

Homework 6 - Recurrent Neural Networks - 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

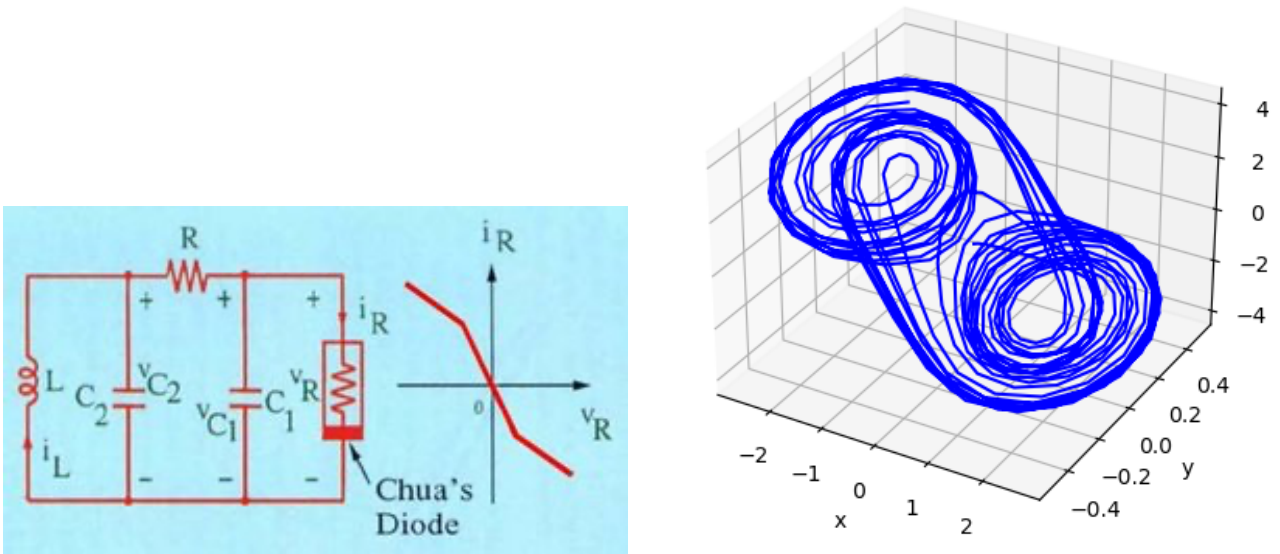


Figure 1: (LEFT) The Chua Circuit, (RIGHT) Chua Attractor, phase plane portrait

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?

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.

3. **Instantiate LSTM Model:** Create a text vectorization layer (`text_vec`) that uses the average word length in the corpus as the max number of tokens and assume an output sequence length equal to the number of unique words (tokens) in the corpus. Assume an integer output mode. Adapt your text vectorization layer to create the vocabulary list. Then create a word embedding layer with input dimension equal to the average word length and output dimension equal to 128 nodes.

Instantiate and compile a bidirectional LSTM layer that takes the text message string as an input and outputs the probability of the input text being SPAM. Your model will have an input layer with shape (1,) and data type `tf.string`. The encoding of the text message will be done by your text vectorization layer (`text_vec`), followed by your word embedding layer. The output of the embedding layer will feed a stack of two bidirectional LSTM layers each with 64 nodes using a tanh activation. You will need to flatten the output of the last LSTM layer. We suggest you introduce a dropout layer, followed by a dense layer with 32 nodes and relu activation, followed by a dense output layer with a single node and sigmoid activation. Compile your model with an adam optimizer, binary crossentropy loss, and accuracy metric.

4. **Train Model:** Create a call back that saves the model with the smallest validation loss and then fit your model using the training data with the testing data being used for validation. Train your model for at least 5 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.