CS 395
Homework 5 – Multivariate LSTMs
20 Points Total
Due in Canvas by 11:59 PM on Sunday, February 24, 2019

Today we continue looking at Long Short-Term Memory networks, or LSTMs. As a reminder, a really good article about LSTMs can be found here: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

In our last lab (HW 4), we looked at univariate LSTMs. In this lab, you will discover how to develop a suite of LSTM models for a multivariate problems.

Multivariate LSTM Models

Multivariate time series data means data where there is more than one observation for each time step.

There are two main models that we may require with multivariate time series data; they are:

- Multiple Input Series.
- Multiple Parallel Series.

Let's take a look at each.

Multiple Input Series

A problem may have two or more *parallel input time series* and *an output time series* that is *dependent* on the input time series.

The input time series are parallel because each series has an observation at the same time steps.

We can demonstrate this with a simple example of two parallel input time series where the output series is the simple addition of the input series.

```
# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
```

We can reshape these three arrays of data as a single dataset where each row is a time step, and each column is a separate time series. This is a standard way of storing parallel time series in a CSV file.

```
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
```

The complete example is listed below.

```
# multivariate data preparation
from numpy import array
from numpy import hstack
# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])

# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))

# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
print(dataset)
```

1. Run the above code once and show the resulting output (1 point).

Running the example prints the dataset with one row per time step and one column for each of the two input and one output parallel time series. As with the univariate time series, we must structure these data into samples with input and output elements.

An LSTM model needs sufficient context to learn a mapping from an input sequence to an output value. LSTMs can support parallel input time series as separate variables or features. Therefore, we need to split the data into samples maintaining the order of observations across the two input sequences.

If we chose three input time steps, then the first sample would look as follows:

Input:

```
10, 15
20, 25
30, 35
```

Output:

```
65
```

That is, the first three timesteps of each parallel series are provided as input to the model and the model associates this with the value in the output series at the third time step, in this case, 65.

We can see that, in transforming the time series into input/output samples to train the model, that we will have to discard some values from the output time series where we do not have values in the input time series at prior time steps. In turn, the choice of the size of the number of input time steps will have an important effect on how much of the training data is used.

We can define a function named split_sequences() that will take a dataset as we have defined it with rows for time steps and columns for parallel series and return input/output samples.

```
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
	X, y = list(), list()
	for i in range(len(sequences)):
		# find the end of this pattern
		end_ix = i + n_steps
		# check if we are beyond the dataset
		if end_ix > len(sequences):
			break
		# gather input and output parts of the pattern
		seq_x, seq_y = sequences[i:end_ix, :-1],
		sequences[end_ix-1, -1]
		X.append(seq_x)
		y.append(seq_y)
	return array(X), array(y)
```

We can test this function on our dataset using three timesteps for each input time series as input. The complete example is listed below.

Make sure you have keras installed if you hadn't done it before in the previous lab:

```
(base) C:\Users\christopher.harris>conda install keras
```

```
# multivariate data preparation
from numpy import array
from numpy import hstack

# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
	X, y = list(), list()
	for i in range(len(sequences)):
		# find the end of this pattern
		end_ix = i + n_steps
		# check if we are beyond the dataset
		if end_ix > len(sequences):
			break
		# gather input and output parts of the pattern
		seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1, -1]
		X.append(seq_x)
		y.append(seq_y)
	return array(X), array(y)

# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
```

```
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps = 3
# convert into input/output
X, y = split_sequences(dataset, n_steps)
print(X.shape, y.shape)
# summarize the data
for i in range(len(X)):
      print(X[i], y[i])
```

2. Run the above code three times and show the resulting output (1 point).

Running the example first prints the shape of the X and y components. We can see that the X component has a three-dimensional structure.

The first dimension is the number of samples, in this case 7. The second dimension is the number of time steps per sample, in this case 3, the value specified to the function. Finally, the last dimension specifies the number of parallel time series or the number of variables, in this case 2 for the two parallel series.

This is the exact three-dimensional structure expected by an LSTM as input. The data is ready to use without further reshaping.

We can then see that the input and output for each sample is printed, showing the three timesteps for each of the two input series and the associated output for each sample.

We are now ready to **fit an LSTM model** on this data.

Any of the varieties of LSTMs in the previous section can be used, such as a Vanilla, Stacked, Bidirectional, CNN, or ConvLSTM model. We will use a Vanilla LSTM where the number of time steps and parallel series (features) are specified for the input layer via the input_shape argument.

```
      # define model
      model = Sequential()
      model.add(LSTM(50, activation='relu', input_shape=(n_steps,
      n_features)))
      model.add(Dense(1))
      model.compile(optimizer='adam', loss='mse')
```

When making a prediction, the model expects three timesteps for two input time series.

We can predict the next value in the output series providing the input values of:

```
      80,    85
      90,    95
      100,  105
```

The shape of the one sample with three timesteps and two variables must be `[1, 3, 2]`.

We would expect the next value in the sequence to be 100 + 105, or 205.

```
# demonstrate prediction
x_input = array([[80, 85], [90, 95], [100, 105]])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
```

The complete example is listed below.

3. Cut and paste the following code and run at least 3 times. Make sure you provide some comment in the output that states that it is Multivariate Vanilla LSTM (you may consider changing the last print statement). Show your input (once) and all outputs (1 point)
4. Change the activation function from 'relu' to 2 other activation functions we have learned about. Make sure the comments indicate that you are using the Multivariate Vanilla LSTM and the activation function you used you may consider changing the last print statement). Run each function at least 3 times. Show code and all outputs (4 points: 2 points each)

```
# multivariate lstm example
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense


# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
    X, y = list(), list()
    for i in range(len(sequences)):
        # find the end of this pattern
        end_ix = i + n_steps
        # check if we are beyond the dataset
        if end_ix > len(sequences):
            break
        # gather input and output parts of the pattern
        seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1, -1]
        X.append(seq_x)
        y.append(seq_y)
    return array(X), array(y)

      X, y = list(), list()
      for i in range(len(sequences)):
            # find the end of this pattern
            end_ix = i + n_steps
            # check if we are beyond the dataset
            if end_ix > len(sequences):
                  break
            # gather input and output parts of the pattern
            seq_x, seq_y = sequences[i:end_ix, :-1], sequences[end_ix-1, -1]
            X.append(seq_x)
            y.append(seq_y)
      return array(X), array(y)

# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
```

```
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps = 3
# convert into input/output
X, y = split_sequences(dataset, n_steps)
# the dataset knows the number of features, e.g. 2
n_features = X.shape[2]
# define model
model = Sequential()
model.add(LSTM(50, activation='relu', input_shape=(n_steps, n_features)))
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=200, verbose=0)
# demonstrate prediction
x_input = array([[80, 85], [90, 95], [100, 105]])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Running the example prepares the data, fits the model, and makes a prediction.

Multiple Parallel Series

An alternate time series problem is the case where there are multiple parallel time series and a value must be predicted for each. For example, given the data from the previous section:

```
[[ 10  15  25]
 [ 20  25  45]
 [ 30  35  65]
 [ 40  45  85]
 [ 50  55 105]
 [ 60  65 125]
 [ 70  75 145]
 [ 80  85 165]
 [ 90  95 185]]
```

We may want to predict the value for each of the three timeseries for the next time step. This might be referred to as multivariate forecasting.

Again, the data must be split into input/output samples in order to train a model.

The first sample of this dataset would be:

Input:

```
10, 15, 25
20, 25, 45
30, 35, 65
```

Output:

```
40, 45, 85
```

The split_sequences() function below will split multiple parallel time series with rows for time steps and one series per column into the required input/output shape.

```
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
	X, y = list(), list()
	for i in range(len(sequences)):
		# find the end of this pattern
		end_ix = i + n_steps
		# check if we are beyond the dataset
		if end_ix > len(sequences)-1:
			break
		# gather input and output parts of the pattern
		seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix, :]
		X.append(seq_x)
		y.append(seq_y)
	return array(X), array(y)
```

We can demonstrate this; the complete example is listed below.

```
# multivariate output data prep
from numpy import array
from numpy import hstack
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
 X, y = list(), list()
 for i in range(len(sequences)):
  # find the end of this pattern
  end_ix = i + n_steps
  # check if we are beyond the dataset
  if end_ix > len(sequences)-1:
   break
  # gather input and output parts of the pattern
  seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix, :]
  X.append(seq_x)
  y.append(seq_y)
 return array(X), array(y)
# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
```

```
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps = 3
# convert into input/output
X, y = split_sequences(dataset, n_steps)
print(X.shape, y.shape)
# summarize the data
for i in range(len(X)):
 print(X[i], y[i])
```

5. Run the above code, showing the code and your resulting output (1 point)

Running the example first prints the shape of the prepared X and y components.

The shape of X is three-dimensional, including the number of samples (6), the number of time steps chosen per sample (3), and the number of parallel time series or features (3).

The shape of y is two-dimensional as we might expect for the number of samples (6) and the number of time variables per sample to be predicted (3).

The data is ready to use in an LSTM model that expects three-dimensional input and two-dimensional output shapes for the X and y components of each sample.

Then, each of the samples is printed showing the input and output components of each sample.

We are now ready to fit an LSTM model on this data.

Any of the varieties of LSTMs in the previous section can be used, such as a Vanilla, Stacked, Bidirectional, CNN, or ConvLSTM model.

We will use a Stacked LSTM where the number of time steps and parallel series (features) are specified for the input layer via the input_shape argument. The number of parallel series is also used in the specification of the number of values to predict by the model in the output layer; again, this is three.

```
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', return_sequences=True,
input_shape=(n_steps, n_features)))
model.add(LSTM(100, activation='relu'))
model.add(Dense(n_features))
model.compile(optimizer='adam', loss='mse')
```

We can predict the next value in each of the three parallel series by providing an input of three time steps for each series.

```
70, 75, 145
80, 85, 165
90, 95, 185
```

The shape of the input for making a single prediction must be 1 sample, 3 timesteps, and 3 features, or `[1, 3, 3]`

```
# demonstrate prediction
x_input = array([[70,75,145], [80,85,165], [90,95,185]])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
```

We would expect the vector output to be:

```
[100, 105, 205]
```

We can tie all of this together and demonstrate a Stacked LSTM for multivariate output time series forecasting below.

6. Cut and paste the following code and run at least 3 times. Make sure you provide some comment in the output that states that it is Multivariate Stacked LSTM (you may consider changing the last print statement). Show your input (once) and all outputs (1 point)
7. Change the activation function from 'relu' to 2 other activation functions we have learned about. Make sure the comments indicate that you are using the Multivariate Stacked LSTM and the activation function you used you may consider changing the last print statement). Run each function at least 3 times. Show code and all outputs (4 points: 2 points each)

```
# multivariate output stacked lstm example
from numpy import array
from numpy import hstack
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
# split a multivariate sequence into samples
def split_sequences(sequences, n_steps):
 X, y = list(), list()
 for i in range(len(sequences)):
  # find the end of this pattern
  end_ix = i + n_steps
  # check if we are beyond the dataset
  if end_ix > len(sequences)-1:
   break
  # gather input and output parts of the pattern
  seq_x, seq_y = sequences[i:end_ix, :], sequences[end_ix, :]
  X.append(seq_x)
  y.append(seq_y)
 return array(X), array(y)
# define input sequence
in_seq1 = array([10, 20, 30, 40, 50, 60, 70, 80, 90])
in_seq2 = array([15, 25, 35, 45, 55, 65, 75, 85, 95])
out_seq = array([in_seq1[i]+in_seq2[i] for i in range(len(in_seq1))])
# convert to [rows, columns] structure
in_seq1 = in_seq1.reshape((len(in_seq1), 1))
```

```
in_seq2 = in_seq2.reshape((len(in_seq2), 1))
out_seq = out_seq.reshape((len(out_seq), 1))
# horizontally stack columns
dataset = hstack((in_seq1, in_seq2, out_seq))
# choose a number of time steps
n_steps = 3
# convert into input/output
X, y = split_sequences(dataset, n_steps)
# the dataset knows the number of features, e.g. 2
n_features = X.shape[2]
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', return_sequences=True,
input_shape=(n_steps, n_features)))
model.add(LSTM(100, activation='relu'))
model.add(Dense(n_features))
model.compile(optimizer='adam', loss='mse')
# fit model
model.fit(X, y, epochs=400, verbose=0)
# demonstrate prediction
x_input = array([[70,75,145], [80,85,165], [90,95,185]])
x_input = x_input.reshape((1, n_steps, n_features))
yhat = model.predict(x_input, verbose=0)
print(yhat)
```

Running the example prepares the data, fits the model, and makes a prediction.

8. What are the primary differences between the two LSTM models? Describe and explain what is occurring differently. (3 points).
9. Provide a single table that illustrates the different LSTM models, the activation functions used, and the average of the 3 results (predictions) you received to 3 decimal places. What are your general findings? Follow the template below. (4 points)

LSTM Model	Activation Function Used	Avg. of 3 Results	General Findings