ASSIGNMENT 1

Problem Statement: Classification with Multilayer Perceptron using Scikit-learn (MNIST Dataset)

CODE:

```
from tensorflow.keras.datasets import mnist
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import time

# Load MNIST dataset
print("Loading MNIST dataset...")
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Flatten images (28x28 → 784 features)
X_train = X_train.reshape(-1, 28*28)
X_test = X_test.reshape(-1, 28*28)

# Normalize pixel values
X_train = X_train / 255.0
X_test = X_test / 255.0

# Standardize data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Define MLP Classifier
mlp = MLPClassifier(
    hidden_layer_sizes=(100,),
```

```
    activation='relu',
    solver='adam',
    batch_size=200,
    max_iter=20,
    random_state=42,
    verbose=True
)

# Tthe model (measure time)
print("\nTraining MLP classifier...")
start_time = time.time()
mlp.fit(X_train, y_train)
end_time = time.time()

training_time = end_time - start_time

# Evaluate performance
y_pred = mlp.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print("\n✅ Model Evaluation Results:")
print(f"Accuracy: {accuracy * 100:.2f}%")
print(f"Training Time: {training_time:.2f} seconds")

# Plot  Loss Curve (Graph)
plt.figure(figsize=(8, 5))
plt.plot(mlp.loss_curve_, marker='o')
plt.title("MLP Classifier Training Loss Curve")
plt.xlabel("Iterations (Epochs)")
plt.ylabel("Loss")
```

```
plt.grid(True)
plt.show()


#  Visualize some sample predictions
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    idx = np.random.randint(0, len(X_test))
    img = X_test[idx].reshape(28, 28)
    plt.imshow(img, cmap='gray')
    plt.title(f"Pred: {y_pred[idx]}\nTrue: {y_test[idx]}")
    plt.axis('off')


plt.tight_layout()
plt.show()
```

OUTPUT

```
Loading MNIST dataset...
Training MLP classifier...
Iteration 1, loss = 0.34997327
Iteration 2, loss = 0.14079123
Iteration 3, loss = 0.09575991
Iteration 4, loss = 0.07115329
Iteration 5, loss = 0.05505706
Iteration 6, loss = 0.04287497
Iteration 7, loss = 0.03379973
Iteration 8, loss = 0.02699543
Iteration 9, loss = 0.02237957
Iteration 10, loss = 0.01735595
Iteration 11, loss = 0.01418406
Iteration 12, loss = 0.01104682
```

Iteration 13, loss = 0.01042444

Iteration 14, loss = 0.00810184

Iteration 15, loss = 0.00606462

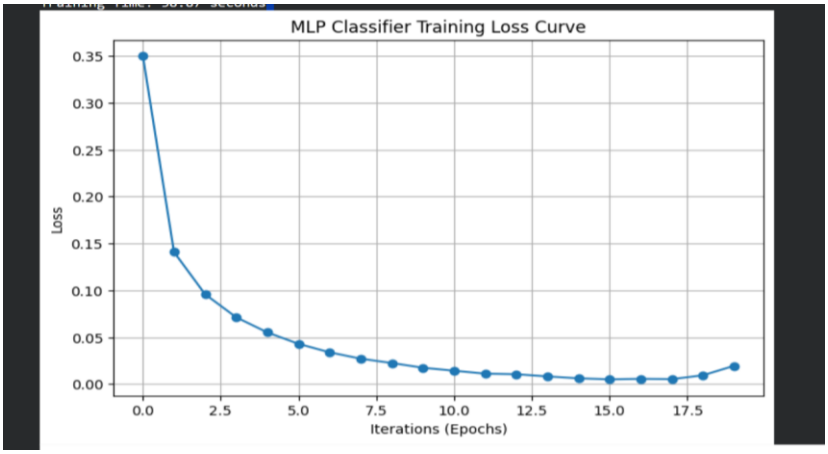
Iteration 16, loss = 0.00493687

Iteration 17, loss = 0.00547189

Iteration 18, loss = 0.00518125

Iteration 19, loss = 0.00925979

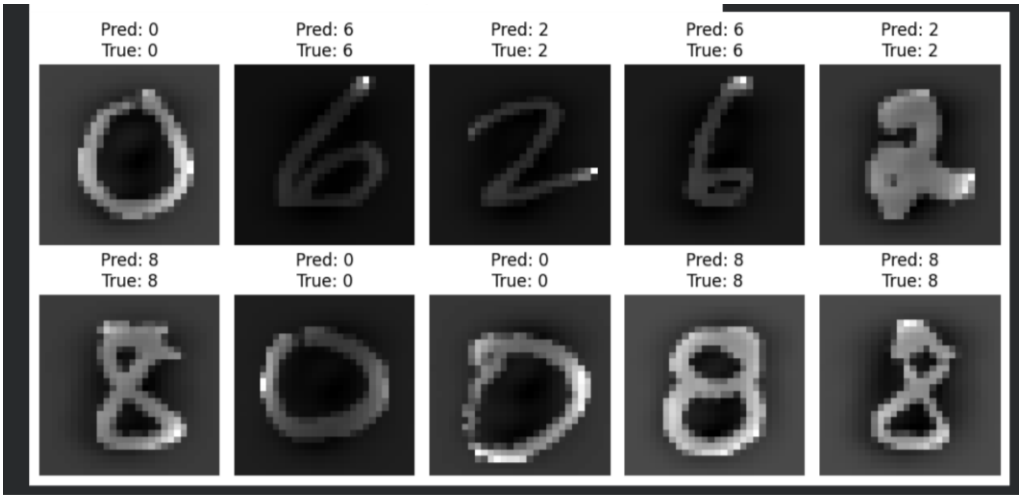
Iteration 20, loss = 0.01936909



Model Evaluation Results:

Accuracy: 97.24%

Training Time: 38.87 seconds



```
# 1. Load the dataset
import tensorflow as tf
from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
# 2. Preprocess the data
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

# Reshape data to add a channel dimension (for grayscale images)
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)

# Convert class vectors to binary class matrices
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_test = tf.keras.utils.to_categorical(y_test, num_classes=10)
```

```
# 3. Build the CNN model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

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')
])
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# 4. Compile the model
model.compile(loss=tf.keras.losses.categorical_crossentropy,
```

```
                optimizer=tf.keras.optimizers.Adam(),
                metrics=['accuracy'])
```

```
# 5. Train the model
batch_size = 128
epochs = 2

history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))
```

```
Epoch 1/2
469/469 ━━━━━━━━━━━━━━━━━━━━ 51s 109ms/step - accuracy: 0.9831 - loss: 0.0541 - val_accuracy: 0.9878 - val_loss: 0.0369
Epoch 2/2
469/469 ━━━━━━━━━━━━━━━━━━━━ 82s 109ms/step - accuracy: 0.9890 - loss: 0.0362 - val_accuracy: 0.9901 - val_loss: 0.0307
```

```
# 6. Evaluate the model
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.030664997175335884
Test accuracy: 0.9901000261306763
```

```
# 1. Load and preprocess the dataset
# Using Labeled Faces in the Wild (LFW) dataset as an example
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Load the dataset, keeping faces with at least 70 images to avoid issues with small classes
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)

# Introspect the images to find the shapes (height, width)
n_samples, h, w = lfw_people.images.shape

# the data has been preprocessed by the fetch_lfw_people function to just
# contain numpy arrays of shape (n_samples, h * w)
X = lfw_people.data
n_features = X.shape[1]

# the target names are the names of the people
y = lfw_people.target
target_names = lfw_people.target_names
n_classes = target_names.shape[0]

print("Total dataset size:")
print("n_samples: %d" % n_samples)
print("n_features: %d" % n_features)
print("n_classes: %d" % n_classes)

# Reshape the data for CNN input (add channel dimension)
X = X.reshape(n_samples, h, w, 1)
```

```
Total dataset size:
n_samples: 1288
n_features: 1850
n_classes: 7
```

```
# 2. Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

```
# 3. Build the CNN model
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

```
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(h, w, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Conv2D(64, kernel_size=(3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(n_classes, activation='softmax')
])
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `input_shape`/`input_dim` argumen
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# 4. Compile the model
model.compile(loss=tf.keras.losses.sparse_categorical_crossentropy,
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])
```

```
# 5. Train the model
batch_size = 32
epochs = 10

history = model.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(X_test, y_test))
```

```
Epoch 1/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 4s 83ms/step - accuracy: 0.3311 - loss: 1.8325 - val_accuracy: 0.4534 - val_loss: 1.6069
Epoch 2/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 3s 82ms/step - accuracy: 0.4043 - loss: 1.6985 - val_accuracy: 0.4534 - val_loss: 1.6052
Epoch 3/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 2s 77ms/step - accuracy: 0.3877 - loss: 1.7225 - val_accuracy: 0.4534 - val_loss: 1.4685
Epoch 4/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 4s 127ms/step - accuracy: 0.4477 - loss: 1.5469 - val_accuracy: 0.5528 - val_loss: 1.2713
Epoch 5/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 3s 81ms/step - accuracy: 0.4951 - loss: 1.3615 - val_accuracy: 0.5870 - val_loss: 1.1404
Epoch 6/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 2s 79ms/step - accuracy: 0.5519 - loss: 1.1993 - val_accuracy: 0.6708 - val_loss: 0.9644
Epoch 7/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 3s 83ms/step - accuracy: 0.6280 - loss: 0.9930 - val_accuracy: 0.7422 - val_loss: 0.8285
Epoch 8/10
```

```
31/31 ━━━━━━━━━━━━━━━━━━━━ 3s 84ms/step - accuracy: 0.6500 - loss: 0.9204 - val_accuracy: 0.7453 - val_loss: 0.7384
Epoch 9/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 4s 130ms/step - accuracy: 0.7391 - loss: 0.7517 - val_accuracy: 0.7981 - val_loss: 0.7016
Epoch 10/10
31/31 ━━━━━━━━━━━━━━━━━━━━ 2s 80ms/step - accuracy: 0.7499 - loss: 0.7097 - val_accuracy: 0.7671 - val_loss: 0.6726
```

```
# 6. Evaluate the model
score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.6726375818252563
Test accuracy: 0.7670807242393494
```

```
# 1. Load and preprocess the dataset
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

# Generate synthetic time series data (sine wave)
data = np.sin(np.arange(0, 100, 0.1))
data = data.reshape(-1, 1)

# Scale the data
scaler = MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(data)

# Create sequences for RNN input
def create_sequences(data, sequence_length):
    X, y = [], []
    for i in range(len(data) - sequence_length):
        X.append(data[i:(i + sequence_length), 0])
        y.append(data[i + sequence_length, 0])
    return np.array(X), np.array(y)

sequence_length = 10
X, y = create_sequences(data, sequence_length)

# Reshape X for RNN input (samples, time steps, features)
X = X.reshape(X.shape[0], X.shape[1], 1)
```

```
# 2. Split the data
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# 3. Build the RNN model
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN, Dense

model = Sequential([
    SimpleRNN(50, activation='relu', input_shape=(sequence_length, 1)),
    Dense(1)
])
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/rnn/rnn.py:199: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Wh
  super().__init__(**kwargs)
```

```
# 4. Compile the model
model.compile(optimizer='adam', loss='mse')
```

```
# 5. Train the model
epochs = 10

history = model.fit(X_train, y_train,
                    epochs=epochs,
                    verbose=1)
```

```
Epoch 1/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - loss: 0.6657
Epoch 2/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0328
Epoch 3/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0033
Epoch 4/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 3.3704e-04
Epoch 5/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.0087e-05
Epoch 6/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 9.6521e-06
Epoch 7/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 3.7057e-06
Epoch 8/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 1.0992e-06
Epoch 9/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 6.4554e-07
Epoch 10/10
25/25 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 4.2984e-07
```

```
# 6. Evaluate the model
loss = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', loss)
```

```
Test loss: 5.951576440565987e-07
```

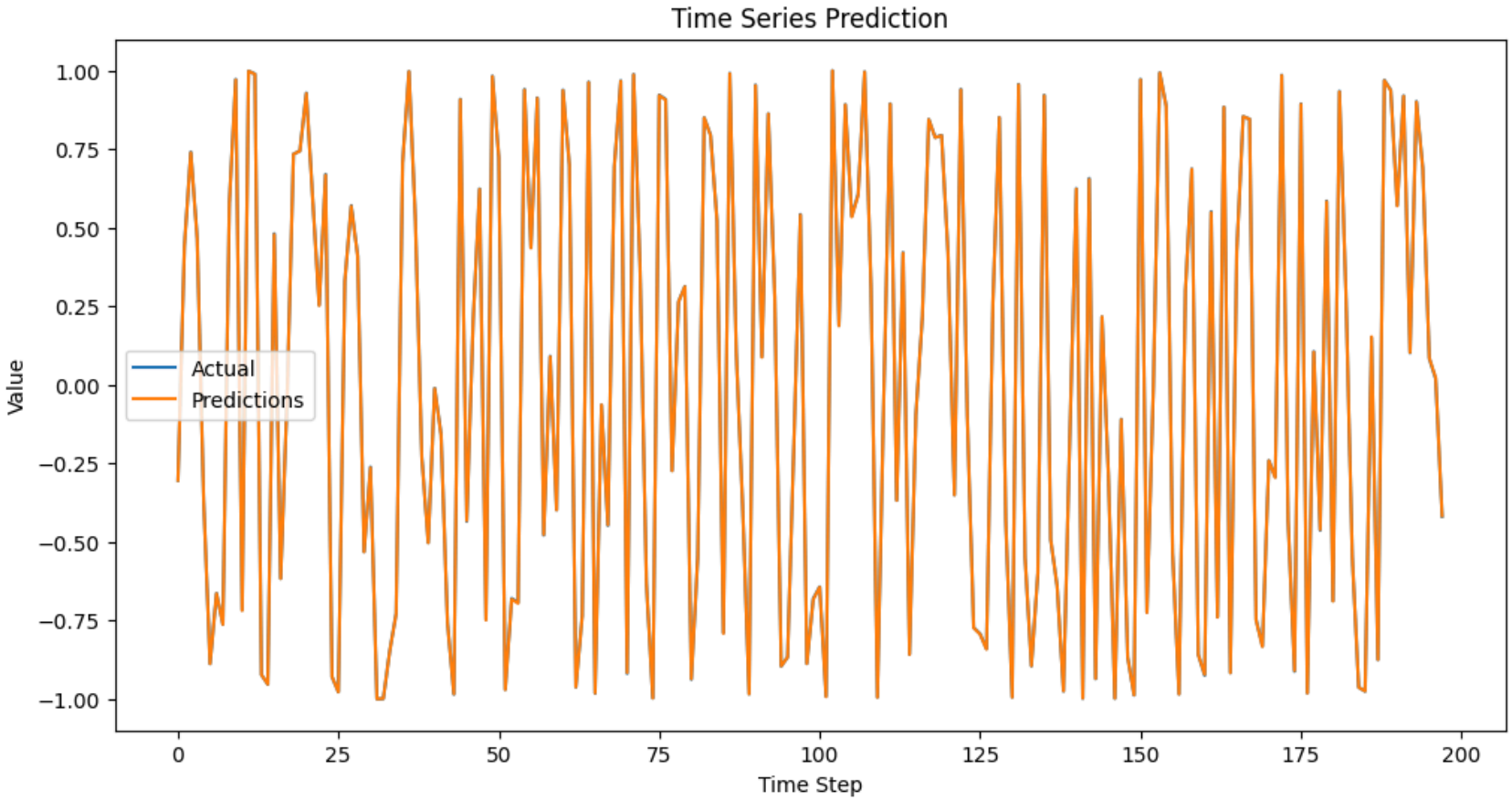
```
# 7. Make predictions
predictions = model.predict(X_test)

# Inverse transform the predictions and actual values to the original scale
predictions = scaler.inverse_transform(predictions)
```

```
predictions = scaler.inverse_transform(predictions)
y_test_original = scaler.inverse_transform(y_test.reshape(-1, 1))

# Plot the predictions vs actual values
plt.figure(figsize=(12, 6))
plt.plot(y_test_original, label='Actual')
plt.plot(predictions, label='Predictions')
plt.title('Time Series Prediction')
plt.xlabel('Time Step')
plt.ylabel('Value')
plt.legend()
plt.show()
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```



ASSIGNMENT 5

Problem Statement: To Analyze and differentiate fake and real image through GAN

CODE:

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Reshape, Flatten, LeakyReLU
from tensorflow.keras.optimizers import Adam
import time


# 1️⃣ Load CIFAR-10 dataset
(X_train, _), (_, _) = cifar10.load_data()
X_train = X_train.astype('float32')
X_train = (X_train - 127.5) / 127.5   # normalize to [-1, 1]
img_shape = (32, 32, 3)
latent_dim = 100


# 2️⃣ Build Generator
def build_generator():
    model = Sequential([
        Dense(256, input_dim=latent_dim),
        LeakyReLU(alpha=0.2),
        Dense(512),
        LeakyReLU(alpha=0.2),
        Dense(1024),
        LeakyReLU(alpha=0.2),
        Dense(np.prod(img_shape), activation='tanh'),
        Reshape(img_shape)
    ])
    return model
```

```
# 3️⃣ Build Discriminator
def build_discriminator():
    model = Sequential([
        Flatten(input_shape=img_shape),
        Dense(512),
        LeakyReLU(alpha=0.2),
        Dense(256),
        LeakyReLU(alpha=0.2),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5), metrics=['accuracy'])
    return model

# Initialize models
generator = build_generator()
discriminator = build_discriminator()
discriminator.trainable = False

# Combine both models into a GAN
gan = Sequential([generator, discriminator])
gan.compile(loss='binary_crossentropy', optimizer=Adam(0.0002, 0.5))

# 4️⃣ Training
epochs = 20  # only 20 to match example
batch_size = 64
half_batch = batch_size // 2

print("Training GAN...")
start_time = time.time()

for epoch in range(epochs):
    # Select random real images
```

```
    idx = np.random.randint(0, X_train.shape[0], half_batch)
    real_imgs = X_train[idx]

    # Generate fake images
    noise = np.random.normal(0, 1, (half_batch, latent_dim))
    fake_imgs = generator.predict(noise)

    # Labels
    real_y = np.ones((half_batch, 1))
    fake_y = np.zeros((half_batch, 1))

    # Train discriminator
    d_loss_real = discriminator.train_on_batch(real_imgs, real_y)
    d_loss_fake = discriminator.train_on_batch(fake_imgs, fake_y)

    real_acc = 100 * d_loss_real[1]
    fake_acc = 100 * d_loss_fake[1]

    # Train generator (wants discriminator to think fake images are real)
    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    valid_y = np.ones((batch_size, 1))
    g_loss = gan.train_on_batch(noise, valid_y)

    # Log like example format
    print(f">{epoch+1} real={real_acc:.0f}% fake={fake_acc:.0f}%")

end_time = time.time()
print(f"\n✅ Training finished in {end_time - start_time:.2f} seconds")

# 5️⃣ Visualize some fake images
noise = np.random.normal(0, 1, (25, latent_dim))
gen_imgs = generator.predict(noise)
```

```
gen_imgs = 0.5 * gen_imgs + 0.5  # Rescale to [0,1]


plt.figure(figsize=(5,5))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.imshow(gen_imgs[i])
    plt.axis('off')
plt.suptitle("Generated CIFAR-10 Images (Fake)")
plt.show()
```

OUTPUT:

Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step

Training GAN...

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step

>1 real=6% fake=50%

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 70ms/step

>2 real=35% fake=27%

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step

>3 real=24% fake=20%

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step

>4 real=18% fake=16%

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step

>5 real=15% fake=14%

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 65ms/step

>6 real=14% fake=13%

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 64ms/step

>7 real=13% fake=12%

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step

>8 real=12% fake=11%

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step

>9 real=11% fake=10%

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step

>10 real=10% fake=9%

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step

>11 real=9% fake=9%

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step

>12 real=9% fake=8%

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step

>13 real=9% fake=8%

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step

>14 real=8% fake=8%

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 169ms/step

>15 real=8% fake=8%

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step

>16 real=8% fake=8%

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step

>17 real=8% fake=8%

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step

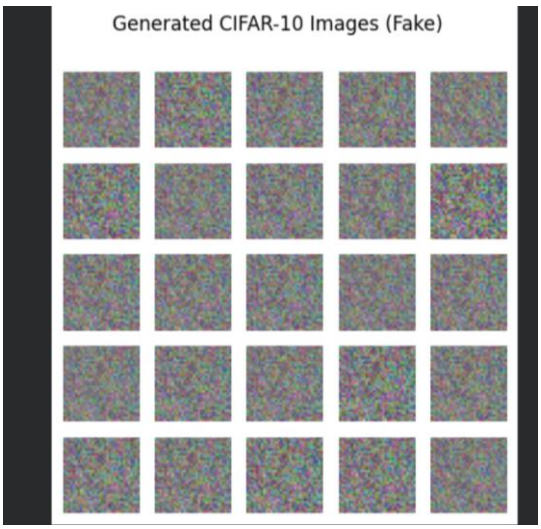
>18 real=8% fake=7%

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 126ms/step

>19 real=8% fake=7%

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step

>20 real=8% fake=8%



✓ **Training finished in 7.61 seconds**

1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 117ms/step

ASSIGNMENT 6

PROBLEM STATEMENT: Sentiment Analysis using LSTM and GloVe Embeddings

CODE:

```
# Sentiment Analysis using LSTM and Keras Embedding (No GloVe)


import re
import numpy as np
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout


# 1️⃣ Load IMDB dataset (pre-tokenized by Keras)
max_words = 10000  # Vocabulary size
max_len = 200      # Maximum review length
print("Loading IMDB dataset...")
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=max_words)


# 2️⃣ Pad sequences to ensure uniform input length
X_train = pad_sequences(X_train, maxlen=max_len)
X_test = pad_sequences(X_test, maxlen=max_len)


# 3️⃣ Custom stopwords list and text cleaning (for any manual text cleaning)
stopwords = ["a","about","above","after","again","against","all","am","an","and","any","are",
        "as","at","be","because","been","before","being","below","between","both","but",
        "by","could","did","do","does","doing","down","during","each","few","for","from",
        "further","had","has","have","having","he","he'd","he'll","he's","her","here",
        "here's","hers","herself","him","himself","his","how","how's","i","i'd","i'll",
        "i'm","i've","if","in","into","is","it","it's","its","itself","let's","me","more",
        "most","my","myself","nor","of","on","once","only","or","other","ought","our",
```

```
        "ours","ourselves","out","over","own","same","she","she'd","she'll","she's",
        "should","so","some","such","than","that","that's","the","their","theirs","them",
        "themselves","then","there","there's","these","they","they'd","they'll","they're",
        "they've","this","those","through","to","too","under","until","up","very","was",
        "we","we'd","we'll","we're","we've","were","what","what's","when","when's","where",
        "where's","which","while","who","who's","whom","why","why's","with","would","you",
        "you'd","you'll","you're","you've","your","yours","yourself","yourselves"]

def clean_text(text):
    text = re.sub(r'<.*?>', '', text)  # Remove HTML tags
    text = re.sub(r'[^a-zA-Z\s]', '', text.lower())  # Remove special characters
    text = ' '.join([word for word in text.split() if word not in stopwords])  # Remove stopwords
    return text

# 4 Build the LSTM model
model = Sequential([
    Embedding(input_dim=max_words, output_dim=128, input_length=max_len),
    LSTM(128, dropout=0.2, recurrent_dropout=0.2),
    Dense(1, activation='sigmoid')
])

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

# 5 Train the model
print("\nTraining model...")
history = model.fit(X_train, y_train,
                    epochs=3,
                    batch_size=128,
```

```
            validation_data=(X_test, y_test),
            verbose=1)

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

# 7 Plot accuracy/loss curves
import matplotlib.pyplot as plt

plt.figure(figsize=(10,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train')
plt.plot(history.history['val_accuracy'], label='Validation')
plt.title('Model Accuracy')
plt.legend()

plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train')
plt.plot(history.history['val_loss'], label='Validation')
plt.title('Model Loss')
plt.legend()

plt.show()
```

OUTPUT:

```
Loading IMDB dataset...
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step

Training model...
Epoch 1/3
```

/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/embedding.py:97: UserWarning: Argument `input_length` is deprecated. Just remove it.

warnings.warn(

196/196 ━━━━━━━━━━━━━━━━━━━━ 309s 2s/step - accuracy: 0.6866 - loss: 0.5711 - val_accuracy: 0.8436 - val_loss: 0.3619

Epoch 2/3

196/196 ━━━━━━━━━━━━━━━━━━━━ 327s 2s/step - accuracy: 0.8694 - loss: 0.3276 - val_accuracy: 0.8288 - val_loss: 0.3839

Epoch 3/3

196/196 ━━━━━━━━━━━━━━━━━━━━ 299s 2s/step - accuracy: 0.8915 - loss: 0.2761 - val_accuracy: 0.8604 - val_loss: 0.3488

782/782 ━━━━━━━━━━━━━━━━━━━━ 65s 84ms/step - accuracy: 0.8600 - loss: 0.3512