ST1504 DEEP LEARNING

Practical 4
Recurrent Neural Network

What you will learn / do in this lab

1. *Create a Recurrent Neural Network (RNN)*
2. *Perform Sentiment Analysis using the RNN*

TABLE OF CONTENTS

1.
OVERVIEW

In this practical we will use python with Keras to create recurrent neural networks (RNN). One of the applications of RNN is in Sentiment Analysis.

INTRODUCTION TO RECURRENT NEURAL NETWORK (RNN)

Recurrent neural networks have the structure that the outputs are fed back into the neural network as input to create cycles in the network graph in an effort to maintain an internal state.

The promise of adding cycles or state to neural networks is that they will be able to explicitly learn and exploit context in sequence prediction problems, such as problems with an order or temporal component.

APPLICATIONS OF RNN

Application areas of RNN includes:

- *Machine Translation*
- *Language Modeling*
- *Natural Language Processing*
- *Speech Recognition*
- *Question Answering*
- *Image Captioning*
- *Handwriting Generation*

2. SENTIMENT ANALYSIS

Sentiment analysis is a natural language processing problem where text is understood and the underlying intent is predicted.

DATA PREPARATION

The Large Movie Review Dataset (often referred to as the **IMDB dataset**) contains 25,000 movie reviews (good or bad) for training and the same amount again for testing. The problem is to determine whether a given movie review has a positive or negative sentiment.

The data was also used as the basis for a Kaggle competition titled "Bag of Words Meets Bags of Popcorn" in late 2014 to early 2015. Accuracy was achieved above 97% with winners achieving 99%.

Keras provides access to the IMDB dataset built-in.

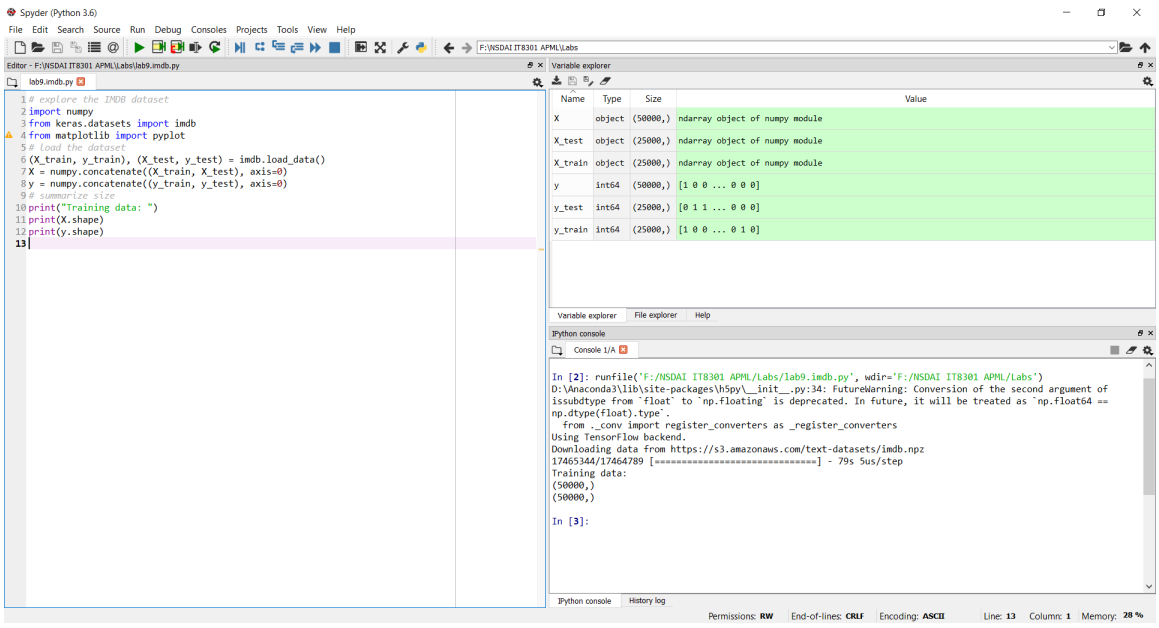
The `keras.datasets.imdb.load_data()` allows you to load the dataset in a format that is ready for use in neural network and deep learning models.

The words have been replaced by integers that indicate the absolute popularity of the word in the dataset. The sentences in each review are therefore comprised of a sequence of integers.

Usefully, the imdb.load_data() provides additional arguments including the number of top words to load (where words with a lower integer are marked as zero in the returned data), the number of top words to skip (to avoid the "the"s) and the maximum length of reviews to support.

Let’s load the dataset and calculate some properties of it. We will start off by loading some libraries and loading the entire IMDB dataset as a training dataset.

```
# explore the IMDB dataset
import numpy as np
from tensorflow.keras.datasets import imdb
from matplotlib import pyplot
# set the random seed
np.random.seed(1)
# load the dataset
(X_train, y_train), (X_test, y_test) = imdb.load_data()
X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)
# summarize size
print("Training data: ")
print(X.shape)
print(y.shape)
```



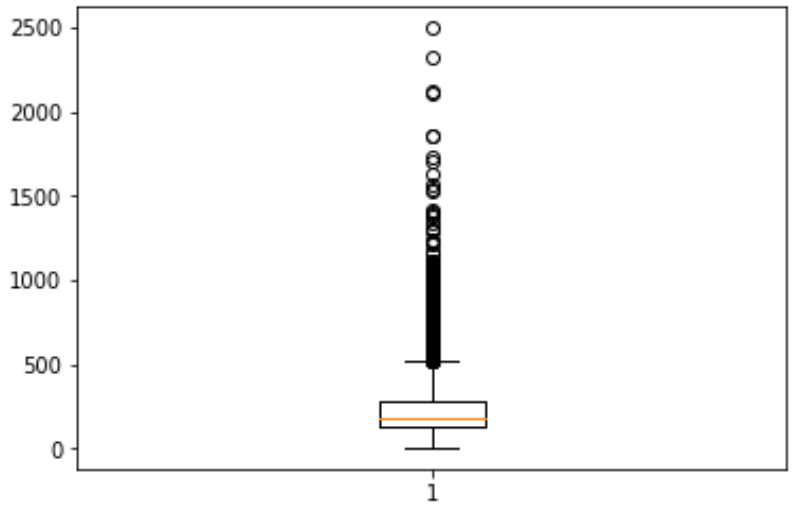
Further exploring the IMDB dataset, we want to determine the number of training samples, the number of label/classes, number of words, and distribution of the reviews length.

```
# explore the IMDB dataset
import numpy as np
from tensorflow.keras.datasets import imdb
from matplotlib import pyplot
np.random.seed(1)
# load the dataset
(X_train, y_train), (X_test, y_test) = imdb.load_data()
X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)
# summarize size
print("Training data: ")
print(X.shape)
print(y.shape)

# Summarize number of classes
print("Classes: ")
print(np.unique(y))
# Summarize number of words
print("Number of words: ")
print(len(np.unique(np.hstack(X))))
# Summarize review length
print("Review length: ")
result = [len(x) for x in X]
print("Mean %.2f words (%f)" % (np.mean(result), np.std(result)))
# plot review length
pyplot.boxplot(result)
pyplot.show()
```

We find the output is:

```
Training data:
(50000,)
(50000,)
Classes:
[0 1]
Number of words:
88585
Review length:
Mean 234.76 words (172.911495)
```



We can see that it is a binary classification problem for good and bad sentiment in the review (only 2 classes: 0 and 1)

Number of words in the reviews is: 88585

Average review length: `234.76 words`

Word Embedding

A recent breakthrough in the field of natural language processing is called **word embedding**. This is a technique where words are encoded as real-valued vectors in a high-dimensional space, where the similarity between words in terms of meaning translates to closeness in the vector space. (see: https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/)

Discrete words are mapped to vectors of continuous numbers. This is useful when working with natural language problems with neural networks and deep learning models are we require numbers as input.

Keras provides a convenient way to convert positive integer representations of words into a word embedding by an **Embedding layer**.

The layer takes arguments that define the mapping including the maximum number of expected words also called the vocabulary size (e.g. the largest integer value that will be seen as an integer). The layer also allows you to specify the dimensionality for each word vector, called the output dimension.

We would like to use a word embedding representation for the IMDB dataset.

Let's say that we are only interested in the first 5,000 most used words in the dataset. Therefore, our vocabulary size will be 5,000. We can choose to use a 32-dimension vector to represent each word. Finally, we may choose to cap the maximum review length at 500 words, truncating reviews longer than that and padding reviews shorter than that with 0 values.

We would then use the Keras utility to truncate or pad the dataset to a length of 500 for each observation using the `sequence.pad_sequences()` function.

```
# explore the IMDB dataset
import numpy as np
from tensorflow.keras.datasets import imdb
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing import sequence
np.random.seed(1)
# load the dataset
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=5000)
X_train = sequence.pad_sequences(X_train, maxlen=500)
X_test = sequence.pad_sequences(X_test, maxlen=500)
X = np.concatenate((X_train, X_test), axis=0)
y = np.concatenate((y_train, y_test), axis=0)
```

Finally, later on, the first layer of our model would be a word embedding layer created using the Embedding class.

```
Embedding(5000, 32, input_length=500)
```

The output of this first layer would be a matrix with the size 32×500 for a given review training or test pattern in integer format.

TRAINING AND EVALUATING THE MODEL

We can start off by developing a simple multi-layer perceptron model with a single hidden layer.

The word embedding representation is a true innovation and we will demonstrate what would have been considered world class results in 2011 with a relatively simple neural network.

Let's start off by importing the classes and functions required for this model and initializing the random number generator to a constant value to ensure we can easily reproduce the results.

Next we will load the IMDB dataset. We will simplify the dataset as discussed during the section on word embeddings. Only the top 5,000 words will be loaded.

We will also use a 50%/50% split of the dataset into training and test.

We will bound reviews at 500 words, truncating longer reviews and zero-padding shorter reviews.

We will use an Embedding layer as the input layer, setting the vocabulary to 5,000, the **word vector size to 32 dimensions** and the input_length to 500. The output of this first layer will be a 32×500 sized matrix as discussed in the previous section.

We will flatten the Embedded layers output to one dimension, then use one dense hidden layer of 250 units with a rectifier activation function. The output layer has one neuron and will use a sigmoid activation to output values of 0 and 1 as predictions.

The model uses logarithmic loss and is optimized using the efficient ADAM optimization procedure.

We can fit the model and use the test set as validation while training. This model overfits very quickly so we will use very few training epochs, in this case just 2.

There is a lot of data so we will use a batch size of 128. After the model is trained, we evaluate its accuracy on the test dataset.

```
# MLP for the IMDB problem
import numpy as np
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing import sequence
# fix random seed for reproducibility
numpy.random.seed(1)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top_words)
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)
# create the model
model = Sequential()
model.add(Embedding(top_words, 32, input_length=max_words))
model.add(Flatten())
model.add(Dense(250, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam', metrics=['accuracy'])
print(model.summary())

# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=2, batch_
size=128, verbose=2)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

3. LSTM

In this section, we will use a Long Short-Term Memory (LSTM) network, a type of recurrent neural network, to perform sentiment analysis on the IMDB dataset.

DATA PREPARATION

The data preparation is similar to what we did in the earlier experiment. We will limit the number of words to 5000 and pad/truncate the reviews to 500 words.

We will map each word onto a 32 length real valued vector. We will also limit the total number of words that we are interested in modeling to the 5000 most frequent words, and zero out the rest. Finally, the sequence length (number of words) in each review varies, so we will constrain each review to be 500 words, truncating long reviews and pad the shorter reviews with zero values.

TRAINING AND EVALUATING THE MODEL

We can quickly develop a small LSTM for the IMDB problem and achieve good accuracy.

The first layer is the Embedded layer that uses 32 length vectors to represent each word. The next layer is the LSTM layer with 100 memory units (smart neurons). Finally, because this is a classification problem we use a Dense output layer with a single neuron and a sigmoid activation function to make 0 or 1 predictions for the two classes (good and bad) in the problem.

Because it is a binary classification problem, log loss is used as the loss function (binary_crossentropy in Keras). The efficient ADAM optimization algorithm is used. The model is fit for only 2 epochs

because it quickly overfits the problem. A large batch size of 64 reviews is used to space out weight updates.

```
# LSTM for the IMDB problem
import numpy as np
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import LSTM
from tensorflow.keras.layers import Embedding
from tensorflow.keras.preprocessing import sequence
# fix random seed for reproducibility
numpy.random.seed(1)
# load the dataset but only keep the top n words, zero the rest
top_words = 5000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=top_words)
max_words = 500
X_train = sequence.pad_sequences(X_train, maxlen=max_words)
X_test = sequence.pad_sequences(X_test, maxlen=max_words)

# create the model
embedding_vector_length = 32
model = Sequential()
model.add(Embedding(top_words, embedding_vector_length,
                    input_length=max_words))
model.add(LSTM(100))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam', metrics=['accuracy'])
print(model.summary())

# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3,
          batch_size=64)
# Final evaluation of the model
scores = model.evaluate(X_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

4.
TOKENIZATION

In this section, we consider the input being actual text, instead of integers. This can be done via the Tokenization function from keras. Tokenization would convert each word to integers. Subsequently, it can be passed into the Embedding layer, RNN layers, etc. as above.

DATA PREPARATION

We will use the "lab4 data.csv" file as input data. This data file comprises 500 quotes. Let us build a model to predict the next word, given as input some number of words of arbitrary length. Since the input data from the file is actual words, we need to convert to integers via the **Tokenization function**.

Refer to the sample code "Lab 4 word predictor example.ipynb" on how to read in an actual text data, carry out tokenization to convert text to integers, then create input-output pairs to train a model to predict the next word.

Since the input can have different length, we need to pad it so that all input would have the same number of words.

Subsequently, we can train an LSTM (or plain RNN, or GRU) to predict the next word. In general, you can repeat the predictions so that the model generates several words, given some input of words.