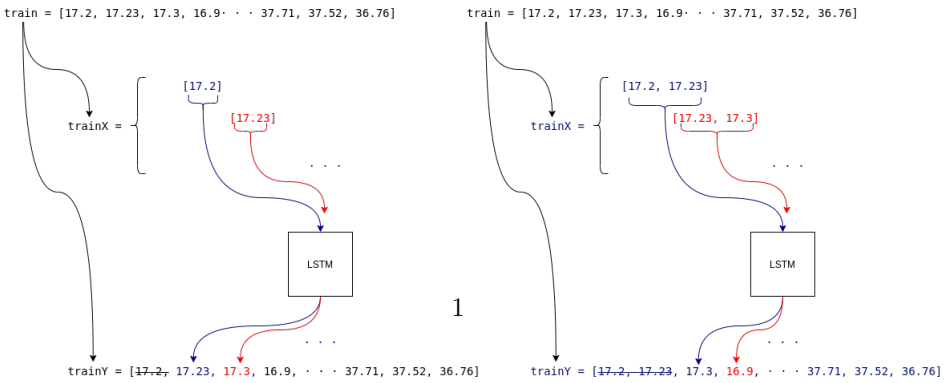
Recursive Neural Networks

Long Short Term Memory (LSTM)

In this part, we focus on using a Deep LSTM Neural Network architecture to provide multidimensional time series forecasting. specifically on stock market datasets to provide momentum indicators of stock price. Based on the available dataset, we are interested in predicting if a stock will rise based on previous information or not.
data.csv contains daily prices. Most of data spans from 2010 to the end 2016, for companies new on stock market date range is shorter.
Exe. 1 Upload the price data set, data.csv using pandas. Check the uploaded dataset and its different columns.
Exe. 2 Regarding the uploaded dataset, the 'symbol' column contains a short name of each company. For instance YHOO represent Yahoo. Get a subset of dataset, contaning YHOO symbol. name this subset of data as yahoo. Get only the high column values (6th column) and save them in a variable namely `yahoo_stock_prices`, and demonstrate high values in a graph. What are your observations?
Exe. 3 Now divide the `yahoo_stock_prices` into `train` and `test` variables. For instance 80% for train and 20% for the test set. Do NOT shuffle the data set because it is a sequential dataset and the its order is important.
Exe. 4 Since, the sequential data are dependent to their previous data, we have to remodel train and test dataset. How does it work?
In order to learn based on a sequence, each element can be dependent only on the previous element of the sequence, or two previous elements of the sequence or more than 2 elements of the sequence. Check the following image, in the left one the `look_back` parameter is 1 while in the right image, each element depends on two previous elements i.e. `look_back = 2`. Write a function as following:



```
import numpy as np
def create_dataset(dataset, look_back):

  return dataX, dataY
```

which receives a dataset and divides it to train and test using the defined look-back. As an example, assume that in the train column we have an array of float values as: `train = array[17.2, 17.23, 17.3, ⋯ 37.71, 37.52, 36.76]`. Assume each data is dependent on its previous data, i.e. if we are looking for a predicted function as f, $f(17.2) = 17.23, f(17.23) = 17.3$ and etc, here `look_back = 1`. Notice the prediction function f will be implented using recursive NN methods in the rest of the exercises.
Now, write the function such that generates the following output:

```
dataX =
[[17.2 ]
 [17.23]
 [17.3 ]
 ...
 [37.21]
 [37.71]
 [37.52]]

dataY = [17.23 17.3  16.9  ... 37.71 37.52 36.76]
```

Generate four arrays namely `trainX, trainY, testX, testY` using `create_dataset()` function from train and test sets.

Notice: **LSTM** function is defined as below in keras 2.0:

```
from keras.layers.recurrent import LSTM

LSTM(units, activation='tanh', recurrent_activation='hard_sigmoid',
     use_bias=True, kernel_initializer='glorot_uniform',
    recurrent_initializer='orthogonal', bias_initializer='zeros',
    unit_forget_bias=True, kernel_regularizer=None,
    recurrent_regularizer=None, bias_regularizer=None,
    activity_regularizer=None, kernel_constraint=None,
    recurrent_constraint=None, bias_constraint=None, dropout=0.0,
    recurrent_dropout=0.0)
```

For more details on the arguments check this link: `http://faroit.com/keras-docs/2.0.2/layers/recurrent/`.
Exe. 5 Define a neural network such as:

$$LSTM, Droupout, LSTM, Droupout, Dense, Activation$$

where the first LSTM has an `input_dim = 1`, and an out put dime i.e. `units =50` with `return_sequences=True`. And the second LSTM has `input_dim = 100` and `return_sequences=False`. Each LSTM is followed with a Dropout and finally the final is a dense layer with an output of size 1 (`units = 1`) and an 'integer' activation function, because here the output is continuous.

Notice that when using the functional API in Keras, layer objects must be created first as in a = Add() and then the layer must be added to the computation graph by invoking the resulting object as in:

```
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.layers import LSTM
from tensorflow.keras.models import Sequential

model = Sequential()

model.add(LSTM(input_dim=1, units=50, return_sequences=True))
model.add(Dropout(0.2))

# complete the rest of the model by yourself.
```

Check the model summary.
Notice: **units:** Positive integer, dimensionality of the output space.
Exe. 5 Compile the model using a mean square error and a rmsprop optimisation.
Exe. 6 Now fit the model on tainX and trainY datasets while defining your `batch_size`, `epochs` and `validation_split`.
Exe. 7 Predict the *textX* with the trained LSTM model and demonstrate the exact and predicted value in a graph. What is your observation?
Exe. 8 Train another model on a different company from the original dataset? How is your prediction method?

Gated Recurrent unit (GRU)

In this section we will build a Text Generator by building a Gated Recurrent Unit (GRU) Network.The data for the described procedure is a collection of short and famous poems by famous poets and is in a poems.txt file downloadable from BrightSpace.
Exe. 9 First read the data from the .csv file and save it into a string variable using:

```
with open('poems.txt', 'r') as file:
    text = file.read()

print(text)
```

Notice : In order to implement any machine learning models on textual data, we first require a vectorial representation of our text:
Exe. 10 Create two dictionaries `char_to_indices` and `indices_to_char` map each unique character to a unique number and vise versa respectively. For your simplicity we give the code:

```
vocabulary = sorted(list(set(text)))

char_to_indices = dict((c, i) for i, c in enumerate(vocabulary))
indices_to_char = dict((i, c) for i, c in enumerate(vocabulary))

```

```
print(vocabulary)
```

Exe. 11 To get valuable data, which we can use to train our model we will split our data up into subsequences with a length of 100 characters using:

```
max_length = 100
steps = 5
sentences = []
next_chars = []
for i in range(0, len(text) - max_length, steps):
  sentences.append(text[i: i + max_length])
  next_chars.append(text[i + max_length])
```

This means, if we have a text as bellow:

```
Buffalo Bill's
defunct
who used to
ride a watersmooth-silver
stallion
and break one two three four five pigeons just like that
Jesus

he was a handsome man
and what i want to know is
how do you like your blueeyed boy
Mister Death
```

and if we select `max_length` = 10 and `steps` = 2, we divide the text to vectors of string of size 10 with windows of size 2. For instance for this example we divide the given texts to different text subsequences as:
[Buffalo Bi],[ffalo Bill], [alo Bill's], etc.

Exe. 12 In order to convert the previous text vectors of 100 size to numerical vector, we will them with a vector of vocabulary size (`len()vocabulary`). Notice that `vocabulary` is the set of all unique vocabularies of the text. For each given text vector, and equivalent vocabulary, we replace the value 1 at the appropriate position, if the word is existed in the selected text vector. For instance, if a set of total vocabularies is [my your her name radio is are red blue orange] and the given text is [my name is red], the text should be represented as: [1001010100]. For the sake of generality the code is given:

```
X = np.zeros((len(sentences), max_length, len(vocabulary)), dtype =
     np.bool)
y = np.zeros((len(sentences), len(vocabulary)), dtype = np.bool)

for i, sentence in enumerate(sentences):
  for t, char in enumerate(sentence):
    X[i, t, char_to_indices[char]] = 1
  y[i, char_to_indices[next_chars[i]]] = 1
```

The GRU is the newer generation of Recurrent Neural networks and is pretty similar to an LSTM. GRU's got rid of the cell state and used the hidden state to transfer information. It also only has two gates, a reset gate and update gate. For more details, check the course subject. To implement GRU, use the following code:

```
keras.layers.GRU(units, activation='tanh', recurrent_activation='
    sigmoid', use_bias=True, kernel_initializer='glorot_uniform',
    recurrent_initializer='orthogonal', bias_initializer='zeros',
    kernel_regularizer=None, recurrent_regularizer=None,
    bias_regularizer=None, activity_regularizer=None,
    kernel_constraint=None, recurrent_constraint=None,
    bias_constraint=None, dropout=0.0, recurrent_dropout=0.0,
    implementation=2, return_sequences=False, return_state=False,
    go_backwards=False, stateful=False, unroll=False, reset_after=
    False)
```

For checking the arguments definition, see `https://keras.io/layers/recurrent/`.
Exe. 13 To be able to generate similar texts later, model your network as:

$$GRU(128), Dense(len(vocabulary))$$

be careful with the `input_shape` of GRU, because the inputs are the numerical vectors equivalent to the text windows. For the dense layer, define an activation as softmax. Finally compile the model with categorical cross entropy and a RMSprop optimiser. We use Softmax or categorical entropy because, we will train the network to output a probability over the vocabulary classes for each input. It is used for multi-class classification.
Exe. 14 fit the trained model on X and Y datasets by defining a `batch_size` and `epochs` values. Check loss and accuracy of your model.
Exe. 15 Now, it is possible to generate a text similar to the trained text of poets. For this purpose, choose part of the original text randomly and separate a 100 size text from the randomly selected part of the text. Using the nn trained model, predict 300 characters and append them to the selected text. In order to predict each new character and appending it to your original selected part of text, we need a loop with a following pseudo code:

1 output $\leftarrow$ selected subtext

2 newchar $\leftarrow$ model.predict(output[-101:-1])

3 output $\leftarrow$ output + newchar

4 go to step 2

be careful with the numerical representation of texts and model prediction output. This is a classification model returning probability of vocabulary elements in the output, the next character is the one with the highest probability.
Exe. 16 Test similar text generator method for another text data source as Sherlock Holmes or Shakespear.