HAR_LSTM

July 31, 2019

In [1]:
```
# Importing Libraries
```

In [1]:
```
import pandas as pd
import numpy as np
```

In [129]:
```
# Activities are the class labels
# It is a 6 class classification
ACTIVITIES = {
    0: 'WALKING',
    1: 'WALKING_UPSTAIRS',
    2: 'WALKING_DOWNSTAIRS',
    3: 'SITTING',
    4: 'STANDING',
    5: 'LAYING',
}

# Utility function to print the confusion matrix
def confusion_matrixn_matrix(Y_true, Y_pred):
    Y_true = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_true, axis=1)])
    Y_pred = pd.Series([ACTIVITIES[y] for y in np.argmax(Y_pred, axis=1)])

    return pd.crosstab(Y_true, Y_pred, rownames=['True'], colnames=['Pred'])
```

0.0.1 Data

In [3]:
```
# Data directory
DATADIR = 'UCI_HAR_Dataset'
```

In [4]:
```
# Raw data signals
# Signals are from Accelerometer and Gyroscope
# The signals are in x,y,z directions
# Sensor signals are filtered to have only body acceleration
# excluding the acceleration due to gravity
# Triaxial acceleration from the accelerometer is total acceleration
SIGNALS = [
    "body_acc_x",
    "body_acc_y",
    "body_acc_z",
```

```
            "body_gyro_x",
            "body_gyro_y",
            "body_gyro_z",
            "total_acc_x",
            "total_acc_y",
            "total_acc_z"
        ]
```

In [5]:
```python
# Utility function to read the data from csv file
def _read_csv(filename):
    return pd.read_csv(filename, delim_whitespace=True, header=None)

# Utility function to load the load
def load_signals(subset):
    signals_data = []

    for signal in SIGNALS:
        filename = f'UCI_HAR_Dataset/{subset}/Inertial Signals/{signal}_{subset}.txt'
        signals_data.append(
            _read_csv(filename).as_matrix()
        )

    # Transpose is used to change the dimensionality of the output,
    # aggregating the signals by combination of sample/timestep.
    # Resultant shape is (7352 train/2947 test samples, 128 timesteps, 9 signals)
    return np.transpose(signals_data, (1, 2, 0))
```

In [6]:
```python
def load_y(subset):
    """
    The objective that we are trying to predict is a integer, from 1 to 6,
    that represents a human activity. We return a binary representation of
    every sample objective as a 6 bits vector using One Hot Encoding
    (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html)
    """
    filename = f'UCI_HAR_Dataset/{subset}/y_{subset}.txt'
    y = _read_csv(filename)[0]

    return pd.get_dummies(y).as_matrix()
```

In [7]:
```python
def load_data():
    """
    Obtain the dataset from multiple files.
    Returns: X_train, X_test, y_train, y_test
    """
    X_train, X_test = load_signals('train'), load_signals('test')
    y_train, y_test = load_y('train'), load_y('test')

    return X_train, X_test, y_train, y_test
```

```
In [8]: # Importing tensorflow
        np.random.seed(42)
        import tensorflow as tf
        tf.set_random_seed(42)

C:\Users\hp\Anaconda3\lib\site-packages\h5py\__init__.py:36: FutureWarning: Conversion of the s
  from ._conv import register_converters as _register_converters

In [9]: # Configuring a session
        session_conf = tf.ConfigProto(
            intra_op_parallelism_threads=1,
            inter_op_parallelism_threads=1
        )

In [10]: # Import Keras
         from keras import backend as K
         sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
         K.set_session(sess)

Using TensorFlow backend.

In [11]: # Importing libraries
         from keras.models import Sequential
         from keras.layers import LSTM
         from keras.layers.core import Dense, Dropout

In [79]: # Initializing parameters
         """
         epochs = 30
         batch_size = 16                  #this configuration is not improving our accuracy
         n_hidden = 64                    # accuracy is around 90.
         drop_out = 0.5


         epochs = 30
         batch_size = 16                   # this is also not working well for us; accuracy 91
         n_hidden = 64
         drop_out = 0.25


         epochs = 30
         batch_size = 32                     #this configuration seems to me as it's doing ove
         n_hidden = 64                        accuracy increase to a certain point and then it
         drop_out = 0.25
         """
```

```
epochs = 30                  # using this configuration we 92.81% almost got 93% accu
batch_size = 32              # after 27 epochs we are getting higher accuracy 93.72 wi
n_hidden = 128               # almost equal to 94%
drop_out = 0.5
```

In [110]:
```
# Utility function to count the number of classes
def _count_classes(y):
    return len(set([tuple(category) for category in y]))
```

In [111]:
```
# Loading the train and test data
X_train, X_test, Y_train, Y_test = load_data()
```

In [112]:
```
timesteps = len(X_train[0])
input_dim = len(X_train[0][0])
n_classes = _count_classes(Y_train)

print(timesteps)
print(input_dim)
print(len(X_train))
```

```
128
9
7352
```

- Defining the Architecture of LSTM

In [83]:
```
import warnings
warnings.filterwarnings("ignore")
# Initiliazing the sequential model
model = Sequential()
# Configuring the parameters
model.add(LSTM(n_hidden, input_shape=(timesteps, input_dim)))
# Adding a dropout layer
model.add(Dropout(drop_out))
# Adding a dense output layer with sigmoid activation
model.add(Dense(n_classes, activation='sigmoid'))
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_13 (LSTM)               (None, 128)               70656
_________________________________________________________________
dropout_13 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_13 (Dense)             (None, 6)                 774
=================================================================
Total params: 71,430
```

```
Trainable params: 71,430
Non-trainable params: 0
_________________________________________________________________
```

In [101]:
```
import warnings
warnings.filterwarnings("ignore")
# Compiling the model
model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

In [85]:
```
import warnings
warnings.filterwarnings("ignore")

from datetime import datetime
start = datetime.now()

# Training the model
model.fit(X_train,
          Y_train,
          batch_size=batch_size,
          validation_data=(X_test, Y_test),
          epochs=epochs)

print("Time taken : ", datetime.now() - start)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 46s 6ms/step - loss: 1.3462 - acc: 0.4006 - val_lo
Epoch 2/30
7352/7352 [==============================] - 44s 6ms/step - loss: 1.2504 - acc: 0.4373 - val_lo
Epoch 3/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.8766 - acc: 0.5884 - val_lo
Epoch 4/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.7814 - acc: 0.6404 - val_lo
Epoch 5/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.6919 - acc: 0.6829 - val_lo
Epoch 6/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.5751 - acc: 0.7703 - val_lo
Epoch 7/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.4046 - acc: 0.8587 - val_lo
Epoch 8/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.2885 - acc: 0.8987 - val_lo
Epoch 9/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.2360 - acc: 0.9146 - val_lo
Epoch 10/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.2128 - acc: 0.9233 - val_lo
```

```
Epoch 11/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1994 - acc: 0.9301 - val_lo
Epoch 12/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1666 - acc: 0.9404 - val_lo
Epoch 13/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1761 - acc: 0.9369 - val_lo
Epoch 14/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1522 - acc: 0.9456 - val_lo
Epoch 15/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1502 - acc: 0.9434 - val_lo
Epoch 16/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1492 - acc: 0.9452 - val_lo
Epoch 17/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1485 - acc: 0.9387 - val_lo
Epoch 18/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1575 - acc: 0.9433 - val_lo
Epoch 19/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1465 - acc: 0.9442 - val_lo
Epoch 20/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1386 - acc: 0.9495 - val_lo
Epoch 21/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1373 - acc: 0.9505 - val_lo
Epoch 22/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1227 - acc: 0.9516 - val_lo
Epoch 23/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1495 - acc: 0.9414 - val_lo
Epoch 24/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1212 - acc: 0.9509 - val_lo
Epoch 25/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1266 - acc: 0.9520 - val_lo
Epoch 26/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1345 - acc: 0.9460 - val_lo
Epoch 27/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1306 - acc: 0.9478 - val_lo
Epoch 28/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1619 - acc: 0.9482 - val_lo
Epoch 29/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1267 - acc: 0.9520 - val_lo
Epoch 30/30
7352/7352 [==============================] - 44s 6ms/step - loss: 0.1284 - acc: 0.9504 - val_lo
Time taken :  0:22:06.036482
```

In [ ]:
```
# Confusion Matrix
print(confusion_matrix(Y_test, model.predict(X_test)))
```

In [22]:
```
score = model.evaluate(X_test, Y_test)
```

```
2947/2947 [==============================] - 1s 461us/step
```

```
In [23]: score
```

```
Out[23]: [0.39372028857254565, 0.9019341703427214]
```

- With a simple 2 layer architecture we got 90.09% accuracy and a loss of 0.30
- We can further imporve the performace with Hyperparameter tuning

0.1 LSTM model with 2 layers

```
In [121]: """                          this configuration gives accuracy of 90.84
          epochs_m2 = 30
          batch_size_m2= 32
          n_hidden_layer1 = 128
          n_hidden_layer2 =32
          drop_out_1 = 0.2
          drop_out_2 = 0.5
          """

          epochs_m2 = 30
          batch_size_m2= 32
          n_hidden_layer1 = 128
          n_hidden_layer2 =64
          drop_out_1 = 0.2
          drop_out_2 = 0.5
```

```
In [122]: # Initiliazing the sequential model
          model2 = Sequential()
          # Configuring the parameters
          model2.add(LSTM(n_hidden_layer1, return_sequences=True, input_shape=(timesteps, inpu
          # Adding a dropout layer
          model2.add(Dropout(drop_out_1))

          model2.add(LSTM(n_hidden_layer2))
          # Adding a dropout layer
          model2.add(Dropout(drop_out_2))
          # Adding a dense output layer with sigmoid activation
          model2.add(Dense(n_classes, activation='sigmoid'))
          model2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_20 (LSTM)               (None, 128, 128)          70656
_________________________________________________________________
dropout_20 (Dropout)         (None, 128, 128)          0
_________________________________________________________________
lstm_21 (LSTM)               (None, 64)                49408
```

```
----------------------------------------------------------------
dropout_21 (Dropout)         (None, 64)                0
----------------------------------------------------------------
dense_18 (Dense)             (None, 6)                 390
================================================================
Total params: 120,454
Trainable params: 120,454
Non-trainable params: 0
----------------------------------------------------------------
```

In [123]:
```python
# Compiling the model
model2.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

In [124]:
```python
import warnings
warnings.filterwarnings("ignore")

from datetime import datetime
start = datetime.now()

# Training the model
model2.fit(X_train,
          Y_train,
          batch_size=batch_size_m2,
          validation_data=(X_test, Y_test),
          epochs=epochs_m2)

print("Time taken : ", datetime.now() - start)
```

```
Train on 7352 samples, validate on 2947 samples
Epoch 1/30
7352/7352 [==============================] - 94s 13ms/step - loss: 1.1846 - acc: 0.4838 - val_
Epoch 2/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.8015 - acc: 0.6364 - val_
Epoch 3/30
7352/7352 [==============================] - 92s 12ms/step - loss: 0.6878 - acc: 0.7087 - val_
Epoch 4/30
7352/7352 [==============================] - 92s 12ms/step - loss: 0.6659 - acc: 0.7042 - val_
Epoch 5/30
7352/7352 [==============================] - 92s 12ms/step - loss: 0.5136 - acc: 0.7734 - val_
Epoch 6/30
7352/7352 [==============================] - 92s 13ms/step - loss: 0.3751 - acc: 0.8637 - val_
Epoch 7/30
7352/7352 [==============================] - 92s 12ms/step - loss: 0.2722 - acc: 0.9154 - val_
Epoch 8/30
7352/7352 [==============================] - 92s 13ms/step - loss: 0.2314 - acc: 0.9242 - val_
```

```
Epoch 9/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1961 - acc: 0.9348 - val_l
Epoch 10/30
7352/7352 [==============================] - 105s 14ms/step - loss: 0.1734 - acc: 0.9389 - val_
Epoch 11/30
7352/7352 [==============================] - 98s 13ms/step - loss: 0.1789 - acc: 0.9403 - val_l
Epoch 12/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1568 - acc: 0.9416 - val_l
Epoch 13/30
7352/7352 [==============================] - 99s 14ms/step - loss: 0.1686 - acc: 0.9425 - val_l
Epoch 14/30
7352/7352 [==============================] - 96s 13ms/step - loss: 0.1430 - acc: 0.9475 - val_l
Epoch 15/30
7352/7352 [==============================] - 92s 13ms/step - loss: 0.1774 - acc: 0.9434 - val_l
Epoch 16/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.1591 - acc: 0.9489 - val_l
Epoch 17/30
7352/7352 [==============================] - 98s 13ms/step - loss: 0.1377 - acc: 0.9494 - val_l
Epoch 18/30
7352/7352 [==============================] - 99s 13ms/step - loss: 0.1294 - acc: 0.9513 - val_l
Epoch 19/30
7352/7352 [==============================] - 98s 13ms/step - loss: 0.2095 - acc: 0.9359 - val_l
Epoch 20/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1286 - acc: 0.9479 - val_l
Epoch 21/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.1432 - acc: 0.9463 - val_l
Epoch 22/30
7352/7352 [==============================] - 92s 13ms/step - loss: 0.1234 - acc: 0.9531 - val_l
Epoch 23/30
7352/7352 [==============================] - 94s 13ms/step - loss: 0.1266 - acc: 0.9504 - val_l
Epoch 24/30
7352/7352 [==============================] - 92s 13ms/step - loss: 0.1574 - acc: 0.9478 - val_l
Epoch 25/30
7352/7352 [==============================] - 92s 13ms/step - loss: 0.1226 - acc: 0.9524 - val_l
Epoch 26/30
7352/7352 [==============================] - 92s 13ms/step - loss: 0.1339 - acc: 0.9456 - val_l
Epoch 27/30
7352/7352 [==============================] - 92s 13ms/step - loss: 0.1276 - acc: 0.9532 - val_l
Epoch 28/30
7352/7352 [==============================] - 92s 13ms/step - loss: 0.1432 - acc: 0.9487 - val_l
Epoch 29/30
7352/7352 [==============================] - 93s 13ms/step - loss: 0.1335 - acc: 0.9489 - val_l
Epoch 30/30
7352/7352 [==============================] - 92s 13ms/step - loss: 0.1427 - acc: 0.9434 - val_l
Time taken :  0:47:09.180834
```

```
In [134]: score = model2.evaluate(X_test, Y_test)
```

```
2947/2947 [==============================] - 7s 2ms/step
```

In [135]:
```python
score
```

Out[135]:
```
[0.31484989217882003, 0.9216152019002375]
```

0.2 Conclusion

In classic machine learning models we got accuracy of around 96% and here we also tried to get the accuracy close to 96% by * tuning the number of lstm units * experimenting with drop out value * and by also adding a secnd hidden layer.

For our first model with single hidden layer I tried various configurations :

In [142]:
```python
from prettytable import PrettyTable
##################################### pretty table for model 1 with 1 lstm layer ###
number       = [1, 2, 3, 4]
epochs       = [30, 30, 30, 30]
batch_size   = [16, 16, 32, 32]
n_hidden     = [64, 64, 64, 128]
drop_out     = [0.25, 0.25, 0.25, 0.5]
accuracy     = [90.6, 91.45, 90.84, 92.81]

# Initializing prettytable # Adding columns
ptable = PrettyTable()
ptable.add_column("Configuration",number)
ptable.add_column("Epochs", epochs)
ptable.add_column("Batch Size",batch_size)
ptable.add_column("Hidden Layer",n_hidden)
ptable.add_column("Dropout",drop_out)
ptable.add_column("Accuracy",accuracy)
#Printing the Table
print(ptable)
```

```
+---------------+--------+------------+--------------+---------+----------+
| Configuration | Epochs | Batch Size | Hidden Layer | Dropout | Accuracy |
+---------------+--------+------------+--------------+---------+----------+
|       1       |   30   |     16     |      64      |   0.25  |   90.6   |
|       2       |   30   |     16     |      64      |   0.25  |  91.45   |
|       3       |   30   |     32     |      64      |   0.25  |  90.84   |
|       4       |   30   |     32     |     128      |   0.5   |  92.81   |
+---------------+--------+------------+--------------+---------+----------+
```

In [144]:
```python
################################# pretty table for model 2 with 2 lstm layesr #######
number          = [1, 2]
epochs          = [30, 30]
batch_size      = [32, 32]
```

```
n_hidden_layer1 = [128, 128]
n_hidden_layer2 = [32, 64]
drop_out_1      = [0.2, 0.5]
drop_out_2      = [0.2, 0.5]
accuracy     = [90.84, 92.16]

# Initializing prettytable # Adding columns
ptable = PrettyTable()
ptable.add_column("Configuration",number)
ptable.add_column("Epochs", epochs)
ptable.add_column("Batch Size",batch_size)
ptable.add_column("Hidden Layer",n_hidden_layer1)
ptable.add_column("Hidden Layer",n_hidden_layer2)
ptable.add_column("Dropout",drop_out_1)
ptable.add_column("Dropout",drop_out_2)
ptable.add_column("Accuracy",accuracy)
#Printing the Table
print(ptable)
```

```
+---------------+--------+------------+--------------+--------------+---------+---------+-----
| Configuration | Epochs | Batch Size | Hidden Layer | Hidden Layer | Dropout | Dropout | Accu
+---------------+--------+------------+--------------+--------------+---------+---------+-----
|       1       |   30   |     32     |     128      |      32      |   0.2   |   0.2   |  90.8
|       2       |   30   |     32     |     128      |      64      |   0.5   |   0.5   |  92.1
+---------------+--------+------------+--------------+--------------+---------+---------+-----
```

- Note: One observation which I made while training our lstm models is that after training the model more than one time we get differnt values of accuracy i.e for same configuration we might get different accuracy values every time we train it over again.

So the results may improve if we train these models again with same configuration. The final configurations for each model i.e model 1 and model 2 are good and we can get accuracy closer to 96%.