**University of Central Missouri**

**Department of Computer Science & Cybersecurity**

**CS5720 Neural network and Deep learning**

**Spring 2025**

**Home Assignment 3. (Cover Ch 7, 8)**

**Student name: YASWANTH POLAROUTHU**

**Submission Requirements:**

* Total Points: 100
* Once finished your assignment push your source code to your repo (GitHub) and explain the work through the ReadMe file properly. Make sure you add your student info in the ReadMe file.
* Submit your GitHub link and video on the BB.
* Comment your code appropriately ***IMPORTANT.***
* Make a simple video about 2 to 3 minutes which includes demonstration of your home assignment and explanation of code snippets.
* Any submission after provided deadline is considered as a late submission.

**Q1: Implementing a Basic Autoencoder**

**Task:** Autoencoders learn to reconstruct input data by encoding it into a lower-dimensional space. You will build a **fully connected autoencoder** and evaluate its performance on image reconstruction.

1. Load the **MNIST dataset** using tensorflow.keras.datasets.
2. Define a **fully connected (Dense) autoencoder**:
   * Encoder: Input layer (784), hidden layer (32).
   * Decoder: Hidden layer (32), output layer (784).
3. Compile and train the autoencoder with **binary cross-entropy loss**.
4. Plot **original vs. reconstructed images** after training.
5. Modify the latent dimension size (e.g., 16, 64) and analyze how it affects the quality of reconstruction.

***Hint:*** *Use Model() from tensorflow.keras.models and Dense() layers.*

*Ans:*

*Code:*

*import tensorflow as tf*

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

*from tensorflow.keras.models import Model*

*import numpy as np*

*import matplotlib.pyplot as plt*

*# Load MNIST dataset*

*(x\_train, \_), (x\_test, \_) = tf.keras.datasets.mnist.load\_data()*

*x\_train = x\_train.astype('float32') / 255.*

*x\_test = x\_test.astype('float32') / 255.*

*x\_train = x\_train.reshape((len(x\_train), 784))*

*x\_test = x\_test.reshape((len(x\_test), 784))*

*# Define Autoencoder*

*latent\_dim = 32 # Modify this for different latent sizes*

*# Encoder*

*input\_img = Input(shape=(784,))*

*encoded = Dense(latent\_dim, activation='relu')(input\_img)*

*# Decoder*

*decoded = Dense(784, activation='sigmoid')(encoded)*

*# Autoencoder Model*

*autoencoder = Model(input\_img, decoded)*

*autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')*

*# Train the Autoencoder*

*autoencoder.fit(x\_train, x\_train, epochs=10, batch\_size=256, shuffle=True, validation\_data=(x\_test, x\_test))*

*# Encode and Decode some images*

*encoded\_imgs = autoencoder.predict(x\_test)*

*# Plot Original vs Reconstructed Images*

*def plot\_images(original, reconstructed, n=10):*

*plt.figure(figsize=(20, 4))*

*for i in range(n):*

*# Original*

*ax = plt.subplot(2, n, i + 1)*

*plt.imshow(original[i].reshape(28, 28), cmap='gray')*

*plt.axis('off')*

*# Reconstructed*

*ax = plt.subplot(2, n, i + 1 + n)*

*plt.imshow(reconstructed[i].reshape(28, 28), cmap='gray')*

*plt.axis('off')*

*plt.show()*

*plot\_images(x\_test, encoded\_imgs)*

*Output:*

A black square with white numbers

AI-generated content may be incorrect.

**Q2: Implementing a Denoising Autoencoder**

**Task:** Denoising autoencoders can reconstruct clean data from noisy inputs. You will train a model to remove noise from images.

1. Modify the **basic autoencoder** from Q2 to a **denoising autoencoder** by adding **Gaussian noise** (mean=0, std=0.5) to input images.
2. Ensure that the **output remains the clean image** while training.
3. Train the model and visualize **noisy vs. reconstructed images**.
4. Compare the **performance of a basic vs. denoising autoencoder** in reconstructing images.
5. Explain one real-world scenario where denoising autoencoders can be useful (e.g., medical imaging, security).

***Hint:*** *Use np.random.normal() to add noise to images before training.*

Ans:

Code:

import tensorflow as tf

from tensorflow.keras.layers import Input, Dense

from tensorflow.keras.models import Model

import numpy as np

import matplotlib.pyplot as plt

# Load MNIST dataset

(x\_train, \_), (x\_test, \_) = tf.keras.datasets.mnist.load\_data()

x\_train = x\_train.astype('float32') / 255.

x\_test = x\_test.astype('float32') / 255.

x\_train = x\_train.reshape((len(x\_train), 784))

x\_test = x\_test.reshape((len(x\_test), 784))

# Add Gaussian noise

noise\_factor = 0.5

x\_train\_noisy = x\_train + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=x\_train.shape)

x\_test\_noisy = x\_test + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=x\_test.shape)

# Clip values to stay within valid range

x\_train\_noisy = np.clip(x\_train\_noisy, 0., 1.)

x\_test\_noisy = np.clip(x\_test\_noisy, 0., 1.)

# Define Denoising Autoencoder

latent\_dim = 32

# Encoder

input\_img = Input(shape=(784,))

encoded = Dense(latent\_dim, activation='relu')(input\_img)

# Decoder

decoded = Dense(784, activation='sigmoid')(encoded)

denoising\_autoencoder = Model(input\_img, decoded)

denoising\_autoencoder.compile(optimizer='adam', loss='binary\_crossentropy')

# Train the autoencoder

denoising\_autoencoder.fit(x\_train\_noisy, x\_train, epochs=10, batch\_size=256, shuffle=True, validation\_data=(x\_test\_noisy, x\_test))

# Denoise test images

denosed\_imgs = denoising\_autoencoder.predict(x\_test\_noisy)

# Plot Noisy vs Reconstructed Images

def plot\_images(original, noisy, reconstructed, n=10):

plt.figure(figsize=(20, 6))

for i in range(n):

# Noisy Image

ax = plt.subplot(3, n, i + 1)

plt.imshow(noisy[i].reshape(28, 28), cmap='gray')

plt.axis('off')

# Original Image

ax = plt.subplot(3, n, i + 1 + n)

plt.imshow(original[i].reshape(28, 28), cmap='gray')

plt.axis('off')

# Reconstructed Image

ax = plt.subplot(3, n, i + 1 + 2 \* n)

plt.imshow(reconstructed[i].reshape(28, 28), cmap='gray')

plt.axis('off')

plt.show()

plot\_images(x\_test, x\_test\_noisy, denosed\_imgs)

Output:

A number in a square

AI-generated content may be incorrect.

**Q3: Implementing an RNN for Text Generation**

**Task:** Recurrent Neural Networks (RNNs) can generate sequences of text. You will train an **LSTM-based RNN** to predict the next character in a given text dataset.

1. Load a **text dataset** (e.g., "Shakespeare Sonnets", "The Little Prince").
2. Convert text into a **sequence of characters** (one-hot encoding or embeddings).
3. Define an **RNN model** using LSTM layers to predict the next character.
4. Train the model and generate new text by **sampling characters** one at a time.
5. Explain the role of **temperature scaling** in text generation and its effect on randomness.

***Hint:*** *Use tensorflow.keras.layers.LSTM() for sequence modeling.*

Ans:

Code:

import tensorflow as tf

import numpy as np

import random

import sys

# Load a small sample text (can be replaced with full dataset)

path = tf.keras.utils.get\_file("shakespeare.txt", "https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt")

text = open(path, encoding='utf-8').read().lower()

print('Corpus length:', len(text))

# Create character-to-index mappings

chars = sorted(list(set(text)))

char\_indices = dict((c, i) for i, c in enumerate(chars))

indices\_char = dict((i, c) for i, c in enumerate(chars))

# Create input sequences and next character labels

maxlen = 40

step = 3

sentences = []

next\_chars = []

for i in range(0, len(text) - maxlen, step):

sentences.append(text[i: i + maxlen])

next\_chars.append(text[i + maxlen])

print('Number of sequences:', len(sentences))

# Vectorize input and output

x = np.zeros((len(sentences), maxlen, len(chars)), dtype=bool)

y = np.zeros((len(sentences), len(chars)), dtype=bool)

for i, sentence in enumerate(sentences):

for t, char in enumerate(sentence):

x[i, t, char\_indices[char]] = 1

y[i, char\_indices[next\_chars[i]]] = 1

# Define the LSTM model

model = tf.keras.models.Sequential([

tf.keras.layers.LSTM(128, input\_shape=(maxlen, len(chars))),

tf.keras.layers.Dense(len(chars), activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy')

# Train the model

model.fit(x, y, batch\_size=128, epochs=5)

# Sampling function with temperature

def sample(preds, temperature=1.0):

preds = np.asarray(preds).astype('float64')

preds = np.log(preds + 1e-8) / temperature

exp\_preds = np.exp(preds)

preds = exp\_preds / np.sum(exp\_preds)

probas = np.random.multinomial(1, preds, 1)

return np.argmax(probas)

# Generate text

def generate\_text(seed, length=400, temperature=1.0):

generated = ''

sentence = seed.lower()[:maxlen]

generated += sentence

for \_ in range(length):

x\_pred = np.zeros((1, maxlen, len(chars)))

for t, char in enumerate(sentence):

if char in char\_indices:

x\_pred[0, t, char\_indices[char]] = 1

preds = model.predict(x\_pred, verbose=0)[0]

next\_index = sample(preds, temperature)

next\_char = indices\_char[next\_index]

generated += next\_char

sentence = sentence[1:] + next\_char

return generated

print("\nGenerated text with temperature=0.5:\n")

print(generate\_text("shall i compare thee to a summer's day? ", temperature=0.5))

Output:

A screenshot of a computer

AI-generated content may be incorrect.

**Q4: Sentiment Classification Using RNN**

**Task:** Sentiment analysis determines if a given text expresses a positive or negative emotion. You will train an **LSTM-based sentiment classifier** using the IMDB dataset.

1. Load the **IMDB sentiment dataset** (tensorflow.keras.datasets.imdb).
2. Preprocess the text data by **tokenization** and **padding** sequences.
3. Train an **LSTM-based model** to classify reviews as **positive or negative**.
4. Generate a **confusion matrix** and classification report (accuracy, precision, recall, F1-score).
5. Interpret why **precision-recall tradeoff** is important in sentiment classification.

***Hint:*** *Use confusion\_matrix and classification\_report from sklearn.metrics.*

Ans:

Code:

import tensorflow as tf

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from sklearn.metrics import confusion\_matrix, classification\_report

import numpy as np

# Load the IMDB dataset

vocab\_size = 10000 # Keep top 10,000 words

maxlen = 200

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.imdb.load\_data(num\_words=vocab\_size)

# Pad sequences to make them equal length

x\_train = pad\_sequences(x\_train, maxlen=maxlen)

x\_test = pad\_sequences(x\_test, maxlen=maxlen)

# Define LSTM model

model = tf.keras.models.Sequential([

tf.keras.layers.Embedding(vocab\_size, 64, input\_length=maxlen),

tf.keras.layers.LSTM(64),

tf.keras.layers.Dense(1, activation='sigmoid')

])

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model

model.fit(x\_train, y\_train, epochs=3, batch\_size=128, validation\_split=0.2)

# Evaluate and predict

y\_pred\_probs = model.predict(x\_test)

y\_pred = (y\_pred\_probs > 0.5).astype(int).flatten()

# Generate confusion matrix and classification report

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

Output:

A screenshot of a computer

AI-generated content may be incorrect.