APPLIED MACHINE LEARNING
LAB ACTIVITIES (LAB 8) (WITH SOLUTIONS)

05/12/19

SEQUENCE DATA PREDICTION USING LSTM

This workbook is designed to guide you through the activities proposed for today's lab. As you will be working independently, feel free to proceed through the text at your own pace, spending more time on the parts that are less familiar to you. The workbook contains both hands-on tasks and links to learning materials such as tutorials, articles and videos. When you are unsure about something, feel free to ask our teaching assistants or use Internet resources to look for a solution. At the end of each section, there will be questions and exercises to verify your understanding of the presented information. You may need to do some research to answer the questions.

1. Time Series Prediction

The problem we are going to look at in this lab is the international airline passengers prediction problem. This is a problem where given a year and a month, the task is to predict the number of international airline passengers in units of 1,000. The data ranges from January 1949 to December 1960 or 12 years, with 144 observations. The dataset is available for free from the DataMarket or Kaggle webpage as a CSV download with the filename $international - airline - passengers.csv$.

https://www.kaggle.com/ternaryrealm/airlines-passenger-data

Below is a sample of the first few lines of the file.

```
"Month","International airline passengers: monthly totals in thousands. Jan 49 ? Dec 60"
"1949-01",112
"1949-02",118
"1949-03",132
"1949-04",129
"1949-05",121
```

Sample Output from Evaluating the Baseline Model.

We can load this dataset easily using the Pandas library. We are not interested in the date, given that each observation is separated by the same interval of one month. Therefore, when we load the dataset, we can exclude the first column. The downloaded dataset also has footer information that we can exclude with the $skipfooter$ argument to $pandas.read\_csv()$ set to 3 for the 3 footer lines. Once loaded we can easily plot the whole dataset. The code to load and plot the dataset is listed below.

```
from pandas import read_csv
import matplotlib.pyplot as plt
dataset = read_csv('international-airline-passengers.csv', usecols=[1], engine='python',
    skipfooter=3)
plt.plot(dataset)
plt.show()
```

LSTMs are sensitive to the scale of the input data, specifically when the $sigmoid$ (default) or $tanh$ activation functions are used. It can be a good practice to rescale the data to the range of 0-to-1, also called normalising. We can easily normalise the dataset using the $MinMaxScaler$ preprocessing class from the scikit-learn library.

```
# normalise the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
```

```
dataset = scaler.fit_transform(dataset)
```

The LSTM network expects the input data (X) to be provided with a specific array structure in the form of: `[samples, time steps, features]`

where
samples is the number of data points we have,
time steps is the number of time-dependent steps that are there in a single data point, and
features refers to the number of variables we have for the corresponding true value in Y.

Our prepared data is in the form: `[samples, features]` and we are framing the problem as one time step for each sample. We can transform the prepared train and test input data into the expected structure using $numpy.reshape()$ as follows:

```
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

The $shape$ attribute for Numpy arrays returns the dimensions of the array. If Y has n rows and m columns, then $Y.shape$ is (n, m). Thus, $Y.shape[0]$ is n.

We are now ready to design and fit our LSTM network for this problem. The network has a visible layer with 1 input, a hidden layer with 4 LSTM blocks or neurons and an output layer that makes a single value prediction. The default sigmoid activation function is used for the LSTM memory blocks. The network is trained for 100 epochs and a batch size of 1 is used.

```
# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam') model.fit(trainX, trainY,
epochs=100, batch_size=1, verbose=2)
```

When defining the input layer of your LSTM network, the network assumes you have 1 or more samples and requires that you specify the number of time steps and the number of features. You can do this by specifying a tuple to the "$input\_shape$" argument.

For example, the model below defines an input layer that expects 1 or more samples, 50 time steps, and 2 features.

For completeness below is the entire code example.

```
# LSTM for international airline passengers problem with regression framing
import numpy
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
  dataX, dataY = [], []
  for i in range(len(dataset)-look_back-1):
    a = dataset[i:(i+look_back), 0]
    dataX.append(a)
    dataY.append(dataset[i + look_back, 0])
  return numpy.array(dataX), numpy.array(dataY)

# LSTM for international airline passengers problem with regression framing
import numpy
```

```
import matplotlib.pyplot as plt
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error

# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
  dataX, dataY = [], []
  for i in range(len(dataset)-look_back-1):
    a = dataset[i:(i+look_back), 0]
    dataX.append(a)
    dataY.append(dataset[i + look_back, 0])
  return numpy.array(dataX), numpy.array(dataY)

# fix random seed for reproducibility
numpy.random.seed(7)

# load the dataset
dataframe = read_csv('international-airline-passengers.csv', usecols=[1],
engine='python', skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')

# normalize the dataset
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)

# split into train and test sets
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]

# reshape into X=t and Y=t+1
look_back = 1
trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# reshape input to be [samples, time steps, features]
trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))

# create and fit the LSTM network
model = Sequential()
model.add(LSTM(4, input_shape=(1, look_back)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)

# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)

# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])

# calculate root mean squared error
trainScore = …
print('Train Score: %.2f RMSE' % (trainScore))
testScore = …
print('Test Score: %.2f RMSE' % (testScore))

# shift train predictions for plotting
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict

# shift test predictions for plotting
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
```

```
# plot baseline and predictions
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

Verify your understanding:

(a) Complete the code block for calculating root mean squared error.

(b) Run the above code block and interpreted the results.

```
Soln:
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))

We can see that the model has an average error of about 23 passengers (in
thousands) on the training dataset and about 52 passengers (in thousands) on the
test dataset. Not that bad.
```

2. Sequence Classification

The problem that we will use to demonstrate sequence learning in this section is the SMS Spam classification problem.

The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged being ham (legitimate) or spam. See below.

https://www.kaggle.com/uciml/sms-spam-collection-dataset/

We can develop a small LSTM for the SMS Spam problem and achieve good accuracy. Let's start off by importing the classes and functions required for this model and initialising the random number generator to a constant value to ensure we can easily reproduce the results.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.models import Model
from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding
from keras.optimizers import RMSprop
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping

# fix random seed for reproducibility
np.random.seed(7)
```

We need to load the spam dataset. We are loading the data into Pandas dataframe.

```
df = pd.read_csv('spam.csv',delimiter=',',encoding='latin-1')
df.head()
```

Verify your understanding:

(c) Run the above code block in Jupyter Notebook and interpret the output.

We now need to drop the columns that are not required. Pandas $dataframe.info()$ function is used to get a concise summary of the dataframe. It comes really handy when doing exploratory analysis of the data.

```
df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'],axis=1,inplace=True)
df.info()
```

Visualising the data may help us to understand the distribution better.

```
sns.countplot(df.v1)
plt.xlabel('Label')
plt.title('Number of ham and spam messages')
```

We need to create input and output vectors and reshape the labels. $LabelEncoder$ can be used to normalise labels and $fit\_transform$ fits label encoder and returns encoded labels.

```
X = df.v2
Y = df.v1
le = LabelEncoder()
Y = le.fit_transform(Y)
Y = Y.reshape(-1,1)
```

We also split the dataset into train (85%) and test (15%) sets.

```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.15)
```

“Tokens” are usually individual words (at least in languages like English) and “tokenization” is taking a text or set of text and breaking it up into its individual words. As each sample in v2 has a sequence of tokens it should be tokenised and added padding to ensure that all the sequences have the same shape. There are many ways of taking the max_len but an arbitrary length of 150 is chosen.

```
max_words = 1000
max_len = 150
tok = Tokenizer(num_words=max_words)
tok.fit_on_texts(X_train)
sequences = tok.texts_to_sequences(X_train)
sequences_matrix = sequence.pad_sequences(sequences,maxlen=max_len)
```

We can now define, compile and fit our LSTM model. The first layer is the Embedded layer that uses 50 length vectors to represent each word. The next layer is the LSTM layer with 64 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 (ham and spam) 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 optimisation algorithm is used. The model is fit for only 5 epochs because it quickly overfits the problem. A large batch size of 128 SMS is used to space out weight updates.

```
def RNN():
    inputs = Input(name='inputs',shape=[max_len])
    layer = Embedding(max_words,50,input_length=max_len)(inputs)
    layer = LSTM(64)(layer)
    layer = Dense(256,name='FC1')(layer)
    layer = Activation('relu')(layer)
    layer = Dropout(0.5)(layer)
    layer = Dense(1,name='out_layer')(layer)
    layer = Activation('sigmoid')(layer)
    model = Model(inputs=inputs,outputs=layer)
    return model
model = RNN()
model.summary()
model.compile(loss='binary_crossentropy',optimizer=’adam’,metrics=['accuracy'])
model.fit(sequences_matrix,Y_train,batch_size=128,epochs=5,validation_split=0.2,callbacks
=[EarlyStopping(monitor='val_loss',min_delta=0.0001)])
```

We need to process the test data as the train data was processed in the previous step.

```
test_sequences = tok.texts_to_sequences(X_test)
test_sequences_matrix = sequence.pad_sequences(test_sequences,maxlen=max_len)
```

Once fit, we estimate the performance of the model on unseen reviews.

```
scores = model.evaluate(test_sequences_matrix,Y_test)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(scores[0],scores[1]))
```

Verify your understanding:

(d) Run the above code block in Jupyter Notebook and interpret the results.

3. References

[1]. Géron, A., 2017. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".
[2]. Raschka, S., 2015. Python machine learning. Packt Publishing Ltd.
[3]. Grus, J., 2019. Data science from scratch: first principles with python. O'Reilly Media.
[4]. Müller, A.C. and Guido, S., 2016. Introduction to machine learning with Python: a guide for data scientists. " O'Reilly Media, Inc.".
[5]. Brownlee, J., 2014. Machine learning mastery.
[6]. Raschka, S., 2015. Python machine learning. Packt Publishing Ltd.
[7]. SAS, 2018, Advanced Predictive Modelling using SAS
[8]. Géron, A., 2019. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.