TECHIN 513 - Basic ML

Instructions

Install the required packages (scikit-learn, TensorFlow, Keras, PyTorch, and, pandas) if they are not already installed.

```
# use pip to install the packages
!pip3 install scikit-learn TensorFlow Keras torch torchvision torchaudio pandas numpy

# Import necessary packages
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
# import RandomForestClassifier from sklearn
import tensorflow as tf
from tensorflow import keras
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense
import torch
import torch.nn as nn
import torch.optim as optim
```

```
Requirement already satisfied: TensorFlow in /usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: Keras in /usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.1.0+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.16.0+cu121)
Requirement already satisfied: torchaudio in /usr/local/lib/python3.10/dist-packages (2.1.0+cu121)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (1.5.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.2.
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (23.5.2
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from Tens
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (16.0.6)
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (0.2.0)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (23.2)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (4.
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from Ten
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (1.60.1)
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist-packages (from TensorFlow) (2.1
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from Tenso
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.13.1)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch) (2.1.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from torchvision) (2.31.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (9.4.
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas) (2023.4)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->Te
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from te
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15-
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->torch
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (3.6
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->ten
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oau
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0
```

```
# Task 1: Load the Iris dataset
iris = datasets.load_iris()
```

```
X = iris.data
y = iris.target

# Task 2: Split the data into training and testing sets
# use train_test_split function to split the data with test_size = 0.2 and random_state = 42

from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Task 3: Train a Random Forest Classifier on the training data
# import RandomForestClassifier from sklearn and fit it with training data

from sklearn.ensemble import RandomForestClassifier

# Create a Random Forest Classifier instance
clf = RandomForestClassifier(random_state=42)
# Fit the classifier to the training data
clf.fit(X_train, y_train)

# Task 4: Evaluate the classifier on the testing data
# use clf.score function to evaluate the classifier on the testing data
# print the accuracy of the classifier

# Evaluate the classifier on the testing data
accuracy = clf.score(X_test, y_test)
print(f"Accuracy of the Random Forest Classifier on the test data: {accuracy:.2f}")

# Task 5: Load the MNIST dataset
# use keras.datasets.mnist.load_data() to load the dataset

from tensorflow.keras.datasets import mnist

# Load the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Task 6: Preprocess the data
# normalize the data by dividing by 255.0
# use to_categorical from keras.utils to one-hot encode the labels

# Normalize the data
train_images = train_images / 255.0
test_images = test_images / 255.0

# One-hot encode the labels
from tensorflow.keras.utils import to_categorical
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Task 7: Define and train a simple neural network using Keras
# use Sequential model from keras.models
# use Dense layer from keras.layers
# use 'adam' as optimizer and 'categorical_crossentropy' as loss function
# use model.fit to train the model

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

# Define the model
model = Sequential([
    Flatten(input_shape=(28, 28)), # Flatten the input images to a vector
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Train the model
model.fit(train_images, train_labels, epochs=5, batch_size=32)


# Task 8: Evaluate the neural network on the testing data
# use model.evaluate to get the test loss and test accuracy

# Evaluate the model
test_loss, test_accuracy = model.evaluate(test_images, test_labels)

# Print test loss and test accuracy
print(f"Test Loss: {test_loss}")
```

```
print(f"Test Accuracy: {test_accuracy}")

# Task 9: Define a simple linear regression model using PyTorch
# create a class LinearRegression that inherit from nn.Module
# define the constructor and forward function
import torch.nn as nn

class LinearRegression(nn.Module):
    def __init__(self):
        super(LinearRegression, self).__init__()
        self.linear = nn.Linear(1, 1) # Assumes input and output size of 1

    def forward(self, x):
        return self.linear(x)

# Task 10: Train the linear regression model on some dummy data and print the weight and bias
# create an instance of LinearRegression
# use nn.MSELoss as criterion, optim.SGD as optimizer
# use model.parameters() as input for optimizer
# use optimizer.step() and criterion to update the model weight and bias
import torch
from torch.optim import SGD

# Create dummy data
x_train = torch.randn(100, 1) * 10
y_train = x_train + 3 * torch.randn(100, 1)

# Create an instance of LinearRegression
model = LinearRegression()

# Define loss function and optimizer
criterion = nn.MSELoss()
optimizer = SGD(model.parameters(), lr=0.001)

# Train the model
for epoch in range(100):
    # Zero the gradients
    optimizer.zero_grad()

    # Forward pass
    predictions = model(x_train)

    # Compute loss
    loss = criterion(predictions, y_train)

    # Backward pass
    loss.backward()

    # Update weights
    optimizer.step()

# Print the weight and bias
print(f"Weight: {model.linear.weight.item()}")
print(f"Bias: {model.linear.bias.item()}")
```

```
Accuracy of the Random Forest Classifier on the test data: 1.00
Epoch 1/5
1875/1875 [==============================] - 9s 4ms/step - loss: 0.2540 - accuracy: 0.9279
Epoch 2/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1110 - accuracy: 0.9673
Epoch 3/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0760 - accuracy: 0.9773
Epoch 4/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0571 - accuracy: 0.9824
Epoch 5/5
1875/1875 [==============================] - 8s 4ms/step - loss: 0.0439 - accuracy: 0.9865
313/313 [==============================] - 1s 2ms/step - loss: 0.0702 - accuracy: 0.9797
Test Loss: 0.07016762346029282
Test Accuracy: 0.9797000288963318
Weight: 1.002183198928833
Bias: 0.421572744846344
```

Bonus

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
```

```
# Define the Enhanced CNN Architecture
class EnhancedCNN(nn.Module):
    def __init__(self):
        super(EnhancedCNN, self).__init__()
        # Convolutional layers
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)  # Input channels = 3 (RGB), Output channels = 32
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)  # Increase channels for higher level features
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)  # Further increase for more complex features
        # Pooling layer to reduce spatial dimensions
        self.pool = nn.MaxPool2d(2, 2)
        # Batch normalization layers to stabilize learning
        self.batchnorm1 = nn.BatchNorm2d(32)
        self.batchnorm2 = nn.BatchNorm2d(64)
        self.batchnorm3 = nn.BatchNorm2d(128)
        # Dropout to prevent overfitting
        self.dropout = nn.Dropout(0.5)
        # Fully connected layers for classification
        self.fc1 = nn.Linear(128 * 4 * 4, 512)  # Flatten output and feed into dense layer
        self.fc2 = nn.Linear(512, 10)  # Final layer outputs 10 classes

    def forward(self, x):
        # Forward pass defining how the data flows through the model
        x = self.pool(F.relu(self.batchnorm1(self.conv1(x))))
        x = self.pool(F.relu(self.batchnorm2(self.conv2(x))))
        x = self.pool(F.relu(self.batchnorm3(self.conv3(x))))
        x = x.view(-1, 128 * 4 * 4)  # Flatten the output for the dense layer
        x = self.dropout(x)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Data Augmentation for Training and Normalization for both Training and Testing
transform_train = transforms.Compose([
    transforms.RandomHorizontalFlip(),  # Randomly flip images horizontally
    transforms.RandomRotation(10),  # Randomly rotate images by up to 10 degrees
    transforms.RandomAffine(0, shear=10, scale=(0.8, 1.2)),  # Random affine transformations
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2),  # Random color jitter
    transforms.ToTensor(),  # Convert images to PyTorch tensors
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])  # Normalize tensors

transform_test = transforms.Compose([
    transforms.ToTensor(),  # Convert images to PyTorch tensors
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])  # Normalize tensors

# Load CIFAR10 dataset with applied transformations
trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform_train)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform_test)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=False, num_workers=2)

# Initialize the network, loss function, and optimizer
net = EnhancedCNN()
criterion = nn.CrossEntropyLoss()  # Suitable for multi-class classification
optimizer = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)  # Stochastic Gradient Descent with momentum
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)  # Learning rate scheduler

# Training the network
for epoch in range(10):  # Loop over the dataset multiple times
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data
        optimizer.zero_grad()  # Zero the parameter gradients
        outputs = net(inputs)  # Forward pass
        loss = criterion(outputs, labels)  # Calculate loss
        loss.backward()  # Backward pass
        optimizer.step()

print('Finished Training')

# Evaluate the network on the test data
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        outputs = net(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f'Accuracy of the network on the 10000 test images: {100 * correct / total:.2f}%')
```

```
Files already downloaded and verified
Files already downloaded and verified
Finished Training
Accuracy of the network on the 10000 test images: 74.69%
```