TECHIN 513 - Basic ML

Freya Yu - 2372732 - yeyfreya@uw.edu

Instructions

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

In [ ]:

```
# use pip to install the packages
# !pip install scikit-learn TensorFlow Keras PyTorch 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

# 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 r
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# 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

clf = RandomForestClassifier()
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
accuracy = clf.score(X_test, y_test)
print(f"Random Forest Classifier Accuracy: {accuracy}")

# Task 5: Load the MNIST dataset
# use keras.datasets.mnist.load_data() to load the dataset
(X_train, y_train), (X_test, y_test) = keras.datasets.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
X_train = X_train / 255.0
X_test = X_test / 255.0
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

```python
# 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
model = Sequential([
    Dense(128, activation='relu', input_shape=(784,)),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['a
model.fit(X_train.reshape(-1, 784), y_train, epochs=10, batch_size=32)

# Task 8: Evaluate the neural network on the testing data
# use model.evaluate to get the test loss and test accuracy
test_loss, test_accuracy = model.evaluate(X_test.reshape(-1, 784), y_test)
print(f"Test Loss: {test_loss}, 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
class LinearRegression(nn.Module):
    def __init__(self):
        super(LinearRegression, self).__init__()
        self.linear = nn.Linear(1, 1)  # Assuming input and output features

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

# Task 10: Train the linear regression model on some dummy data and print th
# 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

model = LinearRegression()
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)

# Dummy data
x_train = torch.tensor([[1.], [2.], [3.]])
y_train = torch.tensor([[2.], [4.], [6.]])

# Training loop
for epoch in range(100):  # number of epochs
    model.train()
    optimizer.zero_grad()
    output = model(x_train)
    loss = criterion(output, y_train)
    loss.backward()
    optimizer.step()

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

```
Random Forest Classifier Accuracy: 1.0
Epoch 1/10
1875/1875 [==============================] - 1s 583us/step - loss: 0.2582 -
accuracy: 0.9257
Epoch 2/10
1875/1875 [==============================] - 1s 579us/step - loss: 0.1114 -
accuracy: 0.9669
Epoch 3/10
1875/1875 [==============================] - 1s 610us/step - loss: 0.0752 -
accuracy: 0.9775
Epoch 4/10
1875/1875 [==============================] - 1s 586us/step - loss: 0.0571 -
accuracy: 0.9825
Epoch 5/10
1875/1875 [==============================] - 1s 603us/step - loss: 0.0437 -
accuracy: 0.9862
Epoch 6/10
1875/1875 [==============================] - 1s 597us/step - loss: 0.0341 -
accuracy: 0.9893
Epoch 7/10
1875/1875 [==============================] - 2s 874us/step - loss: 0.0279 -
accuracy: 0.9910
Epoch 8/10
1875/1875 [==============================] - 1s 592us/step - loss: 0.0223 -
accuracy: 0.9929
Epoch 9/10
1875/1875 [==============================] - 1s 567us/step - loss: 0.0173 -
accuracy: 0.9946
Epoch 10/10
1875/1875 [==============================] - 2s 884us/step - loss: 0.0146 -
accuracy: 0.9955
313/313 [==============================] - 0s 412us/step - loss: 0.0860 - a
ccuracy: 0.9768
Test Loss: 0.0859656035900116, Test Accuracy: 0.9768000245094299
Weight: 1.9921954870224, Bias: 0.017703142017126083
```

Bonus

In [ ]:

```python
# Bonus Task: Implement a Convolutional Neural Network to classify the CIFAR
# use torchvision.datasets.CIFAR10 to load the dataset
# create a class CNN that inherit from nn.Module
# define the constructor, forward function and the network architecture
# use CrossEntropyLoss 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

# !pip install torchvision


import torch
import torchvision
import torchvision.transforms as transforms

import torch.optim as optim
import torch.nn.functional as F

# Transform the data to tensor and normalize it
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

# Load the training and testing sets
```

```
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=4,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)  # 3 input channels, 6 output channe
        self.pool = nn.MaxPool2d(2, 2)  # 2x2 pooling
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  # Fully connected layers
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)  # 10 classes

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)  # Flatten
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


model = CNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

# Training loop
for epoch in range(2):  # loop over the dataset multiple times
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss = 0.0

print('Finished Training')
```

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./da
ta/cifar-10-python.tar.gz
```

```
100%|██████████| 170498071/170498071 [00:18<00:00, 9121693.06it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
[1,  2000] loss: 2.150
[1,  4000] loss: 1.826
[1,  6000] loss: 1.678
[1,  8000] loss: 1.591
[1, 10000] loss: 1.508
[1, 12000] loss: 1.456
[2,  2000] loss: 1.419
[2,  4000] loss: 1.375
[2,  6000] loss: 1.356
[2,  8000] loss: 1.330
[2, 10000] loss: 1.327
[2, 12000] loss: 1.297
Finished Training
```