# lstm

May 11, 2024

## 1 Importing Required Libraries

[1]:
```
from numpy import mean
from numpy import std
from numpy import dstack
from pandas import read_csv
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv1D, LSTM, MaxPooling1D
from keras.utils import to_categorical
from matplotlib import pyplot
```

```
2024-05-11 02:54:20.306517: I tensorflow/core/util/port.cc:113] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-05-11 02:54:20.306967: I external/local_tsl/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2024-05-11 02:54:20.309943: I external/local_tsl/tsl/cuda/cudart_stub.cc:32]
Could not find cuda drivers on your machine, GPU will not be used.
2024-05-11 02:54:20.345944: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX512F AVX512_VNNI FMA, in other
operations, rebuild TensorFlow with the appropriate compiler flags.
2024-05-11 02:54:21.228270: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

## 2 Loaing the File

[2]:
```
# load a single file as a numpy array
def load_file(filepath):
    dataframe = read_csv(filepath, header=None, delim_whitespace=True)
    return dataframe.values

# load a list of files and return as a 3d numpy array
def load_group(filenames, prefix=''):
```

```
    loaded = list()
    for name in filenames:
        data = load_file(prefix + name)
        loaded.append(data)
    # stack group so that features are the 3rd dimension
    loaded = dstack(loaded)
    return loaded

# load a dataset group, such as train or test
def load_dataset_group(group, prefix=''):
    filepath = prefix + group + '/Inertial Signals/'
    # load all 9 files as a single array
    filenames = list()
    # total acceleration
    filenames += ['total_acc_x_'+group+'.txt', 'total_acc_y_'+group+'.txt',␣
 ↪'total_acc_z_'+group+'.txt']
    # body acceleration
    filenames += ['body_acc_x_'+group+'.txt', 'body_acc_y_'+group+'.txt',␣
 ↪'body_acc_z_'+group+'.txt']
    # body gyroscope
    filenames += ['body_gyro_x_'+group+'.txt', 'body_gyro_y_'+group+'.txt',␣
 ↪'body_gyro_z_'+group+'.txt']
    # load input data
    X = load_group(filenames, filepath)
    # load class output
    y = load_file(prefix + group + '/y_'+group+'.txt')
    return X, y

# load the dataset, returns train and test X and y elements
def load_dataset(prefix=''):
    # load all train
    trainX, trainy = load_dataset_group('train', prefix + 'UCI HAR Dataset/')
    print(trainX.shape, trainy.shape)
    # load all test
    testX, testy = load_dataset_group('test', prefix + 'UCI HAR Dataset/')
    print(testX.shape, testy.shape)
    # zero-offset class values
    trainy = trainy - 1
    testy = testy - 1
    # one hot encode y
    trainy = to_categorical(trainy)
    testy = to_categorical(testy)
    print(trainX.shape, trainy.shape, testX.shape, testy.shape)
    return trainX, trainy, testX, testy
```

## 3 Training the Model with Sequential Model

[3]:
```
# # fit and evaluate an LSTM model (do 10-fold cross validation)
# def evaluate_model(trainX, trainy, testX, testy):
#     # define model
#     verbose, epochs, batch_size = 0, 15, 64
#     n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2],␣
 ↪trainy.shape[1]
#     print(n_timesteps, n_features, n_outputs)
#     model = Sequential()
#     model.add(Conv1D(filters=64, kernel_size=3, activation='relu',␣
 ↪input_shape=(n_timesteps,n_features)))
#     model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
#     model.add(Dropout(0.5))
#     model.add(MaxPooling1D(pool_size=2))
#     model.add(Flatten())
#     model.add(Dense(100, activation='relu'))
#     model.add(Dense(n_outputs, activation='softmax'))
#     model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
#     # fit network
#     model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,␣
 ↪verbose=verbose)
#     # evaluate model
#     _, accuracy = model.evaluate(testX, testy, batch_size=batch_size,␣
 ↪verbose=0)
#     return accuracy

from keras.layers import InputLayer

def evaluate_model(trainX, trainy, testX, testy):
    # define model
    verbose, epochs, batch_size = 0, 15, 64
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2],␣
 ↪trainy.shape[1]
    print(n_timesteps, n_features, n_outputs)
    model = Sequential()
    model.add(InputLayer(input_shape=(n_timesteps, n_features)))
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu'))
    model.add(Dropout(0.5))
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())
    model.add(Dense(100, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])
```

```
    # fit network
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,␣
 ↪verbose=verbose)
    # evaluate model
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size, verbose=0)
    return accuracy
```

## 4 Training the Model with LSTM

[4]:
```
def evaluate_lstm_model(trainX, trainy, testX, testy, n_neurons):
    # define model
    verbose, epochs, batch_size = 0, 15, 32
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2],␣
 ↪trainy.shape[1]

    model = Sequential()
    model.add(LSTM(units=n_neurons, input_shape=(n_timesteps, n_features)))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])

    # fit network
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,␣
 ↪verbose=verbose)

    # evaluate model
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size, verbose=0)
    return accuracy
```

## 5 Summarizing the Results Obtained

[5]:
```
# summarize scores
def summarize_results(scores):
    print(scores)
    m, s = mean(scores), std(scores)
    print('Accuracy: %.3f%% (+/-%.3f)' % (m, s))
```

[6]:
```
# run an experiment
def run_experiment(repeats=10, n_neurons=100):
    # load data
    trainX, trainy, testX, testy = load_dataset()
    # repeat experiment
    scores = list()
    for r in range(repeats):
```

```
        score = evaluate_lstm_model(trainX, trainy, testX, testy, n_neurons)
        score = score * 100.0
        print('>#%d: %.3f' % (r+1, score))
        scores.append(score)
    # summarize results
    summarize_results(scores)
```

[7]:
```
# run the experiment
run_experiment()
```

```
(7352, 128, 9) (7352, 1)
(2947, 128, 9) (2947, 1)
(7352, 128, 9) (7352, 6) (2947, 128, 9) (2947, 6)
```

```
/home/uniqueusman/TechUp/myenv/lib/python3.11/site-
packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
>#1: 91.822
>#2: 90.499
>#3: 90.092
>#4: 89.006
>#5: 91.415
>#6: 91.551
>#7: 91.177
>#8: 89.718
>#9: 92.026
>#10: 90.431
[91.82218909263611, 90.49881100654602, 90.09161591529846, 89.00576829910278,
91.41499996185303, 91.55073165893555, 91.17746949195862, 89.71835970878601,
92.02578663825989, 90.43094515800476]
Accuracy: 90.774% (+/-0.935)
```

# 6 Testing Different Number of Neurons

### 6.0.1 Number of Neurons = 100

[8]:
```
def run_experiment(repeats=5, n_neurons=100):
    # load data
    trainX, trainy, testX, testy = load_dataset()
    # repeat experiment
    scores = list()
    for r in range(repeats):
        score = evaluate_lstm_model(trainX, trainy, testX, testy, n_neurons)
        score = score * 100.0
        print('>#%d: %.3f' % (r+1, score))
```

```
        scores.append(score)
    # summarize results
    summarize_results(scores)

run_experiment()
```

```
(7352, 128, 9) (7352, 1)
(2947, 128, 9) (2947, 1)
(7352, 128, 9) (7352, 6) (2947, 128, 9) (2947, 6)
```

```
/home/uniqueusman/TechUp/myenv/lib/python3.11/site-
packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
>#1: 89.786
>#2: 91.686
>#3: 89.922
>#4: 91.211
>#5: 90.668
[89.78622555732727, 91.68646335601807, 89.92195725440979, 91.21140241622925,
90.66847562789917]
Accuracy: 90.655% (+/-0.730)
```

### 6.0.2 Number of Neurons = 10

[9]:
```
def run_experiment(repeats=5, n_neurons=10):
    # load data
    trainX, trainy, testX, testy = load_dataset()
    # repeat experiment
    scores = list()
    for r in range(repeats):
        score = evaluate_lstm_model(trainX, trainy, testX, testy, n_neurons)
        score = score * 100.0
        print('>#%d: %.3f' % (r+1, score))
        scores.append(score)
    # summarize results
    summarize_results(scores)

run_experiment()
```

```
(7352, 128, 9) (7352, 1)
(2947, 128, 9) (2947, 1)
(7352, 128, 9) (7352, 6) (2947, 128, 9) (2947, 6)
```

```
/home/uniqueusman/TechUp/myenv/lib/python3.11/site-
packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
```

```
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
>#1: 87.004
>#2: 87.615
>#3: 88.022
>#4: 87.852
>#5: 84.628
[87.0037317276001, 87.61452436447144, 88.021719455719, 87.85205483436584,
84.62843298912048]
Accuracy: 87.024% (+/-1.247)
```

### 6.0.3 Number of Neurons = 200

[10]:
```
def run_experiment(repeats=5, n_neurons=200):
    # load data
    trainX, trainy, testX, testy = load_dataset()
    # repeat experiment
    scores = list()
    for r in range(repeats):
        score = evaluate_lstm_model(trainX, trainy, testX, testy, n_neurons)
        score = score * 100.0
        print('>#%d: %.3f' % (r+1, score))
        scores.append(score)
    # summarize results
    summarize_results(scores)

run_experiment()
```

```
(7352, 128, 9) (7352, 1)
(2947, 128, 9) (2947, 1)
(7352, 128, 9) (7352, 6) (2947, 128, 9) (2947, 6)
```

```
/home/uniqueusman/TechUp/myenv/lib/python3.11/site-
packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

```
>#1: 90.431
>#2: 89.820
>#3: 90.092
>#4: 92.603
>#5: 90.329
[90.43094515800476, 89.8201584815979, 90.09161591529846, 92.6026463508606,
90.32914638519287]
Accuracy: 90.655% (+/-0.996)
```

# 7 Subjective Questions

**1. Why was this problem suitable for LSTM and nor just any other machine learning algorithm?** - This problem is suitable for LSTM (Long Short-Term Memory) and not just any other machine learning algorithm due to the temporal nature of the data. The data involves sequences of activities over time, and LSTM is particularly well-suited for handling sequences and capturing long-term dependencies. In this case, the LSTM can learn and remember patterns in the sequence of movement data, which is crucial for recognizing different activities that unfold over time. Traditional machine learning algorithms are not as effective in capturing temporal aspects of the data.

**2. Is a many-to-many model a real-world scenario for this problem or a many-to-one model?** - A many-to-one model is more appropriate. The input sequences (sensor data over time) are mapped to a single output (activity label). The goal is to classify the entire sequence of movement data into one of the six activities. A many-to-many model might be more suitable if the task involved predicting a sequence of labels for each time step, which is not the case here.

**3**. In LSTMs, "Short Memory" refers to remembering information over short sequences, while "Long Memory" refers to remembering information over longer sequences.

**4**. An LSTM node consists of the input gate (controls input flow), the forget gate (controls forgetting old information), and the output gate (controls output based on input and memory).

**5**. LSTMs address the vanishing gradient problem in RNNs by using memory cells and gating mechanisms, enabling better long-term dependency modeling.

[ ]: