** 6](#_Toc150013729)

[**خروجی هر نرون خواهیم داشت:** 6](#_Toc150013730)

[**وزن و بایاس هر نورون خروجی با توجه به ضریب آنها در معادله خط تنظیم می شود.** 6](#_Toc150013731)

[۱-۱-۳. اضافه کردن کتابخانه های مورد نیاز 7](#_Toc150013732)

[۱-۲. شبکه عصبی یک لایه 7](#_Toc150013733)

[۱-۲-۱. معماری 7](#_Toc150013734)

[۱-۲-۲. Forward Propagation 7](#_Toc150013735)

[۱-۲-۳. 7](#_Toc150013736)

[**ایجاد شبکه عصبی** 7](#_Toc150013737)

[**تعیین وزن و بایاس و threshold:** 8](#_Toc150013738)

[**ورودی:** 8](#_Toc150013739)

[**تست خروجی:** 8](#_Toc150013740)

[۱-۳. شبکه عصبی دو لایه 8](#_Toc150013741)

[۱-۳-۱. معماری 9](#_Toc150013742)

[**با در نظر گرفتن دو نرون برای لایه مخفی مطابق شکل زیر داریم:** 9](#_Toc150013743)

[۱-۳-۲. کد های این بخش 9](#_Toc150013744)

[**استفاده از شبکه قبلی با اضافه کردن نود های لایه مخفی و تغییر مقادیر شبکه** 9](#_Toc150013745)

[**بررسی و تست خروجی** 9](#_Toc150013746)

[**پاسخ ۲** **- عنوان پرسش دوم به فارسی** 10](#_Toc150013747)

[۱-۲. **عنوان بخش اول** 10](#_Toc150013748)

[**پاسخ ۳** **– عنوان پرسش سوم به فارسی** 11](#_Toc150013749)

[۱-۳. **عنوان بخش اول** 11](#_Toc150013750)

[**پاسخ ۴** **–شبکه‌ی Multi-Layer Perceptron** 12](#_Toc150013751)

[۱-۴**-۱**. **وظیفه** 12](#_Toc150013752)

[۱-۴**-۲**. **پیش پردازش** 12](#_Toc150013753)

[12](#_Toc150013754)

[. ۱-۴**-۲**-۱. **کد های مربوط به این بخش** 13](#_Toc150013755)

[۱-۴**-۳**. **شبکه Teacher** 16](#_Toc150013756)

[. ۱-۴**-۳**-۱. **کد های مربوط به این بخش** 16](#_Toc150013757)

[. ۱-۴**-۳**-**۲**. **Logit** 19](#_Toc150013758)

[۱-۴**-**۴. **شبکه Student** 19](#_Toc150013759)

[. ۱-۴**-**۴-۱. **کد های مربوط به این بخش** 19](#_Toc150013760)

[۱-۴**-**۵. Knowledge Distillation 22](#_Toc150013761)

[۱-۴**-**۵-۱. **کد های مربوط به این بخش** 22](#_Toc150013762)

[. ۱-۴**-**۵-۱. **کد های مربوط به این بخش** 22](#_Toc150013763)

[. 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 23](#_Toc150013764)

[. 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 24](#_Toc150013765)

**شکل‌ها**

شکل 1. عنوان تصویر نمونه 1

**جدول‌ها**

[جدول 1. عنوان جدول نمونه 1](#_Toc115453324)

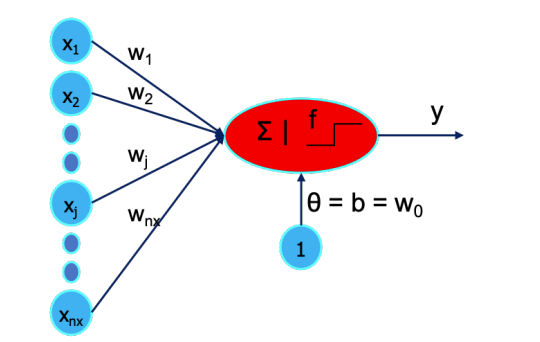
# **پاسخ 1**. **شبکه عصبی Mcculloch-Pitts**

۱-۱. نمایشگر 7-segment

۱-۱-۱. وظیفه

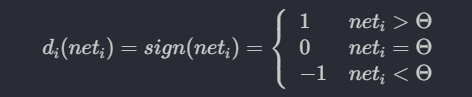
در این مساله باید به کمک نرون های مک کلانژ پیتز مساله تشخیص اعداد بین 6 تا 9 را حل کنیم. در بخش اول این مساله را بدون لایه مخفی و در بخش دوم با یک لایه مخفی انجام میدهیم.

۱-۱-۲. معماری مدل

******درصورتی که در این حالت Theta = 0 برابر threshold در نظر گرفته بشود به ازای**

شکل 1. معماری مدل Mcculloch pits

**خروجی هر نرون خواهیم داشت:**

****

شکل 2. فرمول محاسبه فعال شدن نرون Mcculoch pits

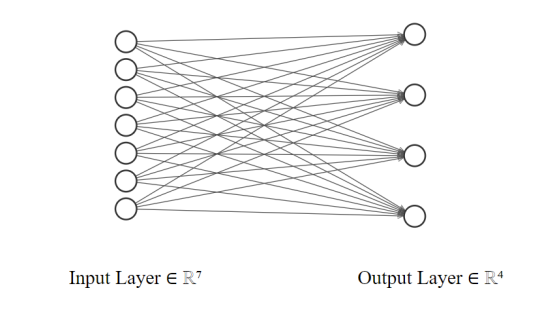
**وزن و بایاس هر نورون خروجی با توجه به ضریب آنها در معادله خط تنظیم می شود.**

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

import numpy as np

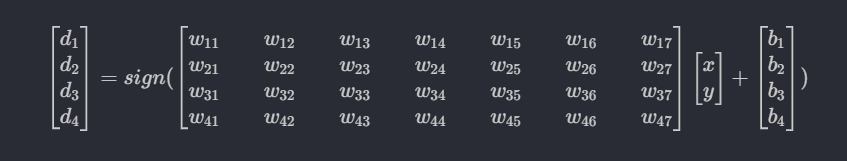
import pandas as pd

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]

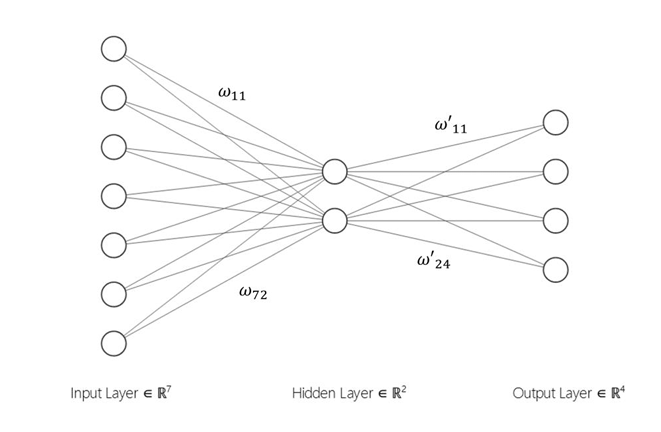
[0, 1, 0, 0]

[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** **- عنوان پرسش دوم به فارسی**

## ۱-۲. **عنوان بخش اول**

متن نمونه

.

# **پاسخ ۳** **– عنوان پرسش سوم به فارسی**

## ۱-۳. **عنوان بخش اول**

متن نمونه

.

# **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 1. 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}')

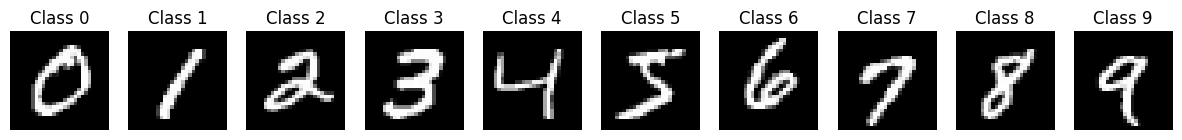
        ax.axis('off')

        class\_shown[label] = True

    if len(class\_shown) == 10:

        break

plt.show()



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*

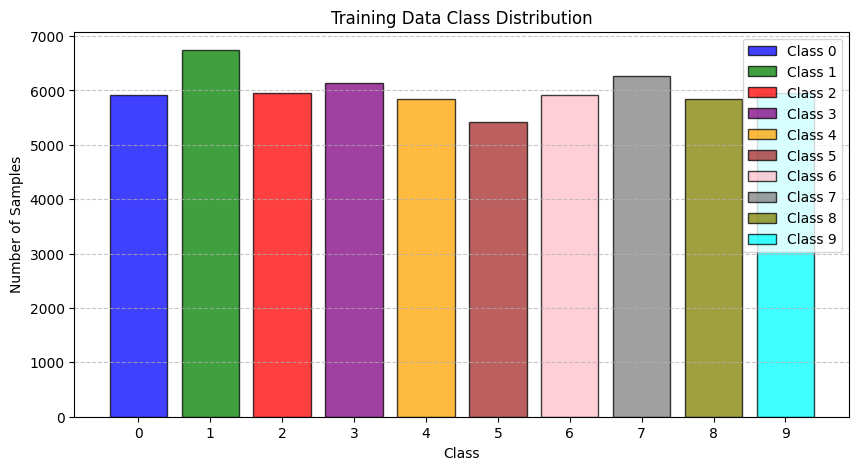
plot\_class\_histogram(train\_dataset, title="Training Data Class Distribution")

plt.show()

*# Plot the histogram for the test data*

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

plt.show()



شکل 2. توزیع داده های کلاس train



شکل 3. 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%

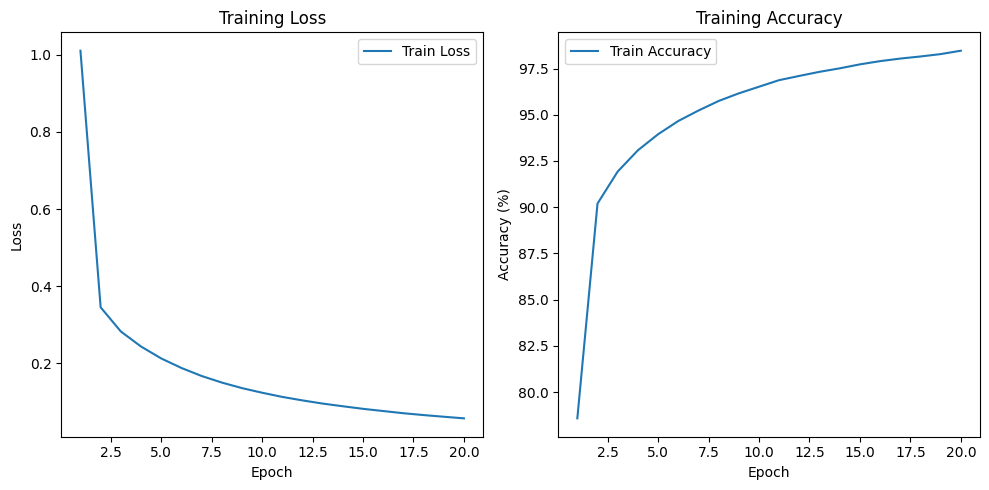
Epoch [17/20] - Loss: 0.0705, Accuracy: 98.04%

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 4. 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')

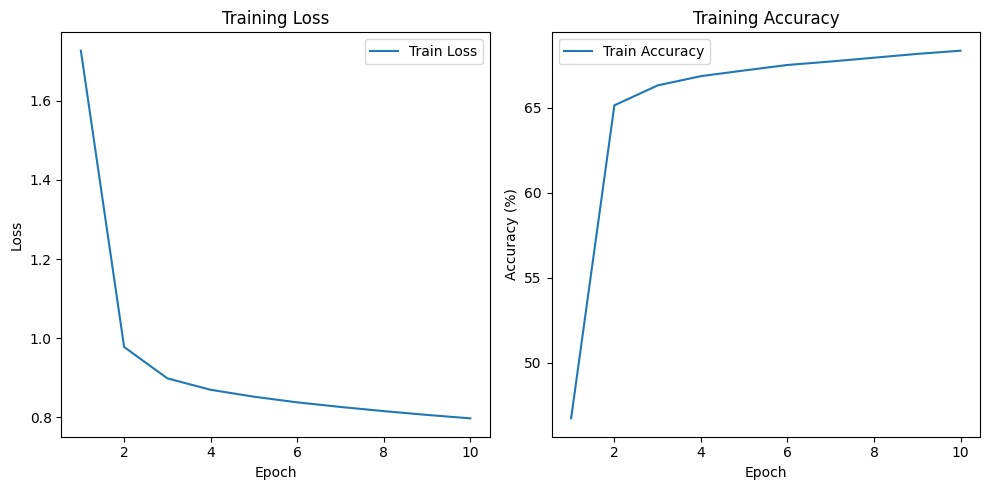
plt.ylabel('Accuracy (%)')

plt.legend()

plt.tight\_layout()

plt.show()

Number of Misclassifications on Test Data: 18983



شکل 5. 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')

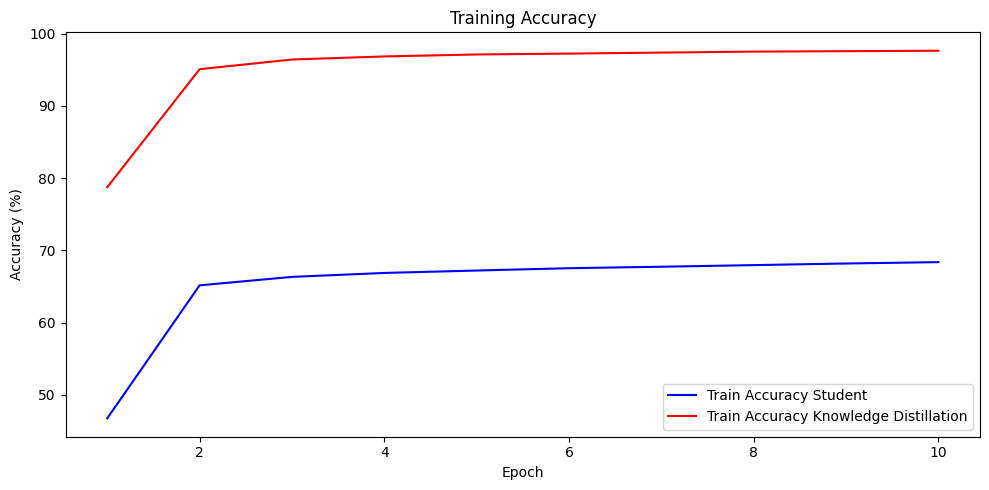
plt.xlabel('Epoch')

plt.ylabel('Accuracy (%)')

plt.legend()

plt.tight\_layout()

plt.show()



شکل 6. 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.