Machine Learning Exercise - VI - Solutions

May 23, 2022

Kannan Singaravelu

* * *

Exercise 10

a) What are deep sequence modeling and its categories?

b) Train a one-to-one sequence LSTM model for a given dataset

X = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

y = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, 200]

c) Train the model with stacked LSTM layers using the above dataset. Use atleast one additional layer when compared to (b).

d) Train a one-to-one sequence LSTM Model with multiple features for the dataset given below

X1 = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40]

X2 = [3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60]

y = [6, 24, 54, 96, 150, 216, 294, 384, 486, 600, 726, 864, 1014, 1176, 1350, 1536, 1734, 1944, 2166, 2400]

e) Train the model with stacked LSTM layers using the above dataset. Use atleast one additional layer when compared to (d).

Solutions

a) What are deep sequence modeling and its categories?

Deep sequence modeling is essentially applying neural network to problems involving sequential processing of data. Sequence data has memory and comes in many forms such as text, audio, video and financial time series. Thus, requiring a different modeling approaches. Sequence problems can be broadly categorized into the following types

1. one-to-one
2. one-to-many
3. many-to-one
4. many-to-many

We'll see two types of these sequence problem : single feature and multiple features. In the former, each timestep has a single feature and in later, each timestep has multiple features.

[ ]:
```
## Data Retrieval and Preprocessing
# Ignore warnings
import warnings
warnings.filterwarnings('ignore')

# Import required libraries
import pandas as pd
import numpy as np

# Import from keras
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense, LSTM
```

One-to-One Single Feature

In one-to-one sequence problem, there is a single input and a single ouput. We'll use LSTM network to the sequence problems. Each input consists of one timestep, which in turn contains a single feature (X).

> The input of the LSTM is always a 3D array *[`batch, timesteps, feature`]* The output of the LSTM could be a 2D array or 3D array depending upon the `return_sequences` `argument` If `return_sequence` is `False`, the output is a 2D array *[`batch, feature`]* If `return_sequence` is `True`, the output is a 3D array *[`batch, timesteps, feature`]*

The `batch` is the number of samples in the input data (20 in this case), `timesteps` are the number of timesteps per sample (1 in this case) and `feature` correspond to the number of features per timestep (1 in this case).

b) Train a one-to-one sequence LSTM model for a given dataset

[ ]:
```
# create sample dataset
X = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
y = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160,␣
↪170, 180, 190, 200]
```

[ ]:
```
print(f'X: {X}')
print(f'y: {y}')
```

```
X: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
y: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170,
180, 190, 200]
```

[ ]:
```
# reshape into 3D
X = np.array(X).reshape(20,1,1)
```

[ ]:
```python
# check the shape
X.shape
```

[ ]: (20, 1, 1)

[ ]:
```python
# convert to array
y = np.array(y)
```

[ ]:
```python
# check the shape
y.shape
```

[ ]: (20,)

[ ]:
```python
# complie model one
model_one = Sequential()
model_one.add(LSTM(50, activation='relu', input_shape=(1,1)))
model_one.add(Dense(1))
model_one.compile(optimizer='adam', loss='mse')
print(model_one.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 50)                10400

 dense (Dense)               (None, 1)                 51

=================================================================
Total params: 10,451
Trainable params: 10,451
Non-trainable params: 0
_________________________________________________________________
None
```

The `return_sequences` argument tells whether to return the output at each time step instead of the final time step. If we set the `return_sequences` to True, the output shape will become a 3D array, instead of a 2D array.

Let's train the model_one now.

[ ]:
```python
# fit model one
model_one.fit(X, y, batch_size=5, epochs=2000, validation_split=0.2, verbose=0)
```

[ ]: <keras.callbacks.History at 0x1706a6d8b50>

[ ]:
```python
# predict the outcome
test_input = np.array([30])
```

```
test_input = test_input.reshape((1, 1, 1))
test_output = model_one.predict(test_input, verbose=0)
print(test_output)
```

```
[[293.39322]]
```

c) One-to-One Single Feature W/stacked LSTM

For the above function and dataset, let's now train our model with stacked LSTM layers.

[ ]:
```
# complie model two
model_two = Sequential()
model_two.add(LSTM(50, activation='relu', return_sequences=True,␣
 ↪input_shape=(1, 1)))
model_two.add(LSTM(50, activation='relu'))
model_two.add(Dense(1))
model_two.compile(optimizer='adam', loss='mse')
print(model_two.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_1 (LSTM)               (None, 1, 50)             10400

 lstm_2 (LSTM)               (None, 50)                20200

 dense_1 (Dense)             (None, 1)                 51

=================================================================
Total params: 30,651
Trainable params: 30,651
Non-trainable params: 0
_________________________________________________________________
None
```

[ ]:
```
# fit model two
model_two.fit(X, y, batch_size=5, epochs=2000, validation_split=0.2, verbose=0)
```

[ ]: <keras.callbacks.History at 0x1706e1a56a0>

[ ]:
```
# predict the outcome
test_output = model_two.predict(test_input, verbose=0)
print(test_output)
```

```
[[296.6909]]
```

d) One-to-One Multiple Feature

In the above examples, each input sample had one timestep where each timestep had one feature. In this example, we will model a one-to-one sequence problem when the input timesteps have multiple features.

[ ]:
```
# create sample dataset
X1 = [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38,␣
 ↪40]
X2 = [3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57,␣
 ↪60]
y = [6, 24, 54, 96, 150, 216, 294, 384, 486, 600, 726, 864, 1014, 1176, 1350,␣
 ↪1536, 1734, 1944, 2166, 2400]
```

[ ]:
```
print(f'X1: {X1}')
print(f'X2: {X2}')
print(f'y:  {y}')
```

```
X1: [2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40]
X2: [3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57,
60]
y:  [6, 24, 54, 96, 150, 216, 294, 384, 486, 600, 726, 864, 1014, 1176, 1350,
1536, 1734, 1944, 2166, 2400]
```

[ ]:
```
# create a feature matrix
X = np.column_stack((X1, X2))
print(X)
```

```
[[ 2  3]
 [ 4  6]
 [ 6  9]
 [ 8 12]
 [10 15]
 [12 18]
 [14 21]
 [16 24]
 [18 27]
 [20 30]
 [22 33]
 [24 36]
 [26 39]
 [28 42]
 [30 45]
 [32 48]
 [34 51]
 [36 54]
 [38 57]
 [40 60]]
```

[ ]:
```
# reshape into 3D
X = np.array(X).reshape(20,1,2)
```

[ ]:
```
# check the shape
X.shape
```

[ ]: (20, 1, 2)

[ ]:
```
# complie model three
model_three = Sequential()
model_three.add(LSTM(50, activation='relu', input_shape=(1, 2)))
model_three.add(Dense(10, activation='relu'))
model_three.add(Dense(1))
model_three.compile(optimizer='adam', loss='mse')
print(model_three.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (None, 50)                10600

 dense_2 (Dense)             (None, 10)                510

 dense_3 (Dense)             (None, 1)                 11

=================================================================
Total params: 11,121
Trainable params: 11,121
Non-trainable params: 0
_________________________________________________________________
None
```

[ ]:
```
# convert to array
y = np.array(y)
```

[ ]:
```
print(y)
```

```
[   6   24   54   96  150  216  294  384  486  600  726  864 1014 1176
 1350 1536 1734 1944 2166 2400]
```

[ ]:
```
# check the shape
y.shape
```

[ ]: (20,)

[ ]:
```
# fit model three
model_three.fit(X, y, batch_size=5, epochs=2000, validation_split=0.2,␣
 ↪verbose=0)
```

[ ]:
```
<keras.callbacks.History at 0x170725244f0>
```

[ ]:
```
# predict the outcome
test_input = np.array([55,80])
test_input = test_input.reshape((1, 1, 2))
test_output = model_three.predict(test_input, verbose=0)
print(test_output)
```

```
[[3545.9573]]
```

e) One-to-One Multiple Features W/stacked LSTM

For the above function and dataset, let's now train our model with stacked LSTM layers.

[ ]:
```
# complie model four
model_four = Sequential()
model_four.add(LSTM(200, activation='relu', return_sequences=True,␣
 ↪input_shape=(1, 2)))
model_four.add(LSTM(200, activation='relu'))
model_four.add(Dense(50, activation='relu'))
model_four.add(Dense(10, activation='relu'))
model_four.add(Dense(1))
model_four.compile(optimizer='adam', loss='mse')
print(model_four.summary())
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_4 (LSTM)               (None, 1, 200)            162400

 lstm_5 (LSTM)               (None, 200)               320800

 dense_4 (Dense)             (None, 50)                10050

 dense_5 (Dense)             (None, 10)                510

 dense_6 (Dense)             (None, 1)                 11

=================================================================
Total params: 493,771
Trainable params: 493,771
Non-trainable params: 0
_________________________________________________________________
```

```
None
```

[ ]:
```
# fit model four
model_four.fit(X, y, batch_size=5, epochs=2000, validation_split=0.2, verbose=0)
```

[ ]:
```
<keras.callbacks.History at 0x170761be5b0>
```

[ ]:
```
# predict the outcome
test_output = model_four.predict(test_input, verbose=0)
print(test_output)
```

```
[[3474.777]]
```

Note: The data is not treated for feature scaling or in/out sample as the objective here is to showcase the application of sequence modeling.

References

- Keras API Documentaion
- TensorFlow API Documentation
- Scikit-Learn Preprocessing
- Python Resources