Set 1:

(i) Import all necessary libraries required for creating the neural network model given below.

(ii) Construct the model using Sequential API. Consider the corresponding activation functions as specified in the layer type and choose the number of units in each layer accordingly.

(iii) Compile the model with the following details and print the model as shown below:

(a) Optimizer = Adam

(b) Loss function = Choose appropriately from the layer types

(c) Metrics = Choose appropriately considering the binary class classification Layer (Type)

| | Output Shape | Param # |
|---|---|---|
| layer1 - tanh (dense) | (None, 4) | 12 |
| layer2 - tanh (dense) | (None, 2) | 10 |
| layer3 - sigmoid (dense) | (None, 1) | 3 |

Total params: 25

Trainable params: 25

Non-trainable params: 0 Desc:

The model consists of three dense (fully connected) layers.

First Layer: A dense layer with 4 neurons and a 'tanh' activation function, taking an unspecified input shape.

Second Layer: A dense layer with 2 neurons, also using the 'tanh' activation function.

Third Layer: A dense layer with 1 neuron and a 'sigmoid' activation function, making it suitable for binary classification.

**Code:**

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential from
tensorflow.keras.layers import Dense
# Constructing the model model =
Sequential([
  Dense(4, activation='tanh', input_shape=(None, )),  # Layer 1 Dense(2,
  activation='tanh'),  # Layer 2
```

```
    Dense(1, activation='sigmoid')  # Layer 3
])
# Compiling the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])  # Printing the model summary
model.summary()
```

Set 2:

(i) Construct a vector consisting of first 24 integers using numpy.

(ii) Convert that numpy vector into a tensor of rank 3.

(iii) Write your observations on how the elements of the rank 3 tensor vector got rearranged.

(iv) Get the version of TensorFlow running on your machine.

(v) Get the type, number of physical devices available on your machine, and test whether the GPU is available. Print what GPU is available.

(vi) What is the need for setting a seed value in any random number generation?

DESC:

This script demonstrates key concepts in deep learning and TensorFlow:

Neural Network Construction: A simple feedforward neural network is built using Keras' Sequential API, consisting of three layers with 'tanh' and 'sigmoid' activations for binary classification.

Vector to Tensor Conversion: A NumPy vector of the first 24 integers is reshaped into a rank-3 tensor (2,3,4), illustrating how data can be structured for deep learning models.

Observations on Reshaping: The transformation maintains the sequential order of elements while organizing them into two blocks of three rows and four columns each.

Checking TensorFlow Version & GPU Availability: The script verifies the installed TensorFlow version and detects if a GPU is available for faster computations.

Importance of Setting Seed: Setting a seed ensures reproducibility, making experiments consistent and easier to debug in machine learning applications.

Code:

2) **import tensorflow**

**as tf import numpy as**

**np**

**from tensorflow import keras**

**from tensorflow.keras.models import**

**Sequential from tensorflow.keras.layers**

**import Dense**

**# Constructing the model**

**model = Sequential([**

**Dense(4, activation='tanh', input_shape=(None, )), #**

**Layer 1 Dense(2, activation='tanh'), # Layer 2**

**Dense(1, activation='sigmoid') # Layer 3**

**])**

**# Compiling the model**

```
model.compile(optimizer='adam', loss='binary_crossentropy',

metrics=['accuracy'])  # Printing the model summary

model.summary()

# Constructing a vector of first 24

integers vector = np.arange(1, 25)

print("Vector:", vector)

# Converting the vector into a rank 3 tensor

tensor = vector.reshape(2, 3, 4)

print("Rank 3 Tensor:\n", tensor)

# Observations on the rearrangement of elements

print("Observations: The elements are reshaped into a 2x3x4 structure, where the first
dimension (2) represents two blocks, ")

print("each containing 3 rows and 4 columns. The elements maintain their sequential
order within the reshaped dimensions.")

# Getting TensorFlow version

print("TensorFlow Version:",

tf._version_) # Checking for GPU

availability

gpus =

tf.config.list_physical_devices('GPU') if

gpus:

  print("GPU is available:", gpus)

else:

  print("No GPU available")

# Importance of setting seed in random number generation

print("Setting a seed ensures reproducibility, meaning that the random numbers
generated will be the same each time the code is run. This is crucial for debugging and
consistency in machine learning experiments.")
```

Set 3.Write a code for implementing LSTM model for sentiment analysis of imdb movie review using RMSProp optimizer

Desc: This script implements an LSTM model for sentiment analysis on the IMDB movie review dataset using TensorFlow and Keras:

Dataset Preparation: The IMDB dataset is loaded with the top 10,000 frequent words, and reviews are truncated or padded to a fixed length of 200 words for uniformity.

Model Architecture: The model consists of an Embedding layer to convert words into dense vectors, an LSTM layer with dropout for sequential processing, and a Dense output layer with a sigmoid activation for binary classification.

Compilation & Loss Function: The model is compiled using the RMSprop optimizer and binary cross-entropy loss, which is ideal for classification tasks.

Performance Optimization: The script checks for GPU availability to utilize hardware acceleration for faster training.

Application: This model is used to classify movie reviews as either positive or negative, making it a practical example of NLP-based sentiment analysis

**Code:**

3) **import tensorflow**

**as tf import numpy as**

**np**

**from tensorflow import keras**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense, LSTM,**

**Embedding from tensorflow.keras.datasets import imdb**

**from tensorflow.keras.preprocessing import**

**sequence # Load IMDB dataset**

**max_features = 10000 # Number of words to consider as**

**features maxlen = 200 # Cut reviews after this number of**

**words**

**(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)**

**# Preprocess data**

**x_train = sequence.pad_sequences(x_train,**

**maxlen=maxlen) x_test =**

**sequence.pad_sequences(x_test, maxlen=maxlen) #**

**Constructing LSTM model for sentiment analysis**

```
model = Sequential([
   Embedding(max_features, 128, input_length=maxlen),  # Embedding
   layer LSTM(64, dropout=0.2, recurrent_dropout=0.2),  # LSTM layer
   Dense(1, activation='sigmoid')  # Output layer
])
# Compiling the model
model.compile(optimizer='rmsprop', loss='binary_crossentropy',
metrics=['accuracy'])  # Printing the model summary
model.summary()
# Checking for GPU availability
gpus =
tf.config.list_physical_devices('GPU') if
gpus:
   print("GPU is available:", gpus)
else:
   print("No GPU available")
```

Set 04,07(both same)
Implenent a vgg16&19 CNN architecture model to classify multi image and check it's accuracy
Desc:
This script implements VGG16 and VGG19-based deep learning models for multi-class image classification:
Data Preparation: It uses ImageDataGenerator for data augmentation and preprocessing, normalizing images and splitting them into training and validation sets.

VGG16 Model: The pre-trained VGG16 model (without the top classification layer) is loaded and frozen to retain learned features, followed by a Flatten and Dense layer for classification.

VGG19 Model: Similarly, the VGG19 model is used as a feature extractor with a customized classification head.

Compilation and Optimization: Both models use the Adam optimizer and categorical cross-entropy loss, suitable for multi-class classification tasks.

Hardware Optimization: The script checks for GPU availability to speed up training using hardware acceleration

**Code:**

```
import tensorflow as tf

import numpy as np

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.applications import VGG16,

VGG19 from tensorflow.keras.layers import Dense,

Flatten

from tensorflow.keras.preprocessing.image import

ImageDataGenerator # Define paths to dataset

data_dir = "path_to_dataset"  # Change this to your dataset

path img_size = (224, 224)

batch_size = 32

# Data augmentation and preprocessing

datagen = ImageDataGenerator(rescale=1./255,

validation_split=0.2) train_generator =

datagen.flow_from_directory(

  data_dir,

  target_size=img_size,

  batch_size=batch_size,
```

```
    class_mode='categorical
    ', subset='training'
)
val_generator =
    datagen.flow_from_directory( data_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical
    ', subset='validation'
)
# Load VGG16 model
vgg16_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
vgg16_model.trainable = False
#Constructing VGG16-based model
model_vgg16 = Sequential([
    vgg16_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dense(train_generator.num_classes,
    activation='softmax')
])# Compile VGG16 model
model_vgg16.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy']) model_vgg16.summary()
# Load VGG19 model
vgg19_model = VGG19(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
vgg19_model.trainable = False
# Constructing VGG19-based model
model_vgg19 = Sequential([
    vgg19_model,
    Flatten(),
    Dense(256, activation='relu'),
```

```
    Dense(train_generator.num_classes, activation='softmax')
])
# Compile VGG19 model
model_vgg19.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy']) model_vgg19.summary()
# Checking for GPU availability
gpus =
tf.config.list_physical_devices('GPU') if
gpus:
    print("GPU is available:", gpus)
else:
    print("No GPU available")
```

Set 5:

classification of multicategorical images using cnn model from mnist dataset and vary epochs 5,10,15 and calculating accuracy with respect to corresponding epochs

desc:

CNN model for multi-class classification on the MNIST dataset and evaluates accuracy over different epochs:

Data Preparation: The MNIST dataset (handwritten digits) is loaded, normalized, and reshaped into a 4D tensor format suitable for CNNs. Labels are one-hot encoded for classification.

CNN Model Architecture: The model consists of two convolutional layers (Conv2D) with ReLU activation, MaxPooling layers for feature extraction, followed by Flatten, Dense, and Dropout layers to prevent overfitting.

Compilation & Loss Function: The model is compiled using the Adam optimizer and categorical cross-entropy loss, ideal for multi-class classification.

Training with Varying Epochs: The model is trained with 5, 10, and 15 epochs, and test accuracy is evaluated after each run to analyze its effect on performance.

Hardware Optimization: The script checks for GPU availability to accelerate training using hardware-based computations.

**Code:**

```
import tensorflow as tf

import numpy as np

from tensorflow import keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.applications import VGG16, VGG19

from tensorflow.keras.layers import Dense, Flatten, Dropout, Conv2D,

MaxPooling2D from tensorflow.keras.preprocessing.image import

ImageDataGenerator

from tensorflow.keras.datasets import

mnist from tensorflow.keras.utils import

to_categorical # Load MNIST dataset

(x_train, y_train), (x_test, y_test) =

mnist.load_data() # Preprocess data

x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0

x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') /

255.0 y_train = to_categorical(y_train, 10)

y_test = to_categorical(y_test, 10)
```

```
# Build CNN model for multi-category
classification cnn_model = Sequential([
  Conv2D(32, (3,3), activation='relu',
  input_shape=(28,28,1)), MaxPooling2D((2,2)),
  Conv2D(64, (3,3), activation='relu'),
  MaxPooling2D((2,2)),
  Flatten(),
  Dense(128,
  activation='relu'),
  Dropout(0.5),
  Dense(10, activation='softmax')
])
# Compile CNN model
cnn_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
cnn_model.summary()
# Train CNN model with varying epochs and track
accuracy for epochs in [5, 10, 15]:
  print(f"Training with {epochs} epochs:")
  history = cnn_model.fit(x_train, y_train, epochs=epochs, batch_size=32,
validation_data=(x_test, y_test))
  test_loss, test_acc = cnn_model.evaluate(x_test, y_test)
  print(f"Test Accuracy after {epochs} epochs:
  {test_acc}\n")
# Checking for GPU availability
gpus =
tf.config.list_physical_devices('GPU') if
gpus:
  print("GPU is available:", gpus)
else:
  print("No GPU available")
```

Set 8:

1. To build DL model to demonstrate use of drop out layer
2. To build DL model using Transfer learning

DESC:

two deep learning models: one utilizing a Dropout layer and another leveraging Transfer Learning with VGG16:

Dropout Model: A fully connected neural network with Dropout layers (50% dropout) is implemented to reduce overfitting by randomly deactivating neurons during training.

Transfer Learning with VGG16: The pre-trained VGG16 model (without the top classification layers) is used as a feature extractor, followed by Flatten, Dense, and Dropout layers for classification.

Data Preprocessing & Augmentation: The ImageDataGenerator is used for normalizing and augmenting images, improving model generalization.

Compilation & Optimization: Both models use the Adam optimizer and categorical cross-entropy loss, making them suitable for multi-class classification tasks.

Hardware Optimization: The script checks for GPU availability, which can significantly speed up training and inference in deep learning models.

Code:

1) import tensorflow as tf

```
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
# Load MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# Normalize data (scale pixel values between 0 and 1)
x_train, x_test = x_train / 255.0, x_test / 255.0
# Convert labels to categorical format
y_train, y_test = to_categorical(y_train, 10)
# Build Deep Learning model with Dropout
model = Sequential([
    Flatten(input_shape=(28, 28)),  # Flatten 2D image to 1D
    Dense(512, activation='relu'),
```

```
    Dropout(0.5),  # Dropout to reduce overfitting
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')  # Output layer (10 classes)
])
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Model summary
model.summary()
# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=32, validation_data=(x_test, y_test))
# Evaluate model performance
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc}")
# Check for GPU availability
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    print("GPU is available:", gpus)
else:
    print("No GPU available")
```

2)
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.applications import VGG16
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Define dataset path (Change this to your dataset directory)
data_dir = "path_to_dataset"
img_size = (224, 224)
batch_size = 32
```

```
# Data Augmentation and Preprocessing
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
train_generator = datagen.flow_from_directory(
    data_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)
val_generator = datagen.flow_from_directory(
    data_dir,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)
# Load Pre-trained VGG16 Model (without top layers)
vgg16_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
vgg16_model.trainable = False  # Freeze layers
# Build Transfer Learning Model
model = Sequential([
    vgg16_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(train_generator.num_classes, activation='softmax')  # Output layer
])
# Compile the Model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Model Summary
model.summary()
```

```
# Train the Model
model.fit(train_generator, epochs=10, validation_data=val_generator)
# Check for GPU Availability
gpus = tf.config.list_physical_devices('GPU')
if gpus:
    print("GPU is available:", gpus)
else:
    print("No GPU available")
```

Set 9:

i) implement a dl model to show the usage of early stopping

ii) implement a dl model to show the usage of data Augmentation technique

desc:

a Deep Learning model for House Price Prediction using Early Stopping to prevent overfitting:

Data Preprocessing: The dataset is loaded, and features are normalized using StandardScaler for improved model performance.

Train-Test Split: The data is split into 80% training and 20% testing to evaluate model performance.

Neural Network Architecture: A fully connected neural network with ReLU activation and Dropout layers (30%) is built to improve generalization.

Early Stopping: The model monitors validation loss and stops training if it does not improve for 5 consecutive epochs, restoring the best weights.

Evaluation & GPU Check: The model is evaluated using Mean Absolute Error (MAE), a key metric for regression, and GPU availability is checked for hardware acceleration

Code:

**9a)import tensorflow as tf**

**import numpy as np**

**import pandas as pd**

**from tensorflow import keras**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.layers import Dense,**

**Dropout from tensorflow.keras.callbacks import**

**EarlyStopping from sklearn.model_selection import**

**train_test_split from sklearn.preprocessing import**

**StandardScaler**

**# Load dataset (Replace 'house_prices.csv' with the actual dataset**

**path) df = pd.read_csv('house_prices.csv')**

**# Preprocess data**

**X = df.drop(columns=['price']) # Features**

**y = df['price'] # Target variable**

**# Normalize features**

**scaler = StandardScaler()**

**X =**

**scaler.fit_transform(X)**

```
# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) # Build House Price Prediction Model with Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
house_price_model = Sequential([
   Dense(128, activation='relu',
   input_shape=(X_train.shape[1],)), Dropout(0.3),
   Dense(64,
   activation='relu'),
   Dropout(0.3),
   Dense(32, activation='relu'),
   Dense(1)  # Output layer for regression
])
# Compile the model
house_price_model.compile(optimizer='adam', loss='mse', metrics=['mae'])
house_price_model.summary()
# Train the model with Early Stopping
house_price_model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test,
y_test), callbacks=[early_stopping])
# Evaluate the model
test_loss, test_mae = house_price_model.evaluate(X_test,
y_test) print(f"Test MAE: {test_mae}")
# Checking for GPU availability
gpus =
tf.config.list_physical_devices('GPU') if
gpus:
   print("GPU is available:", gpus)
else:
   print("No GPU available")
```

```
9b) import tensorflow as
tf import numpy as np
import pandas as pd
```

```
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D,
Flatten from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import
EarlyStopping from sklearn.model_selection import
train_test_split from sklearn.preprocessing import
StandardScaler
# Load dataset (Replace 'house_prices.csv' with the actual dataset
path) df = pd.read_csv('house_prices.csv')
# Preprocess data
X = df.drop(columns=['price']) # Features
y = df['price'] # Target variable
# Normalize features
scaler = StandardScaler()
X =
scaler.fit_transform(X) #
Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Build House Price Prediction Model with Early Stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
house_price_model = Sequential([
  Dense(128, activation='relu',
  input_shape=(X_train.shape[1],)), Dropout(0.3),
  Dense(64,
  activation='relu'),
  Dropout(0.3),
  Dense(32, activation='relu'),
  Dense(1) # Output layer for regression
])
# Compile the model
house_price_model.compile(optimizer='adam', loss='mse', metrics=['mae'])
house_price_model.summary()
```

```
# Train the model with Early Stopping

house_price_model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test,
y_test), callbacks=[early_stopping])

# Evaluate the model

test_loss, test_mae = house_price_model.evaluate(X_test,

y_test) print(f"Test MAE: {test_mae}")

# Implementing Data Augmentation

image_data_generator = ImageDataGenerator(

   rescale=1./255,

   rotation_range=30,

   width_shift_range=0.2,

   height_shift_range=0.2,

   shear_range=0.2,

   zoom_range=0.2,

   horizontal_flip=True,

   fill_mode='nearest'

)

# Example usage (assuming an image dataset is available)

# train_data = image_data_generator.flow_from_directory('path_to_images', target_size=(224, 224),
batch_size=32, class_mode='categorical')

# Checking for GPU availability

gpus =

tf.config.list_physical_devices('GPU') if

gpus:

   print("GPU is available:", gpus)

else:

   print("No GPU available")
```

Set-10:

implement a Long Short-Term Memory (LSTM) for predicting time series data. (Next Word Prediction)

desc:

a CNN for multi-category classification on MNIST and an LSTM for next-word prediction.

CNN Model for Image Classification:

Uses Conv2D & MaxPooling layers to extract image features.

Dropout (50%) is applied to prevent overfitting.

The model is trained for 5, 10, and 15 epochs to compare accuracy trends.

LSTM for Next-Word Prediction:

A Tokenizer processes text into numerical sequences.

Sequences are padded to ensure uniform input size.

The model includes an Embedding layer, LSTM layer (100 units), and a Dense softmax layer to predict the next word.

Evaluation & GPU Check: The script verifies GPU availability for faster computation, making it suitable for deep learning tasks.

**Code:**

**10)import tensorflow as tf**

**import numpy as np**

**from tensorflow import keras**

**from tensorflow.keras.models import Sequential**

**from tensorflow.keras.applications import VGG16, VGG19**

**from tensorflow.keras.layers import Dense, Flatten, Dropout, Conv2D, MaxPooling2D, LSTM, Embedding**

**from tensorflow.keras.preprocessing.image import**

**ImageDataGenerator from tensorflow.keras.datasets import mnist**

**from tensorflow.keras.utils import to_categorical**

**from tensorflow.keras.preprocessing.sequence import pad_sequences**

**from tensorflow.keras.preprocessing.text import Tokenizer**

**# Load MNIST dataset**

**(x_train, y_train), (x_test, y_test) =**

**mnist.load_data() # Preprocess data**

**x_train = x_train.reshape(-1, 28, 28, 1).astype('float32') / 255.0**

```
x_test = x_test.reshape(-1, 28, 28, 1).astype('float32') / 255.0

y_train = to_categorical(y_train, 10)

y_test = to_categorical(y_test, 10)

# Build CNN model for multi-category classification

cnn_model = Sequential([

  Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),

  MaxPooling2D((2,2)),

  Conv2D(64, (3,3), activation='relu'),

  MaxPooling2D((2,2)),

  Flatten(),

  Dense(128,

  activation='relu'),

  Dropout(0.5),

  Dense(10, activation='softmax')

])

# Compile CNN model

cnn_model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

cnn_model.summary()

# Train CNN model with varying epochs and track accuracy

for epochs in [5, 10, 15]:

  print(f"Training with {epochs} epochs:")

  history = cnn_model.fit(x_train, y_train, epochs=epochs, batch_size=32, validation_data=(x_test,
y_test))

  test_loss, test_acc = cnn_model.evaluate(x_test, y_test)

  print(f"Test Accuracy after {epochs} epochs: {test_acc}\n")

# Implementing LSTM for Next Word Prediction

corpus = ["hello how are you", "hi what are you doing", "where are you going", "good morning
have a nice day", "stay happy and healthy"]

# Tokenization

tokenizer = Tokenizer()

tokenizer.fit_on_texts(corpus)

total_words = len(tokenizer.word_index) +

1 # Creating sequences
```

```
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)
# Padding sequences
max_sequence_length = max(len(seq) for seq in input_sequences)
input_sequences = pad_sequences(input_sequences, maxlen=max_sequence_length,
padding='pre') X, y = input_sequences[:, :-1], input_sequences[:, -1]
y = to_categorical(y, num_classes=total_words)
# Build LSTM model
lstm_model = Sequential([
    Embedding(total_words, 10, input_length=max_sequence_length - 1),
    LSTM(100),
    Dense(total_words, activation='softmax')
])
# Compile LSTM model
lstm_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
lstm_model.summary()
# Train LSTM model
lstm_model.fit(X, y, epochs=100,
verbose=1) # Checking for GPU availability
gpus =
tf.config.list_physical_devices('GPU') if
gpus:
    print("GPU is available:", gpus)
else:
    print("No GPU available")
```

Set 11:

House price prediction

Desc:

house price prediction model using deep learning.

Data Preprocessing:

Loads the house prices dataset and separates features (X) from the target variable (y).

Uses StandardScaler for feature normalization to improve model convergence.

Splits data into training (80%) and testing (20%) sets.

Deep Learning Model:

A Sequential model with three Dense layers (128, 64, 32 units, ReLU activation).

Dropout (30%) is added to prevent overfitting.

The output layer has one neuron (for price prediction) without activation (regression task).

Model Compilation & Training:

Uses Adam optimizer, Mean Squared Error (MSE) loss, and Mean Absolute Error (MAE) as a metric.

Trains for 50 epochs with batch size 32, tracking validation performance.

Evaluation & GPU Check:

Evaluates model performance on the test set using MAE.

Checks for GPU availability for optimized deep learning training.

**Code:**

```
import tensorflow as tf
import numpy as np
import pandas as pd
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense,
Dropout from sklearn.model_selection import
train_test_split from sklearn.preprocessing import
StandardScaler
# Load dataset (Replace 'house_prices.csv' with the actual dataset
path) df = pd.read_csv('house_prices.csv')
# Preprocess data
X = df.drop(columns=['price'])  # Features
```

```
y = df['price']  # Target variable
# Normalize features
scaler = StandardScaler()
X =
scaler.fit_transform(X) #
Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42) # Build House Price Prediction Model
house_price_model = Sequential([
   Dense(128, activation='relu',
   input_shape=(X_train.shape[1],)), Dropout(0.3),
   Dense(64,
   activation='relu'),
   Dropout(0.3),
   Dense(32, activation='relu'),
   Dense(1)  # Output layer for regression
])
# Compile the model
house_price_model.compile(optimizer='adam', loss='mse', metrics=['mae'])
house_price_model.summary()
# Train the model
house_price_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test,
y_test)) # Evaluate the model
test_loss, test_mae = house_price_model.evaluate(X_test,
y_test) print(f"Test MAE: {test_mae}")
# Checking for GPU availability
gpus =
tf.config.list_physical_devices('GPU') if
gpus:
   print("GPU is available:", gpus)
else:
   print("No GPU available")
```