**Week 5 Report Summer Internship**

Name: Mohamed Elfaki

University: Eastern Mediterranean University (Famagusta, Cyprus)

Department: Software Engineering

Training Organization: Cloud Solutions for IT & Communications Co, AI department

**Outline:**

1. Introduction
2. NLP Fundamentals
3. Text preprocessing and Embeddings
4. RNN Fundamentals
5. Sentiment RNN Implementation
6. References
7. **Introduction**

In this week of my internship (week 5), I plan to implement a sentiment RNN using the IMDb reviews dataset. In order to do that, I will start with the basics. I will first go over NLP fundamentals, then move onto text preprocessing and embeddings and finally study RNN fundamentals.

1. **NLP Fundamentals**

NLP (Natural Language Processing) refers to computers understanding human language/speech. Its core concepts consist of tokenizing the text received and turning it all into lower case. Words that don’t carry meaning such as “the” and “is” are removed, and words are reduced to their base form (jumping -> jump).

1. **Text Preprocessing and Embeddings**

To change raw human language to a format a machine learning model can understand, text preprocessing is needed. Text preprocessing consists of many core steps including lowercasing, tokenization, stop word removal, punctuation removal, and lemmatization.

Lowercasing changes all upper-case letter to lower case. Tokenization takes all sentences and splits them into words; stop word removal removes any unnecessary words that do not add meaning to the sentence. Punctuation removal removes any punctuation such as commas and full stops, and lemmatization reduces words to their root.

Another core function is embedding, which is essentially when words (tokens) are switched to vectors (numbers) which models can understand. In a good embedding words such as laptop and computer are close.

1. **RNN Fundamentals**

A recurrent neural network is a type of neural network in which a neuron’s output is calculated based on its weights, bias and a hidden state containing the memory of any previous timesteps. This allows a flexible input to the neural network, while CNNs and traditional NNs used fixed inputs.

In an RNN the hidden state is saved and added to the current neuron to calculate the output; this allows the RNN to calculate based on both past and present data. This makes RNNs very useful in tasks that require memory of past data, such as language comprehension and graph analysis.

1. **RNN Implementation**

To implement an RNN, I first approached this without using any libraries. I tried to make an RNN from scratch without PyTorch or TensorFlow, only NumPy and pandas. However, this was much more difficult than I anticipated, and I encountered many roadblocks; the math for back propagation was complex and I had to use a simple approximation version due to time constraints.

After I completed my forward and backward pass, as well as the update function, I was ready to run my RNN. But I achieved a result I did not expect, I got 49% accuracy on a dataset with 2 label classes. This means that my model’s predictions were 1% worse than random guessing.

After realizing that getting high accuracy on this required more complex precise back propagation calculations, I went into the math and implemented the precise back propagation calculations. Then I ran my code again only to arrive at 51% accuracy, a negligible increase from my previous result.

I did more research and came to a conclusion, this by scratch RNN implementation would need a ton of tweaks before it gave good results.

After realizing this, I switched to PyTorch. Having already understood the inner workings of a basic RNN, PyTorch usage will not be a complete black box anymore. I then implemented an RNN in PyTorch but used a specific type called an LSTM that retains longer memories.

I used Adam as my optimizer, and BCEWithLogitsloss due to the label being 2 classes. My vocabulary only consisted of the 5000 most common words to speed up runtime. Through running this new implementation, I got 84% accuracy which is a huge jump from my original 49%.

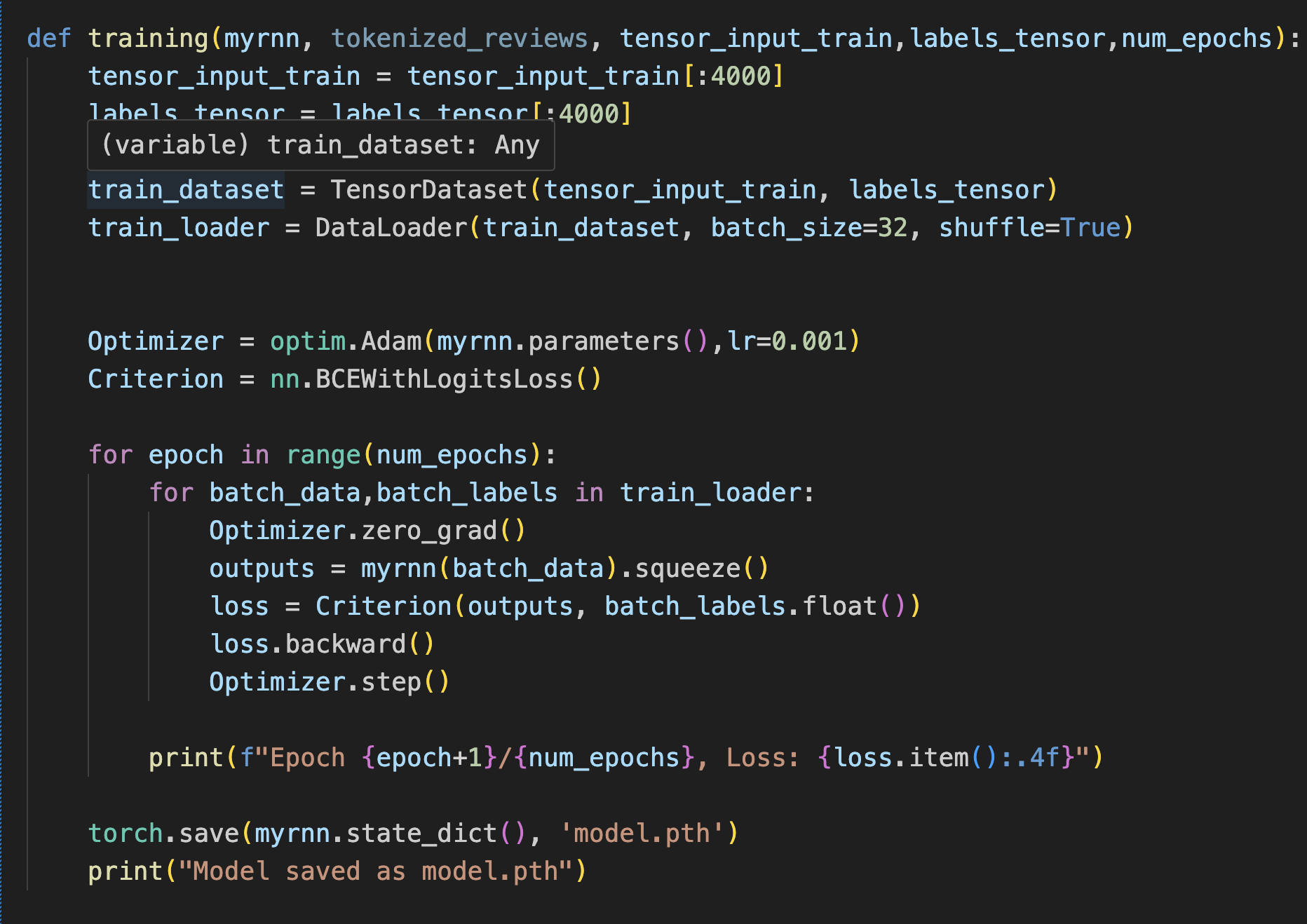
A screenshot of a graph

AI-generated content may be incorrect.

A close-up of a text

AI-generated content may be incorrect.

A sample of the code:



**6.References**

<https://en.wikipedia.org/wiki/Recurrent_neural_network>

<https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks>

<https://www.ibm.com/think/topics/recurrent-neural-networks>

<https://docs.pytorch.org/docs/stable/generated/torch.nn.RNN.html>