Final Exam Code

README

Run each code file separate in google Colab, use the tiny Shakespeare input (add to folder list).
run both should take less than 10 minutes, alter the epochs for increase accuracy at processing and time detriment.
output should be provided and also downloaded to your pc
Thank You
-Matt Foreman

FinalSimpleRNN.ipynb

```
# Import needed libraries
import tensorflow as tf  # TensorFlow and Keras handle model building/training
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.utils import to_categorical, pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
import matplotlib.pyplot as plt  # Matplotlib is used to plot the graph

# Load / normalize dataset
with open("input.txt", "r", encoding="utf-8") as f:  # Reads text from file and converts to lowercase
    text = f.read().lower()

# Tokenize (word-level)
tokeniZZR = Tokenizer()
tokeniZZR.fit_on_texts([text])  # Converts text into integers, Keras Tokenizer gives a unique index
maxWords = len(tokeniZZR.word_index) + 1  # +1 to account for padding token index=0

# Create n-gram sequence
ninput = []
for line in text.split('\n'):  # Each line is split into incremental n-gram sequences (the, king, shall)
    listOfTokens = tokeniZZR.texts_to_sequences([line])[0]
    for i in range(2, len(listOfTokens)):
        nGramSequences = listOfTokens[:i+1]  # Creates n-gram sequences
        ninput.append(nGramSequences)  # Appends to list of sequences

# Pad sequences and separate labeled
lengthOfSequences = max([len(seq) for seq in ninput])  # All sequences are padded to the same length
```

```
ninput = pad_sequences(ninput, maxlen=lengthOfSequences, padding='pre')
ninput = ninput[:50000]  # optional limit to control memory use (Google Colab can't handle it all)

x, labeled = ninput[:, :-1], ninput[:, -1]
y = to_categorical(labeled, num_classes=maxWords)  # one-hot encode target labels

# Build the model using Keras Sequential API
# Dense layer outputs a probability distribution over the vocabulary for the next word
model = Sequential()
model.add(Embedding(input_dim=maxWords, output_dim=64))  # input_dim = vocab size
model.add(SimpleRNN(32))  # 32 units; returns final output only
model.add(Dense(maxWords, activation='softmax'))  # softmax to predict next word probabilities

# Compile the model
# Loss categorical_crossentropy for multi-class classification problem which is the case here
# Optimizer: Adam adaptive learning rate this is a good default for many problems
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Train the model for some number of epochs
# x = input sequences, y = target next words
prevData = model.fit(x, y, epochs=4, verbose=1)

# Plot and save training loss curve     # Loss curve is a plot of the loss function value over
epochs
plt.plot(prevData.history['loss'])
plt.title('Training Loss Curve')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.grid(True)
plt.savefig("lossCurve.png")  # saves to file
plt.show()

# Generate new text, greedy decoding argmax
originText = "thy name is"  # Starts with a phrase and generates one word at a time
followingStatements = 10

for _ in range(followingStatements):  # Generates number of words
    listOfTokens = tokeniZZR.texts_to_sequences([originText])[0]  # Tokenize seed text
    listOfTokens = pad_sequences([listOfTokens], maxlen=lengthOfSequences-1, padding='pre')
# Pad sequence to match model input
    anticipatiedWords = model.predict(listOfTokens, verbose=0)  # Predict next word
    predictWords = tf.argmax(anticipatiedWords[0]).numpy()  # Get index of highest probability
word
    outputWord = tokeniZZR.index_word.get(predictWords, "")  # Get word from index
```

```
    originText += " " + outputWord  # Append predicted word to seed text

print("\nGeneratedSequence:")  # Print generated text
print(originText)

# Save generated text to file
# This allows download later
with open("dataOutput.txt", "w") as f:
    f.write("SimpleRNN Output:\n" + originText + "\n")

# These lines initiate download in the Colab UI
from google.colab import files
files.download("lossCurve.png")  # Downloads the loss curve image
files.download("dataOutput.txt") # Downloads the generated text
```

.....

FinalGRUTransform(extraCredit).ipynb

```
# install keras-nlp (only for colab)
!pip install keras-nlp --quiet  # gives us transformer encoder later (KerasNLP Docs)
# import libraries
import tensorflow as tf  # tensorflow core (TensorFlow Docs)
from tensorflow.keras.preprocessing.text import Tokenizer  # tokenizer maps text to ints (Keras Tokenizer API)
from tensorflow.keras.utils import to_categorical, pad_sequences  # for one-hot and input shape (Keras Utils API)
from tensorflow.keras.models import Sequential  # base model style (Keras Sequential API)
from tensorflow.keras.layers import Embedding, GRU, Dense, Input  # main layers (Keras Layer Docs)
from tensorflow.keras.layers import GlobalAveragePooling1D  # needed for transformer shape flatten (Keras Layer Docs)
import keras_nlp  # extra library for transformer (KerasNLP)
from keras_nlp.layers import TransformerEncoder  # pretrained encoder block (KerasNLP Encoder API)
import matplotlib.pyplot as plt  # plot loss curve (Matplotlib)
import numpy as np  # numpy just in case (NumPy)

# load input text
with open("input.txt", "r", encoding="utf-8") as f:
    rawTXT = f.read().lower()  # read and normalize lowercase (Python I/O)
```

```
# tokenize words
tokeniZZR = Tokenizer()
tokeniZZR.fit_on_texts([rawTXT])  # build vocab from words (Keras Tokenizer API)
maxWords = len(tokeniZZR.word_index) + 1  # total vocab size (plus 0 pad token) (Keras Docs)

# create n-gram sequences
ninput = []
for linex in rawTXT.split('\n'):  # split into lines (Basic Python string ops)
    tokList = tokeniZZR.texts_to_sequences([linex])[0]  # turn words into ints
    for i in range(2, len(tokList)):
        nGrams = tokList[:i+1]  # n-gram window grows per token (NLP n-gram concept)
        ninput.append(nGrams)  # build up training data

# pad input and one-hot label
maxLen = max([len(s) for s in ninput])  # get max length of sequence (Keras Padding API)
ninput = pad_sequences(ninput, maxlen=maxLen, padding='pre')  # pad shorter to match (Keras
pad_sequences)
ninput = ninput[:50000]  # limit size so colab doesn't crash (Colab memory tip)

xGRU, labelGRU = ninput[:, :-1], ninput[:, -1]  # split inputs and labels
yGRU = to_categorical(labelGRU, num_classes=maxWords)  # turn label into one-hot (Keras
to_categorical)

#  build the GRU model
modelGRU = Sequential()
modelGRU.add(Embedding(input_dim=maxWords, output_dim=64))  # word embedding (Keras
Embedding Layer)
modelGRU.add(GRU(64))  # GRU cell, handles past memory (Cho et al., 2014)
modelGRU.add(Dense(maxWords, activation='softmax'))  # predict next word in vocab (Softmax
Output Layer)
modelGRU.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])  #
standard compile (Keras Compile API)

#  build the transformer model
modelTRANS = Sequential([
    Input(shape=(maxLen - 1,)),  # input shape is one less than padded (Keras Input Layer)
    Embedding(input_dim=maxWords, output_dim=64),  # same embedding (Keras Embedding
Layer)
    TransformerEncoder(intermediate_dim=128, num_heads=2),  # 2-head encoder block
(KerasNLP TransformerEncoder)
    GlobalAveragePooling1D(),  # flatten the sequence (Keras GlobalAvgPooling)
    Dense(maxWords, activation='softmax')  # output word distribution (Softmax Layer)
])
```

```
modelTRANS.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# same loss + opt (Keras Compile)

# train both models
prevGRU = modelGRU.fit(xGRU, yGRU, epochs=3, verbose=1)  # run GRU model training
(Keras Training Loop)
prevTRANS = modelTRANS.fit(xGRU, yGRU, epochs=3, verbose=1)  # run Transformer model
training (Keras Training Loop)

# plot loss curves
plt.plot(prevGRU.history['loss'], label='GRU')  # GRU loss over epochs (Matplotlib Line Plot)
plt.plot(prevTRANS.history['loss'], label='Transformer')  # Transformer loss over epochs
(Matplotlib)
plt.title('Training Loss Comparison')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)  # add grid lines (Matplotlib Grid)
plt.savefig("lossCurve.png")  # save plot to file (Matplotlib savefig)
plt.show()

#  make text from model
def genWords(net, seedz, steps=10):  # greedy generator loop
    result = seedz
    for _ in range(steps):  # generate n words
        tokenz = tokeniZZR.texts_to_sequences([result])[0]  # tokenize seed (Keras Tokenizer)
        tokenz = pad_sequences([tokenz], maxlen=maxLen-1, padding='pre')  # pad to input shape
(Keras Padding)
        pred = net.predict(tokenz, verbose=0)  # predict next word (Keras predict)
        bestIndex = tf.argmax(pred[0]).numpy()  # pick best word (TensorFlow argmax)
        nextWord = tokeniZZR.index_word.get(bestIndex, "")  # convert index back to word (Keras
Tokenizer)
        result += " " + nextWord
    return result  # return full generated sentence

# generate some output
outGRU = genWords(modelGRU, "thy name is")  # sample from GRU
outTRANS = genWords(modelTRANS, "thy name is")  # sample from transformer

# save to file
with open("dataOutput.txt", "w") as f:  # save text (Python file I/O)
    f.write("GRU Output:\n" + outGRU + "\n\n")
    f.write("Transformer Output:\n" + outTRANS + "\n")
```

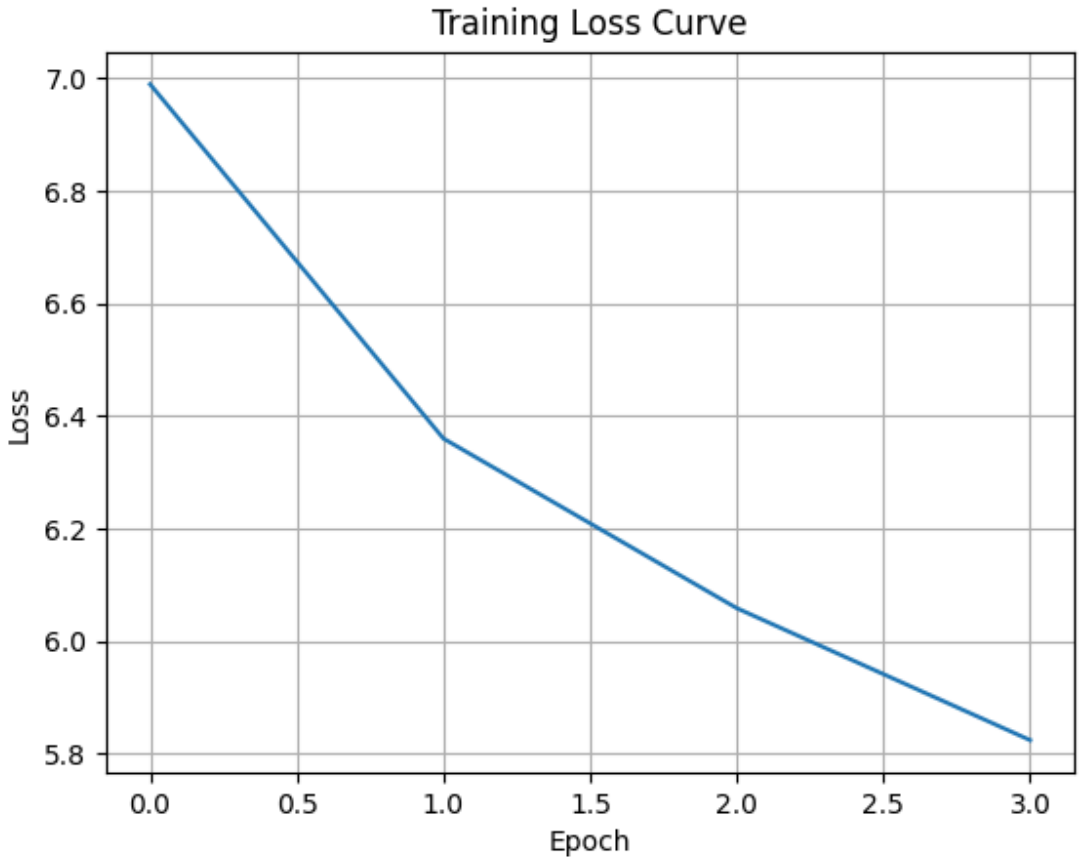
```
# Show the output in Colab as well
with open("dataOutput.txt", "r") as f:
    print(f.read())

# download in colab
from google.colab import files  # colab download API
files.download("lossCurve.png")  # download plot (Colab UI)
files.download("dataOutput.txt")  # download text
```

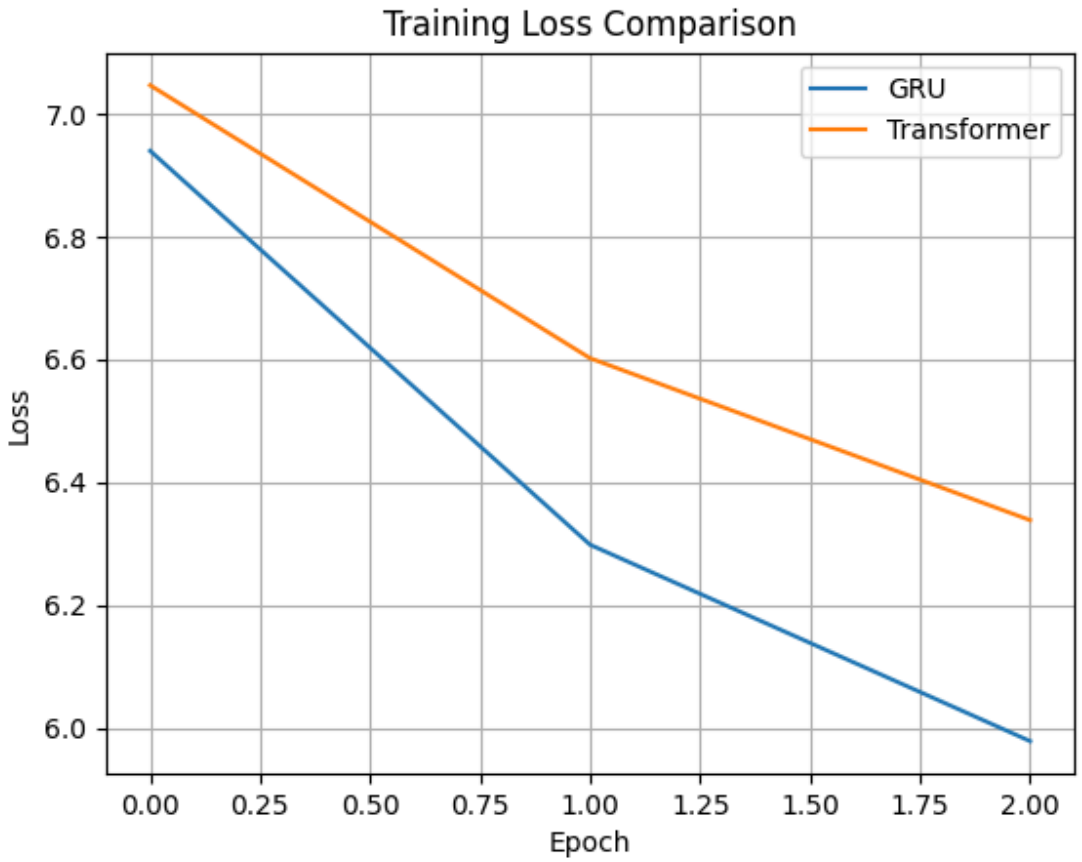
OUTPUTS

SimpleRNN Output:
thy name is a king and i have not a king and i

GRU Output:
thy name is the duke of the world of the world of the

Transformer Output:
thy name is and and and a son of death death death death





Compare and contrast
GRU performed better but similar to simpleRNN would need more time to have more epoch iterations to confirm winner

CITATIONS

TensorFlow/Keras API Documentation
TensorFlow Developers. Keras API Reference. https://keras.io/api/
Used for constructing models (Sequential), compiling, training, and evaluating deep learning architectures including SimpleRNN, GRU, Dense, Embedding, and optimizers.

Keras Tokenizer and Utilities
TensorFlow Developers. Text Vectorization Utilities. https://keras.io/api/preprocessing/text/
Used for Tokenizer (to turn text into sequences), pad_sequences (to pad sequences to fixed length), and to_categorical (for one-hot encoding labels).

Keras Layers: SimpleRNN, GRU, Dense, Embedding, GlobalAveragePooling1D
TensorFlow Developers. Keras Layer Documentation. https://keras.io/api/layers/
Provides all standard layers used in model construction. SimpleRNN and GRU handle sequence memory, Dense for output prediction, Embedding for word vectorization.

Cho et al. (2014) — Gated Recurrent Unit (GRU)
Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. EMNLP.
https://arxiv.org/abs/1406.1078
Source of the GRU architecture used in your second model.

KerasNLP TransformerEncoder
KerasNLP Team. TransformerEncoder Layer Documentation.
https://keras.io/api/keras_nlp/layers/transformer_encoder/
Used for the transformer model's encoder block, which supports multi-head self-attention and intermediate feedforward layers.

Google Colab File Operations
Google Research. Google Colab Documentation.
https://research.google.com/colaboratory/faq.html
for downloading output files from the notebook using files.download().

Matplotlib Visualization Library
Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90–95. https://matplotlib.org
Used to visualize and export the training loss curve (plt.plot, plt.savefig, etc.).

NumPy Numerical Computing Library

Harris, C. R., Millman, K. J., van der Walt, S. J., et al. (2020). Array programming with NumPy. Nature 585, 357–362. https://numpy.org/doc/
Supporting usage (e.g., optional in data preparation), particularly when used in examples.

Python Standard Library: File I/O
Python Software Foundation. Built-in Functions — open().
https://docs.python.org/3/library/functions.html#open
Used to read and write files for the input text and generated output (open(), read(), write()).

Hochreiter & Schmidhuber (1997) — LSTM Background
Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735
While not directly used, this is foundational for understanding GRU and Transformer memory handling.