**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:**

**Bhargav Sai Allam 700752883**

**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.*

Output:

Downloading data from [https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz](https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz" \t "_blank)

**11490434/11490434** ━━━━━━━━━━━━━━━━━━━━ **0s** 0us/step

Epoch 1/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **4s** 10ms/step - loss: 0.3813 - val\_loss: 0.1924

Epoch 2/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 4ms/step - loss: 0.1822 - val\_loss: 0.1541

Epoch 3/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 4ms/step - loss: 0.1495 - val\_loss: 0.1331

Epoch 4/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 4ms/step - loss: 0.1310 - val\_loss: 0.1210

Epoch 5/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 4ms/step - loss: 0.1201 - val\_loss: 0.1130

Epoch 6/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 5ms/step - loss: 0.1127 - val\_loss: 0.1073

Epoch 7/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 5ms/step - loss: 0.1074 - val\_loss: 0.1031

Epoch 8/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 4ms/step - loss: 0.1035 - val\_loss: 0.1000

Epoch 9/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 3ms/step - loss: 0.1006 - val\_loss: 0.0977

Epoch 10/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 3ms/step - loss: 0.0987 - val\_loss: 0.0959

**313/313** ━━━━━━━━━━━━━━━━━━━━ **1s** 2ms/step

A black square with white text

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).

Answer:

One real-world scenario where denoising autoencoders are highly useful is **medical imaging**. Medical scans such as X-rays, MRIs, and CT scans often contain noise due to hardware limitations, low radiation doses, or environmental factors. Denoising autoencoders can help enhance these images by removing noise while preserving critical details, leading to better diagnosis and treatment planning.

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

Output :

Epoch 1/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **3s** 6ms/step - loss: 0.3725 - val\_loss: 0.2252

Epoch 2/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 3ms/step - loss: 0.2121 - val\_loss: 0.1824

Epoch 3/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 3ms/step - loss: 0.1785 - val\_loss: 0.1638

Epoch 4/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 3ms/step - loss: 0.1622 - val\_loss: 0.1521

Epoch 5/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 3ms/step - loss: 0.1513 - val\_loss: 0.1443

Epoch 6/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 3ms/step - loss: 0.1439 - val\_loss: 0.1388

Epoch 7/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 3ms/step - loss: 0.1389 - val\_loss: 0.1350

Epoch 8/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 3ms/step - loss: 0.1354 - val\_loss: 0.1326

Epoch 9/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 4ms/step - loss: 0.1333 - val\_loss: 0.1312

Epoch 10/10

**235/235** ━━━━━━━━━━━━━━━━━━━━ **1s** 4ms/step - loss: 0.1318 - val\_loss: 0.1302

**313/313** ━━━━━━━━━━━━━━━━━━━━ **1s** 2ms/step

A collage of numbers

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.

Answer :

This implementation trains an LSTM-based RNN to generate text character-by-character. It also demonstrates how temperature scaling affects randomness in text generation:

* **Low temperature (0.2):** More deterministic, repeats common patterns.
* **Medium temperature (1.0):** Balanced creativity and coherence.
* **High temperature (1.5):** More random, sometimes nonsensical.

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

Output:

Epoch 1/10

**2905/2905** ━━━━━━━━━━━━━━━━━━━━ **27s** 8ms/step - loss: 2.5426

Epoch 2/10

**2905/2905** ━━━━━━━━━━━━━━━━━━━━ **24s** 8ms/step - loss: 1.8427

Epoch 3/10

**2905/2905** ━━━━━━━━━━━━━━━━━━━━ **25s** 8ms/step - loss: 1.6638

Epoch 4/10

**2905/2905** ━━━━━━━━━━━━━━━━━━━━ **24s** 8ms/step - loss: 1.5691

Epoch 5/10

**2905/2905** ━━━━━━━━━━━━━━━━━━━━ **24s** 8ms/step - loss: 1.4993

Epoch 6/10

**2905/2905** ━━━━━━━━━━━━━━━━━━━━ **25s** 9ms/step - loss: 1.4526

Epoch 7/10

**2905/2905** ━━━━━━━━━━━━━━━━━━━━ **41s** 8ms/step - loss: 1.4104

Epoch 8/10

**2905/2905** ━━━━━━━━━━━━━━━━━━━━ **41s** 8ms/step - loss: 1.3829

Epoch 9/10

**2905/2905** ━━━━━━━━━━━━━━━━━━━━ **41s** 8ms/step - loss: 1.3526

Epoch 10/10

**2905/2905** ━━━━━━━━━━━━━━━━━━━━ **41s** 9ms/step - loss: 1.3341

Generated Text at Temperature 0.2:

shall i compare thee to a summer's daynaeese

ioi ilaosasa ioi io io io

io ieao wt taei io io io io.ya,yt

aiei auaaalri aela'

irioo' io io io io io io io

io ieaall ieiaileeeeeeeeeeso.a

hsasm ieao io io io io aiei aiei

aoea ioi ilaosi aor,ieaelsyi.la,yllese,a,ylsls ieooo'a,yt,i aea

aoa ieio ie' ioi ilaoo,i aeasyi tasa

aoalylne ilaosa' io io io io.ya,ylse,aeael,i aeasyi tasa

auli ieaaileeeeeeeeeeeeeeeeeso

ieio iei aoea ioilee.a,yt,i

tasa

Generated Text at Temperature 1.0:

shall i compare thee to a summer's dayescctm.y io ta?

aoi ieaoh' idauor uoaoei.rlr elsyo.aaua';yt?

svisy,o,ai' io io aula aidlti

tisa'lnnmslsa igstlcfa' ieauiyrbna.ael,i aaamltcsi.la iu

oscaie' aoellseaur,,wth iooroeael

imtmilyi,rba;elhi,aetrael;a

aiela,a iio ioi inmli mlsftnnsa':aio,a

hra!i sltm-aor;i tasy-nlrldelee;o-iontasr;tsee ao

wtbeee wco;i,y io iei io io?oyael ieauri,si,o,i

aona iulo iul ieiro ioberlaipela;iuila.yi:eo au;rt'

a

Generated Text at Temperature 1.5:

shall i compare thee to a summer's dayxe.ei!eimi-ie.aipe yru iusese

uonri!re.keoireyi : ii aeew?

tamlo,y imilsi ia,y',mi,fh yra,aaary,o

iraselrnta;,o,m' iraow bala trar.i-

jgt'l' iu isa;a,ra

aaafee,rlls

eeisi alrtttni, np-asi o.lla,ah.iairlr;th oeioeihtesa'.ieo?aoeaee' unsil,i,trtcelra ccvsre pi

?sa ioasi smlr ieyaseo,a''-a ntra:yi mgte.osis bo

aai.yyoseeellri?ywt!oieoaaall,

wne ioyahe?a?mla ioweyi ty?la,a,la g-mna aei:aseism'

mrkbo',

**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.*

import tensorflow as tf

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

from sklearn.model\_selection import train\_test\_split

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

import numpy as np

# 1. Load the IMDB sentiment dataset

num\_words = 10000 # Consider only the top 10,000 most frequent words

maxlen = 200 # Maximum sequence length

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=num\_words)

# 2. Preprocess the text data

# Pad sequences to ensure uniform length

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

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

# 3. Train an LSTM-based model

embedding\_dim = 128

lstm\_units = 128

model = Sequential([

Embedding(num\_words, embedding\_dim, input\_length=maxlen),

LSTM(lstm\_units),

Dense(1, activation='sigmoid') # Output layer with sigmoid for binary classification

])

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

epochs = 5

batch\_size = 128

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

# 4. Generate confusion matrix and classification report

# Make predictions on the test set

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

y\_pred = np.round(y\_pred\_probs).astype(int)

# Generate confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:")

print(cm)

# Generate classification report

cr = classification\_report(y\_test, y\_pred)

print("\nClassification Report:")

print(cr)

# 5. Interpret why precision-recall tradeoff is important in sentiment classification.

print("\nInterpretation of Precision-Recall Tradeoff in Sentiment Classification:")

print("""

In sentiment classification, the precision-recall tradeoff highlights the balance

between the accuracy of positive predictions (precision) and the ability to

identify all actual positive instances (recall).

Consider a scenario where we want to identify positive reviews for a product.

High Precision: If our model has high precision, it means that when it predicts

a review as positive, it is very likely to be actually positive. This is important

to avoid falsely promoting negative reviews. However, to achieve high precision,

the model might be very conservative and miss some actual positive reviews

(resulting in lower recall).

High Recall: If our model has high recall, it means that it identifies a large

proportion of all the actual positive reviews. This is important to ensure that

we don't miss many positive opinions. However, to achieve high recall, the model

might be more liberal in its positive predictions, leading to some negative

reviews being incorrectly classified as positive (resulting in lower precision).

The importance of the tradeoff depends on the specific application:

- For applications where falsely identifying a negative sentiment as positive

has significant consequences (e.g., flagging potentially harmful content),

high precision might be prioritized.

- For applications where missing positive sentiment is more costly (e.g.,

identifying enthusiastic customers), high recall might be prioritized.

The F1-score, which is the harmonic mean of precision and recall, provides a

single metric to balance both aspects. Choosing the right balance depends on the

specific goals and costs associated with false positives and false negatives in

the sentiment classification task.

""")

Output :

Downloading data from [https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz](https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz" \t "_blank)

**17464789/17464789** ━━━━━━━━━━━━━━━━━━━━ **0s** 0us/step

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input\_length` is deprecated. Just remove it.

warnings.warn(

Epoch 1/5

**157/157** ━━━━━━━━━━━━━━━━━━━━ **13s** 36ms/step - accuracy: 0.6319 - loss: 0.6165 - val\_accuracy: 0.8472 - val\_loss: 0.3554

Epoch 2/5

**157/157** ━━━━━━━━━━━━━━━━━━━━ **5s** 28ms/step - accuracy: 0.8882 - loss: 0.2845 - val\_accuracy: 0.8652 - val\_loss: 0.3435

Epoch 3/5

**157/157** ━━━━━━━━━━━━━━━━━━━━ **7s** 39ms/step - accuracy: 0.9247 - loss: 0.2047 - val\_accuracy: 0.8660 - val\_loss: 0.3623

Epoch 4/5

**157/157** ━━━━━━━━━━━━━━━━━━━━ **7s** 21ms/step - accuracy: 0.9401 - loss: 0.1619 - val\_accuracy: 0.8688 - val\_loss: 0.3841

Epoch 5/5

**157/157** ━━━━━━━━━━━━━━━━━━━━ **3s** 18ms/step - accuracy: 0.9617 - loss: 0.1108 - val\_accuracy: 0.8620 - val\_loss: 0.3977

**782/782** ━━━━━━━━━━━━━━━━━━━━ **4s** 5ms/step

Confusion Matrix:

[[10430 2070]

[ 1438 11062]]

Classification Report:

precision recall f1-score support

0 0.88 0.83 0.86 12500

1 0.84 0.88 0.86 12500

accuracy 0.86 25000

macro avg 0.86 0.86 0.86 25000

weighted avg 0.86 0.86 0.86 25000