

Homework 6 - Natural Language Processing - Spring 2025

Essay Question: *Sequence to Sequence Learning for Natural Language Processing (NLP) switched from recurrent network networks (RNN) to transformer models after 2017. The neural attention mechanism used by Transformer models had a number of advantages over earlier RNN models. Write a one paragraph essay describing what neural attention does and identify at least 3 reasons why the use of attention in Transformer models worked better than earlier RNN models for NLP. Conclude your essay with a sentence summarizing the message of your paragraph.*

Notebook Assignment 1 (RNN): Chua's circuit is the simplest electronic circuit exhibiting chaos as verified in numerous laboratory experiments, computer simulations, and rigorous mathematical analyses. The circuit diagram is shown in Fig. 1. It contains five circuit elements. The first four elements are standard linear passive devices: namely, an inductor (L), resistor (R), and two capacitors (C_1 and C_2). These devices setup a standard oscillator circuit. What gives rise to chaos is a nonlinear element whose current-voltage characteristic (i_R vs v_R) that is shown on the right side of Fig. 1 has a negative slope, which essentially means it is an active element with a negative resistance. Physically this active nonlinear diode can be realized using linear op-amp circuits. In this notebook assignment, you will use an LSTM to predict the future steady state behavior of Chua's circuit.

By rescaling the circuit variables v_{C_1} , v_{C_2} , and i_L , we obtain the following dimensionless equations for Chua's circuit involving 3 state variables x , y , and z and 2 dimensionless parameters α and β

$$\begin{aligned}\dot{x}(t) &= \alpha(y(t) - x - \phi(x(t))) \\ \dot{y}(t) &= x(t) - y(t) + z(t) \\ \dot{z}(t) &= -\beta y(t)\end{aligned}$$

where $\phi(x)$ is defined as a piecewise-linear function

$$\phi(x) = m_1 x + \frac{1}{2}(m_0 - m_1)(|x + 1| - |x - 1|)$$

with m_0 and m_1 denoting the slopes of the inner and outer segments of the piecewise linear function shown in Fig. 1.

One can visualize the chaotic behavior of this circuit by numerically integrating the circuit's differential equations (1) and generating a phase plane portrait of the resulting trajectory. A phase portrait plots the points of the state trajectory, $\{x(t), y(t), z(t)\}$, in the state space and draws edges between temporally adjacent points as shown on right side of Fig. 1. What this phase portrait shows is that the long-term steady-state behavior is not strictly periodic and demonstrates a great deal of sensitivity to perturbations of the initial state. This sensitivity to initial conditions is usually seen as the hallmark of a *chaotic* dynamical system.

1. **Simulate Chua's Circuit:** Use the SciPy function `odeint` to numerically integrate Chua's state equation (1) over the time interval $t \in [0, 5]$ seconds. Sample the state every 0.1 seconds assuming $\alpha = 16$, $\beta = 28$, $m_0 = -1.2$ and $m_1 = -0.7$. Use your simulation to generate 25,000 different trajectories from random initial state vectors, (x_0, y_0, z_0) whose components are uniformly sampled over the interval $[-0.5, 0.5]$ in an i.i.d. manner. Use your simulated

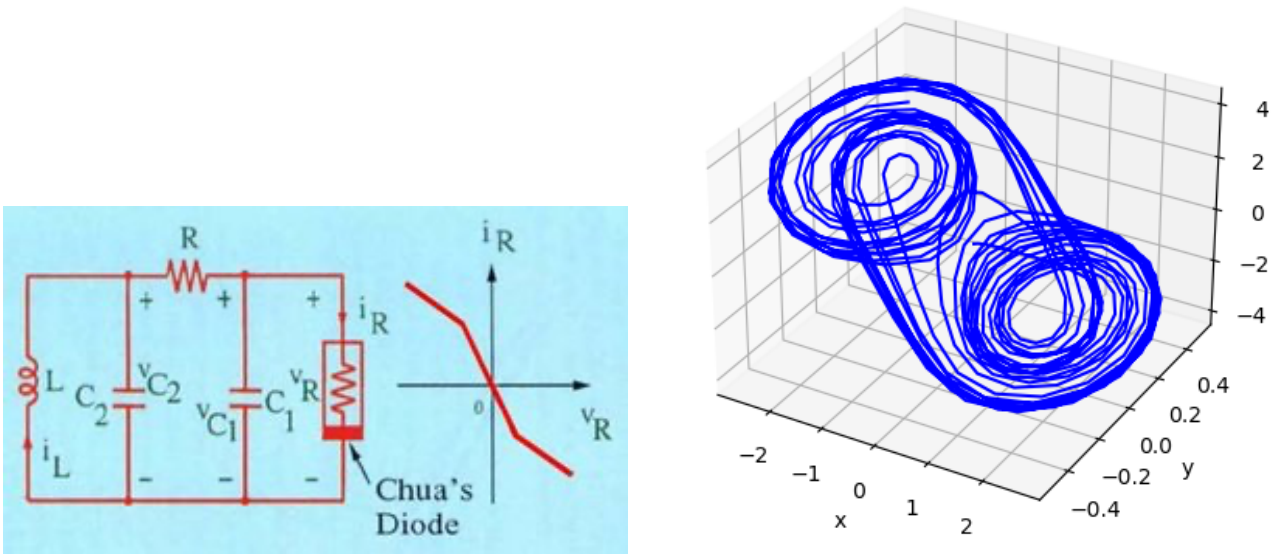


Figure 1: (LEFT) The Chua Circuit, (RIGHT) Chua Attractor, phase plane portrait

trajectories to create a numpy array (`input_bucket`) of shape $(25000, 28, 3)$ whose i th slice is formed by taking the slice `raw_data[i, :28, :]` and adding zero mean white noise whose samples are uniformly distributed between $[-0.1, 0.1]$. So `input_bucket` consists of noise corrupted versions of the first 2.8 seconds of each simulated trajectory. Create another numpy array (`target_bucket`) of shape $(25000, 6)$ whose i th slice first three components equal the noise-free trajectory state `raw_data[48, :]` and whose last three components equal the noise free trajectory state `raw_data[49, :]`. So `target_bucket` are the targets formed from the last two points in the noise free simulated trajectories. Generate a phase portrait for one of the inputs and associated targets. Show the input trajectory in blue and the target in red.

2. **Train LSTM Model:** Split the data in `input_bucket` and `target_bucket` into training and testing set assuming a 25% testing split. Split the training data into a p-training and validation set assuming 25% validation split. Create three tensorflow dataset objects for these data sets assuming a 128 batch size. Instantiate and train a three layer stack of LSTM networks each with 256 nodes (no dropout) for 150 epochs using your dataset objects. Note that this may take a couple of hours to train because your model will have about 1.3 million weights.
3. **Evaluate Model:** Show the training curves and compute the test MAE for the model with the smallest validation loss. For the trained layered model, generate all of the predictions made by your model on the test dataset's inputs. Randomly select 10 of these predictions and plot the 3D phase portrait of the input (blue), the actual target (red), and the predicted target (green - dashed). Based on these samples, does your model appear to predict the future behavior of this chaotic system?

```
In [2]: #####
#
# DEPENDENCIES: python 3.12, tensorflow 3.18.0
# main libraries: numpy, scipy, matplotlib, tensorflow
# sublibraries: tensorflow.keras.layers, scipy.integrate.odeint, matplotlib.
# HW6utils training_curves

import numpy as np
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from scipy.integrate import odeint
import matplotlib.pyplot as plt

from HW6utils import training_curves
```

```
In [5]: #####
#
# TO DO:
# 1) Use the scipy function odeint to numerically integrate Chua's state equations over the interval [0,30] with initial states  $x(0)=.1$ ,  $y(0)=z(0)=0$ . Sample the trajectory every 0.1 seconds. Assume the parameters for the system equations are  $\alpha = 16$ ,  $\beta = 28$ ,  $m_0 = -1.2$ , and  $m_1 = -0.7$ .
# 2) Use your simulation to generate 25,000 different starting trajectories for the circuit where each sample trajectory is run over the interval [0,5] and sampled every 0.1 seconds. The only difference between these trajectories is a randomly selected initial state, whose components are uniformly distributed in a random manner over [-0.5,0.5].
# 3) From your 25000 trajectories create a numpy array (input_bucket) of shape (25000,6) whose ith slice (i,:) contains the states of the ith trajectory's first 2.8 seconds. Add a slice of white noise whose samples are uniformly distributed between [-.1,.1]. Create a numpy array (target_bucket) of shape (25000,6) whose ith slice (i,:) has the two trajectory points in the noise-free trajectory. Plot one of the 3D trajectories showing the data input samples and associated targets, showing the input in blue.

t_0 = 0
dt = 1e-1
t_final = 5
t = np.arange(t_0, t_final, dt)
num_points = int(t_final/dt)

alpha = 16
beta = 28
m0 = -1.2
m1 = -0.7

def chua(u,t):
    x, y, z = u
```

```

    phi = m1*x+0.5*(m0-m1)*(abs(x+1)-abs(x-1))
    randx = np.random.normal(0,5e-5,3)
    dxdt = alpha*(y-x-phi)
    dydt = x-y+z
    dzdt = -beta*y
    dudt = [dxdt, dydt, dzdt]
    return dudt

delay = 22
target_length = 2

# integrate ode system
num_runs = 25000
input_bucket = np.ndarray((num_runs,num_points-delay,3))
target_bucket = np.ndarray((num_runs,target_length*3))

for i in range(num_runs):
    if i%1000==0:
        print(str(i)+'-',end='')
    # initial conditions
    u0 = .5*np.random.uniform(-1,1,3)
    # integrate ode system
    sol = odeint(chua, u0, t)
    noise = .1*np.random.uniform(-1,1,(sol.shape))
    noise[num_points-target_length:,:]=0
    sol += noise
    input_bucket[i] = sol[:num_points-delay,:]
    tmp_target = sol[num_points-target_length:,:]
    target_bucket[i] = np.reshape(tmp_target,target_length*3)

sample = int(np.random.uniform(0,num_runs,1)[0])

input_sample = input_bucket[sample,:,:]
target_sample = target_bucket[sample,:]
target_sample = np.reshape(target_sample,(2,3))

fig = plt.figure(1)
ax = fig.add_subplot(111, projection='3d')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')

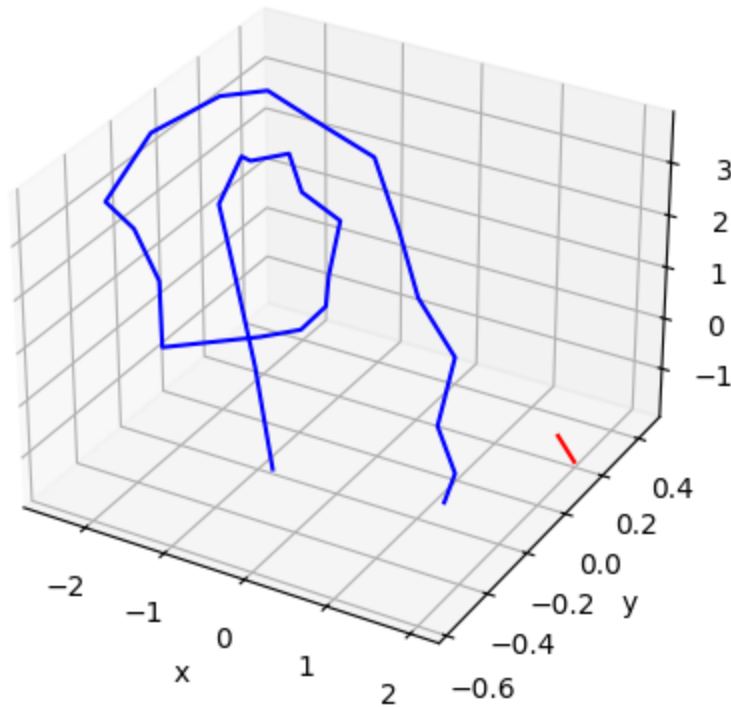
ax.plot(input_sample[:,0],input_sample[:,1],input_sample[:,2],'b')
ax.plot(target_sample[:,0],target_sample[:,1],target_sample[:,2],'r')
tstring = 'input sample = '+str(sample)
ax.set_title(tstring)

```

0-1000-2000-3000-4000-5000-6000-7000-8000-9000-10000-11000-12000-13000-14000
-15000-16000-17000-18000-19000-20000-21000-22000-23000-24000-

Out[5]: Text(0.5, 0.92, 'input sample = 1721')

input sample = 1721



```
In [ ]: #####
#
# TO DO:
# 1) Split the data in input_bucket and target_bucket into training and test
#    Split the training data into a p-training and validation set assuming 2
# 2) Create three tensorflow dataset objects for these data sets assuming a
# 3) Instantiate and train a three layer stack of LSTM networks each with 25
#    using your dataset objects. Show the training curves and compute the tes
#    with the smallest validation loss.
#
# NOTE: training this model takes about 2-3 hours.

num_all_training_samples = int(num_runs*.75)
all_training_inputs = input_bucket[:num_all_training_samples,:,:)
all_training_targets = target_bucket[:num_all_training_samples,:,:)
testing_inputs = input_bucket[num_all_training_samples,:,:)
testing_targets = target_bucket[num_all_training_samples,:,:)

num_ptraining_samples = int(num_all_training_samples*.75)
ptraining_inputs = all_training_inputs[:num_ptraining_samples,:,:)
ptraining_targets = all_training_targets[:num_ptraining_samples,:,:)
validation_inputs = all_training_inputs[num_ptraining_samples,:,:)
validation_targets = all_training_targets[num_ptraining_samples,:,:)

batch_size = 128

ptrain_ds = tf.data.Dataset.from_tensor_slices((ptraining_inputs,ptraining_t
ptrain_ds = ptrain_ds.batch(batch_size)

val_ds = tf.data.Dataset.from_tensor_slices((validation_inputs,validation
```

```

val_ds      = val_ds.batch(batch_size)

test_ds     = tf.data.Dataset.from_tensor_slices((testing_inputs,testing_target))
test_ds     = test_ds.batch(batch_size)

inputs = keras.Input(shape=(num_points-delay, 3))
x = layers.LSTM(256,return_sequences=True)(inputs)
x = layers.LSTM(256,return_sequences=True)(x)
x = layers.LSTM(256)(x)
outputs = layers.Dense(target_length*3)(x)
model = keras.Model(inputs, outputs)

model.compile(optimizer="rmsprop", loss = "mse", metrics=["mae"])

model.summary()

callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="HW6-best-model-1.keras",
        save_best_only = True,
        monitor = "val_loss"
    )
]

num_epochs = 150

history = model.fit(ptrain_ds,
                    epochs = num_epochs,
                    validation_data = val_ds,
                    callbacks = callbacks)

best_model = keras.models.load_model("HW6-best-model-1.keras")
test_loss, test_mae = best_model.evaluate(test_ds,verbose=1)
print(f"Test MAE = {test_mae :.2f}")

training_curves(history)

```



















Model: "functional_1"

Layer (type)	Output Shape	Par
input_layer_1 (InputLayer)	(None, 28, 3)	
lstm_3 (LSTM)	(None, 28, 256)	266
lstm_4 (LSTM)	(None, 28, 256)	525
lstm_5 (LSTM)	(None, 256)	525
dense_1 (Dense)	(None, 6)	1

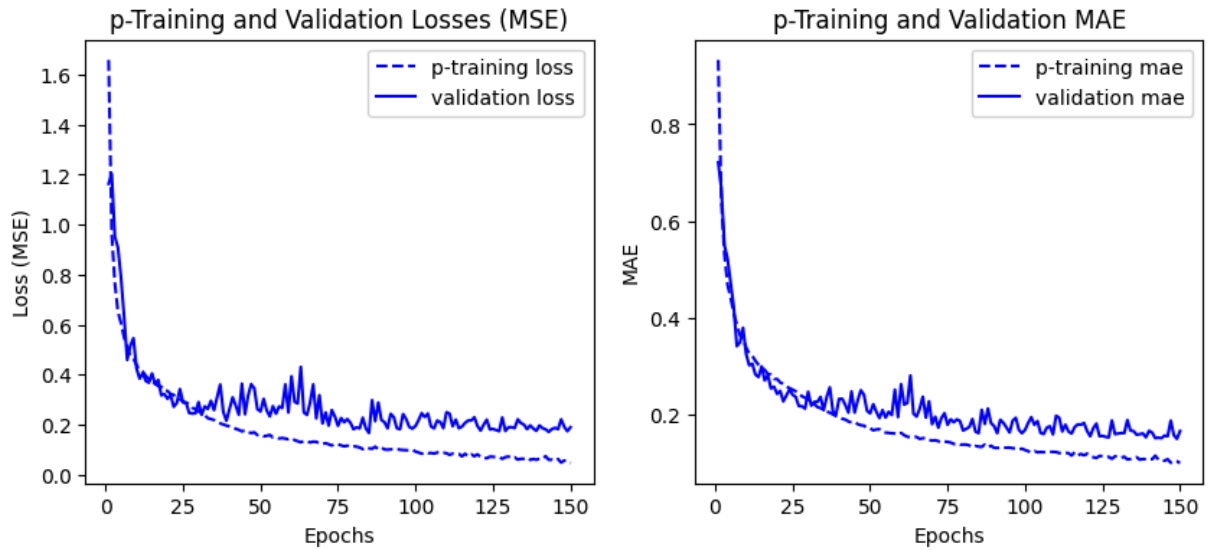
Total params: 1,318,406 (5.03 MB)

Trainable params: 1,318,406 (5.03 MB)

Non-trainable params: 0 (0.00 B)

al_loss: 0.1901 - val_mae: 0.1624
Epoch 132/150
110/110  27s 242ms/step - loss: 0.0654 - mae: 0.1119 - v
al_loss: 0.1863 - val_mae: 0.1621
Epoch 133/150
110/110  27s 242ms/step - loss: 0.0589 - mae: 0.1080 - v
al_loss: 0.2212 - val_mae: 0.1890
Epoch 134/150
110/110  29s 260ms/step - loss: 0.0617 - mae: 0.1083 - v
al_loss: 0.2064 - val_mae: 0.1675
Epoch 135/150
110/110  28s 256ms/step - loss: 0.0652 - mae: 0.1109 - v
al_loss: 0.2003 - val_mae: 0.1663
Epoch 136/150
110/110  28s 257ms/step - loss: 0.0660 - mae: 0.1124 - v
al_loss: 0.1822 - val_mae: 0.1589
Epoch 137/150
110/110  27s 250ms/step - loss: 0.0589 - mae: 0.1066 - v
al_loss: 0.1971 - val_mae: 0.1602
Epoch 138/150
110/110  28s 257ms/step - loss: 0.0650 - mae: 0.1106 - v
al_loss: 0.1847 - val_mae: 0.1614
Epoch 139/150
110/110  26s 232ms/step - loss: 0.0567 - mae: 0.1076 - v
al_loss: 0.1753 - val_mae: 0.1536
Epoch 140/150
110/110  26s 239ms/step - loss: 0.0592 - mae: 0.1075 - v
al_loss: 0.1970 - val_mae: 0.1670
Epoch 141/150
110/110  26s 232ms/step - loss: 0.0582 - mae: 0.1057 - v
al_loss: 0.1859 - val_mae: 0.1632
Epoch 142/150
110/110  25s 224ms/step - loss: 0.0765 - mae: 0.1169 - v
al_loss: 0.1814 - val_mae: 0.1518
Epoch 143/150
110/110  25s 226ms/step - loss: 0.0608 - mae: 0.1079 - v
al_loss: 0.1713 - val_mae: 0.1528
Epoch 144/150
110/110  25s 227ms/step - loss: 0.0616 - mae: 0.1077 - v
al_loss: 0.1824 - val_mae: 0.1516
Epoch 145/150
110/110  24s 222ms/step - loss: 0.0563 - mae: 0.1051 - v
al_loss: 0.1873 - val_mae: 0.1571
Epoch 146/150
110/110  25s 228ms/step - loss: 0.0598 - mae: 0.1072 - v
al_loss: 0.1821 - val_mae: 0.1556
Epoch 147/150
110/110  25s 225ms/step - loss: 0.0440 - mae: 0.0981 - v
al_loss: 0.2213 - val_mae: 0.1880
Epoch 148/150
110/110  26s 239ms/step - loss: 0.0605 - mae: 0.1078 - v
al_loss: 0.1916 - val_mae: 0.1579
Epoch 149/150
110/110  25s 225ms/step - loss: 0.0520 - mae: 0.1036 - v
al_loss: 0.1739 - val_mae: 0.1504
Epoch 150/150

110/110 ————— 26s 236ms/step - loss: 0.0413 - mae: 0.0976 - val_loss: 0.1897 - val_mae: 0.1665
 49/49 ————— 5s 96ms/step - loss: 0.1600 - mae: 0.1580
 Test MAE = 0.15



```
In [7]: #####
#
# TO DO:
# 1) For the trained layered model, generate all of the predictions made by
#    test dataset's inputs. Randomly select 10 of these predictions and plot
#    portrait of the input (blue), the actual target (red), and the predicted
#    samples, does your model appear to predict the future behavior of this
#    chaotic system.

best_model = keras.models.load_model("HW6-best-model-1.keras")
test_loss, test_mae = best_model.evaluate(test_ds, verbose=1)
print(f"Test MAE = {test_mae :.2f}")

predictions = best_model.predict(testing_inputs)

num_testing_samples = num_runs - num_all_training_samples
fig = plt.figure(figsize=plt.figaspect(0.5))
for i in range(10):
    sample = int(np.random.uniform(0, num_testing_samples, 1)[0])
    input_sample = testing_inputs[sample, :, :]
    target_sample = testing_targets[sample, :]
    target_sample = np.reshape(target_sample, (2, 3))
    predict_sample = predictions[sample]
    predict_sample = np.reshape(predict_sample, (2, 3))

    ix = i%2
    iy = int(i/2)

    ax = fig.add_subplot(2, 5, i+1, projection='3d')

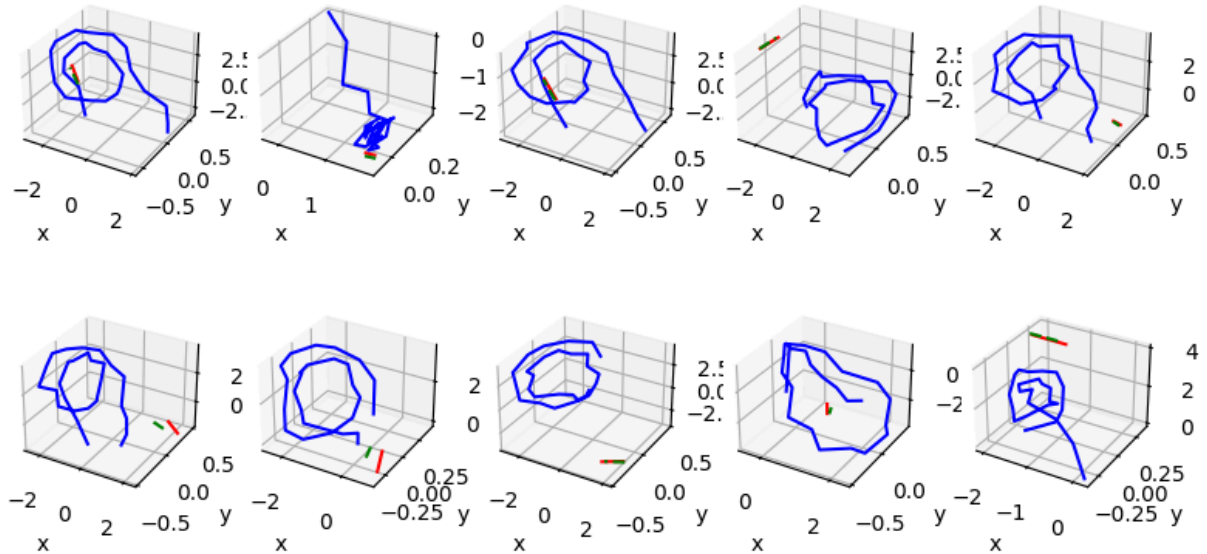
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')
```

```
ax.plot(input_sample[:,0],input_sample[:,1],input_sample[:,2],'b')
ax.plot(target_sample[:,0],target_sample[:,1],target_sample[:,2],'r')
ax.plot(predict_sample[:,0],predict_sample[:,1],predict_sample[:,2],'g--')
```

49/49 ————— 4s 81ms/step - loss: 0.1454 - mae: 0.1527

Test MAE = 0.15

196/196 ————— 9s 45ms/step



Notebook Assignment 2 (SMS spam detection): This notebook assignment uses deep learning to detect spam messages in a manner similar to the sentiment analysis problem discussed in the textbook. We will use a Kaggle data set (SMS Spam Detection Dataset). This assignment makes use of Python's pandas and sklearn libraries, most of which are used in the HW6utils script.

1. **Load the SMS Dataset:** Use the `load_sms_dataset` function from `HW6utils.py` to load the SMS dataset. This dataset consists of 5572 text messages and the targets are labeled 1 (for spam) or 0 (for ham). In loading the data set used an 80/20 split to form the training and testing datasets. Determine the number of training and testing samples. Print out the first 10 input text messages and print out whether they are HAM or SPAM.
2. **Create a Naive Bayes Baseline:** Scikit-learn (aka sklearn) is a free and open-source machine learning library for Python. One sklearn function is a multinomial naive Bayes model for text (you built a similar function in earlier HW). We've encapsulated the training of Sklearn's Naive Bayes model as a `HW6utils` function. Use this function to print a report characterizing the Naive Bayes baseline's metrics

$$\begin{aligned}\text{Precision} &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \\ \text{Sensitivity} &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \\ \text{F1 Score} &= \frac{2 \times \text{Precision} \times \text{Sensitivity}}{\text{Precision} + \text{Recall}}\end{aligned}$$

Ideally we want models that have high precision and sensitivity. The F1 score is the harmonic mean of these two measures and is commonly used to evaluate a classifier's performance.

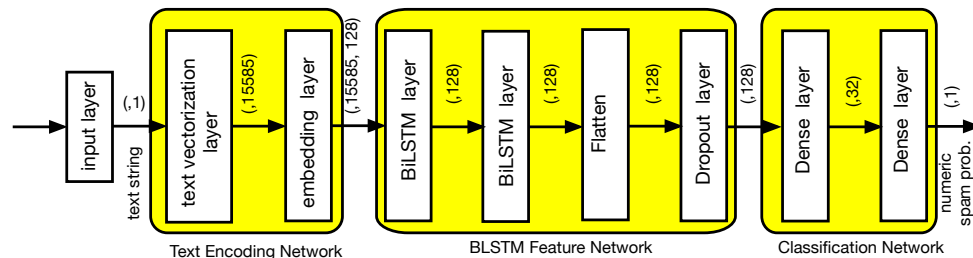


Figure 2: SPAM model architecture

3. **Instantiate SPAM Model:** The architecture for your SPAM model is shown in Fig. 2. The architecture has 3 sequentially connected networks. The *text encoding network* takes the text string input and encodes it as an output sequence length equals the number of unique words (tokens) in the corpus. This encoded sequence is then run through an embedding layer that transforms each token into a numeric feature vector of length 128. This block has been instantiated in the HW6utils.py file. The second network is a *BiLSTM feature network* formed from two Bidirectional LSTM layers with 64 nodes each using a *tanh* activation function. To help regularize learning, we added a dropout layer after the BiLSTM feature network. The final network is a *spam classification* formed from two dense layers feeding down to 32 (relu activation) and then a single numeric output (sigmoid activation) representing the probability of the input text being a SPAM message.

4. **Train Model:** Fit your model using the training data with the testing data being used for validation. Train your model for at least 2 epochs. Note that this model can take a very long time to train, about 1 hour for each epoch. Use Sklearn functions `classification_report` and `confusion_matrix` to evaluate your model's accuracy using 0.5 prediction threshold. Print out the model's accuracy and compare to the NB model's accuracy. Print out the full classification report for your model and compare against the NB model's report. Compute the confusion matrix for your model, print out a heatmap of the confusion matrix, and compare it to the confusion matrix of the NB model.

```
In [17]: #####
#
# DEPENDENCIES: Python 3.12.9, Tensorflow 2.18.0
# Python Libraries: numpy, pandas, matplotlib, seaborn, tensorflow, sklearn
#
# sublibraries: matplotlib.pyplot, tensorflow.keras, sklearn
# tensorflow.keras.layers, tensorflow.keras.layers.TextVectorization
# sklearn.naive_bayes.MultinomialNB
# sklearn.metrics.confusion_matrix

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import TextVectorization

from HW6utils import load_sms_dataset, naive_bayes_report

from sklearn.metrics import confusion_matrix
```

```
In [18]: #####
#
# T0 D0:
# 1) Use load_sms_dataset from HW6utils to load the SMS dataset.
# This dataset inputs are 5572 text messages and the targets
# are 1 (spam message) or 0 (ham message). Specify that the
# function use an 80/20 split in forming the training and testing dataset
# 2) Determine the number of training and testing samples. Print out the fi
# and print out whether they are HAM or SPAM.

filename = "data/spam.csv"
train_split = 0.8
(train_x, train_y), (test_x, test_y), (avg_words_len, total_words_len) = load_sms_dataset(filename, train_split)

print(f"shape of training samples = {train_x.shape}")
print(f"shape of testing samples = {test_x.shape}\n")

for sample in range(10):
    if train_y[sample]==0:
        print(f"HAM:\n {train_x[sample]}\n")
    else:
        print(f"SPAM:\n {train_x[sample]}\n")
```

Average number of tokens in all sentences = 15
 Total number of unique words in corpus = 15585
 shape of training samples = (4457,)
 shape of testing samples = (1115,)

HAM:

Go until jurong point, crazy.. Available only in bugis n great world la e b
 uffet... Cine there got amore wat...

HAM:

Ok lar... Joking wif u oni...

SPAM:

Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA
 to 87121 to receive entry question(std txt rate)T&C's apply 08452810075over1
 8's

HAM:

U dun say so early hor... U c already then say...

HAM:

Nah I don't think he goes to usf, he lives around here though

SPAM:

FreeMsg Hey there darling it's been 3 week's now and no word back! I'd like
 some fun you up for it still? Tb ok! XxX std chgs to send, å£1.50 to rcv

HAM:

Even my brother is not like to speak with me. They treat me like aids paten
 t.

HAM:

As per your request 'Melle Melle (Oru Minnaminunginte Nurungu Vettam)' has
 been set as your callertune for all Callers. Press *9 to copy your friends C
 allertune

SPAM:

WINNER!! As a valued network customer you have been selected to receive a
 £900 prize reward! To claim call 09061701461. Claim code KL341. Valid 12 hou
 rs only.

SPAM:

Had your mobile 11 months or more? U R entitled to Update to the latest col
 our mobiles with camera for Free! Call The Mobile Update Co FREE on 08002986
 030

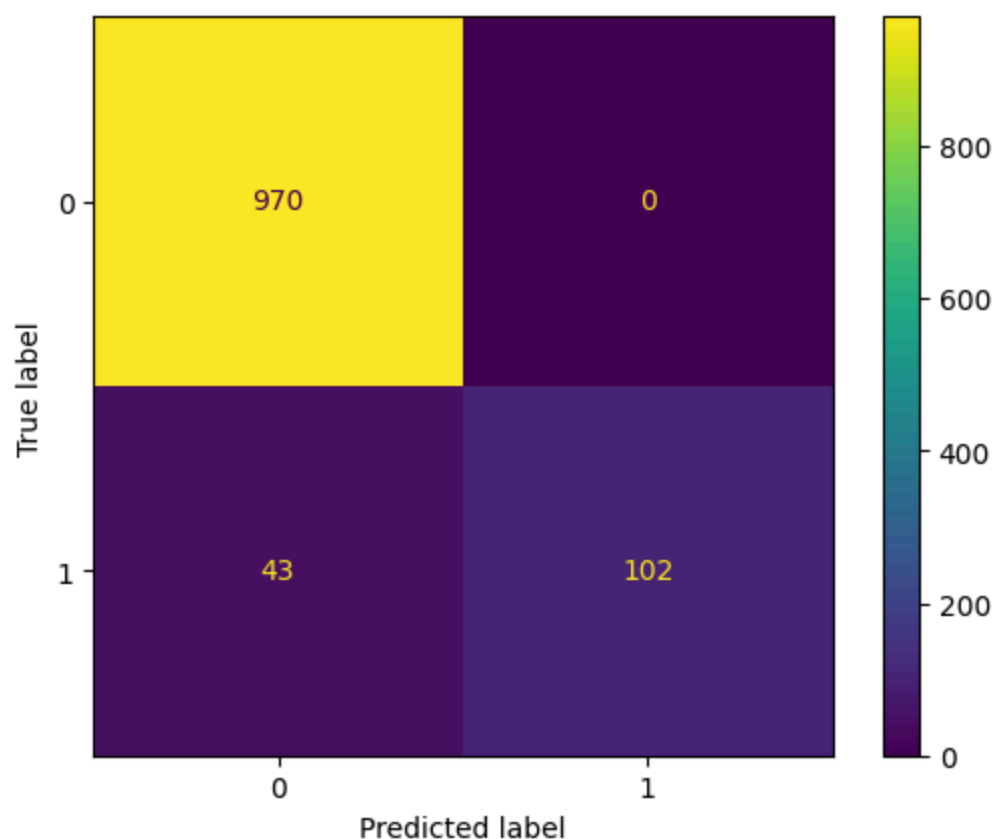
```
In [19]: #####
#
# TO DO:
# 1) scikit-learn (a.k.a. sklearn) is a free and open-source
# machine learning library for the Python programming language.
# One sklearn function of use to us id its multinomial naive
# bayes model for text. We've encapsulated the determination
# of the sklearn naive bayes model in a HW6utils function.
# Describe what this function is doing and use it to print
```

```
# a report characterize the naive bayes baseline metrics our
# trained LSTM model will need to beat.
# 2) The sklearn NB model report returns three metrics;
# precision, recall and f1-score. Look through the documentation
# and describe what each of these metrics actually measures.
```

```
naive_bayes_report(train_x, train_y, test_x, test_y)
```

NB baseline accuracy = 96.14%

	precision	recall	f1-score	support
0	0.96	1.00	0.98	970
1	1.00	0.70	0.83	145
accuracy			0.96	1115
macro avg	0.98	0.85	0.90	1115
weighted avg	0.96	0.96	0.96	1115



```
In [ ]: #####
#
# TO DO:
# 1) create a text vectorization layer (text_vec) that uses
# the average word length in the corpus (15) as the output_sequence_length
# and assumes an output sequence length equal to the number of unique
# words (tokens) in the corpus. Assume an integer output mode.
# 2) Adapt the textvectorization layer to create a vocabulary.
# 3) Create an embedding layer with input_dim = avg_words_len and output_dim
# 4) Create a bidirectional LSTM layer that takes the text message strings
# as inputs and outputs the probability of the input message being SPAM.
# The model has an input layer with shape (1,) and data type tf.string.
```

```

# This is followed by the textvectorization layer and then the embedding
# We then follow with a bidirectional LSTM of 64 nodes and a tanh activation
# This is followed by another bidirectional LSTM of 64 nodes, then a Flatten
# a dropout layer (0.1 probability) which feeds a dense layer of 32 nodes
# The output layer is another dense layer with a single node and sigmoid
#

# create TextVectorization Layer
text_vec = TextVectorization(
    #max_tokens = avg_words_len,
    standardize = 'lower_and_strip_punctuation',
    #output_mode = 'int',
    output_sequence_length=avg_words_len
)

text_vec.adapt(train_x)

#create Embedding Layer

embedding_layer = layers.Embedding(
    input_dim=total_words_len,
    output_dim=128,
    embeddings_initializer='uniform',
    #input_length=avg_words_len,
)

#####
# instantiate and compile model

inputs = layers.Input(shape=(1,), dtype=tf.string)
x = text_vec(inputs)
x = embedding_layer(x)
x = layers.Bidirectional(layers.LSTM(64, activation='tanh', return_sequences=True))
x = layers.Bidirectional(layers.LSTM(64))(x)
#x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
x = layers.Dense(32, activation='relu')(x)
outputs = layers.Dense(1, activation='sigmoid')(x)
model = keras.Model(inputs, outputs)

model.summary()

model.compile(optimizer = "adam", loss= keras.losses.BinaryCrossentropy(), metrics=

```

Model: "functional_5"

Layer (type)	Output Shape	Par
input_layer_5 (InputLayer)	(None, 1)	
text_vectorization_5 (TextVectorization)	(None, 15)	
embedding_5 (Embedding)	(None, 15, 128)	1,994
bidirectional_10 (Bidirectional)	(None, 15, 128)	98
bidirectional_11 (Bidirectional)	(None, 128)	98
dropout_5 (Dropout)	(None, 128)	
dense_10 (Dense)	(None, 32)	4
dense_11 (Dense)	(None, 1)	

Total params: 2,196,673 (8.38 MB)

Trainable params: 2,196,673 (8.38 MB)

Non-trainable params: 0 (0.00 B)

```
In [ ]: # #####
#
# TO DO:
# 1) Create a call back that saves the model with the smallest validation loss
# using the training data with the testing data being used for validation.
# 5 epochs. Note that this model can take a very long time to train, about
```

```
In [21]: callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="HW6-best-model-2a.keras",
        save_best_only = True,
        monitor = "val_loss"
    )
]
num_epochs = 10
history = model.fit(train_x, train_y, epochs = num_epochs,
                    validation_data = (test_x, test_y),
                    batch_size = 128,
                    callbacks = callbacks)
```

```

Epoch 1/10
35/35 ————— 5s 43ms/step - accuracy: 0.8648 - loss: 0.4231 -
val_accuracy: 0.9767 - val_loss: 0.1036
Epoch 2/10
35/35 ————— 1s 36ms/step - accuracy: 0.9791 - loss: 0.0833 -
val_accuracy: 0.9767 - val_loss: 0.0798
Epoch 3/10
35/35 ————— 2s 43ms/step - accuracy: 0.9941 - loss: 0.0294 -
val_accuracy: 0.9803 - val_loss: 0.0768
Epoch 4/10
35/35 ————— 2s 53ms/step - accuracy: 0.9974 - loss: 0.0147 -
val_accuracy: 0.9740 - val_loss: 0.1047
Epoch 5/10
35/35 ————— 2s 54ms/step - accuracy: 0.9974 - loss: 0.0070 -
val_accuracy: 0.9794 - val_loss: 0.0914
Epoch 6/10
35/35 ————— 2s 53ms/step - accuracy: 0.9995 - loss: 0.0026 -
val_accuracy: 0.9767 - val_loss: 0.1146
Epoch 7/10
35/35 ————— 2s 51ms/step - accuracy: 0.9998 - loss: 8.4459e-0
4 - val_accuracy: 0.9776 - val_loss: 0.1262
Epoch 8/10
35/35 ————— 2s 54ms/step - accuracy: 0.9981 - loss: 0.0061 -
val_accuracy: 0.9785 - val_loss: 0.1030
Epoch 9/10
35/35 ————— 2s 55ms/step - accuracy: 0.9987 - loss: 0.0021 -
val_accuracy: 0.9803 - val_loss: 0.1036
Epoch 10/10
35/35 ————— 2s 53ms/step - accuracy: 1.0000 - loss: 1.8712e-0
4 - val_accuracy: 0.9785 - val_loss: 0.1100

```

```

In [24]: #####
#
# TO DO:
# 1) For the best model, use the sklearn functions classification_report
#    and confusion_matrix to evaluate your model using a 0.5 prediction thre
#    your model's accuracy and compare to NB baseline accuracy. Print out t
#    classification report for your model and compare to NB model's report.
#    confusion matrix for your model, display its heatmap and compare to the
#    matrix of the NB baseline.

best_model = keras.models.load_model("HW6-best-model-2a.keras")

from sklearn.metrics import classification_report, accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

threshold = 0.5
predictions = best_model.predict(test_x)
predictions = (predictions >= threshold).astype('int32')
accuracy = accuracy_score(test_y, predictions)
print(f"test accuracy = {accuracy*100:.2f}%")
print(classification_report(test_y, predictions))

cm = confusion_matrix(test_y, predictions)

```

```
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot()
```

35/35 ————— 1s 12ms/step

test accuracy = 98.03%

	precision	recall	f1-score	support
0	0.99	0.99	0.99	970
1	0.91	0.94	0.93	145
accuracy			0.98	1115
macro avg	0.95	0.97	0.96	1115
weighted avg	0.98	0.98	0.98	1115

Out[24]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1625000e0>

