Model LSTM evaluation

Implemented by :

Vo Minh Chanh - 1910057

Nguyen Lap Quan - 191931

Executive Summary:
This report is used for the purpose of a university thesis, aimed at evaluating the performance of the LSTM model.

Introduction:
The machine learning model used is the LSTM model, which is employed for the purpose of predicting gestures and actions collected from devices integrated with flex sensors and angular acceleration sensors.

The model is applied to a dataset collected from users; therefore, it is necessary to adjust the model parameters accordingly to evaluate the accuracy of each architecture.

Data Description:

The data is collected from 15 gestures, divided into two separate files: a training dataset and a testing dataset.
Each row of data is recorded for 2.4 seconds for each gesture.
Each row of data includes 240*7 columns of data, where each 240 continuous data points horizontally represent data from one sensor. For example, for the i-th sensor, its data ranges from position 240*i to 240*i + 239. There are a total of 7 sensors.

LSTM Model Used:

- The LSTM model is employed for predicting gestures and actions collected from devices integrated with flex sensors and angular acceleration sensors.

- The model is applied to a dataset collected from users; therefore, it is necessary to adjust the model parameters accordingly to evaluate the accuracy of each architecture.

Training environment:

```
name: tf-gpu
channels:
  - conda-forge
  - defaults
dependencies:
  - asttokens=2.0.5=pyhd3eb1b0_0
  - backcall=0.2.0=pyhd3eb1b0_0
  - ca-certificates=2024.3.11=haa95532_0
  - colorama=0.4.6=py39haa95532_0
  - comm=0.2.1=py39haa95532_0
  - cudatoolkit=11.2.2=h933977f_10
  - cudnn=8.1.0.77=h3e0f4f4_0
  - debugpy=1.6.7=py39hd77b12b_0
  - decorator=5.1.1=pyhd3eb1b0_0
  - exceptiongroup=1.2.0=py39haa95532_0
  - executing=0.8.3=pyhd3eb1b0_0
  - importlib-metadata=7.0.1=py39haa95532_0
  - importlib_metadata=7.0.1=hd3eb1b0_0
  - ipykernel=6.28.0=py39haa95532_0
  - ipython=8.15.0=py39haa95532_0
  - jedi=0.18.1=py39haa95532_1
  - jupyter_client=8.6.0=py39haa95532_0
  - jupyter_core=5.5.0=py39haa95532_0
  - libsodium=1.0.18=h62dcd97_0
  - matplotlib-inline=0.1.6=py39haa95532_0
  - nest-asyncio=1.6.0=py39haa95532_0
  - openssl=3.0.13=h2bbff1b_2
  - packaging=23.2=py39haa95532_0
  - parso=0.8.3=pyhd3eb1b0_0
  - pickleshare=0.7.5=pyhd3eb1b0_1003
  - platformdirs=3.10.0=py39haa95532_0
  - prompt-toolkit=3.0.43=py39haa95532_0
  - psutil=5.9.0=py39h2bbff1b_0
  - pure_eval=0.2.2=pyhd3eb1b0_0
  - pygments=2.15.1=py39haa95532_1
  - python=3.9.19=h1aa4202_1
  - pywin32=305=py39h2bbff1b_0
  - pyzmq=25.1.2=py39hd77b12b_0
  - setuptools=69.5.1=py39haa95532_0
  - six=1.16.0=pyhd3eb1b0_1
  - sqlite=3.45.3=h2bbff1b_0
  - stack_data=0.2.0=pyhd3eb1b0_0
  - tornado=6.3.3=py39h2bbff1b_0
  - traitlets=5.7.1=py39haa95532_0
  - typing_extensions=4.11.0=py39haa95532_0
  - vc=14.2=h2eaa2aa_1
  - vs2015_runtime=14.29.30133=h43f2093_3
  - wcwidth=0.2.5=pyhd3eb1b0_0
  - wheel=0.43.0=py39haa95532_0
  - zeromq=4.3.5=hd77b12b_0
  - zipp=3.17.0=py39haa95532_0
  - pip:
      - absl-py==2.1.0
      - astunparse==1.6.3
      - cachetools==5.3.2
      - certifi==2023.11.17
      - charset-normalizer==3.3.2
      - contourpy==1.2.0
      - cycler==0.12.1
      - flatbuffers==23.5.26
      - fonttools==4.47.2
      - future==0.18.3
      - gast==0.4.0
      - google-auth==2.26.2
      - google-auth-oauthlib==0.4.6
      - google-pasta==0.2.0
      - grpcio==1.60.0
      - h5py==3.10.0
      - idna==3.6
      - importlib-resources==6.1.1
      - iso8601==2.1.0
      - joblib==1.3.2
      - keras==2.10.0
      - keras-preprocessing==1.1.2
      - kiwisolver==1.4.5
      - libclang==16.0.6
      - markdown==3.5.2
      - markupsafe==2.1.3
      - matplotlib==3.8.2
      - numpy==1.26.3
      - oauthlib==3.2.2
      - opt-einsum==3.3.0
      - pandas==2.1.4
      - pillow==10.2.0
      - pip==23.3.2
      - protobuf==3.19.6
      - pyasn1==0.5.1
      - pyasn1-modules==0.3.0
      - pyparsing==3.1.1
      - python-dateutil==2.8.2
      - pytz==2023.3.post1
      - pyyaml==6.0.1
      - requests==2.31.0
      - requests-oauthlib==1.3.1
      - rsa==4.9
      - scikit-learn==1.4.0
      - scipy==1.11.4
      - serial==0.0.97
      - tensorboard==2.10.1
      - tensorboard-data-server==0.6.1
      - tensorboard-plugin-wit==1.8.1
      - tensorflow==2.10.1
      - tensorflow-estimator==2.10.0
      - tensorflow-io-gcs-filesystem==0.31.0
      - termcolor==2.4.0
      - threadpoolctl==3.2.0
      - typing-extensions==4.9.0
      - tzdata==2023.4
      - urllib3==2.1.0
      - werkzeug==3.0.1
      - wrapt==1.16.0
```

Hardware: NVIDIA GeForce GTX 1660 Supper

CASE 1:

Default architecture of the model*:

```
def evaluate_model(trainX, trainy, testX, testy,r = 0):
    # define model
    path = './model/train'+str(r)+'.h5'
    verbose, epochs, batch_size = 2, 15, 4000
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2],
trainy.shape[1]
    print("time steps = ", trainX.shape[1])
    #define model
    model = Sequential()
    model.add(LSTM(units = 100, input_shape = (n_timesteps, n_features)))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(units = 100, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,
verbose=verbose)
    model.save(path)
    # evaluate model
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size,
verbose=2)
    return model,accuracy
```

*The model is based on the paper "LSTMs for Human Activity Recognition Time Series Classification
link:
https://machinelearningmastery.com/how-to-develop-rnn-models-for-human-activity-recogniti on-time-series-classification/

result:

```
[73.247, 78.243, 85.173, 72.28, 77.599]
Accuracy: 77.308% (+/-4.574)
```

CASE 2 :

Change units value

units = 100 -> 128

```
-> model.add(LSTM(units = 128, input_shape = (n_timesteps, n_features)))
```

units = 100 -> 64

```
-> model.add(Dense(units = 64, activation='relu'))
```

```
def evaluate_model(trainX, trainy, testX, testy,r = 0):
    # define model
    path = './model/train'+str(r)+'.h5'
    verbose, epochs, batch_size = 2, 15, 4000
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2],
trainy.shape[1]
    print("time steps = ", trainX.shape[1])
    #define model
    model = Sequential()
    model.add(LSTM(units = 128, input_shape = (n_timesteps, n_features)))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(units = 64, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,
verbose=verbose)
    model.save(path)
    # evaluate model
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size,
verbose=2)
    return model,accuracy
```

result:

```
[72.764, 73.731, 77.679, 83.159, 83.239]
Accuracy: 78.114% (+/-4.466)
```

CASE 3 :
Change epochs = 15 -> 300

```
def evaluate_model(trainX, trainy, testX, testy,r = 0):
    # define model
    path = './model/train'+str(r)+'.h5'
    verbose, epochs, batch_size = 2, 300, 4000
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2],
trainy.shape[1]
    print("time steps = ", trainX.shape[1])
    #define model
    model = Sequential()
    model.add(LSTM(units = 128, input_shape = (n_timesteps, n_features)))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(units = 64, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,
verbose=verbose)
    model.save(path)
    # evaluate model
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size,
verbose=2)
    return model,accuracy
```

Result:

```
[99.355, 99.194, 99.355, 99.919, 99.839]
Accuracy: 99.532% (+/-0.290)
```

CASE 4 :
Change dropout = 0.5 -> 0.2
Change unit value layer 1 = 100 -> 128, layer 3 = 100 ->64

```
def evaluate_model(trainX, trainy, testX, testy,r = 0):
    # define model
    path = './model/train'+str(r)+'.h5'
    verbose, epochs, batch_size = 2, 300, 4000
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2],
trainy.shape[1]
    print("time steps = ", trainX.shape[1])
    #define model
    model = Sequential()
    model.add(LSTM(units = 128, input_shape = (n_timesteps, n_features)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(units = 64, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,
verbose=verbose)
    model.save(path)
    # evaluate model
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size,
verbose=2)
    return model,accuracy
```

result:

```
[99.839, 99.114, 99.919, 100.0, 99.678]
Accuracy: 99.710% (+/-0.316)
```

CASE 5 :
Change dropout = 0.5 ->0.2

```
def evaluate_model(trainX, trainy, testX, testy,r = 0):
    # define model
    path = './model/train'+str(r)+'.h5'
    verbose, epochs, batch_size = 2, 300, 4000
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2],
trainy.shape[1]
    print("time steps = ", trainX.shape[1])
    #define model
    model = Sequential()
    model.add(LSTM(units = 100, input_shape = (n_timesteps, n_features)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(units = 100, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,
verbose=verbose)
    model.save(path)
    # evaluate model
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size,
verbose=2)
    return model,accuracy
```

result :

```
[99.275, 99.597, 99.436, 100.0, 98.872]
Accuracy: 99.436% (+/-0.371)
```

CASE 6:

Queue size 240 -> 120

```
def evaluate_model(trainX, trainy, testX, testy,r = 0):
    # define model
    path = './model/train'+str(r)+'.h5'
    verbose, epochs, batch_size = 1, 300, 4000
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2],
trainy.shape[1]
    print("time steps = ", trainX.shape[1])
    #define model
    model = Sequential()
    model.add(LSTM(units = 100, input_shape = (n_timesteps, n_features)))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(Dense(units = 100, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,
verbose=verbose)
    model.save(path)
    # evaluate model
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size,
verbose = verbose)
    return model,accuracy
```

result:

```
[99.355, 99.517, 99.678, 98.55, 99.678]
Accuracy: 99.356% (+/-0.420)
```

CASE 7:

```
queue size 240 ->120

dropout 0.5 -> 02
def evaluate_model(trainX, trainy, testX, testy,r = 0):
    # define model
    path = './model/train'+str(r)+'.h5'
    verbose, epochs, batch_size = 1, 300, 4000
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2],
trainy.shape[1]
    print("time steps = ", trainX.shape[1])
    #define model
    model = Sequential()
    model.add(LSTM(units = 100, input_shape = (n_timesteps, n_features)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(units = 100, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,
verbose=verbose)
    model.save(path)
    # evaluate model
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size,
verbose = verbose)
    return model,accuracy

result:
[99.517, 99.517, 99.919, 99.275, 99.517]
Accuracy: 99.549% (+/-0.207)
```

CASE 8:

Change queue size 240 -> 120
Units layer 1 100-> 128
Units layer 3 100 -> 64

```
def evaluate_model(trainX, trainy, testX, testy,r = 0):
    # define model
    path = './model/train'+str(r)+'.h5'
    verbose, epochs, batch_size = 1, 300, 4000
    n_timesteps, n_features, n_outputs = trainX.shape[1], trainX.shape[2],
trainy.shape[1]
    print("time steps = ", trainX.shape[1])
    #define model
    model = Sequential()
    model.add(LSTM(units = 128, input_shape = (n_timesteps, n_features)))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(units = 64, activation='relu'))
    model.add(Dense(n_outputs, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
    model.fit(trainX, trainy, epochs=epochs, batch_size=batch_size,
verbose=verbose)
    model.save(path)
    # evaluate model
    _, accuracy = model.evaluate(testX, testy, batch_size=batch_size,
verbose = verbose)
    return model,accuracy
```

result:

```
[99.919, 99.597, 99.839, 99.678, 99.758]
Accuracy: 99.758% (+/-0.114)
```

Conclusion:
The best accuracy for the test cases is case 4 with **accuracy = 99.710**% (+/-0.316)