

System call Anomaly Detection- Deep Learning

Type *Markdown* and LaTeX: α^2

ADFA Dataset Preprocessing:

1. The system call language model estimates the probability distribution of the next call in a sequence given the sequence of previous calls.
2. We assume that the host system generates a finite number of system calls.
3. We index each system call by using an integer starting from 1 and denote the fixed set of all possible system calls in the system as $S = \{1, \dots, K\}$. Let $x = x_1x_2 \dots x_l (x_i \in S)$ denote a sequence of l system calls.

LSTM Based Model :

1. At the Input Layer, the call at each time step x_i is fed into the model in the form of one-hot encoding,
in other words, a K dimensional vector with all elements zero except position x_i .
2. At the Embedding Layer*, incoming calls are embedded to continuous space by multiplying embedding matrix W ,
which should be learned.
3. At the Hidden Layer*, the LSTM unit has an internal state, and this state is updated recurrently at each time step.
4. At the Output Layer, a softmax activation function is used to produce the estimation of normalized probability values of possible calls coming next in the sequence.

References for systemcalls:

1. http://osinside.net/syscall/system_call_table.htm
2. <https://www.cs.unm.edu/~immsec/systemcalls.htm>
3. <https://github.com/karpathy/char-rnn>
4. https://keras.io/losses/#categorical_crossentropy
5. <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

ADFA Dataset Preprocessing

In [1]:

```
1  # -*- coding: utf-8 -*-
2  """
3  Created on Thu Aug  1 13:52:35 2019
4
5  @author: kuna
6  """
7
8  #!/usr/bin/env python
9  # -*- coding: utf-8 -*-
10
11
12  import pickle
13  import sys
14
15  # import warnings filter
16  from warnings import simplefilter
17  # ignore all future warnings
18  simplefilter(action='ignore', category=FutureWarning)
19  # ignore all user warnings
20  simplefilter(action='ignore', category=UserWarning)
21
22  def saveintopickle(obj, filename):
23      with open(filename, 'wb') as handle:
24          pickle.dump(obj, handle, protocol=pickle.HIGHEST_PROTOCOL)
25
26      print ("[Pickle]: save object into {}".format(filename))
27      return
28
29
30
31  def loadfrompickle(filename):
32      with open(filename, 'rb') as handle:
33          b = pickle.load(handle)
34      return b
35
36
37
38  #draw the process bar
39  def drawProgressBar(percent, barLen = 20):
40      sys.stdout.write("\r")
41      progress = ""
42      for i in range(barLen):
43          if i < int(barLen * percent):
44              progress += "="
45          else:
46              progress += " "
47      sys.stdout.write("[ %s ] %.2f%%" % (progress, percent * 100))
48      sys.stdout.flush()
```

In [2]:

```

1 import numpy as np
2 #import io_helper
3
4
5 random_data_dup = 10 # each sample randomly duplicated between 0 and 9 times, see drop
6
7
8 def dropin(X, y):
9     """
10     The name suggests the inverse of dropout, i.e. adding more samples. See Data Augmen
11     http://simaaron.github.io/Estimating-rainfall-from-weather-radar-readings-using-rec
12     :param X: Each row is a training sequence
13     :param y: The target we train and will later predict
14     :return: new augmented X, y
15     """
16     print("X shape:", X.shape)
17     print("y shape:", y.shape)
18     X_hat = []
19     y_hat = []
20     for i in range(0, len(X)):
21         for j in range(0, np.random.random_integers(0, random_data_dup)):
22             X_hat.append(X[i, :])
23             y_hat.append(y[i])
24     return np.asarray(X_hat), np.asarray(y_hat)
25
26
27
28 def preprocess():
29
30     arrayfile = "./array_test.pickle"
31     array = loadfrompickle(arrayfile)
32     #print(type(array))
33     #print(array)
34     x_train = array[:, :-1]
35     y_train = array[:, -1]
36
37     print ("The train data size is that ")
38     print (x_train.shape)
39     print (y_train.shape)
40     return (x_train,y_train)
41
42 def preprocess_val():
43
44     arrayfile = "./array_val.pickle"
45     array = loadfrompickle(arrayfile)
46     #print(type(array))
47     #print(array)
48     x_test = array[:, :-1]
49     y_test = array[:, -1]
50
51     print ("The train data size is that ")
52     print (x_test.shape)
53     print (y_test.shape)
54     return (x_test,y_test)
55
56 #if __name__ == "__main__":
57 #    preprocess()

```


In [3]:

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4
5  import os
6  import sys
7  import numpy as np
8
9  #import io_helper
10
11 def readfilesfromAdir(dataset):
12     #read a list of files
13     files = os.listdir(dataset)
14     files_absolute_paths = []
15     for i in files:
16         files_absolute_paths.append(dataset+str(i))
17     return files_absolute_paths
18
19
20 file = "ADFA-LD/Training_Data_Master/UTD-0001.txt"
21 #this is used to read a char sequence from
22 def readCharsFromFile(file):
23     channel_values = open(file).read().split()
24     #print (len(channel_values))
25     #channel_values is a list
26     return channel_values
27     #print (channel_values[800:819])
28
29 def get_attack_subdir(path):
30     subdirectories = os.listdir(path)
31     for i in range(0,len(subdirectories)):
32         subdirectories[i] = path + subdirectories[i]
33
34     print (subdirectories)
35     return (subdirectories)
36
37
38 def get_all_call_sequences(dire):
39     files = readfilesfromAdir(dire)
40     allthelist = []
41     print (len(files))
42
43     for eachfile in files:
44         if not eachfile.endswith("DS_Store"):
45             allthelist.append(readCharsFromFile(eachfile))
46         else:
47             print ("Skip the file "+ str(eachfile))
48
49     elements = []
50     for item in allthelist:
51         for key in item:
52             if key not in elements:
53                 elements.append(key)
54
55     elements = map(int,elements)
56     elements = sorted(elements)
57
58     print ("The total unique elements:")
59     print (elements)

```

```

60
61     print ("The maximum number of elements:")
62     print (max(elements))
63
64     #print ("The length elements:")
65     #print (len(elements))
66     print (len(allthelist))
67
68     #clean the all list data set
69     _max = 0
70     for i in range(0,len(allthelist)):
71         _max = max(_max,len(allthelist[i]))
72         allthelist[i] = list(map(int,allthelist[i]))
73         #print(allthelist[i])
74
75
76     print ("The maximum length of a sequence is that {}".format(_max))
77
78     return (allthelist)
79
80 ## shift the data for analysis
81 def shift(seq, n):
82     n = n % len(seq)
83     return seq[n:] + seq[:n]
84
85
86 def convertToOneHot(vector, num_classes=None):
87     """
88     Converts an input 1-D vector of integers into an output
89     2-D array of one-hot vectors, where an i'th input value
90     of j will set a '1' in the i'th row, j'th column of the
91     output array.
92
93     Example:
94         v = np.array((1, 0, 4))
95         one_hot_v = convertToOneHot(v)
96         print one_hot_v
97
98         [[0 1 0 0 0]
99          [1 0 0 0 0]
100          [0 0 0 0 1]]
101     """
102
103     assert isinstance(vector, np.ndarray)
104     assert len(vector) > 0
105
106     if num_classes is None:
107         num_classes = np.max(vector)+1
108     else:
109         assert num_classes > 0
110         assert num_classes >= np.max(vector)
111
112     result = np.zeros(shape=(len(vector), num_classes))
113     result[np.arange(len(vector)), vector] = 1
114     return result.astype(int)
115
116     """
117     The num_class here is set as 341
118     """
119
120     #one function do one thing

```

```

121 def sequence_n_gram_parsing(alist,n_gram=20,num_class=341):
122     if len(alist) <= n_gram:
123         return alist
124
125     ans = []
126     for i in range(0,len(alist)-n_gram+1,1):
127         tmp = alist[i:i+n_gram]
128         oneHot = convertToOneHot(np.asarray(tmp), num_class)
129         #print(tmp)
130         #print(np.asarray(tmp))
131         #print(oneHot)
132         ans.append(oneHot)
133
134     #transform into nmup array
135     ans = np.array(ans)
136     return (ans)
137
138
139 def lists_of_list_into_big_matrix(allthelist,n_gram=20):
140
141     print("lists_of_list_into_big_matrix")
142     print(len(allthelist))
143     array = sequence_n_gram_parsing(allthelist[0])
144     #print(len(allthelist[0]))
145     #print(allthelist[0])
146     #print(len(array))
147     #print(array)
148
149     for i in range(1,len(allthelist),1):
150
151         tmp = sequence_n_gram_parsing(allthelist[i])
152
153         #print ("tmp shape")
154         #print(tmp)
155         #print (len(tmp))
156
157         array = np.concatenate((array, tmp), axis=0)
158         #print(allthelist[i])
159         #print(array)
160
161         percent = (i+0.0)/len(allthelist)
162         #io_helper.drawProgressBar(percent)
163         drawProgressBar(percent)
164
165         if (len(array)> 20000):
166             break
167         #print ("array shape")
168         #print (array.shape)
169         #print(len(allthelist[1]))
170         #print(allthelist[1])
171         #print(len(array))
172         #print(array)
173         #break
174
175     print (array.shape)
176     print ("done")
177     #io_helper.saveintopickle(array,"array_test.pickle")
178     saveintopickle(array,"array_test.pickle")
179
180
181 def lists_of_list_into_big_matrix_val(allthelist,n_gram=20):

```

```

182
183 array = sequence_n_gram_parsing(allthelist[0])
184
185 for i in range(1,len(allthelist),1):
186     tmp = sequence_n_gram_parsing(allthelist[i])
187
188     # print ("tmp shape")
189     # print (tmp.shape)
190
191     array = np.concatenate((array, tmp), axis=0)
192
193
194     percent = (i+0.0)/len(allthelist)
195     #io_helper.drawProgressBar(percent)
196     drawProgressBar(percent)
197
198     if (len(array)> 20000):
199         break
200     #print ("array shape")
201     #print (array.shape)
202
203
204 print (array.shape)
205 print ("done")
206 #io_helper.saveintopickle(array,"array_test.pickle")
207 saveintopickle(array,"array_val.pickle")
208
209
210 if __name__ == "__main__":
211     dirc = "ADFA-LD/Training_Data_Master/"
212     dirc_val = "ADFA-LD/Validation_Data_Master/"
213     dic_attack = "ADFA-LD/Attack_Data_Master/Adduser_1/"
214     #train1 = get_all_call_sequences(dirc)
215
216     #test = [i for i in range(0,300)]
217     #array = sequence_n_gram_parsing(test)
218     #print (type(array))
219     #print (array.shape)
220
221     #get_attack_subdir(dic_attack)
222     #print ("XXXXXXXXXXXXXXXXXXXX")
223     #val1 = get_all_call_sequences(dirc_val)
224
225     #dirc_test = "Test/"
226     #att_test = get_all_call_sequences(dirc_test)
227     #lists_of_list_into_big_matrix(att_test)
228
229     att = get_all_call_sequences(dirc)
230     lists_of_list_into_big_matrix(att)
231
232     att_val = get_all_call_sequences(dirc_val)
233     lists_of_list_into_big_matrix_val(att_val)
234

```

834

Skip the file ADFA-LD/Training_Data_Master/.DS_Store

The total unique elements:

[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 19, 20, 21, 26, 27, 30, 33, 37, 38, 39, 40, 41, 42, 43, 45, 54, 57, 60, 63, 64, 65, 66, 75, 77, 78, 83, 85, 91, 93, 94, 96, 97, 99, 102, 104, 110, 114, 117, 118, 119, 120, 122, 125, 128, 132, 133, 140, 141, 142, 143, 144, 146, 148, 155, 157, 158, 159,


```
160, 162, 163, 168, 172, 174, 175, 176, 179, 180, 183, 184, 185, 191, 192,
194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208,
209, 211, 212, 213, 214, 219, 220, 221, 224, 226, 228, 229, 230, 231, 233,
234, 240, 242, 243, 252, 254, 255, 256, 258, 259, 260, 264, 265, 266, 268,
269, 270, 272, 289, 292, 293, 295, 298, 300, 301, 307, 308, 309, 311, 314,
320, 322, 331, 332, 340]
```

```
The maximum number of elements:
```

```
340
```

```
833
```

```
The maximum length of a sequence is that 2948
```

```
lists_of_list_into_big_matrix
```

```
833
```

```
[ = ] 8.52%(20298, 20, 341)
```

```
done
```

```
[Pickle]: save object into array_test.pickle
```

```
4373
```

```
Skip the file ADFA-LD/Validation_Data_Master/.DS_Store
```

```
The total unique elements:
```

```
[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 19, 20, 21, 22, 26, 27, 30, 3
3, 37, 38, 39, 40, 41, 42, 43, 45, 54, 57, 60, 61, 63, 64, 65, 66, 75, 77,
78, 79, 83, 85, 90, 91, 93, 94, 96, 97, 99, 102, 104, 110, 111, 114, 116,
117, 118, 119, 120, 122, 124, 125, 128, 132, 133, 136, 140, 141, 142, 143,
144, 146, 148, 150, 151, 154, 155, 156, 157, 158, 159, 160, 162, 163, 168,
172, 174, 175, 176, 177, 179, 180, 181, 183, 184, 185, 186, 187, 190, 191,
192, 194, 195, 196, 197, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208,
209, 210, 211, 212, 213, 214, 215, 216, 219, 220, 221, 224, 226, 228, 229,
231, 234, 240, 243, 252, 254, 255, 256, 258, 259, 260, 264, 265, 266, 268,
269, 270, 272, 289, 292, 293, 295, 296, 298, 300, 301, 306, 307, 308, 309,
311, 314, 320, 324, 328, 331, 332, 340]
```

```
The maximum number of elements:
```

```
340
```

```
4372
```

```
The maximum length of a sequence is that 4494
```

```
[ ] 1.26%(21238, 20, 341)
```

```
done
```

```
[Pickle]: save object into array_val.pickle
```

LSTM Based Model

In [4]:

```

1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import matplotlib.pyplot as plt
5  import numpy as np
6  import time
7  from keras.layers.core import Dense, Activation, Dropout
8  from keras.layers.recurrent import LSTM
9  from keras.models import Sequential
10 from keras.models import model_from_json
11 from keras.layers.embeddings import Embedding
12
13 #import preprocess
14
15 # Global hyper-parameters
16 sequence_length = 19
17 epochs = 1
18 batch_size = 50
19 feature_dimension = 341
20 top_words = 5000
21
22 def save_model_weight_into_file(model, modelname="model.json", weight="model.h5"):
23     model_json = model.to_json()
24     with open(modelname, "w") as json_file:
25         json_file.write(model_json)
26     # serialize weights to HDF5
27     model.save_weights(weight)
28     print("Saved model to disk in {} and {}".format(modelname,weight))
29
30
31 def load_model_and_wieght_from_file(modelname="model.json", weight="model.h5"):
32
33     json_file = open(modelname, 'r')
34     loaded_model_json = json_file.read()
35     json_file.close()
36     loaded_model = model_from_json(loaded_model_json)
37     # Load weights into new model
38     loaded_model.load_weights(weight)
39     print("Loaded model from disk, you can do more analysis more")
40
41     pass
42
43
44 def build_model():
45     model = Sequential()
46     layers = {'input': feature_dimension, 'hidden1': 64, 'hidden2': 256, 'hidden3': 10
47
48     model.add(LSTM(
49         input_length=sequence_length,
50         input_dim=layers['input'],
51         output_dim=layers['hidden1'],
52         return_sequences=True))
53     model.add(Dropout(0.2))
54
55     model.add(LSTM(
56         layers['hidden2'],
57         return_sequences=True))
58     model.add(Dropout(0.2))
59

```

```

60     model.add(LSTM(
61         layers['hidden3'],
62         return_sequences=False))
63     model.add(Dropout(0.2))
64
65     model.add(Dense(
66         output_dim=layers['output'], activation='softmax'))
67     #model.add(Activation("Linear"))
68
69     start = time.time()
70
71     model.compile(loss="categorical_crossentropy", optimizer='rmsprop', metrics=['acc
72     #model.compile(loss="mse", optimizer="rmsprop")
73
74     #print ("Compilation Time :"%(time.time() - start))
75     return model
76
77 from keras.callbacks import EarlyStopping
78
79 def run_network(model=None, data=None):
80
81     global_start_time = time.time()
82
83     if data is None:
84         print ('Loading data... ')
85         # train on first 700 samples and test on next 300 samples (has anomaly)
86         X_train, y_train = preprocess()
87     else:
88         X_train, y_train = data
89
90     print ("X_train, y_train,shape")
91     print (X_train.shape)
92     print (y_train.shape)
93     print ('\nData Loaded. Compiling...\n')
94
95     if model is None:
96         model = build_model()
97         #model = build_model_2()
98         print("Training...")
99         model.fit(
100             X_train, y_train,
101             batch_size=batch_size,
102             epochs=epochs,
103             validation_split=0.3)
104         model.summary()
105         print("Done Training...")
106
107     #predicted = model.predict(X_test)
108     #print("Reshaping predicted")
109     #predicted = np.reshape(predicted, (predicted.size,))
110
111
112
113
114     """
115     except KeyboardInterrupt:
116         print("prediction exception")
117         print 'Training duration (s) : ', time.time() - global_start_time
118         return model, y_test, 0
119
120     try:

```

```
121     plt.figure(1)
122     plt.subplot(311)
123     plt.title("Actual Test Signal w/Anomalies")
124     plt.plot(y_test[:len(y_test)], 'b')
125     plt.subplot(312)
126     plt.title("Predicted Signal")
127     plt.plot(predicted[:len(y_test)], 'g')
128     plt.subplot(313)
129     plt.title("Squared Error")
130     mse = ((y_test - predicted) ** 2)
131     plt.plot(mse, 'r')
132     plt.show()
133 except Exception as e:
134     print("plotting exception")
135     print (str(e))
136 print ('Training duration (s) : '% (time.time() - global_start_time))
137
138     return model, y_test, predicted
139 """
140
141 #if __name__ == "__main__":
142 # run_network()
```

Using TensorFlow backend.

Train LSTM Model

In [5]:

```

1 global_start_time = time.time()
2
3 model=None
4
5 print ('Loading data... ')
6 # train on first 700 samples and test on next 300 samples (has anomaly)
7 X_train, y_train = preprocess()
8
9 print ("X_train, y_train,shape")
10 print (X_train.shape)
11 print (y_train.shape)
12 print ('\nData Loaded. Compiling...\n')
13
14 if model is None:
15     model = build_model()
16     print("Training...")
17     history = model.fit(
18         X_train, y_train,
19         batch_size=batch_size,
20         epochs=epochs,
21         validation_split=0.3,
22         callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)]
23     )
24     model.summary()
25     print("Done Training...")

```

Loading data...

The train data size is that

(20298, 19, 341)

(20298, 341)

X_train, y_train,shape

(20298, 19, 341)

(20298, 341)

Data Loaded. Compiling...

WARNING:tensorflow:From C:\Users\kuna\AppData\Local\Continuum\anaconda3\lib\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.

WARNING:tensorflow:From C:\Users\kuna\AppData\Local\Continuum\anaconda3\lib\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`.

Training...

WARNING:tensorflow:From C:\Users\kuna\AppData\Local\Continuum\anaconda3\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops) is deprecated and will be removed in a future version.

Instructions for updating:

Use tf.cast instead.

Train on 14208 samples, validate on 6090 samples

Epoch 1/1

14208/14208 [=====] - 32s 2ms/step - loss: 2.7770 -
acc: 0.2367 - val_loss: 2.8779 - val_acc: 0.2154

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 19, 64)	103936
dropout_1 (Dropout)	(None, 19, 64)	0
lstm_2 (LSTM)	(None, 19, 256)	328704
dropout_2 (Dropout)	(None, 19, 256)	0
lstm_3 (LSTM)	(None, 100)	142800
dropout_3 (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 341)	34441
Total params: 609,881		
Trainable params: 609,881		
Non-trainable params: 0		

Done Training...

In [6]:

```

1  #import pandas as pd
2
3  #def LoadData(file):
4      # for reading also binary mode is important
5      #     dbfile = open(file, 'rb')
6      #     db = pickle.load(dbfile)
7      #     for keys in db:
8      #         print(keys, '=>', db[keys])
9      #     dbfile.close()
10
11 #if __name__ == '__main__':
12 #     LoadData("./array_test.pickle")
13 #df_val = pd.read_pickle("./array_val.pickle")
14 #df_val.head()
```

Run model on Validation Data

In [7]:

```

1 # https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd1
2
3 X_test, y_test = preprocess_val()
4
5 print ("X_test, y_test, shape")
6 print (X_test.shape)
7 print (y_test.shape)
8
9 print("Validating...")
10 predicted = model.predict(X_test)
11 print("Done Validating...")
12 print(predicted)
13

```

The train data size is that

(21238, 19, 341)

(21238, 341)

X_test, y_test, shape

(21238, 19, 341)

(21238, 341)

Validating...

Done Validating...

```

[[2.8147128e-05 3.8867351e-02 5.9301241e-05 ... 3.2527638e-05
 4.0639268e-05 6.0572522e-04]
 [2.7337572e-05 4.2425249e-02 5.7118334e-05 ... 3.0709063e-05
 3.9311104e-05 5.8961258e-04]
 [3.2080068e-05 4.1573644e-02 6.1957377e-05 ... 3.6363443e-05
 4.3218708e-05 6.0780835e-04]
 ...
 [1.8595829e-06 1.2511486e-03 3.5294606e-06 ... 2.1812916e-06
 1.6237399e-06 6.7461682e-05]
 [1.8240867e-06 1.3079355e-03 3.5116327e-06 ... 2.1674750e-06
 1.6114174e-06 6.7553679e-05]
 [1.8013474e-06 1.3025296e-03 3.4824586e-06 ... 2.1350809e-06
 1.6007832e-06 6.6800356e-05]]

```

How did our model perform?

In [8]:

```

1
2 score, accuracy = model.evaluate(X_test, y_test, verbose=2, batch_size=batch_size)
3 print('Score : %.2f'%(score))
4 print('Validation Accuracy : %.2f'%(accuracy))

```

Score : 3.08

Validation Accuracy : 0.19

In [9]:

```
1 plt.title('Loss')
2 plt.plot(history.history['loss'], label='train')
3 plt.plot(history.history['val_loss'], label='test')
4 plt.legend()
5 plt.show();
```

In [10]:

```
1 history.history
```

Out[10]:

```
{'val_loss': [2.8778519998434535],
 'val_acc': [0.21543513882556573],
 'loss': [2.776978516363883],
 'acc': [0.23669763501452468]}
```

In [11]:

```
1 plt.title('Accuracy')
2 plt.plot(history.history['acc'], label='train')
3 plt.plot(history.history['val_acc'], label='test')
4 plt.legend()
5 plt.show();
```

How to Test with new syscall sequence ??

In []:

```
1
```

Train LSTM simpler model

In [12]:

```
1 # https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-  
2  
3 word_vec_length = 19  
4 char_vec_length = 341  
5 output_labels = 341  
6  
7  
8 #hidden_nodes = 4000 # int(2/3 * (word_vec_length * char_vec_length))  
9 hidden_nodes = 100  
10 print(f"The number of hidden nodes is {hidden_nodes}.")  
11  
12 def build_model_2():  
13     # Build the model  
14     print('Build model...')  
15     model = Sequential()  
16     model.add(LSTM(hidden_nodes, return_sequences=False, input_shape=(word_vec_length,  
17     model.add(Dropout(0.2))  
18     model.add(Dense(units=output_labels))  
19     model.add(Activation('softmax'))  
20     model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])  
21     #print ("Compilation Time :"%(time.time() - start))  
22     return model
```

The number of hidden nodes is 100.

In [13]:

```

1 global_start_time = time.time()
2
3 model=None
4
5 print ('Loading data... ')
6 # train on first 700 samples and test on next 300 samples (has anomaly)
7 X_train, y_train = preprocess()
8
9 print ("X_train, y_train,shape")
10 print (X_train.shape)
11 print (y_train.shape)
12 print ('\nData Loaded. Compiling...\n')
13
14 batch_size=32
15 model = build_model_2()
16 print("Training...")
17 model.fit(X_train, y_train, batch_size=batch_size, epochs=10, validation_data=(X_test,
18 model.summary()
19 print("Done Training...")
20

```

Loading data...

The train data size is that

(20298, 19, 341)

(20298, 341)

X_train, y_train,shape

(20298, 19, 341)

(20298, 341)

Data Loaded. Compiling...

Build model...

Training...

Train on 20298 samples, validate on 21238 samples

Epoch 1/10

20298/20298 [=====] - 22s 1ms/step - loss: 2.8397

- acc: 0.2396 - val_loss: 2.5586 - val_acc: 0.4236

Epoch 2/10

20298/20298 [=====] - 21s 1ms/step - loss: 1.9795

- acc: 0.4184 - val_loss: 2.1248 - val_acc: 0.4901

Epoch 3/10

20298/20298 [=====] - 21s 1ms/step - loss: 1.6778

- acc: 0.5024 - val_loss: 2.0376 - val_acc: 0.5023

Epoch 4/10

20298/20298 [=====] - 21s 1ms/step - loss: 1.5040

- acc: 0.5570 - val_loss: 1.9283 - val_acc: 0.5027

Epoch 5/10

20298/20298 [=====] - 20s 977us/step - loss: 1.39

88 - acc: 0.5863 - val_loss: 1.9624 - val_acc: 0.5107

Epoch 6/10

20298/20298 [=====] - 20s 1ms/step - loss: 1.3225

- acc: 0.6038 - val_loss: 1.9617 - val_acc: 0.5203

Epoch 7/10

20298/20298 [=====] - 21s 1ms/step - loss: 1.2664

- acc: 0.6213 - val_loss: 1.9898 - val_acc: 0.5073

Epoch 8/10

20298/20298 [=====] - 20s 998us/step - loss: 1.21

96 - acc: 0.6373 - val_loss: 2.0098 - val_acc: 0.5102

Epoch 9/10

20298/20298 [=====] - 21s 1ms/step - loss: 1.1759

- acc: 0.6471 - val_loss: 2.0388 - val_acc: 0.5134

Epoch 10/10

20298/20298 [=====] - 20s 980us/step - loss: 1.14

51 - acc: 0.6549 - val_loss: 2.0311 - val_acc: 0.5157

Layer (type)	Output Shape	Param #
lstm_4 (LSTM)	(None, 100)	176800
dropout_4 (Dropout)	(None, 100)	0
dense_2 (Dense)	(None, 341)	34441
activation_1 (Activation)	(None, 341)	0

Total params: 211,241

Trainable params: 211,241

Non-trainable params: 0

Done Training...

In [14]:

```

1 score, accuracy = model.evaluate(X_train, y_train, verbose=2, batch_size=batch_size)
2 print('Train Score : %.2f'%(score))
3 print('Train Validation Accuracy : %.2f'%(accuracy))

```

Train Score : 1.05

Train Validation Accuracy : 0.68

In [15]:

```

1 score, accuracy = model.evaluate(X_test, y_test, verbose=2, batch_size=batch_size)
2 print('Test Score : %.2f'%(score))
3 print('Test Validation Accuracy : %.2f'%(accuracy))

```

Test Score : 2.03

Test Validation Accuracy : 0.52

In [16]:

```

1  ## k-fold validation
2  from sklearn.model_selection import StratifiedKFold
3  import numpy
4
5  # fix random seed for reproducibility
6  seed = 7
7  numpy.random.seed(seed)
8
9  # split into input (X) and output (Y) variables
10 X = X_train
11 Y = y_train
12 Y

```

Out[16]:

```

array([[0, 1, 0, ..., 0, 0, 0],
       [0, 1, 0, ..., 0, 0, 0],
       [0, 1, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])

```

In [17]:

```

1  # define 10-fold cross validation test harness
2  #kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
3  #cvscores = []
4  #for train, test in kfold.split(X, Y):
5  # # create model
6  # model = Sequential()
7  # model.add(Dense(12, input_dim=341, activation='relu'))
8  # model.add(Dense(8, activation='relu'))
9  # model.add(Dense(1, activation='sigmoid'))
10 # # Compile model
11 # model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
12 # # Fit the model
13 # model.fit(X[train], Y[train], epochs=150, batch_size=10, verbose=0)
14 # # evaluate the model
15 # scores = model.evaluate(X[test], Y[test], verbose=0)
16 # print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
17 # cvscores.append(scores[1] * 100)
18 #print("%.2f%% (+/- %.2f%%)" % (numpy.mean(cvscores), numpy.std(cvscores)))

```

In [18]:

```

1  # https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-
2
3  word_vec_length = 19
4  char_vec_length = 341
5  output_labels = 341
6
7
8  hidden_nodes = 100 # int(2/3 * (word_vec_length * char_vec_length))
9  print(f"The number of hidden nodes is {hidden_nodes}.")
10
11 def build_model_3():
12     # Build the model
13     print('Build model...')
14     model = Sequential()
15     model.add(LSTM(hidden_nodes, return_sequences=False, input_shape=(word_vec_length,
16     model.add(Dropout(0.5))
17     model.add(Dense(units=output_labels))
18     model.add(Activation('softmax'))
19     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
20     #print ("Compilation Time :"%(time.time() - start))
21     return model
22
23 global_start_time = time.time()
24
25 model=None
26
27 print ('Loading data... ')
28 # train on first 700 samples and test on next 300 samples (has anomaly)
29 X_train, y_train = preprocess()
30
31 print ("X_train, y_train,shape")
32 print (X_train.shape)
33 print (y_train.shape)
34 print ('\nData Loaded. Compiling...\n')
35
36 batch_size=32
37 model = build_model_3()
38 print("Training...")
39 model.fit(X_train, y_train, batch_size=batch_size, epochs=10, validation_data=(X_test,
40 model.summary()
41 print("Done Training...")

```

The number of hidden nodes is 100.

Loading data...

The train data size is that

(20298, 19, 341)

(20298, 341)

X_train, y_train,shape

(20298, 19, 341)

(20298, 341)

Data Loaded. Compiling...

Build model...

Training...

Train on 20298 samples, validate on 21238 samples

Epoch 1/10

20298/20298 [=====] - 24s 1ms/step - loss: 0.0118 -
acc: 0.9971 - val_loss: 0.0100 - val_acc: 0.9973

Epoch 2/10

20298/20298 [=====] - 20s 1ms/step - loss: 0.0086 -
acc: 0.9973 - val_loss: 0.0086 - val_acc: 0.9976 0.99 - ETA: 5s - loss: 0. -
ETA: 4s - loss: 0.00 - ETA: 3s - loss: 0.0087 - acc: 0.9 - ETA: 3s - loss:
0.0087 - - ETA: 2s - loss: 0. - ETA: 1s - loss: 0.

Epoch 3/10

20298/20298 [=====] - 23s 1ms/step - loss: 0.0076 -
acc: 0.9975 - val_loss: 0.0083 - val_acc: 0.9977

Epoch 4/10

20298/20298 [=====] - 24s 1ms/step - loss: 0.0070 -
acc: 0.9977 - val_loss: 0.0080 - val_acc: 0.9978

Epoch 5/10

20298/20298 [=====] - 22s 1ms/step - loss: 0.0059 -
acc: 0.9980 - val_loss: 0.0078 - val_acc: 0.9977

Epoch 9/10

20298/20298 [=====] - 20s 1ms/step - loss: 0.0058 -
acc: 0.9981 - val_loss: 0.0077 - val_acc: 0.9978

Epoch 10/10

20298/20298 [=====] - 20s 994us/step - loss: 0.0056
- acc: 0.9981 - val_loss: 0.0078 - val_acc: 0.9978

Layer (type)	Output Shape	Param #
=====		
lstm_5 (LSTM)	(None, 100)	176800
=====		
dropout_5 (Dropout)	(None, 100)	0
=====		
dense_3 (Dense)	(None, 341)	34441
=====		
activation_2 (Activation)	(None, 341)	0
=====		

Total params: 211,241

Trainable params: 211,241

Non-trainable params: 0

Done Training...

In [19]:

```

1  def build_model_4():
2      # Build the model
3      print('Build model...')
4      model = Sequential()
5      model.add(LSTM(hidden_nodes, return_sequences=False, input_shape=(word_vec_length,
6      model.add(Dropout(0.2))
7      model.add(Dense(units=output_labels))
8      model.add(Activation('softmax'))
9      model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
10     #print ("Compilation Time :"%(time.time() - start))
11     return model
12
13 global_start_time = time.time()
14
15 model=None
16
17 print ('Loading data... ')
18 # train on first 700 samples and test on next 300 samples (has anomaly)
19 X_train, y_train = preprocess()
20
21 print ("X_train, y_train,shape")
22 print (X_train.shape)
23 print (y_train.shape)
24 print ('\nData Loaded. Compiling...\n')
25
26 batch_size=32
27 model = build_model_4()
28 print("Training...")
29 model.fit(X_train, y_train, batch_size=batch_size, epochs=10, validation_data=(X_test,
30 model.summary()
31 print("Done Training...")

```

Loading data...

The train data size is that

(20298, 19, 341)

(20298, 341)

X_train, y_train,shape

(20298, 19, 341)

(20298, 341)

Data Loaded. Compiling...

Build model...

Training...

Train on 20298 samples, validate on 21238 samples

Epoch 1/10

20298/20298 [=====] - 23s 1ms/step - loss: 0.0112

- acc: 0.9971 - val_loss: 0.0099 - val_acc: 0.9975

Epoch 2/10

20298/20298 [=====] - 20s 986us/step - loss: 0.00

82 - acc: 0.9974 - val_loss: 0.0085 - val_acc: 0.9977

Epoch 3/10

20298/20298 [=====] - 20s 996us/step - loss: 0.00

71 - acc: 0.9976 - val_loss: 0.0080 - val_acc: 0.9978

Epoch 4/10

20298/20298 [=====] - 21s 1ms/step - loss: 0.0065

- acc: 0.9978 - val_loss: 0.0078 - val_acc: 0.9978

Epoch 5/10
 20298/20298 [=====] - 22s 1ms/step - loss: 0.0061
 - acc: 0.9979 - val_loss: 0.0078 - val_acc: 0.9978
 Epoch 6/10
 20298/20298 [=====] - 22s 1ms/step - loss: 0.0058
 - acc: 0.9980 - val_loss: 0.0078 - val_acc: 0.9977
 Epoch 7/10
 20298/20298 [=====] - 21s 1ms/step - loss: 0.0056
 - acc: 0.9981 - val_loss: 0.0079 - val_acc: 0.9977
 Epoch 8/10
 20298/20298 [=====] - 21s 1ms/step - loss: 0.0054
 - acc: 0.9981 - val_loss: 0.0080 - val_acc: 0.9977
 Epoch 9/10
 20298/20298 [=====] - 22s 1ms/step - loss: 0.0053
 - acc: 0.9982 - val_loss: 0.0080 - val_acc: 0.9977
 Epoch 10/10
 20298/20298 [=====] - 20s 994us/step - loss: 0.00
 51 - acc: 0.9982 - val_loss: 0.0080 - val_acc: 0.9977

Layer (type)	Output Shape	Param #
=====		
lstm_6 (LSTM)	(None, 100)	176800
=====		
dropout_6 (Dropout)	(None, 100)	0
=====		
dense_4 (Dense)	(None, 341)	34441
=====		
activation_3 (Activation)	(None, 341)	0
=====		

Total params: 211,241
 Trainable params: 211,241
 Non-trainable params: 0

Done Training...

In [20]:

```

1  def build_model_5():
2      # Build the model
3      print('Build model...')
4      model = Sequential()
5      model.add(LSTM(hidden_nodes, return_sequences=False, input_shape=(word_vec_length,
6      #model.add(Dropout(0.2))
7      model.add(Dense(units=output_labels))
8      model.add(Activation('softmax'))
9      model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
10     #print ("Compilation Time :"%(time.time() - start))
11     return model
12
13 global_start_time = time.time()
14
15 model=None
16
17 print ('Loading data... ')
18 # train on first 700 samples and test on next 300 samples (has anomaly)
19 X_train, y_train = preprocess()
20
21 print ("X_train, y_train,shape")
22 print (X_train.shape)
23 print (y_train.shape)
24 print ('\nData Loaded. Compiling...\n')
25
26 batch_size=32
27 model = build_model_5()
28 print("Training...")
29 model.fit(X_train, y_train, batch_size=batch_size, epochs=10, validation_data=(X_test,
30 model.summary()
31 print("Done Training...")

```

Loading data...

The train data size is that

(20298, 19, 341)

(20298, 341)

X_train, y_train,shape

(20298, 19, 341)

(20298, 341)

Data Loaded. Compiling...

Build model...

Training...

Train on 20298 samples, validate on 21238 samples

Epoch 1/10

20298/20298 [=====] - 28s 1ms/step - loss: 0.0111

- acc: 0.9971 - val_loss: 0.0100 - val_acc: 0.9973

Epoch 2/10

20298/20298 [=====] - 21s 1ms/step - loss: 0.0080

- acc: 0.9974 - val_loss: 0.0085 - val_acc: 0.9977

Epoch 3/10

20298/20298 [=====] - 22s 1ms/step - loss: 0.0069

- acc: 0.9977 - val_loss: 0.0081 - val_acc: 0.9978

Epoch 4/10

20298/20298 [=====] - 21s 1ms/step - loss: 0.0062

- acc: 0.9979 - val_loss: 0.0080 - val_acc: 0.9978

Epoch 5/10
 20298/20298 [=====] - 21s 1ms/step - loss: 0.0059
 - acc: 0.9980 - val_loss: 0.0079 - val_acc: 0.9978
 Epoch 6/10
 20298/20298 [=====] - 22s 1ms/step - loss: 0.0056
 - acc: 0.9981 - val_loss: 0.0080 - val_acc: 0.9977
 Epoch 7/10
 20298/20298 [=====] - 22s 1ms/step - loss: 0.0054
 - acc: 0.9982 - val_loss: 0.0081 - val_acc: 0.9977
 Epoch 8/10
 20298/20298 [=====] - 22s 1ms/step - loss: 0.0052
 - acc: 0.9982 - val_loss: 0.0081 - val_acc: 0.9977
 Epoch 9/10
 20298/20298 [=====] - 21s 1ms/step - loss: 0.0050
 - acc: 0.9983 - val_loss: 0.0080 - val_acc: 0.9977
 Epoch 10/10
 20298/20298 [=====] - 21s 1ms/step - loss: 0.0049
 - acc: 0.9983 - val_loss: 0.0082 - val_acc: 0.9978

Layer (type)	Output Shape	Param #
lstm_7 (LSTM)	(None, 100)	176800
dense_5 (Dense)	(None, 341)	34441
activation_4 (Activation)	(None, 341)	0
Total params: 211,241		
Trainable params: 211,241		
Non-trainable params: 0		

Done Training...

In [21]:

```
1 score, accuracy = model.evaluate(X_train, y_train, verbose=2, batch_size=batch_size)
2 print('Train Score : %.2f'%(score))
3 print('Train Validation Accuracy : %.2f'%(accuracy))
```

Train Score : 0.00
 Train Validation Accuracy : 1.00

In [22]:

```
1 score, accuracy = model.evaluate(X_test, y_test, verbose=2, batch_size=batch_size)
2 print('Test Score : %.2f'%(score))
3 print('Test Validation Accuracy : %.2f'%(accuracy))
```

Test Score : 0.01
 Test Validation Accuracy : 1.00

LSTM for Binary Classification of SystemCalls

In [244]:

```

1  def preprocess():
2
3      arrayfile = "./array_test.pickle"
4      array = loadfrompickle(arrayfile)
5      #print(type(array))
6      #print(array)
7      x_train = array[:, :]
8      print (x_train.shape)
9      x_train = x_train.reshape(40099, 20, 1)
10     y_train = np.zeros((40099,1))
11
12     print ("The train data size is that ")
13     print (x_train.shape)
14     print (y_train.shape)
15     return (x_train,y_train)
16
17 def preprocess_val():
18
19     arrayfile = "./array_val.pickle"
20     array = loadfrompickle(arrayfile)
21     #print(type(array))
22     #print(array)
23     x_test = array[:, :]
24     print (x_test.shape)
25     x_test = x_test.reshape(40142, 20, 1)
26     y_test = np.zeros((40142,1))
27
28     print ("The validation data size is that ")
29     print (x_test.shape)
30     print (y_test.shape)
31     return (x_test,y_test)
32
33 def preprocess_attack():
34
35     arrayfile = "./array_attack.pickle"
36     array = loadfrompickle(arrayfile)
37     #print(type(array))
38     #print(array)
39     x_attack = array[:, :]
40     x_attack = x_attack.reshape(6184, 20, 1)
41     y_attack = np.ones((6184,1))
42
43     print ("The attack data size is that ")
44     print (x_attack.shape)
45     print (y_attack.shape)
46     return (x_attack,y_attack)
47
48
49 """
50 The num_class here is set as 1
51 """
52
53 #one function do one thing
54 def sequence_n_gram_parsing_noencoding(alist,n_gram=20,num_class=1):
55     if len(alist) <= n_gram:
56         return alist
57
58     ans = []
59     for i in range(0,len(alist)-n_gram+1,1):

```

```

60     tmp = alist[i:i+n_gram]
61     #oneHot = convertToOneHot(np.asarray(tmp), num_class)
62     #print(tmp)
63     #print(np.asarray(tmp))
64     #print(oneHot)
65     ans.append(tmp)
66
67     #transform into nmup array
68     ans = np.array(ans)
69     return (ans)
70
71
72 def lists_of_list_into_big_matrix(allthelist,n_gram=20):
73
74     #print("lists_of_list_into_big_matrix train")
75     #print(len(allthelist))
76     array = sequence_n_gram_parsing_noencoding(allthelist[0])
77     #print(len(allthelist[0]))
78     #print(allthelist[0])
79     #print(len(array))
80     #print(array)
81
82     for i in range(1,len(allthelist),1):
83
84         tmp = sequence_n_gram_parsing_noencoding(allthelist[i])
85
86         #print ("tmp shape")
87         #print(tmp.shape)
88         #print(array.shape)
89         #print (len(tmp))
90
91         array = np.concatenate((array, tmp), axis=0)
92         #print(allthelist[i])
93         #print(array)
94
95         percent = (i+0.0)/len(allthelist)
96         #io_helper.drawProgressBar(percent)
97         drawProgressBar(percent)
98
99         if (len(array)> 40000):
100             break
101         #print ("array shape")
102         #print (array.shape)
103         #print(len(allthelist[1]))
104         #print(allthelist[1])
105         #print(len(array))
106         #print(array)
107         #break
108
109     print (array.shape)
110     print ("done")
111     #io_helper.saveintopickle(array,"array_test.pickle")
112     saveintopickle(array,"array_test.pickle")
113
114
115 def lists_of_list_into_big_matrix_val(allthelist,n_gram=20):
116
117     #print("lists_of_list_into_big_matrix validation")
118     #print(len(allthelist))
119     array = sequence_n_gram_parsing_noencoding(allthelist[0])
120     #print(len(allthelist[0]))

```

```
121     #print(allthelist[0])
122     #print(len(array))
123     #print(array)
124
125     for i in range(1,len(allthelist),1):
126         tmp = sequence_n_gram_parsing_noencoding(allthelist[i])
127
128         # print ("tmp shape")
129         # print (tmp.shape)
130
131         array = np.concatenate((array, tmp), axis=0)
132
133
134         percent = (i+0.0)/len(allthelist)
135         #io_helper.drawProgressBar(percent)
136         drawProgressBar(percent)
137
138         if (len(array)> 40000):
139             break
140         #print ("array shape")
141         #print (array.shape)
142
143
144     print (array.shape)
145     print ("done")
146     #io_helper.saveintopickle(array,"array_test.pickle")
147     saveintopickle(array,"array_val.pickle")
148
149 def get_all_call_sequences_attack(dire):
150     # List of attacks
151     attack = ['Adduser', 'Hydra_FTP', 'Hydra_SSH', 'Java_Meterpreter', 'Meterpreter', 'Web_
152     #attack = ['Adduser' , 'Hydra_FTP']
153     for term in attack:
154         in_address = dire+term
155         for i in range (1,11):
156             files = readfilesfromAdir(in_address+"_"+str(i)+"/")
157
158     allthelist = []
159     #print(files)
160     #print (len(files))
161
162     for eachfile in files:
163         if not eachfile.endswith("DS_Store"):
164             allthelist.append(readCharsFromFile(eachfile))
165         else:
166             print ("Skip the file "+ str(eachfile))
167
168     elements = []
169     for item in allthelist:
170         for key in item:
171             if key not in elements:
172                 elements.append(key)
173
174     elements = map(int,elements)
175     elements = sorted(elements)
176
177     print ("The total unique elements:")
178     print (elements)
179
180     print ("The maximum number of elements:")
181     print (max(elements))
```

```

182
183     #print ("The Length elements:")
184     #print (len(elements))
185     print (len(allthelist))
186
187     #clean the all list data set
188     _max = 0
189     for i in range(0,len(allthelist)):
190         _max = max(_max,len(allthelist[i]))
191         allthelist[i] = list(map(int,allthelist[i]))
192         #print(allthelist[i])
193
194
195     print ("The maximum length of a sequence is that {}".format(_max))
196
197     return (allthelist)
198
199 def lists_of_list_into_big_matrix_attack(allthelist,n_gram=20):
200
201     array = sequence_n_gram_parsing_noencoding(allthelist[0])
202
203     for i in range(1,len(allthelist),1):
204         tmp = sequence_n_gram_parsing_noencoding(allthelist[i])
205
206         # print ("tmp shape")
207         #print (tmp.shape)
208         #print (array.shape)
209
210         array = np.concatenate((array, tmp), axis=0)
211
212
213         percent = (i+0.0)/len(allthelist)
214         #io_helper.drawProgressBar(percent)
215         drawProgressBar(percent)
216
217         if (len(array)> 40000):
218             break
219         #print ("array shape")
220         #print (array.shape)
221
222
223     print (array.shape)
224     print ("done")
225     #io_helper.saveintopickle(array,"array_test.pickle")
226     saveintopickle(array,"array_attack.pickle")
227     #pickle2csv("array_attack.pickle", "attack.csv")
228
229
230
231
232 if __name__ == "__main__":
233     dirc = "ADFA-LD/Training_Data_Master/"
234     dirc_val = "ADFA-LD/Validation_Data_Master/"
235     dic_attack = "ADFA-LD/Attack_Data_Master/"
236
237     att = get_all_call_sequences(dirc)
238     lists_of_list_into_big_matrix(att)
239
240     att_val = get_all_call_sequences(dirc_val)
241     lists_of_list_into_big_matrix_val(att_val)
242

```

```

243 att_attack = get_all_call_sequences_attack(dic_attack)
244 lists_of_list_into_big_matrix_attack(att_attack)
245
246 test_split = 0.2
247
248 X_train_p, y_train_p = preprocess()
249
250 X_test_p, y_test_p = preprocess_val()
251
252 X_attack_p, y_attack_p = preprocess_attack()
253
254 X_a1, X_a2 = np.array_split(X_attack_p, 2)
255 y_a1, y_a2 = np.array_split(y_attack_p, 2)
256
257 X_train = np.concatenate([X_train_p, X_a1])
258 y_train = np.concatenate([y_train_p, y_a1])
259
260 X_test = np.concatenate([X_test_p, X_a2])
261 y_test = np.concatenate([y_test_p, y_a2])
262
263
264

```

834

Skip the file ADFA-LD/Training_Data_Master/.DS_Store

The total unique elements:

```

[1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 19, 20, 21, 26, 27, 30, 33, 3
7, 38, 39, 40, 41, 42, 43, 45, 54, 57, 60, 63, 64, 65, 66, 75, 77, 78, 83,
85, 91, 93, 94, 96, 97, 99, 102, 104, 110, 114, 117, 118, 119, 120, 122, 1
25, 128, 132, 133, 140, 141, 142, 143, 144, 146, 148, 155, 157, 158, 159,
160, 162, 163, 168, 172, 174, 175, 176, 179, 180, 183, 184, 185, 191, 192,
194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208,
209, 211, 212, 213, 214, 219, 220, 221, 224, 226, 228, 229, 230, 231, 233,
234, 240, 242, 243, 252, 254, 255, 256, 258, 259, 260, 264, 265, 266, 268,
269, 270, 272, 289, 292, 293, 295, 298, 300, 301, 307, 308, 309, 311, 314,
320, 322, 331, 332, 340]

```

The maximum number of elements:

340

833

The maximum length of a sequence is that 2948

[==] 14.53%(40099, 20)

done

[251]: save object into array test with

In [252]:

```

1 X_train_t = np.concatenate((X_train, y_train[:,None]), axis=1)#np.random.shuffle
2 X_train_t.shape
3 np.random.shuffle(X_train_t)
4 y_train = X_train_t[:, -1]
5 X_train = X_train_t[:, :20, :]

```

In [253]:

```

1 X_test_t = np.concatenate((X_test, y_test[:,None]), axis=1)#np.random.shuffle
2 X_test_t.shape
3 np.random.shuffle(X_test_t)
4 y_test = X_test_t[:, -1]
5 X_test = X_test_t[:, :20, :]

```

In [254]:

```

1 pprint.pprint(X_train_t[0,:20,:])
2 type(X_train_t[:, :20,:])
3 X_train_t[:, :20,:].shape

```

```

array([[ 33.],
       [192.],
       [  6.],
       [ 33.],
       [  6.],
       [192.],
       [125.],
       [197.],
       [197.],
       [197.],
       [197.],
       [ 85.],
       [ 85.],
       [174.],
       [174.],
       [174.],
       [174.],
       [195.],
       [  3.],
       [  3.]])

```

Out[254]:

(43191, 20, 1)

In [255]:

```

1 X_train.shape, y_train.shape, X_test.shape, y_test.shape , X_attack.shape, y_attack.sh

```

Out[255]:

```

((43191, 20, 1),
 (43191, 1),
 (43234, 20, 1),
 (43234, 1),
 (6184, 20, 1),
 (6184, 1))

```


In [290]:

```

1 word_vec_length = 20
2 char_vec_length = 1
3 output_labels = 1
4 hidden_nodes = 13 # int(2/3 * (word_vec_length * char_vec_length))
5 epochs = 10
6 batch_size = 32
7
8
9 def build_model_6():
10     # Build the model
11     print('Build model...')
12     model = Sequential()
13     model.add(LSTM(hidden_nodes, return_sequences=False, input_shape=(word_vec_length,
14     #model.add(Dropout(0.2))
15     model.add(Dense(units=output_labels))
16     #model.add(Activation('softmax'))
17     model.add(Dense(units=output_labels, activation='sigmoid'))
18     model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
19     #print ("Compilation Time :"%(time.time() - start))
20     return model
21
22
23
24
25 global_start_time = time.time()
26
27 model=None
28
29 print ('Loading data... ')
30 # train on first 700 samples and test on next 300 samples (has anomaly)
31
32 print ("X_train, y_train,shape")
33 print (X_train.shape)
34 print (y_train.shape)
35 print ('\nData Loaded. Compiling...\n')
36
37
38
39 batch_size=32
40 model = build_model_6()
41 print("Training...")
42 history = model.fit(
43     X_train, y_train,
44     batch_size=batch_size,
45     epochs=epochs,
46     validation_split=0.01,
47     callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)]
48
49 model.summary()
50 print("Done Training...")

```

Loading data...
X_train, y_train,shape
(43191, 20, 1)
(43191, 1)

Data Loaded. Compiling...

```

Build model...
Training...
Train on 42759 samples, validate on 432 samples
Epoch 1/10
42759/42759 [=====] - 27s 627us/step - loss: 0.25
58 - acc: 0.8988 - val_loss: 0.1816 - val_acc: 0.9306
Epoch 2/10
42759/42759 [=====] - 21s 496us/step - loss: 0.16
51 - acc: 0.9272 - val_loss: 0.1562 - val_acc: 0.9329
Epoch 3/10
42759/42759 [=====] - 22s 503us/step - loss: 0.15
18 - acc: 0.9305 - val_loss: 0.1500 - val_acc: 0.9329
Epoch 4/10
42759/42759 [=====] - 22s 510us/step - loss: 0.14
45 - acc: 0.9324 - val_loss: 0.1481 - val_acc: 0.9306
Epoch 5/10
42759/42759 [=====] - 22s 512us/step - loss: 0.13
80 - acc: 0.9352 - val_loss: 0.1405 - val_acc: 0.9375
Epoch 6/10
42759/42759 [=====] - 22s 520us/step - loss: 0.14
08 - acc: 0.9372 - val_loss: 0.1384 - val_acc: 0.9421
Epoch 7/10
42759/42759 [=====] - 22s 523us/step - loss: 0.13
06 - acc: 0.9395 - val_loss: 0.1367 - val_acc: 0.9444
Epoch 8/10
42759/42759 [=====] - 22s 509us/step - loss: 0.12
61 - acc: 0.9411 - val_loss: 0.1205 - val_acc: 0.9468
Epoch 9/10
42759/42759 [=====] - 22s 523us/step - loss: 0.12
13 - acc: 0.9440 - val_loss: 0.1112 - val_acc: 0.9560
Epoch 10/10
42759/42759 [=====] - 22s 515us/step - loss: 0.11
31 - acc: 0.9458 - val_loss: 0.1089 - val_acc: 0.9560

```

Layer (type)	Output Shape	Param #
lstm_41 (LSTM)	(None, 13)	780
dense_59 (Dense)	(None, 1)	14
dense_60 (Dense)	(None, 1)	2

```

Total params: 796
Trainable params: 796
Non-trainable params: 0

```

Done Training...

In [291]:

```

1  ### Plotting the change in the loss over the epochs.
2  # https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for

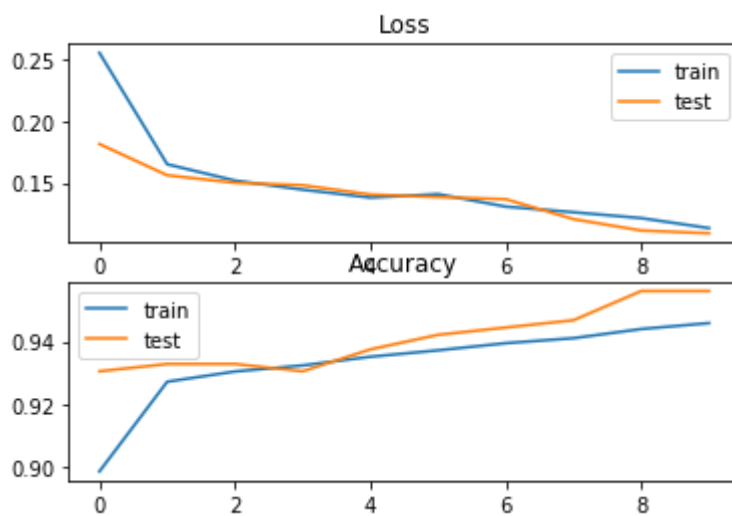
```

In [292]:

```

1  # plot loss during training
2  from matplotlib import pyplot
3
4  pyplot.subplot(211)
5  pyplot.title('Loss')
6  pyplot.plot(history.history['loss'], label='train')
7  pyplot.plot(history.history['val_loss'], label='test')
8  pyplot.legend()
9  # plot accuracy during training
10 pyplot.subplot(212)
11 pyplot.title('Accuracy')
12 pyplot.plot(history.history['acc'], label='train')
13 pyplot.plot(history.history['val_acc'], label='test')
14 pyplot.legend()
15 pyplot.show()

```



In [293]:

```

1  # predict probabilities for test set
2  yhat_probs = model.predict(X_test, verbose=0)
3  # predict crisp classes for test set
4  yhat_classes = model.predict_classes(X_test, verbose=0)

```

In [294]:

```

1  # reduce to 1d array
2  yhat_probs = yhat_probs[:, 0]
3  yhat_classes = yhat_classes[:, 0]

```

In [295]:

```

1 from sklearn import metrics
2
3 print(metrics.classification_report(yhat_classes, y_test))

```

	precision	recall	f1-score	support
0	0.91	0.94	0.92	39123
1	0.20	0.15	0.17	4111
micro avg	0.86	0.86	0.86	43234
macro avg	0.56	0.54	0.55	43234
weighted avg	0.85	0.86	0.85	43234

In [296]:

```

1 from sklearn.metrics import (confusion_matrix, precision_recall_curve, auc,
2                               roc_curve, recall_score, classification_report, f1_score,
3                               precision_recall_fscore_support)
4
5 from sklearn.metrics import cohen_kappa_score
6 from sklearn.metrics import roc_auc_score
7 from sklearn.metrics import confusion_matrix
8
9 # accuracy: (tp + tn) / (p + n)
10 accuracy = accuracy_score(y_test, yhat_classes)
11 print('Accuracy: %f' % accuracy)
12 # precision tp / (tp + fp)
13 precision = precision_score(y_test, yhat_classes, average='weighted', labels=np.unique
14 print('Precision: %f' % precision)
15 # recall: tp / (tp + fn)
16 recall = recall_score(y_test, yhat_classes, average='weighted', labels=np.unique(yhat_
17 print('Recall: %f' % recall)
18 # f1: 2 tp / (2 tp + fp + fn)
19 f1 = f1_score(y_test, yhat_classes, average='weighted', labels=np.unique(yhat_classes)
20 print('F1 score: %f' % f1)

```

Accuracy: 0.862261
Precision: 0.880766
Recall: 0.862261
F1 score: 0.871119

In [297]:

```

1 # kappa
2 kappa = cohen_kappa_score(y_test, yhat_classes)
3 print('Cohens kappa: %f' % kappa)
4

```

Cohens kappa: 0.099771

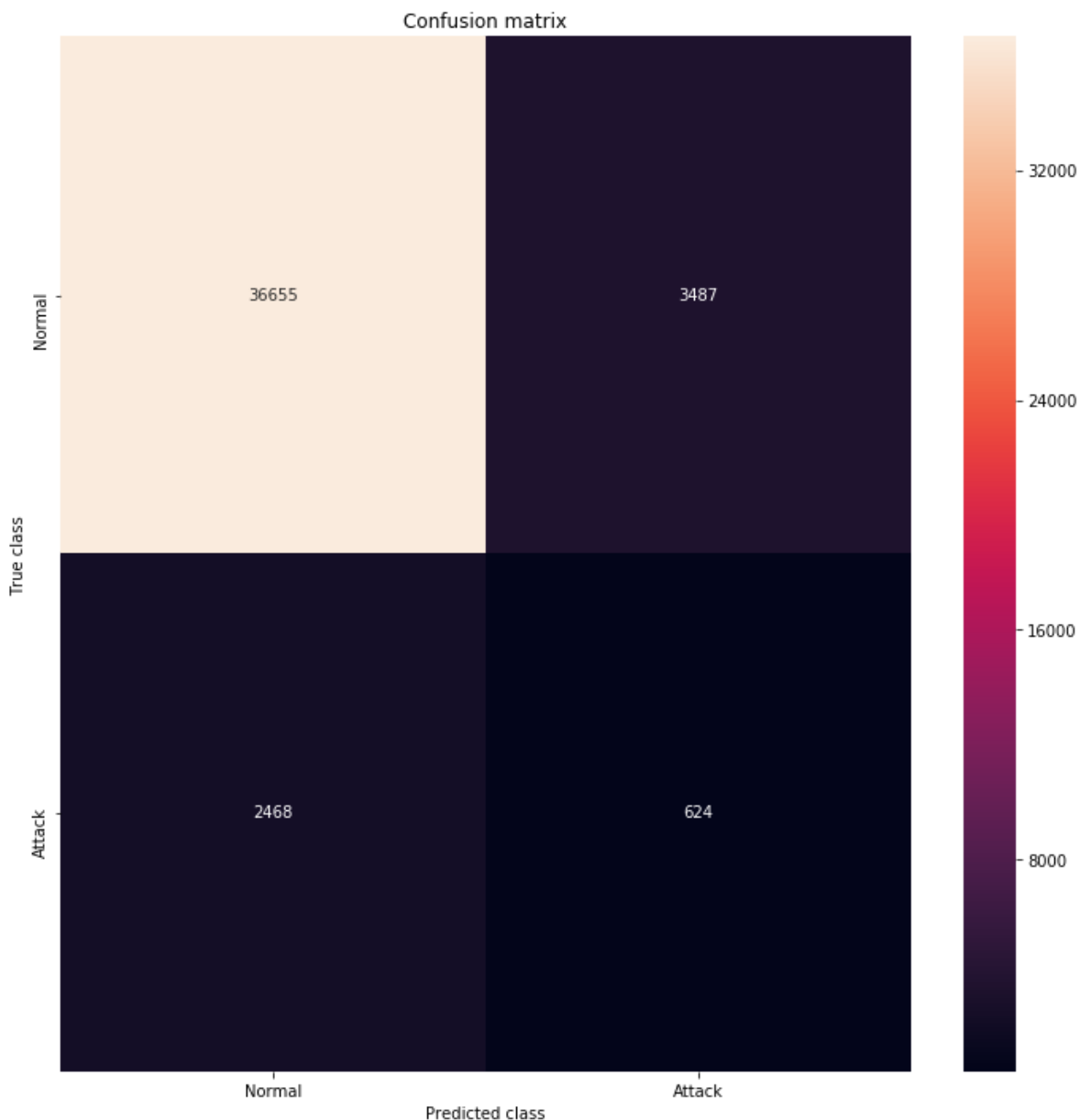
In [298]:

```
1 # ROC AUC
2 auc = roc_auc_score(y_test, yhat_probs)
3 print('ROC AUC: %f' % auc)
4
```

ROC AUC: 0.601735

In [299]:

```
1 # confusion matrix
2 import seaborn as sns
3 LABELS = ["Normal", "Attack"]
4
5 matrix = confusion_matrix(y_test, yhat_classes)
6 plt.figure(figsize=(12, 12))
7 sns.heatmap(matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d");
8 plt.title("Confusion matrix")
9 plt.ylabel('True class')
10 plt.xlabel('Predicted class')
11 plt.show()
```



Input/Output Data to LSTM for Sequence Prediction

In [36]:

```

1  #https://stackabuse.com/solving-sequence-problems-with-lstm-in-keras/
2  import numpy
3  numpy.set_printoptions(threshold=numpy.nan)
4
5  def int_to_onehot(n, n_classes):
6      v = [0] * n_classes
7      v[n] = 1
8      return v
9
10 def onehot_to_int(v):
11     return v.index(1)
12
13 X_train, y_train, X_test, y_test
14 import pprint
15
16 pprint.pprint(X_train[:, :, :])
17
18 # systemcall trace-1 length = 819,
19 # [6, 6, 63, 6, 42, 120, 6, 195, 120, 6, 6, 114, 114, 1, 1, 252, 252,
20 # 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 252, 252, 252, 252, 252, 252,
21 # 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 1, 1, 252, 1,
22 # 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 1, 1, 1, 1, 1, 1, 252,
23 # 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 1, 1, 1, 1, 1, 1,
24 # 1, 1, 1, 1, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
25 # 1, 252, 1, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252,
26 # 252, 252, 252, 252, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 252, 1,
27 # 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 252, 252, 1, 252, 1, 1, 1,
28 # 1, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 252, 252, 252, 252, 1, 1, 1, 1,
29 # 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
30 # 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
31 # 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
32 # 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
33 # 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 252, 252, 252, 252, 252,
34 # 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 1, 1, 1, 1, 1,
35 # 252, 252, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 1, 1, 1, 1, 1, 1, 1,
36 # 1, 1, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
37 # 1, 1, 1, 1, 1, 1, 252, 252, 252, 252, 252, 1, 1, 252, 1, 252, 252, 252,
38 # 252, 252, 1, 1, 252, 252, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
39 # 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
40 # 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
41 # 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 1, 1, 1, 1, 1,
42 # 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
43 # 1, 1, 1, 1, 252, 1, 1, 252, 1, 1, 252, 1, 1, 252, 252, 1, 1, 1, 1, 1, 1,
44 # 1, 1, 1, 252, 1, 1, 1, 1, 1, 1, 252, 252, 252, 1, 1, 1, 1, 1, 1, 1,
45 # 1, 1, 1, 1, 1, 1, 1, 1, 252, 1, 1, 1, 1, 1, 1, 252, 1, 1, 1, 1, 1, 1,
46 # 252, 1, 1, 1, 1, 1, 1, 252, 252, 1, 1, 1, 1, 1, 1, 252, 252, 252, 252, 1,
47 # 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 252, 252, 252, 252, 252,
48 # 252, 252, 252, 252, 252, 252, 252, 252, 252, 1, 252, 252, 252, 252, 252,
49 # 252, 252, 252, 1, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252,
50 # 252, 252, 252, 252, 252, 1, 252, 252, 1, 252, 252, 1, 1, 252, 252, 252,
51 # 1, 1, 252, 252, 252, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 252, 252, 252,
52 # 252, 252, 252, 1, 252, 252, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
53 # 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252,
54 # 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252,
55 # 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 1, 252, 252, 1,
56 # 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 1,
57 # 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 1, 1, 1, 1, 252, 252, 252, 252,
58 # 252, 252, 252, 1, 252, 1, 1, 252, 1, 1, 252, 1, 252, 252, 252, 252, 252,
59 # 252, 252, 252, 252, 252, 1, 252, 1, 1, 252, 1, 252, 252, 252, 1, 252,

```

[illegible]

In [38]:

```

1
2 # Sequence [114 ,162, 114, 114 ,162, 114, 162, 162]
3 # [X ->114, 162 ,114, 114 ,162, 114, 162 Y-> 162]
4
5 test_input = array([[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
6     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
7     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
8     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
9     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
10    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
11    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
12    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
13    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
14    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
15    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
16    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
17    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
18    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
19    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
20    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
21    0, 0, 0, 0, 0],
22    [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
23     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
24     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
25     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
26     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
27     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
28     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
29     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
30     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
31     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
32     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
33     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
34     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
35     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
36     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
37     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
38     0, 0, 0, 0, 0],
39    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
40     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
41     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
42     1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
43     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
44     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
45     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
46     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
47     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
48     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
49     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
50     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
51     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
52     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
53     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
54     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
55     0, 0, 0, 0, 0],
56    [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
57     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
58     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
59     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

```

47/49

NameError: name 'array' is not defined

In []:

```
1 # https://towardsdatascience.com/step-by-step-understanding-lstm-autoencoder-layers-ff  
2 # https://towardsdatascience.com/lstm-autoencoder-for-extreme-rare-event-classification
```