Open in Colab

T81-558: Applications of Deep Neural Networks

Module 10: Time Series in Keras

- 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. Running the following code will map your GDrive to `/content/drive`.

In [1]:
```
try:
    from google.colab import drive
    drive.mount('/content/drive', force_remount=True)
    COLAB = True
    print("Note: using Google CoLab")
    %tensorflow_version 2.x
except:
    print("Note: not using Google CoLab")
    COLAB = False
```

```
Mounted at /content/drive
Note: using Google CoLab
```

Part 10.5: Programming Transformers with Keras

This section shows an example of a transformer encoder to predict sunspots. You can find the data files needed for this example at the following location.

- Sunspot Data Files
- Download Daily Sunspots - 1/1/1818 to now.

The following code loads the sunspot file:

In [2]:
```
import pandas as pd
import os

names = ['year', 'month', 'day', 'dec_year', 'sn_value' ,
         'sn_error', 'obs_num', 'extra']
df = pd.read_csv(
    "https://data.heatonresearch.com/data/t81-558/SN_d_tot_V2.0.csv",
    sep=';',header=None,names=names,
    na_values=['-1'], index_col=False)

print("Starting file:")
print(df[0:10])

print("Ending file:")
print(df[-10:])
```

```
Starting file:
   year  month  day  dec_year  sn_value  sn_error  obs_num  extra
0  1818      1    1  1818.001        -1       NaN        0      1
1  1818      1    2  1818.004        -1       NaN        0      1
2  1818      1    3  1818.007        -1       NaN        0      1
3  1818      1    4  1818.010        -1       NaN        0      1
4  1818      1    5  1818.012        -1       NaN        0      1
5  1818      1    6  1818.015        -1       NaN        0      1
6  1818      1    7  1818.018        -1       NaN        0      1
7  1818      1    8  1818.021        65      10.2        1      1
8  1818      1    9  1818.023        -1       NaN        0      1
9  1818      1   10  1818.026        -1       NaN        0      1
Ending file:
       year  month  day  dec_year  sn_value  sn_error  obs_num  extra
72855  2017      6   21  2017.470        35       1.0       41      0
72856  2017      6   22  2017.473        24       0.8       39      0
72857  2017      6   23  2017.475        23       0.9       40      0
72858  2017      6   24  2017.478        26       2.3       15      0
72859  2017      6   25  2017.481        17       1.0       18      0
72860  2017      6   26  2017.484        21       1.1       25      0
72861  2017      6   27  2017.486        19       1.2       36      0
72862  2017      6   28  2017.489        17       1.1       22      0
72863  2017      6   29  2017.492        12       0.5       25      0
72864  2017      6   30  2017.495        11       0.5       30      0
```

As you can see, there is quite a bit of missing data near the end of the file. We want to find the starting index where the missing data no longer occurs. This technique is somewhat sloppy; it would be better to find a use for the data between missing values.

However, the point of this example is to show how to use a transformer encoder with a somewhat simple time series.

In [3]:
```
# Find the last zero and move one beyond
start_id = max(df[df['obs_num'] == 0].index.tolist())+1
print(start_id)
df = df[start_id:] # Trim the rows that have missing observations
```

```
11314
```

Divide into training and test/validation sets.

In [4]:
```
df['sn_value'] = df['sn_value'].astype(float)
df_train = df[df['year']<2000]
df_test = df[df['year']>=2000]

spots_train = df_train['sn_value'].tolist()
spots_test = df_test['sn_value'].tolist()

print("Training set has {} observations.".format(len(spots_train)))
print("Test set has {} observations.".format(len(spots_test)))
```

```
Training set has 55160 observations.
Test set has 6391 observations.
```

The **to_sequences** function takes linear time series data into an **x** and **y** where **x** is all possible sequences of seq_size. After each **x** sequence, this function places the next value into the **y** variable. These **x** and **y** data can train a time-series neural network.

In [5]:
```
import numpy as np

def to_sequences(seq_size, obs):
    x = []
    y = []

    for i in range(len(obs)-SEQUENCE_SIZE):
        #print(i)
        window = obs[i:(i+SEQUENCE_SIZE)]
        after_window = obs[i+SEQUENCE_SIZE]
        window = [[x] for x in window]
        #print("{} - {}".format(window,after_window))
        x.append(window)
        y.append(after_window)

    return np.array(x),np.array(y)


SEQUENCE_SIZE = 10
x_train,y_train = to_sequences(SEQUENCE_SIZE,spots_train)
x_test,y_test = to_sequences(SEQUENCE_SIZE,spots_test)

print("Shape of training set: {}".format(x_train.shape))
print("Shape of test set: {}".format(x_test.shape))
```

```
Shape of training set: (55150, 10, 1)
Shape of test set: (6381, 10, 1)
```

We can view the results of the **to_sequences** encoding of the sunspot data.

In [6]:
```
print(x_train.shape)
```

```
(55150, 10, 1)
```

Next, we create the transformer_encoder; I obtained this function from a Keras example. This layer includes residual connections, layer normalization, and dropout. This resulting layer can be stacked multiple times. We implement the projection layers with the Keras Conv1D.

In [7]:
```
from tensorflow import keras
from tensorflow.keras import layers

def transformer_encoder(inputs, head_size, num_heads, ff_dim, dropout=0):
    # Normalization and Attention
    x = layers.LayerNormalization(epsilon=1e-6)(inputs)
    x = layers.MultiHeadAttention(
        key_dim=head_size, num_heads=num_heads, dropout=dropout
    )(x, x)
    x = layers.Dropout(dropout)(x)
    res = x + inputs

    # Feed Forward Part
    x = layers.LayerNormalization(epsilon=1e-6)(res)
    x = layers.Conv1D(filters=ff_dim, kernel_size=1, activation="relu")(x)
    x = layers.Dropout(dropout)(x)
    x = layers.Conv1D(filters=inputs.shape[-1], kernel_size=1)(x)
    return x + res
```

The following function is provided to build the model, including the attention layer.

In [8]:
```
def build_model(
    input_shape,
    head_size,
    num_heads,
    ff_dim,
    num_transformer_blocks,
    mlp_units,
    dropout=0,
    mlp_dropout=0,
):
    inputs = keras.Input(shape=input_shape)
    x = inputs
    for _ in range(num_transformer_blocks):
        x = transformer_encoder(x, head_size, num_heads, ff_dim, dropout)

    x = layers.GlobalAveragePooling1D(data_format="channels_first")(x)
    for dim in mlp_units:
        x = layers.Dense(dim, activation="relu")(x)
        x = layers.Dropout(mlp_dropout)(x)
```

```
    outputs = layers.Dense(1)(x)
    return keras.Model(inputs, outputs)
```

We are now ready to build and train the model.

In [9]:
```
input_shape = x_train.shape[1:]

model = build_model(
    input_shape,
    head_size=256,
    num_heads=4,
    ff_dim=4,
    num_transformer_blocks=4,
    mlp_units=[128],
    mlp_dropout=0.4,
    dropout=0.25,
)

model.compile(
    loss="mean_squared_error",
    optimizer=keras.optimizers.Adam(learning_rate=1e-4)
)
#model.summary()

callbacks = [keras.callbacks.EarlyStopping(patience=10, \
    restore_best_weights=True)]

model.fit(
    x_train,
    y_train,
    validation_split=0.2,
    epochs=200,
    batch_size=64,
    callbacks=callbacks,
)

model.evaluate(x_test, y_test, verbose=1)
```

```
Epoch 1/200
690/690 [==============================] - 25s 16ms/step - loss: 1919.4844 -
val_loss: 463.2157
Epoch 2/200
690/690 [==============================] - 11s 16ms/step - loss: 1113.0945 -
val_loss: 365.1375
Epoch 3/200
690/690 [==============================] - 11s 16ms/step - loss: 873.6814 -
val_loss: 345.2026
Epoch 4/200
690/690 [==============================] - 11s 16ms/step - loss: 789.0035 -
val_loss: 329.4594
Epoch 5/200
690/690 [==============================] - 11s 15ms/step - loss: 762.3812 -
val_loss: 324.1521
Epoch 6/200
690/690 [==============================] - 11s 15ms/step - loss: 750.8315 -
val_loss: 323.2135
Epoch 7/200
690/690 [==============================] - 11s 15ms/step - loss: 744.6664 -
val_loss: 326.0743
Epoch 8/200
690/690 [==============================] - 11s 15ms/step - loss: 737.4108 -
val_loss: 312.2708
Epoch 9/200
690/690 [==============================] - 11s 15ms/step - loss: 719.6789 -
val_loss: 307.9902
Epoch 10/200
690/690 [==============================] - 11s 16ms/step - loss: 715.9462 -
val_loss: 296.0132
Epoch 11/200
690/690 [==============================] - 11s 15ms/step - loss: 712.4185 -
val_loss: 299.6908
Epoch 12/200
690/690 [==============================] - 11s 15ms/step - loss: 709.5587 -
val_loss: 297.3573
Epoch 13/200
690/690 [==============================] - 11s 16ms/step - loss: 703.4562 -
val_loss: 293.3137
Epoch 14/200
690/690 [==============================] - 11s 15ms/step - loss: 714.5865 -
val_loss: 310.3690
Epoch 15/200
690/690 [==============================] - 11s 15ms/step - loss: 706.6390 -
val_loss: 304.2530
Epoch 16/200
690/690 [==============================] - 11s 15ms/step - loss: 701.1292 -
val_loss: 297.4577
Epoch 17/200
690/690 [==============================] - 10s 15ms/step - loss: 705.1274 -
val_loss: 326.7051
Epoch 18/200
690/690 [==============================] - 11s 15ms/step - loss: 696.4119 -
val_loss: 290.4462
Epoch 19/200
690/690 [==============================] - 10s 15ms/step - loss: 701.0396 -
```

```
val_loss: 324.6311
Epoch 20/200
690/690 [==============================] - 11s 17ms/step - loss: 694.8000 -
val_loss: 304.3717
Epoch 21/200
690/690 [==============================] - 11s 16ms/step - loss: 697.7822 -
val_loss: 314.0597
Epoch 22/200
690/690 [==============================] - 11s 15ms/step - loss: 696.5428 -
val_loss: 296.1778
Epoch 23/200
690/690 [==============================] - 11s 16ms/step - loss: 688.9620 -
val_loss: 291.3384
Epoch 24/200
690/690 [==============================] - 11s 15ms/step - loss: 692.5215 -
val_loss: 294.7356
Epoch 25/200
690/690 [==============================] - 10s 15ms/step - loss: 695.5998 -
val_loss: 309.7605
Epoch 26/200
690/690 [==============================] - 11s 16ms/step - loss: 688.1234 -
val_loss: 289.6525
Epoch 27/200
690/690 [==============================] - 11s 16ms/step - loss: 680.7135 -
val_loss: 287.5633
Epoch 28/200
690/690 [==============================] - 11s 15ms/step - loss: 689.2556 -
val_loss: 306.3144
Epoch 29/200
690/690 [==============================] - 11s 15ms/step - loss: 686.9375 -
val_loss: 294.5692
Epoch 30/200
690/690 [==============================] - 10s 15ms/step - loss: 689.2764 -
val_loss: 295.0640
Epoch 31/200
690/690 [==============================] - 11s 15ms/step - loss: 683.3184 -
val_loss: 306.8054
Epoch 32/200
690/690 [==============================] - 11s 16ms/step - loss: 679.1677 -
val_loss: 311.3470
Epoch 33/200
690/690 [==============================] - 11s 15ms/step - loss: 683.5298 -
val_loss: 292.4295
Epoch 34/200
690/690 [==============================] - 11s 15ms/step - loss: 689.2239 -
val_loss: 298.1823
Epoch 35/200
690/690 [==============================] - 11s 15ms/step - loss: 682.8902 -
val_loss: 297.4239
Epoch 36/200
690/690 [==============================] - 11s 15ms/step - loss: 679.1320 -
val_loss: 289.7046
Epoch 37/200
690/690 [==============================] - 11s 16ms/step - loss: 673.3400 -
val_loss: 297.0687
200/200 [==============================] - 1s 5ms/step - loss: 214.5603
```

Out[9]: 214.56031799316406

Finally, we evaluate the model with RMSE.

In [10]:
```
from sklearn import metrics

pred = model.predict(x_test)
score = np.sqrt(metrics.mean_squared_error(pred,y_test))
print("Score (RMSE): {}".format(score))
```

```
Score (RMSE): 14.647875946283007
```