DL

1. Build a deep neural network model start with linear regression using a single variable.

Program:

```
import torch
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

data = fetch_california_housing()
X, y = data.data[:, 0], data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train.reshape(-1, 1))
X_test = scaler.transform(X_test.reshape(-1, 1))
X_train = torch.tensor(X_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32).unsqueeze(1)
model = nn.Linear(1, 1)
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
for epoch in range(100):
   predictions = model(X_train)
   loss = criterion(predictions, y_train)

   optimizer.zero_grad()
   loss.backward()
   optimizer.step()
plt.figure(figsize=(10, 6))
plt.scatter(X_train.numpy(), y_train.numpy(), label='Actual Data')
predictions = model(X_train).detach().numpy()
plt.plot(X_train.numpy(), predictions, color='red', label='Prediction Line')
plt.xlabel('Median Income')
plt.ylabel('House Value')
plt.title('True vs Predicted House Values (Single Feature)')
plt.legend()
plt.show()
```

2. Build a deep neural network model start with linear regression using a multiple variable

Program:

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from mpl_toolkits.mplot3d import Axes3D
data = fetch_california_housing()
X = data.data[:, :2]
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.float32).view(-1, 1)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32).view(-1, 1)
model = nn.Linear(X_train.shape[1], 1)  # Two features
criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01)
for epoch in range(1000):
   model.train()
   y_pred = model(X_train_tensor)
   loss = criterion(y_pred, y_train_tensor)
   optimizer.zero_grad()
   loss.backward()
   optimizer.step()
x_min, x_max = X_train[:, 0].min(), X_train[:, 0].max()
y_min, y_max = X_train[:, 1].min(), X_train[:, 1].max()
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                     np.linspace(y_min, y_max, 100))
grid_points = torch.tensor(np.c_[xx.ravel(), yy.ravel()], dtype=torch.float32)
predictions = model(grid_points).detach().numpy()
zz = predictions.reshape(xx.shape)
fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(xx, yy, zz, cmap='viridis', alpha=0.7)
ax.scatter(X_test[:, 0], X_test[:, 1], y_test, color='r', label='Test data', s=50)
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Target Value')
ax.set_title('3D Linear Regression Surface and Test Data')
plt.legend() plt.show() {next line}
```

3. Write a program to convert speech to text

Program:

```
import torch
from transformers import Wav2Vec2ForCTC, Wav2Vec2Processor
import librosa
import numpy as np
processor = Wav2Vec2Processor.from_pretrained("facebook/wav2vec2-large-960h")
model = Wav2Vec2ForCTC.from_pretrained("facebook/wav2vec2-large-960h")

audio_file = r'/content/harvard.wav'  speech, sample_rate = librosa.load(audio_file, sr=16000)
input_values = processor(speech, return_tensors="pt").input_values
with torch.no_grad():
    logits = model(input_values).logits
predicted_ids = torch.argmax(logits, dim=-1)
transcription = processor.decode(predicted_ids[0])

print("Transcription: ", transcription)
```

4. Write a program to convert text to speech.

Program:

```
pip install speechbrain
pip install speechbrain torchaudio datasets

from speechbrain.pretrained import Tacotron2, HIFIGAN
import torchaudio
tacotron2 = Tacotron2.from_hparams(source="speechbrain/tts-tacotron2-ljspeech",
savedir="tmpdir_tts")
hifi_gan = HIFIGAN.from_hparams(source="speechbrain/tts-hifigan-ljspeech", savedir="tmpdir_hifigan")
text = input()
mel_output, mel_length, alignment = tacotron2.encode_text(text)
waveforms = hifi_gan.decode_batch(mel_output)
torchaudio.save('output.wav', waveforms.squeeze(1), 22050)
```

5. Write a program to convert video into frames.

Program:

```
import cv2
from transformers import pipeline
from PIL import Image
import os
model = pipeline("image-classification", model="google/vit-base-patch16-224")

def process_video(video_path, output_dir):
    cap = cv2.VideoCapture(video_path)
    if not cap.isOpened():
        print("Error: Cannot open video.")
        return

    os.makedirs(output_dir, exist_ok=True)
    frame_idx = 0

    while True:
        ret, frame = cap.read()
        if not ret:
            break
        frame_path = os.path.join(output_dir, f"frame_{frame_idx}.jpg")
        cv2.imwrite(frame_path, frame)
        results = model(Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)))
        print(f"Frame {frame_idx}: {results}")

        frame_idx += 1

    cap.release()
    print(f"Processed {frame_idx} frames.")
process_video("/content/5_seconds_video_with_4_images.mp4", "tempframes")
```

Classification :

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.utils import to_categorical

# Step 1: Generate Random Dataset for Multi-Class Classification
np.random.seed(42)
X = np.random.rand(200, 4)                  # 200 samples, 4 input features
y = np.random.randint(0, 3, 200)            # 3 classes (0, 1, 2)

# Step 2: One-Hot Encode the Labels
y = to_categorical(y, num_classes=3)

# Step 3: Normalize Features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Step 4: Split the Dataset (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 5: Define the Multi-Class Classification Model
model = keras.Sequential([
   layers.Dense(16, activation='relu', input_shape=(4,)),  # First Hidden Layer with 16 neurons
   layers.Dense(8, activation='relu'),                  # Second Hidden Layer with 8 neurons
   layers.Dense(3, activation='softmax')                 # Output Layer with 3 neurons (one for each class)
])

# Step 6: Compile the Model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Step 7: Train the Model
model.fit(X_train, y_train, epochs=100, batch_size=16, verbose=1, validation_data=(X_test, y_test))

# Step 8: Test the Model with New Data
new_input = np.random.rand(1, 4)             # Generate new input
new_input = scaler.transform(new_input)       # Normalize input before prediction
predicted_output = model.predict(new_input)    # Predict class probabilities
predicted_class = np.argmax(predicted_output)  # Get the class with the highest probability

print("New Input:", new_input)
print("Predicted Class Probabilities:", predicted_output)
print("Predicted Class:", predicted_class)
```

Regression:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split

# Step 1: Generate Random Dataset for Regression
np.random.seed(42)
X = np.random.rand(100, 4)                # 100 samples, 4 input features
y = np.random.rand(100, 1) * 100          # 100 continuous target values (e.g., prices, temperatures)

# Step 2: Split the Dataset (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 3: Define the Regression Model
model = keras.Sequential([
   layers.Dense(8, activation='relu', input_shape=(4,)),  # First Hidden Layer with 8 neurons
   layers.Dense(8, activation='relu'),                  # Second Hidden Layer with 8 neurons
   layers.Dense(1)                                   # Output Layer (1 neuron for continuous value)
])

# Step 4: Compile the Model
model.compile(optimizer='adam', loss='mse', metrics=['mae'])

# Step 5: Train the Model
model.fit(X_train, y_train, epochs=100, batch_size=16, verbose=1, validation_data=(X_test, y_test))

# Step 6: Test the Model with New Data
new_input = np.random.rand(1, 4)             # New random input
predicted_output = model.predict(new_input)    # Predict continuous output

print("New Input:", new_input)
print("Predicted Output (Regression):", predicted_output)
```