Open in Colab

T81-558: Applications of Deep Neural Networks

Module 10: Time Series in Keras
Part 10.1: Time Series Data Encoding for Deep Learning

- Instructor: Jeff Heaton, McKelvey School of Engineering, Washington University in St. Louis
- For more information visit the class website.

Module 10 Material

- **Part 10.1: Time Series Data Encoding for Deep Learning** [Video] [Notebook]
- Part 10.2: Programming LSTM with Keras and TensorFlow [Video] [Notebook]
- Part 10.3: Text Generation with Keras and TensorFlow [Video] [Notebook]
- Part 10.4: Introduction to Transformers [Video] [Notebook]
- Part 10.5: Transformers for Timeseries [Video] [Notebook]

Google CoLab Instructions

The following code ensures that Google CoLab is running the correct version of TensorFlow.

In [1]:
```
try:
    %tensorflow_version 2.x
    COLAB = True
    print("Note: using Google CoLab")
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Note: not using Google CoLab
```

Part 10.1: Time Series Data Encoding

There are many different methods to encode data over time to a neural network. In this chapter, we will examine time series encoding and recurrent networks, two topics that are logical to put together because they are both methods for dealing with data

that spans over time. Time series encoding deals with representing events that occur over time to a neural network. This encoding is necessary because a feedforward neural network will always produce the same output vector for a given input vector. Recurrent neural networks do not require encoding time series data because they can automatically handle data that occur over time.

The variation in temperature during the week is an example of time-series data. For instance, if we know that today's temperature is 25 degrees Fahrenheit and tomorrow's temperature is 27 degrees, the recurrent neural networks and time series encoding provide another option to predict the correct temperature for the week. Conversely, a traditional feedforward neural network will always respond with the same output for a given input. If we train a feedforward neural network to predict tomorrow's temperature, it should return a value of 27 for 25. It will always output 27 when given 25 might hinder its predictions. Surely the temperature of 27 will not always follow 25. It would be better for the neural network to consider the temperatures for days before the prediction. Perhaps the temperature over the last week might allow us to predict tomorrow's temperature. Therefore, recurrent neural networks and time series encoding represent two different approaches to representing data over time to a neural network.

Previously we trained neural networks with input (x) and expected output (y). X was a matrix, the rows were training examples, and the columns were values to be predicted. The x value will now contain sequences of data. The definition of the y value will stay the same.

Dimensions of the training set (x):

- Axis 1: Training set elements (sequences) (must be of the same size as y size)
- Axis 2: Members of sequence
- Axis 3: Features in data (like input neurons)

Previously, we might take as input a single stock price to predict if we should buy (1), sell (-1), or hold (0). The following code illustrates this encoding.

In [2]:
```
x = [
    [32],
    [41],
    [39],
    [20],
    [15]
]

y = [
    1,
    -1,
    0,
    -1,
    1
```

```
]

print(x)
print(y)
```

```
[[32], [41], [39], [20], [15]]
[1, -1, 0, -1, 1]
```

The following code builds a CSV file from scratch. To see it as a data frame, use the following:

In [3]:
```
from IPython.display import display, HTML
import pandas as pd
import numpy as np

x = np.array(x)
print(x[:,0])


df = pd.DataFrame({'x':x[:,0], 'y':y})
display(df)
```

```
[32 41 39 20 15]
/tmp/ipykernel_5068/2644240216.py:2: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major releas
e of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and bet
ter interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54
466

  import pandas as pd
```

	x	y
0	32	1
1	41	-1
2	39	0
3	20	-1
4	15	1

You might want to put volume in with the stock price. The following code shows how to add a dimension to handle the volume.

In [4]:
```
x = [
    [32,1383],
    [41,2928],
    [39,8823],
    [20,1252],
```

```
    [15,1532]
]

y = [
    1,
    -1,
    0,
    -1,
    1
]

print(x)
print(y)
```

```
[[32, 1383], [41, 2928], [39, 8823], [20, 1252], [15, 1532]]
[1, -1, 0, -1, 1]
```

Again, very similar to what we did before. The following shows this as a data frame.

In [5]:
```
from IPython.display import display, HTML
import pandas as pd
import numpy as np

x = np.array(x)
print(x[:,0])


df = pd.DataFrame({'price':x[:,0], 'volume':x[:,1], 'y':y})
display(df)
```

```
[32 41 39 20 15]
```

	price	volume	y
0	32	1383	1
1	41	2928	-1
2	39	8823	0
3	20	1252	-1
4	15	1532	1

Now we get to sequence format. We want to predict something over a sequence, so the data format needs to add a dimension. You must specify a maximum sequence length. The individual sequences can be of any size.

In [6]:
```
x = [
    [[32,1383],[41,2928],[39,8823],[20,1252],[15,1532]],
    [[35,8272],[32,1383],[41,2928],[39,8823],[20,1252]],
    [[37,2738],[35,8272],[32,1383],[41,2928],[39,8823]],
    [[34,2845],[37,2738],[35,8272],[32,1383],[41,2928]],
    [[32,2345],[34,2845],[37,2738],[35,8272],[32,1383]],
]
```

```
y = [
    1,
    -1,
    0,
    -1,
    1
]

print(x)
print(y)
```

```
[[[32, 1383], [41, 2928], [39, 8823], [20, 1252], [15, 1532]], [[35, 8272],
[32, 1383], [41, 2928], [39, 8823], [20, 1252]], [[37, 2738], [35, 8272], [3
2, 1383], [41, 2928], [39, 8823]], [[34, 2845], [37, 2738], [35, 8272], [32,
1383], [41, 2928]], [[32, 2345], [34, 2845], [37, 2738], [35, 8272], [32, 13
83]]]
[1, -1, 0, -1, 1]
```

Even if there is only one feature (price), you must use 3 dimensions.

In [7]:
```
x = [
    [[32],[41],[39],[20],[15]],
    [[35],[32],[41],[39],[20]],
    [[37],[35],[32],[41],[39]],
    [[34],[37],[35],[32],[41]],
    [[32],[34],[37],[35],[32]],
]

y = [
    1,
    -1,
    0,
    -1,
    1
]

print(x)
print(y)
```

```
[[[32], [41], [39], [20], [15]], [[35], [32], [41], [39], [20]], [[37], [3
5], [32], [41], [39]], [[34], [37], [35], [32], [41]], [[32], [34], [37], [3
5], [32]]]
[1, -1, 0, -1, 1]
```

Module 10 Assignment

You can find the first assignment here: assignment 10

In [ ]:
```

```