Tut 1

Installing Anaconda or Miniconda and Setting Up TensorFlow and Keras

Step 1: Install Anaconda or Miniconda

Option 1: Install Anaconda

Anaconda is a full-fledged distribution that includes Python, Jupyter Notebook, and many data science libraries.

1. Download the Anaconda installer from the official website: https://www.anaconda.com/download
2. Choose the version based on your operating system (Windows, macOS, Linux)
3. Follow the installation instructions:
    - On Windows: Run the `.exe` installer and follow the prompts.
    - On macOS/Linux: Use the `.sh` installer in the terminal.
4. After installation, restart your terminal or command prompt.

Option 2: Install Miniconda (Lightweight Alternative)

Miniconda is a minimal installer for conda, useful if you want to save space and install only necessary packages.

1. Download Miniconda from https://docs.conda.io/en/latest/miniconda.html
2. Follow the installation steps similar to Anaconda.

Step 2: Set Up a Conda Environment for TensorFlow and Keras

Once Anaconda/Miniconda is installed, create a virtual environment to avoid conflicts with other Python libraries.

1. Open a terminal (Anaconda Prompt, Command Prompt, or Terminal)
2. Run the following command to create an environment (e.g., named `tensorflow_env`):

```
3. conda create --name tensorflow_env python=3.9
```

4. Activate the environment:
    - On Windows:
    - `conda activate tensorflow_env`
    - On macOS/Linux:
    - `source activate tensorflow_env`

Step 3: Install TensorFlow and Keras

With the environment activated, install TensorFlow and Keras:

```
pip install tensorflow
```

This will install both TensorFlow and Keras (since Keras is integrated into TensorFlow 2.x).

To verify the installation, run:

```
python -c "import tensorflow as tf; print(tf.__version__)"
```

This should print the installed TensorFlow version.

Step 4: Install Additional Packages (Optional)

You may also need the following packages for deep learning projects:

```
pip install numpy pandas matplotlib jupyterlab scikit-learn
```

If using Anaconda, you can also install them with:

```
conda install numpy pandas matplotlib jupyterlab scikit-learn
```

Step 5: Running Jupyter Notebook (Optional)

To use Jupyter Notebook with TensorFlow:

```
jupyter notebook
```

Step 6: Test a Simple TensorFlow and Keras Model

Create a simple neural network to verify the installation:

```
import tensorflow as tf
from tensorflow import keras

# Define a simple model
model = keras.Sequential([
    keras.layers.Dense(10, activation='relu', input_shape=(5,)),
    keras.layers.Dense(1, activation='sigmoid')
])

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

print("Model created successfully!")
```

Tut 2

Introduction to Google Colab and Working with GPUs and TPUs

Google Colab (Colaboratory) is a cloud-based Jupyter Notebook environment that allows users to write and execute Python code with free access to GPUs and TPUs. This makes it an excellent tool for deep learning, machine learning, and large-scale computational tasks.

Step 1: Accessing Google Colab

1. Open your web browser and go to Google Colab.
2. Sign in with your Google account.
3. Click on **New Notebook** to create a new Colab notebook.

Step 2: Setting Up the Runtime with GPU or TPU

By default, Colab runs on a standard CPU environment. To enable GPU or TPU:

1. Click on **Runtime** in the top menu.
2. Select **Change runtime type**.
3. Under "Hardware accelerator", choose **GPU** or **TPU**.
4. Click **Save**.

Step 3: Checking GPU and TPU Availability

Once you enable GPU or TPU, verify that it is available.

Checking GPU Availability

```
import tensorflow as tf
print("GPU Available:", tf.config.list_physical_devices('GPU'))
```

Checking TPU Availability

```
import tensorflow as tf
try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    print("TPU Available")
except:
    print("No TPU found")
```

Step 4: Installing and Importing Libraries

Colab comes with many pre-installed libraries, but you may need additional ones.

```
!pip install tensorflow numpy pandas matplotlib
```

For specific versions, use:

```
!pip install tensorflow==2.9
```

Step 5: Uploading and Accessing Files in Google Colab

Upload Files from Local Machine

```
from google.colab import files
uploaded = files.upload()
```

Mount Google Drive for Persistent Storage

```
from google.colab import drive
drive.mount('/content/drive')
```

Step 6: Running a Simple Deep Learning Model on GPU

Here's a simple neural network using TensorFlow and Keras:

```
import tensorflow as tf
from tensorflow import keras

# Define a basic model
model = keras.Sequential([
    keras.layers.Dense(128, activation='relu', input_shape=(784,)),
    keras.layers.Dense(10, activation='softmax')
])

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

print("Model created successfully!")
```

Step 7: Running a Model on TPU

Use the TPU strategy to distribute computations across TPU cores:

```
import tensorflow as tf

try:
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver()
    tf.config.experimental_connect_to_cluster(tpu)
    tf.tpu.experimental.initialize_tpu_system(tpu)
    strategy = tf.distribute.TPUStrategy(tpu)
    print("Running on TPU!")
except:
    print("No TPU found")
```

Tut 3

Developing a Simple Perceptron (Single-Layer Neural Network)

Introduction

A perceptron is one of the simplest types of artificial neural networks. It is a single-layer neural network used for binary classification problems. This tutorial will guide you through the development of a simple perceptron using Python and TensorFlow/Keras.

Step 1: Install Required Libraries

Ensure you have the necessary libraries installed. If you haven't installed TensorFlow yet, use:

```
pip install tensorflow numpy matplotlib scikit-learn
```

Step 2: Import Necessary Modules

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Step 3: Generate or Load Data

For this example, we'll generate a simple dataset using NumPy.

```
# Generate a simple dataset (binary classification)
n_samples = 200
np.random.seed(42)
X = np.random.rand(n_samples, 2) * 2 - 1  # Features between -1 and 1
y = (X[:, 0] + X[:, 1] > 0).astype(int)  # Labels: 1 if sum > 0, else 0
```

Step 4: Preprocess Data

```
# Split 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)

# Normalize features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Step 5: Build the Perceptron Model

A perceptron consists of a single layer with one neuron.

```
# Define a simple perceptron model
model = keras.Sequential([
    keras.layers.Dense(1, activation='sigmoid', input_shape=(2,))  # Single
neuron with sigmoid activation
])

# Compile the model
model.compile(optimizer='sgd', loss='binary_crossentropy',
metrics=['accuracy'])
```

Step 6: Train the Model

```
# Train the perceptron
history = model.fit(X_train, y_train, epochs=50, batch_size=10,
validation_data=(X_test, y_test))
```

Step 7: Evaluate the Model

```
# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

Step 8: Visualizing the Decision Boundary

```
# Plot decision boundary
def plot_decision_boundary(model, X, y):
    x_min, x_max = X[:, 0].min() - 0.1, X[:, 0].max() + 0.1
    y_min, y_max = X[:, 1].min() - 0.1, X[:, 1].max() + 0.1
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100), np.linspace(y_min,
y_max, 100))

    Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)

    plt.contourf(xx, yy, Z, levels=[0, 0.5, 1], colors=['blue', 'red'],
alpha=0.3)
    plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k', cmap=plt.cm.binary)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title('Perceptron Decision Boundary')
    plt.show()

plot_decision_boundary(model, X_test, y_test)
```

Tut 4

Designing and Developing a Model for Text Generation Using LSTM

Introduction

Long Short-Term Memory (LSTM) networks are a type of Recurrent Neural Network (RNN) designed to process sequential data while maintaining long-range dependencies. One of their powerful applications is text generation, where a model learns to predict the next word or character in a sequence. In this tutorial, we will design and develop an LSTM-based model for text generation using TensorFlow and Keras.

Step 1: Install Required Libraries

Ensure you have TensorFlow and other necessary libraries installed.

```
pip install tensorflow numpy matplotlib scikit-learn pandas
```

Step 2: Import Necessary Modules

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import LSTM, Embedding, Dense
from tensorflow.keras.models import Sequential
```

Step 3: Load and Prepare Data

We will use a sample text dataset and preprocess it for training.

```
# Sample text data for training
data = """Deep learning enables models to learn patterns from data.
Recurrent neural networks are powerful for sequential data.
LSTM networks are designed to handle long-term dependencies.
Text generation is an exciting application of LSTMs."""

# Tokenize words
tokenizer = Tokenizer()
tokenizer.fit_on_texts([data])
total_words = len(tokenizer.word_index) + 1

# Convert text to sequences
input_sequences = []
for line in data.split('\n'):
    token_list = tokenizer.texts_to_sequences([line])[0]
```

```
    for i in range(1, len(token_list)):
        input_sequences.append(token_list[:i+1])

# Pad sequences
max_sequence_length = max([len(seq) for seq in input_sequences])
input_sequences = pad_sequences(input_sequences,
maxlen=max_sequence_length, padding='pre')

# Split data into input (X) and output (y)
X, y = input_sequences[:, :-1], input_sequences[:, -1]
y = keras.utils.to_categorical(y, num_classes=total_words)
```

Step 4: Build an LSTM Model

```
# Define LSTM model
def build_lstm_model():
    model = Sequential([
        Embedding(total_words, 64, input_length=max_sequence_length-1),
        LSTM(100, return_sequences=True),
        LSTM(100),
        Dense(100, activation='relu'),
        Dense(total_words, activation='softmax')
    ])

    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    return model
```

Step 5: Train the LSTM Model

```
# Create the model
lstm_model = build_lstm_model()

# Train the model
lstm_model.fit(X, y, epochs=100, verbose=1)
```

Step 6: Generate Text

```
import random

def generate_text(seed_text, next_words, model, max_sequence_length):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list],
maxlen=max_sequence_length-1, padding='pre')
        predicted = np.argmax(model.predict(token_list), axis=-1)[0]

        for word, index in tokenizer.word_index.items():
            if index == predicted:
                seed_text += " " + word
                break
    return seed_text

# Generate new text based on a seed phrase
seed_text = "Text generation"
generated_text = generate_text(seed_text, 10, lstm_model,
max_sequence_length)print(generated_text)
```

Tut 5

Designing and Developing a Basic CNN for Image Classification

In this tutorial, we will design and develop a **basic Convolutional Neural Network (CNN)** for image classification using **TensorFlow and Keras**. We will use the **MNIST dataset** (handwritten digits) as an example.

Step 1: Install Required Libraries

Ensure you have the necessary libraries installed. Run the following command if needed:

pip install numpy tensorflow keras matplotlib

Step 2: Import Required Libraries

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
```

Step 3: Load and Preprocess the Dataset

We will use the **MNIST dataset**, which consists of **28x28 grayscale images** of handwritten digits (0-9).

```
# Load dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
# Normalize pixel values to range 0-1
X_train, X_test = X_train / 255.0, X_test / 255.0

# Reshape the data to fit CNN input format (batch_size, height, width, channels)
X_train = X_train.reshape(-1, 28, 28, 1)
X_test = X_test.reshape(-1, 28, 28, 1)

# Convert labels to categorical format (one-hot encoding)
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)

# Display an example image
plt.imshow(X_train[0].reshape(28, 28), cmap='gray')
plt.title(f"Label: {np.argmax(y_train[0])}")
plt.show()
```

Step 4: Build the CNN Model

A basic CNN architecture consists of:

- **Convolutional Layers** to extract features
- **Max Pooling Layers** to reduce dimensionality
- **Flattening** to convert feature maps into a single vector
- **Dense Layers** for classification

```
# Build CNN Model
model = Sequential([
    Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(28,28,1)),
    MaxPooling2D(pool_size=(2,2)),

    Conv2D(64, kernel_size=(3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),

    Flatten(),
```

```
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')  # Output layer with 10 classes
])

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

# Display model architecture
model.summary()
```

Step 5: Train the Model

```
# Train the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32)
```

Step 6: Evaluate the Model

```
# Evaluate on test data
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {test_acc:.4f}")
```

Step 7: Make Predictions

```
# Make predictions
predictions = model.predict(X_test)

# Display a test image and predicted label
index = 0  # Change index to test different images
plt.imshow(X_test[index].reshape(28, 28), cmap='gray')
plt.title(f"Predicted Label: {np.argmax(predictions[index])}")
plt.show()
```

Tut 6

Using Transfer Learning in CNN for Image Classification

In this tutorial, we will use **transfer learning** to classify images using a pre-trained **CNN model (VGG16)**. Instead of training a deep network from scratch, we will **fine-tune** a pre-trained model on a custom dataset (e.g., **Cats vs. Dogs**).

Step 1: Install Required Libraries

Ensure you have the necessary libraries installed:

```
pip install numpy tensorflow keras matplotlib
```

Step 2: Import Required Libraries

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Step 3: Load Pre-Trained VGG16 Model

We will use **VGG16**, which is pre-trained on **ImageNet** and contains knowledge of **millions of images**.
We exclude the **top layers (fully connected layers)** to customize the model for our own classification task.

```
# Load VGG16 model without top layers
```

```
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(150, 150, 3))


# Freeze the convolutional base
for layer in base_model.layers:
    layer.trainable = False
```

Step 4: Build the Transfer Learning Model

We will **add new layers** to customize the pre-trained model for our classification task.

```
# Add custom layers on top of VGG16
model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(2, activation='softmax')  # Output layer for 2 classes (Cats & Dogs)
])


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


# Display model summary
model.summary()
```

Step 5: Load and Prepare Data

We will use **ImageDataGenerator** to load images from a directory and apply data augmentation.

```
# Define directories (update paths as needed)
train_dir = "path/to/train"
val_dir = "path/to/validation"
```

```
# Data augmentation for training images
train_datagen = 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)

# Only rescaling for validation images
val_datagen = ImageDataGenerator(rescale=1./255)

# Load training data
train_generator = train_datagen.flow_from_directory(train_dir, target_size=(150, 150),
                                                    batch_size=32, class_mode='categorical')

# Load validation data
val_generator = val_datagen.flow_from_directory(val_dir, target_size=(150, 150),
                                                batch_size=32, class_mode='categorical')
```

Step 6: Train the Model

```
# Train the model
history = model.fit(train_generator, validation_data=val_generator, epochs=10)
```

Step 7: Evaluate and Test the Model

```
# Evaluate on validation set
loss, accuracy = model.evaluate(val_generator)
print(f"Validation Accuracy: {accuracy:.4f}")

# Predict on a single image
import cv2
```

```
# Load an image for testing
image_path = "path/to/test/image.jpg"
image = cv2.imread(image_path)
image = cv2.resize(image, (150, 150))
image = np.expand_dims(image, axis=0) / 255.0  # Normalize

# Make prediction
prediction = model.predict(image)
predicted_class = np.argmax(prediction)
print(f"Predicted Class: {predicted_class}")
```

Tut 7

Designing and Developing a Simple RNN for Sequence Classification

In this tutorial, we will design and develop a **simple Recurrent Neural Network (RNN)** using **TensorFlow/Keras** to classify **sentiment in text data (positive or negative reviews).**

Step 1: Install Required Libraries

Ensure you have the necessary libraries installed:

pip install numpy tensorflow keras matplotlib

Step 2: Import Required Libraries

```
import numpy as np

import tensorflow as tf

from tensorflow import keras

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import SimpleRNN, Dense, Embedding

from tensorflow.keras.preprocessing.text import Tokenizer

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

Step 3: Prepare the Dataset

We will use a **small dataset of movie reviews** (positive and negative sentiments).

```
# Sample dataset (10 text reviews with labels)

reviews = [

    "I love this movie", "This film was amazing", "Best movie ever", "Really enjoyed it",
```

```
    "It was a fantastic experience", "I hate this movie", "Worst film I have seen",
    "Not good at all", "Terrible experience", "Completely disappointing"
]

labels = [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]  # 1 = Positive, 0 = Negative
```

Tokenization and Sequence Preparation

```
# Tokenizing the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(reviews)
total_words = len(tokenizer.word_index) + 1

# Convert text to sequences
sequences = tokenizer.texts_to_sequences(reviews)

# Padding sequences to make them the same length
max_seq_length = max([len(seq) for seq in sequences])
X = pad_sequences(sequences, maxlen=max_seq_length, padding='post')

# Convert labels to NumPy array
y = np.array(labels)
```

Step 4: Build the Simple RNN Model

A simple **Recurrent Neural Network (RNN)** consists of:

- **Embedding Layer**: Converts words into dense vectors
- **SimpleRNN Layer**: Processes sequences

- **Dense Layer**: Outputs classification (positive/negative)

```
# Build Simple RNN Model
model = Sequential([
    Embedding(input_dim=total_words, output_dim=8, input_length=max_seq_length),
    SimpleRNN(16, activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification (Positive/Negative)
])

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

# Display model summary
model.summary()
```

Step 5: Train the Model

```
# Train the model
history = model.fit(X, y, epochs=20, batch_size=2, verbose=1)
```

Step 6: Make Predictions

```
# Function to classify new reviews
def predict_sentiment(text):
    sequence = tokenizer.texts_to_sequences([text])
    sequence = pad_sequences(sequence, maxlen=max_seq_length, padding='post')
    prediction = model.predict(sequence)[0][0]
```

```
    return "Positive" if prediction > 0.5 else "Negative"


# Test with a new review
print(predict_sentiment("This movie was fantastic"))
print(predict_sentiment("I did not like this film at all"))
```

Tut 8

Designing and Developing an RNN with LSTM for Sentiment Analysis

In this tutorial, we will design and develop an **RNN with LSTM (Long Short-Term Memory)** to classify **sentiment in text data (positive or negative movie reviews)** using **TensorFlow and Keras**.

Step 1: Install Required Libraries

Ensure you have the necessary libraries installed:

```
pip install numpy tensorflow keras matplotlib
```

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

Step 3: Prepare the Dataset

We will use a **small dataset of movie reviews** (positive and negative sentiments).

```
# Sample dataset (10 text reviews with labels)
reviews = [
    "I love this movie", "This film was amazing", "Best movie ever", "Really enjoyed it",
    "It was a fantastic experience", "I hate this movie", "Worst film I have seen",
    "Not good at all", "Terrible experience", "Completely disappointing"
]

labels = [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]  #1 = Positive, 0 = Negative
```

Tokenization and Sequence Preparation

```
# Tokenizing the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(reviews)
total_words = len(tokenizer.word_index) + 1

# Convert text to sequences
sequences = tokenizer.texts_to_sequences(reviews)

# Padding sequences to make them the same length
max_seq_length = max([len(seq) for seq in sequences])
X = pad_sequences(sequences, maxlen=max_seq_length, padding='post')

# Convert labels to NumPy array
y = np.array(labels)
```

Step 4: Build the LSTM Model

An **LSTM (Long Short-Term Memory)** model is designed to handle **long-term dependencies** in sequences.

```
# Build LSTM Model

model = Sequential([

    Embedding(input_dim=total_words, output_dim=8, input_length=max_seq_length),

    LSTM(16, activation='tanh', return_sequences=False),

    Dense(1, activation='sigmoid')  # Binary classification (Positive/Negative)

])



# Compile the model

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])



# Display model summary

model.summary()
```

Step 5: Train the Model

```
# Train the model

history = model.fit(X, y, epochs=20, batch_size=2, verbose=1)
```

Step 6: Make Predictions

```
# Function to classify new reviews

def predict_sentiment(text):

    sequence = tokenizer.texts_to_sequences([text])

    sequence = pad_sequences(sequence, maxlen=max_seq_length, padding='post')
```

```
    prediction = model.predict(sequence)[0][0]
    return "Positive" if prediction > 0.5 else "Negative"

# Test with a new review
print(predict_sentiment("This movie was fantastic"))
print(predict_sentiment("I did not like this film at all"))
```

Tut 9

Designing and Developing an RNN with GRU for Sentiment Analysis

In this tutorial, we will design and develop an **RNN with GRU (Gated Recurrent Unit)** to classify **sentiment in text data (positive or negative movie reviews)** using **TensorFlow and Keras**.

Step 1: Install Required Libraries

Ensure you have the necessary libraries installed:

```
pip install numpy tensorflow keras matplotlib
```

Step 2: Import Required Libraries

```
import numpy as np

import tensorflow as tf

from tensorflow import keras

import matplotlib.pyplot as plt

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import GRU, Dense, Embedding

from tensorflow.keras.preprocessing.text import Tokenizer

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

Step 3: Prepare the Dataset

We will use a **small dataset of movie reviews** (positive and negative sentiments).

```
# Sample dataset (10 text reviews with labels)
reviews = [
    "I love this movie", "This film was amazing", "Best movie ever", "Really enjoyed it",
    "It was a fantastic experience", "I hate this movie", "Worst film I have seen",
    "Not good at all", "Terrible experience", "Completely disappointing"
]

labels = [1, 1, 1, 1, 1, 0, 0, 0, 0, 0]  # 1 = Positive, 0 = Negative
```

Tokenization and Sequence Preparation

```
# Tokenizing the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(reviews)
total_words = len(tokenizer.word_index) + 1

# Convert text to sequences
sequences = tokenizer.texts_to_sequences(reviews)

# Padding sequences to make them the same length
max_seq_length = max([len(seq) for seq in sequences])
X = pad_sequences(sequences, maxlen=max_seq_length, padding='post')

# Convert labels to NumPy array
y = np.array(labels)
```

Step 4: Build the GRU Model

A **GRU (Gated Recurrent Unit)** model is designed to **efficiently capture long-term dependencies** in sequences while using fewer parameters than LSTMs.

```
# Build GRU Model

model = Sequential([

    Embedding(input_dim=total_words, output_dim=8, input_length=max_seq_length),

    GRU(16, activation='tanh', return_sequences=False),

    Dense(1, activation='sigmoid')  # Binary classification (Positive/Negative)

])


# Compile the model

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


# Display model summary

model.summary()
```

Step 5: Train the Model

```
# Train the model

history = model.fit(X, y, epochs=20, batch_size=2, verbose=1)
```

Step 6: Make Predictions

```
# Function to classify new reviews

def predict_sentiment(text):
```

```
    sequence = tokenizer.texts_to_sequences([text])
    sequence = pad_sequences(sequence, maxlen=max_seq_length, padding='post')
    prediction = model.predict(sequence)[0][0]
    return "Positive" if prediction > 0.5 else "Negative"

# Test with a new review
print(predict_sentiment("This movie was fantastic"))
print(predict_sentiment("I did not like this film at all"))
```

Tut 10

Designing and Developing a Text Generation Model Using LSTM

In this tutorial, we will **design and develop a text generation model using LSTM (Long Short-Term Memory)** in TensorFlow/Keras. The model will be trained on a sample text corpus and will generate new text sequences based on learned patterns.

Step 1: Install Required Libraries

Ensure you have the necessary libraries installed:

pip install numpy tensorflow keras matplotlib

Step 2: Import Required Libraries

import numpy as np

import tensorflow as tf

from tensorflow import keras

import matplotlib.pyplot as plt

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences

from tensorflow.keras.models import Sequential

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

Step 3: Prepare the Dataset

We will use a **sample text corpus** for training.

Sample text dataset

text = """Deep learning is a subset of machine learning in artificial intelligence.

It enables computers to learn from data and make intelligent decisions.

Neural networks are used to process and understand complex patterns.

LSTM networks are useful for text generation and natural language processing.

With enough training, deep learning models can generate realistic text.

The power of AI is transforming the world in multiple ways."""

Tokenization and Sequence Preparation

```
# Tokenizing the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts([text])
total_words = len(tokenizer.word_index) + 1  # Total unique words

# Convert text into sequences
input_sequences = []
words = text.split()

for i in range(1, len(words)):
    sequence = words[:i + 1]
    input_sequences.append(tokenizer.texts_to_sequences([" ".join(sequence)])[0])

# Padding sequences to make them the same length
max_seq_length = max([len(seq) for seq in input_sequences])
input_sequences = pad_sequences(input_sequences, maxlen=max_seq_length, padding='pre')
```

```
# Separate inputs (X) and targets (y)

X, y = input_sequences[:, :-1], input_sequences[:, -1]

y = keras.utils.to_categorical(y, num_classes=total_words)
```

Step 4: Build the LSTM Model

The model consists of:

- **Embedding Layer**: Converts words into dense vectors
- **LSTM Layer**: Captures sequential dependencies
- **Dense Layer**: Predicts the next word

```
# Build LSTM Model

model = Sequential([

    Embedding(input_dim=total_words, output_dim=10, input_length=max_seq_length - 1),

    LSTM(100, return_sequences=True),

    LSTM(100),

    Dense(total_words, activation='softmax')

])


# Compile the model

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


# Display model summary

model.summary()
```

Step 5: Train the Model

```
# Train the model
history = model.fit(X, y, epochs=200, verbose=1)
```

Step 6: Generate New Text

```
import random


# Function to generate text
def generate_text(seed_text, next_words=10):
    for _ in range(next_words):
        sequence = tokenizer.texts_to_sequences([seed_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_seq_length - 1, padding='pre')


        predicted = np.argmax(model.predict(sequence), axis=-1)[0]
        output_word = ""


        for word, index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break


        seed_text += " " + output_word
    return seed_text


# Test text generation
print(generate_text("Deep learning"))
```

```
print(generate_text("Artificial intelligence"))
```