02_stacked_lstm_with_feature_embeddings

September 29, 2021

1 Stacked LSTMs for Time Series Classification

We'll now build a slightly deeper model by stacking two LSTM layers using the Quandl stock price data (see the stacked_lstm_with_feature_embeddings notebook for implementation details). Furthermore, we will include features that are not sequential in nature, namely indicator variables for identifying the equity and the month.

1.1 Run inside docker container for GPU acceleration

See tensorflow guide and more detailed instructions

`docker run -it -p 8889:8888 -v /path/to/machine-learning-for-trading/18_recurrent_neural_nets:/`
`--name tensorflow tensorflow/tensorflow:latest-gpu-py3 bash`

Inside docker container: `jupyter notebook --ip 0.0.0.0 --no-browser --allow-root`

1.2 Imports

[18]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, date
from sklearn.metrics import mean_squared_error, roc_auc_score
from sklearn.preprocessing import minmax_scale
from keras.callbacks import ModelCheckpoint, EarlyStopping
from keras.models import Sequential, Model
from keras.layers import Dense, LSTM, Input, concatenate, Embedding, Reshape
import keras
import keras.backend as K
import tensorflow as tf
```

[19]:
```
sns.set_style('whitegrid')
np.random.seed(42)
K.clear_session()
```

1.3 Data

Data produced by the notebook build_dataset.

[20]:
```
data = pd.read_hdf('data.h5', 'returns_weekly')
data = data.drop([c for c in data.columns if str(c).startswith('year')], axis=1)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1167341 entries, 2009-01-01 to 2017-12-01
Data columns (total 66 columns):
ticker       1167341 non-null int64
1            1167341 non-null float64
2            1167341 non-null float64
3            1167341 non-null float64
4            1167341 non-null float64
5            1167341 non-null float64
6            1167341 non-null float64
7            1167341 non-null float64
8            1167341 non-null float64
9            1167341 non-null float64
10           1167341 non-null float64
11           1167341 non-null float64
12           1167341 non-null float64
13           1167341 non-null float64
14           1167341 non-null float64
15           1167341 non-null float64
16           1167341 non-null float64
17           1167341 non-null float64
18           1167341 non-null float64
19           1167341 non-null float64
20           1167341 non-null float64
21           1167341 non-null float64
22           1167341 non-null float64
23           1167341 non-null float64
24           1167341 non-null float64
25           1167341 non-null float64
26           1167341 non-null float64
27           1167341 non-null float64
28           1167341 non-null float64
29           1167341 non-null float64
30           1167341 non-null float64
31           1167341 non-null float64
32           1167341 non-null float64
33           1167341 non-null float64
34           1167341 non-null float64
35           1167341 non-null float64
36           1167341 non-null float64
```

```
37          1167341 non-null float64
38          1167341 non-null float64
39          1167341 non-null float64
40          1167341 non-null float64
41          1167341 non-null float64
42          1167341 non-null float64
43          1167341 non-null float64
44          1167341 non-null float64
45          1167341 non-null float64
46          1167341 non-null float64
47          1167341 non-null float64
48          1167341 non-null float64
49          1167341 non-null float64
50          1167341 non-null float64
51          1167341 non-null float64
52          1167341 non-null float64
label       1167341 non-null int64
month_1     1167341 non-null uint8
month_2     1167341 non-null uint8
month_3     1167341 non-null uint8
month_4     1167341 non-null uint8
month_5     1167341 non-null uint8
month_6     1167341 non-null uint8
month_7     1167341 non-null uint8
month_8     1167341 non-null uint8
month_9     1167341 non-null uint8
month_10    1167341 non-null uint8
month_11    1167341 non-null uint8
month_12    1167341 non-null uint8
dtypes: float64(52), int64(2), uint8(12)
memory usage: 503.2 MB
```

1.4 Train-test split

To respect the time series nature of the data, we set aside the data at the end of the sample as hold-out or test set. More specifically, we'll use the data for 2018.

[4]:
```
window_size=52
ticker = 1
months = 12
n_tickers = data.ticker.nunique()
```

[5]:
```
train_data = data[:'2016']
test_data = data['2017']
del data
```

For each train and test dataset, we generate a list with three input arrays containing the return series, the stock ticker (converted to integer values), and the month (as an integer), as shown here:

[6]:
```
X_train = [
    train_data.loc[:, list(range(1, window_size+1))].values.reshape(-1,␣
↪window_size , 1),
    train_data.ticker,
    train_data.filter(like='month')
]
y_train = train_data.label
[x.shape for x in X_train], y_train.shape
```

[6]:
```
([(1035424, 52, 1), (1035424,), (1035424, 12)], (1035424,))
```

[7]:
```
# keep the last year for testing
X_test = [
    test_data.loc[:, list(range(1, window_size+1))].values.reshape(-1,␣
↪window_size , 1),
    test_data.ticker,
    test_data.filter(like='month')
]
y_test = test_data.label
[x.shape for x in X_test], y_test.shape
```

[7]:
```
([(131917, 52, 1), (131917,), (131917, 12)], (131917,))
```

1.5 Custom Metric

[8]:
```
def roc_auc(y_true, y_pred):
    # any tensorflow metric
    value, update_op = tf.metrics.auc(y_true, y_pred)

    # find all variables created for this metric
    metric_vars = [i for i in tf.local_variables() if 'auc_roc' in i.name.
↪split('/')[1]]

    # Add metric variables to GLOBAL_VARIABLES collection.
    # They will be initialized for new session.
    for v in metric_vars:
        tf.add_to_collection(tf.GraphKeys.GLOBAL_VARIABLES, v)

    # force to update metric values
    with tf.control_dependencies([update_op]):
        value = tf.identity(value)
        return value
```

[9]:
```
# source: https://github.com/keras-team/keras/issues/3230
def auc(y_true, y_pred):
    ptas = tf.stack([binary_PTA(y_true, y_pred, k) for k in np.linspace(0, 1,␣
↪1000)], axis=0)
```

```python
    pfas = tf.stack([binary_PFA(y_true, y_pred, k) for k in np.linspace(0, 1,␣
↪1000)], axis=0)
    pfas = tf.concat([tf.ones((1,)), pfas], axis=0)
    binSizes = -(pfas[1:] - pfas[:-1])
    s = ptas * binSizes
    return K.sum(s, axis=0)


def binary_PFA(y_true, y_pred, threshold=K.variable(value=0.5)):
    """prob false alert for binary classifier"""
    y_pred = K.cast(y_pred >= threshold, 'float32')
    # N = total number of negative labels
    N = K.sum(1 - y_true)
    # FP = total number of false alerts, alerts from the negative class labels
    FP = K.sum(y_pred - y_pred * y_true)
    return FP / (N + 1)


def binary_PTA(y_true, y_pred, threshold=K.variable(value=0.5)):
    """prob true alerts for binary classifier"""
    y_pred = K.cast(y_pred >= threshold, 'float32')
    # P = total number of positive labels
    P = K.sum(y_true)
    # TP = total number of correct alerts, alerts from the positive class labels
    TP = K.sum(y_pred * y_true)
    return TP / (P + 1)
```

```
WARNING:tensorflow:From
/home/stefan/.pyenv/versions/miniconda3-latest/envs/ml4t/lib/python3.6/site-
packages/tensorflow/python/framework/op_def_library.py:263: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
```

1.6 Define the Model Architecture

The functional API of Keras makes it easy to design architectures with multiple inputs and outputs. This example illustrates a network with three inputs, as follows:

- A two stacked LSTM layers with 25 and 10 units respectively
- An embedding layer that learns a 10-dimensional real-valued representation of the equities
- A one-hot encoded representation of the month

This can be constructed using just a few lines - see e.g., - the general Keras documentation, - the LTSM documentation.

Make sure you are initializing your optimizer given the keras-recommended approach for RNNs

We begin by defining the three inputs with their respective shapes, as described here:

```
[ ]: returns = Input(shape=(window_size, n_features), name='Returns')
tickers = Input(shape=(1,), name='Tickers')
months = Input(shape=(12,), name='Months')
```

1.6.1 LSTM Layers

To define stacked LSTM layers, we set the `return_sequences` keyword to `True`. This ensures that the first layer produces an output that conforms to the expected three-dimensional input format. Note that we also use dropout regularization and how the functional API passes the tensor outputs from one layer to the subsequent layer:

```
[10]: lstm1_units = 25
lstm2_units = 10
n_features = 1
```

```
[ ]: lstm1 = LSTM(units=lstm1_units,
             input_shape=(window_size, n_features),
             name='LSTM1',
             dropout=.2,
             return_sequences=True)(returns)
lstm_model = LSTM(units=lstm2_units,
             dropout=.2,
             name='LSTM2')(lstm1)
```

1.6.2 Embedding Layer

The embedding layer requires the `input_dim` keyword, which defines how many embeddings the layer will learn, the `output_dim` keyword, which defines the size of the embedding, and the `input_length` keyword to set the number of elements passed to the layer (here only one ticker per sample).

To combine the embedding layer with the LSTM layer and the months input, we need to reshape (or flatten) it, as follows:

```
[11]: ticker_embedding = Embedding(input_dim=n_tickers,
                             output_dim=10,
                             input_length=1)(tickers)
ticker_embedding = Reshape(target_shape=(10,))(ticker_embedding)
```

```
WARNING:tensorflow:From
/home/stefan/.pyenv/versions/miniconda3-latest/envs/ml4t/lib/python3.6/site-
packages/keras/backend/tensorflow_backend.py:3445: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed
in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
```

1.6.3 Concatenate Model components

Now we can concatenate the three tensors and add fully-connected layers to learn a mapping from these learned time series, ticker, and month indicators to the outcome, a positive or negative return in the following week, as shown here:

[ ]:
```
merged = concatenate([lstm_model, ticker_embedding, months], name='Merged')
hidden_dense = Dense(10, name='FC1')(merged)
output = Dense(1, name='Output')(hidden_dense)

rnn = Model(inputs=[returns, tickers, months], outputs=output)
```

The summary lays out this slightly more sophisticated architecture with 29,371 parameters, as follows:

[12]:
```
rnn.summary()
```

```
------------------------------------------------------------------------------------------------
------------------
Layer (type)                    Output Shape         Param #     Connected to
================================================================================================
==================
Returns (InputLayer)            (None, 52, 1)        0

------------------------------------------------------------------------------------------------
------------------
Tickers (InputLayer)            (None, 1)            0

------------------------------------------------------------------------------------------------
------------------
LSTM1 (LSTM)                    (None, 52, 25)       2700        Returns[0][0]

------------------------------------------------------------------------------------------------
------------------
embedding_1 (Embedding)         (None, 1, 10)        24890       Tickers[0][0]

------------------------------------------------------------------------------------------------
------------------
LSTM2 (LSTM)                    (None, 10)           1440        LSTM1[0][0]

------------------------------------------------------------------------------------------------
------------------
reshape_1 (Reshape)             (None, 10)           0
embedding_1[0][0]

------------------------------------------------------------------------------------------------
------------------
Months (InputLayer)             (None, 12)           0

------------------------------------------------------------------------------------------------
------------------
Merged (Concatenate)            (None, 32)           0           LSTM2[0][0]
                                                                 reshape_1[0][0]
                                                                 Months[0][0]

------------------------------------------------------------------------------------------------
------------------
```

```
FC1 (Dense)                    (None, 10)          330         Merged[0][0]
--------------------------------------------------------------------------------
------------------
Output (Dense)                 (None, 1)           11          FC1[0][0]
================================================================================
==================
Total params: 29,371
Trainable params: 29,371
Non-trainable params: 0
--------------------------------------------------------------------------------
------------------
```

1.7 Train the Model

We compile the model to compute a custom auc metric as follows:

[13]:
```
rnn.compile(loss='binary_crossentropy',
            optimizer='adam',
           metrics=['accuracy', auc])
```

[14]:
```
rnn_path = 'models/quandl.lstm_months_{}_{}.weights.best.hdf5'.
 ↪format(lstm1_units, lstm2_units)
checkpointer = ModelCheckpoint(filepath=rnn_path,
                               monitor='val_loss',
                               save_best_only=True,
                               save_weights_only=True,
                               period=5)
```

[15]:
```
early_stopping = EarlyStopping(monitor='val_loss',
                               patience=5,
                               restore_best_weights=True)
```

[ ]:
```
result = rnn.fit(X_train,
                 y_train,
                 epochs=50,
                 batch_size=32,
                 validation_data=(X_test, y_test),
                 callbacks=[checkpointer, early_stopping],
                 verbose=1)
```

Training stops after 18 epochs, producing a test area under the curve (AUC) of 0.63 for the best model with 13 rounds of training (each of which takes around an hour on a single GPU).

[ ]:
```
loss_history = pd.DataFrame(result.history)
loss_history
```

[ ]:
```
def which_metric(m):
    return m.split('_')[-1]
```

```
[ ]: loss_history.groupby(which_metric, axis=1).plot(figsize=(14, 6));
```

1.8 Evaluate model performance

```
[ ]: test_predict = pd.Series(rnn.predict(X_test).squeeze(), index=y_test.index)
```

```
[ ]: roc_auc_score(y_score=test_predict, y_true=y_test)
```

```
[ ]: rnn.load_weights(rnn_path)
```

```
[ ]: test_predict = pd.Series(rnn.predict(X_test).squeeze(), index=y_test.index)
```

```
[ ]: roc_auc_score(y_score=test_predict, y_true=y_test)
```

```
[ ]: score
```

```
[ ]: predictions = (test_predict.to_frame('prediction').assign(data='test')
                    .append(train_predict.to_frame('prediction').
      ↪assign(data='train')))
     predictions.info()
```

```
[ ]: results = sp500_scaled.join(predictions).dropna()
     results.info()
```

```
[ ]: corr = {}
     for run, df in results.groupby('data'):
         corr[run] = df.SP500.corr(df.prediction)
```

```
[ ]: sp500_scaled['Train Prediction'] = pd.Series(train_predict.squeeze(),␣
      ↪index=y_train.index)
     sp500_scaled['Test Prediction'] = pd.Series(test_predict.squeeze(),␣
      ↪index=y_test.index)
```

```
[ ]: training_error = np.sqrt(rnn.evaluate(X_train, y_train, verbose=0))
     testing_error = np.sqrt(rnn.evaluate(X_test, y_test, verbose=0))
     print('Training Error: {:.4f} | Test Error: {:.4f}'.format(training_error,␣
      ↪testing_error))
```

```
[ ]: sns.set_style('whitegrid')
```