

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 [24]:

```
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 [25]:

```

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 [26]:

```

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/"
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, 12

```
8, 132, 133, 140, 141, 142, 143, 144, 146, 148, 155, 157, 158, 159, 160, 16
2, 163, 168, 172, 174, 175, 176, 179, 180, 183, 184, 185, 191, 192, 194, 19
5, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 21
1, 212, 213, 214, 219, 220, 221, 224, 226, 228, 229, 230, 231, 233, 234, 24
0, 242, 243, 252, 254, 255, 256, 258, 259, 260, 264, 265, 266, 268, 269, 27
0, 272, 289, 292, 293, 295, 298, 300, 301, 307, 308, 309, 311, 314, 320, 32
2, 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, 33,
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, 1
18, 119, 120, 122, 124, 125, 128, 132, 133, 136, 140, 141, 142, 143, 144, 14
6, 148, 150, 151, 154, 155, 156, 157, 158, 159, 160, 162, 163, 168, 172, 17
4, 175, 176, 177, 179, 180, 181, 183, 184, 185, 186, 187, 190, 191, 192, 19
4, 195, 196, 197, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 21
0, 211, 212, 213, 214, 215, 216, 219, 220, 221, 224, 226, 228, 229, 231, 23
4, 240, 243, 252, 254, 255, 256, 258, 259, 260, 264, 265, 266, 268, 269, 27
0, 272, 289, 292, 293, 295, 296, 298, 300, 301, 306, 307, 308, 309, 311, 31
4, 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 [5]:

```

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 [79]:

```

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...

Training...

Train on 14208 samples, validate on 6090 samples

Epoch 1/1

14208/14208 [=====] - 32s 2ms/step - loss: 2.7681

- acc: 0.2387 - val_loss: 2.8872 - val_acc: 0.1819

| Layer (type) | Output Shape | Param # |
|-----------------------|-----------------|---------|
| lstm_23 (LSTM) | (None, 19, 64) | 103936 |
| dropout_22 (Dropout) | (None, 19, 64) | 0 |
| lstm_24 (LSTM) | (None, 19, 256) | 328704 |
| dropout_23 (Dropout) | (None, 19, 256) | 0 |
| lstm_25 (LSTM) | (None, 100) | 142800 |
| dropout_24 (Dropout) | (None, 100) | 0 |
| dense_13 (Dense) | (None, 341) | 34441 |
| Total params: 609,881 | | |

Trainable params: 609,881
Non-trainable params: 0

Done Training...

In [7]:

```
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 [8]:

```

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...

```

[[1.2335530e-06 1.0198790e-03 8.9691122e-07 ... 1.0258395e-06
 1.3941190e-06 3.1251540e-05]
 [1.0490133e-06 1.0960293e-03 7.4805553e-07 ... 8.8167792e-07
 1.1079958e-06 2.8315437e-05]
 [1.5564030e-06 1.4356853e-03 1.0348892e-06 ... 1.2527308e-06
 1.4993083e-06 3.9803337e-05]
 ...
 [4.5564093e-06 1.5376091e-02 2.3351497e-06 ... 3.3488861e-06
 4.1597473e-06 1.1217311e-04]
 [3.9790325e-06 1.5034483e-02 2.0294226e-06 ... 2.9392420e-06
 3.6070917e-06 1.0151930e-04]
 [3.7426239e-06 1.4799395e-02 1.9216072e-06 ... 2.7753281e-06
 3.4382588e-06 9.7567863e-05]]

```

How did our model perform?

In [9]:

```

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 : 3.01

Validation Accuracy : 0.28

In [80]:

```

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 [37]:

```
1 history.history
```

Out[37]:

```
{'val_loss': [2.9082302657645718],
 'val_acc': [0.19474548491693677],
 'loss': [2.8591575310943096],
 'acc': [0.21107826600331595]}
```

In [81]:

```
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 systemcall sequence ??

In []:

```
1
```

Train LSTM simpler model

In [82]:

```
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 print(f"The number of hidden nodes is {hidden_nodes}.")
10
11 def build_model_2():
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(0.2))
17     model.add(Dense(units=output_labels))
18     model.add(Activation('softmax'))
19     model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
20     #print ("Compilation Time :"%(time.time() - start))
21     return model
```

The number of hidden nodes is 4000.

In [83]:

```

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 [=====] - 2387s 118ms/step - loss: 3.06

39 - acc: 0.2968 - val_loss: 2.7892 - val_acc: 0.4133

Epoch 2/10

20298/20298 [=====] - 2296s 113ms/step - loss: 2.15

57 - acc: 0.5167 - val_loss: 2.3318 - val_acc: 0.4857

Epoch 3/10

20298/20298 [=====] - 2240s 110ms/step - loss: 1.56

23 - acc: 0.6057 - val_loss: 2.3203 - val_acc: 0.5069

Epoch 4/10

20298/20298 [=====] - 2263s 112ms/step - loss: 1.39

43 - acc: 0.6450 - val_loss: 2.1959 - val_acc: 0.5048

Epoch 5/10

20298/20298 [=====] - 2293s 113ms/step - loss: 1.13

44 - acc: 0.6758 - val_loss: 2.1536 - val_acc: 0.5035

Epoch 6/10

20298/20298 [=====] - 2309s 114ms/step - loss: 1.03

04 - acc: 0.6982 - val_loss: 2.1643 - val_acc: 0.5033

Epoch 7/10

20298/20298 [=====] - 2341s 115ms/step - loss: 0.98

06 - acc: 0.7183 - val_loss: 2.2458 - val_acc: 0.5089

Epoch 8/10

20298/20298 [=====] - 2351s 116ms/step - loss: 0.89

93 - acc: 0.7389 - val_loss: 2.2913 - val_acc: 0.5078

Epoch 9/10

20298/20298 [=====] - 2289s 113ms/step - loss: 0.82

27 - acc: 0.7563 - val_loss: 2.2352 - val_acc: 0.5294

Epoch 10/10

20298/20298 [=====] - 2275s 112ms/step - loss: 0.74

31 - acc: 0.7794 - val_loss: 2.3909 - val_acc: 0.5145

| Layer (type) | Output Shape | Param # |
|---------------------------|--------------|----------|
| lstm_32 (LSTM) | (None, 4000) | 69472000 |
| dropout_28 (Dropout) | (None, 4000) | 0 |
| dense_20 (Dense) | (None, 341) | 1364341 |
| activation_6 (Activation) | (None, 341) | 0 |

Total params: 70,836,341

Trainable params: 70,836,341

Non-trainable params: 0

Done Training...

In [85]:

```

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

```

Score : 2.39

Validation Accuracy : 0.51

In [93]:

```

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[93]:

```

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 [85]:

```
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 [80]:

```

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 [=====] - 28s 1ms/step - loss: 0.0113 -

acc: 0.9971 - val_loss: 0.0097 - val_acc: 0.9973

Epoch 2/10

20298/20298 [=====] - 20s 975us/step - loss: 0.0085

- acc: 0.9973 - val_loss: 0.0089 - val_acc: 0.9976

Epoch 3/10

20298/20298 [=====] - 19s 953us/step - loss: 0.0075

- acc: 0.9975 - val_loss: 0.0083 - val_acc: 0.9977

Epoch 4/10

20298/20298 [=====] - 20s 993us/step - loss: 0.0070

- acc: 0.9977 - val_loss: 0.0079 - val_acc: 0.9978

Epoch 5/10

20298/20298 [=====] - 19s 935us/step - loss: 0.0066

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

Epoch 6/10

20298/20298 [=====] - 19s 949us/step - loss: 0.0063

- acc: 0.9979 - val_loss: 0.0078 - val_acc: 0.9978

Epoch 7/10

20298/20298 [=====] - 20s 966us/step - loss: 0.0061

- acc: 0.9979 - val_loss: 0.0079 - val_acc: 0.9978

Epoch 8/10

20298/20298 [=====] - 19s 939us/step - loss: 0.0059

- acc: 0.9980 - val_loss: 0.0080 - val_acc: 0.9978

Epoch 9/10

20298/20298 [=====] - 20s 965us/step - loss: 0.0058

- acc: 0.9981 - val_loss: 0.0080 - val_acc: 0.9978

Epoch 10/10

20298/20298 [=====] - 20s 979us/step - loss: 0.0056

- acc: 0.9981 - val_loss: 0.0080 - val_acc: 0.9977

| Layer (type) | Output Shape | Param # |
|---------------------------|--------------|---------|
| ===== | | |
| lstm_26 (LSTM) | (None, 100) | 176800 |
| ===== | | |
| dropout_25 (Dropout) | (None, 100) | 0 |
| ===== | | |
| dense_14 (Dense) | (None, 341) | 34441 |
| ===== | | |
| activation_8 (Activation) | (None, 341) | 0 |
| ===== | | |
| Total params: 211,241 | | |
| Trainable params: 211,241 | | |
| Non-trainable params: 0 | | |

Done Training...

In [81]:

```

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 [=====] - 25s 1ms/step - loss: 0.0113 -
acc: 0.9971 - val_loss: 0.0099 - val_acc: 0.9973

Epoch 2/10

20298/20298 [=====] - 20s 972us/step - loss: 0.0083
- acc: 0.9974 - val_loss: 0.0087 - val_acc: 0.9977

Epoch 3/10

20298/20298 [=====] - 20s 973us/step - loss: 0.0071
- acc: 0.9976 - val_loss: 0.0081 - val_acc: 0.9978

Epoch 4/10

20298/20298 [=====] - 20s 964us/step - loss: 0.0065
- acc: 0.9978 - val_loss: 0.0078 - val_acc: 0.9978

Epoch 5/10

20298/20298 [=====] - 19s 956us/step - loss: 0.0061

```

- acc: 0.9979 - val_loss: 0.0078 - val_acc: 0.9978
Epoch 6/10
20298/20298 [=====] - 19s 940us/step - loss: 0.0058
- acc: 0.9980 - val_loss: 0.0077 - val_acc: 0.9978
Epoch 7/10
20298/20298 [=====] - 20s 988us/step - loss: 0.0056
- acc: 0.9981 - val_loss: 0.0078 - val_acc: 0.9978
Epoch 8/10
20298/20298 [=====] - 19s 960us/step - loss: 0.0054
- acc: 0.9982 - val_loss: 0.0079 - val_acc: 0.9978
Epoch 9/10
20298/20298 [=====] - 19s 950us/step - loss: 0.0053
- acc: 0.9982 - val_loss: 0.0078 - val_acc: 0.9978
Epoch 10/10
20298/20298 [=====] - 19s 956us/step - loss: 0.0051
- acc: 0.9983 - val_loss: 0.0080 - val_acc: 0.9978

```

| Layer (type) | Output Shape | Param # |
|---------------------------|--------------|---------|
| ===== | | |
| lstm_27 (LSTM) | (None, 100) | 176800 |
| ===== | | |
| dropout_26 (Dropout) | (None, 100) | 0 |
| ===== | | |
| dense_15 (Dense) | (None, 341) | 34441 |
| ===== | | |
| activation_9 (Activation) | (None, 341) | 0 |
| ===== | | |

```

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

```

Done Training...

In [82]:

```

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 [=====] - 23s 1ms/step - loss: 0.0110 -

acc: 0.9971 - val_loss: 0.0096 - val_acc: 0.9975

Epoch 2/10

20298/20298 [=====] - 19s 945us/step - loss: 0.0079

- acc: 0.9975 - val_loss: 0.0085 - val_acc: 0.9977

Epoch 3/10

20298/20298 [=====] - 19s 950us/step - loss: 0.0068

- acc: 0.9977 - val_loss: 0.0079 - val_acc: 0.9978

Epoch 4/10

20298/20298 [=====] - 20s 963us/step - loss: 0.0062

- acc: 0.9979 - val_loss: 0.0081 - val_acc: 0.9978

Epoch 5/10

20298/20298 [=====] - 19s 928us/step - loss: 0.0059

```

- acc: 0.9980 - val_loss: 0.0079 - val_acc: 0.9977
Epoch 6/10
20298/20298 [=====] - 19s 920us/step - loss: 0.0056
- acc: 0.9981 - val_loss: 0.0079 - val_acc: 0.9978
Epoch 7/10
20298/20298 [=====] - 18s 898us/step - loss: 0.0054
- acc: 0.9982 - val_loss: 0.0080 - val_acc: 0.9978
Epoch 8/10
20298/20298 [=====] - 18s 909us/step - loss: 0.0052
- acc: 0.9982 - val_loss: 0.0082 - val_acc: 0.9977
Epoch 9/10
20298/20298 [=====] - 19s 926us/step - loss: 0.0050
- acc: 0.9983 - val_loss: 0.0081 - val_acc: 0.9977
Epoch 10/10
20298/20298 [=====] - 19s 928us/step - loss: 0.0049
- acc: 0.9983 - val_loss: 0.0083 - val_acc: 0.9978

```

| Layer (type) | Output Shape | Param # |
|----------------------------|--------------|---------|
| ===== | | |
| lstm_28 (LSTM) | (None, 100) | 176800 |
| ===== | | |
| dense_16 (Dense) | (None, 341) | 34441 |
| ===== | | |
| activation_10 (Activation) | (None, 341) | 0 |
| ===== | | |

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

Done Training...

In [83]:

```

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

```

Score : 0.01
 Validation Accuracy : 1.00

In [49]:

```

1  def preprocess():
2
3      arrayfile = "./array_test.pickle"
4      array = loadfrompickle(arrayfile)
5      #print(type(array))
6      #print(array)
7      x_train = array[:, :-1]
8      y_train = array[:, -1]
9
10     print ("The train data size is that ")
11     print (x_train.shape)
12     print (y_train.shape)
13     return (x_train,y_train)
14
15 def preprocess_val():
16
17     arrayfile = "./array_val.pickle"
18     array = loadfrompickle(arrayfile)
19     #print(type(array))
20     #print(array)
21     x_test = array[:, :-1]
22     y_test = array[:, -1]
23
24     print ("The validation data size is that ")
25     print (x_test.shape)
26     print (y_test.shape)
27     return (x_test,y_test)
28
29 def preprocess_attack():
30
31     arrayfile = "./array_attack.pickle"
32     array = loadfrompickle(arrayfile)
33     #print(type(array))
34     #print(array)
35     x_attack = array[:, :-1]
36     y_attack = array[:, -1]
37
38     print ("The attack data size is that ")
39     print (x_attack.shape)
40     print (y_attack.shape)
41     return (x_attack,y_test)
42
43
44
45 def convertToOneHot(vector, num_classes=None):
46     """
47     Converts an input 1-D vector of integers into an output
48     2-D array of one-hot vectors, where an i'th input value
49     of j will set a '1' in the i'th row, j'th column of the
50     output array.
51
52     Example:
53         v = np.array((1, 0, 4))
54         one_hot_v = convertToOneHot(v)
55         print one_hot_v
56
57         [[0 1 0 0 0]
58          [1 0 0 0 0]
59          [0 0 0 0 1]]

```

```

60     """
61
62     assert isinstance(vector, np.ndarray)
63     assert len(vector) > 0
64
65     if num_classes is None:
66         num_classes = np.max(vector)+1
67     else:
68         assert num_classes > 0
69         assert num_classes >= np.max(vector)
70
71     result = np.zeros(shape=(len(vector), num_classes))
72     result[np.arange(len(vector)), vector] = 1
73     return result.astype(int)
74
75     """
76     The num_class here is set as 341
77     """
78
79     #one function do one thing
80     def sequence_n_gram_parsing(alist,n_gram=10,num_class=341):
81         if len(alist) <= n_gram:
82             return alist
83
84         ans = []
85         for i in range(0,len(alist)-n_gram+1,1):
86             tmp = alist[i:i+n_gram]
87             oneHot = convertToOneHot(np.asarray(tmp), num_class)
88             #print(tmp)
89             #print(np.asarray(tmp))
90             #print(oneHot)
91             ans.append(oneHot)
92
93         #transform into nmup array
94         ans = np.array(ans)
95         return (ans)
96
97
98     def lists_of_list_into_big_matrix(allthelist,n_gram=10):
99
100         print("lists_of_list_into_big_matrix train")
101         print(len(allthelist))
102         array = sequence_n_gram_parsing(allthelist[0])
103         print(len(allthelist[0]))
104         print(allthelist[0])
105         print(len(array))
106         print(array)
107
108         for i in range(1,len(allthelist),1):
109
110             tmp = sequence_n_gram_parsing(allthelist[i])
111
112             #print ("tmp shape")
113             #print(tmp)
114             #print (len(tmp))
115
116             array = np.concatenate((array, tmp), axis=0)
117             #print(allthelist[i])
118             #print(array)
119
120             percent = (i+0.0)/len(allthelist)

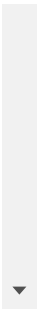
```

```

121     #io_helper.drawProgressBar(percent)
122     drawProgressBar(percent)
123
124     if (len(array)> 20000):
125         break
126     #print ("array shape")
127     #print (array.shape)
128     #print(len(allthelist[1]))
129     #print(allthelist[1])
130     #print(len(array))
131     #print(array)
132     #break
133
134     print (array.shape)
135     print ("done")
136     #io_helper.saveintopickle(array,"array_test.pickle")
137     saveintopickle(array,"array_test.pickle")
138
139
140 def lists_of_list_into_big_matrix_val(allthelist,n_gram=10):
141
142     print("lists_of_list_into_big_matrix validation")
143     print(len(allthelist))
144     array = sequence_n_gram_parsing(allthelist[0])
145     print(len(allthelist[0]))
146     print(allthelist[0])
147     print(len(array))
148     print(array)
149
150     for i in range(1,len(allthelist),1):
151         tmp = sequence_n_gram_parsing(allthelist[i])
152
153         # print ("tmp shape")
154         # print (tmp.shape)
155
156         array = np.concatenate((array, tmp), axis=0)
157
158
159         percent = (i+0.0)/len(allthelist)
160         #io_helper.drawProgressBar(percent)
161         drawProgressBar(percent)
162
163         if (len(array)> 20000):
164             break
165         #print ("array shape")
166         #print (array.shape)
167
168
169     print (array.shape)
170     print ("done")
171     #io_helper.saveintopickle(array,"array_test.pickle")
172     saveintopickle(array,"array_val.pickle")
173
174
175 def lists_of_list_into_big_matrix_attack(allthelist,n_gram=10):
176
177     array = sequence_n_gram_parsing(allthelist[0])
178
179     for i in range(1,len(allthelist),1):
180         tmp = sequence_n_gram_parsing(allthelist[i])
181

```

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



In [50]:

```

1 word_vec_length = 9
2 char_vec_length = 341
3 output_labels = 341
4 hidden_nodes = 100 # int(2/3 * (word_vec_length * char_vec_length))
5 epochs = 10
6 batch_size = 10
7
8 def build_model_6():
9     # Build the model
10    print('Build model...')
11    model = Sequential()
12    model.add(LSTM(hidden_nodes, return_sequences=False, input_shape=(word_vec_length,
13    #model.add(Dropout(0.2))
14    model.add(Dense(units=output_labels))
15    model.add(Activation('softmax'))
16    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
17    #print ("Compilation Time :"%(time.time() - start))
18    return model
19
20
21
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
37
38 batch_size=32
39 model = build_model_6()
40 print("Training...")
41 history = model.fit(
42     X_train, y_train,
43     batch_size=batch_size,
44     epochs=epochs,
45     validation_split=0.3,
46     callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)]
47 model.summary()
48 print("Done Training...")

```

Loading data...

The train data size is that

(20450, 9, 341)

(20450, 341)

X_train, y_train,shape

(20450, 9, 341)

(20450, 341)

Data Loaded. Compiling...

Build model...

Training...

Train on 14315 samples, validate on 6135 samples

Epoch 1/10

14315/14315 [=====] - 12s 821us/step - loss: 0.0108
- acc: 0.9972 - val_loss: 0.0100 - val_acc: 0.9972

Epoch 2/10

14315/14315 [=====] - 7s 481us/step - loss: 0.0078
- acc: 0.9975 - val_loss: 0.0090 - val_acc: 0.9974

Epoch 3/10

14315/14315 [=====] - 8s 535us/step - loss: 0.0070
- acc: 0.9977 - val_loss: 0.0086 - val_acc: 0.9974

Epoch 4/10

14315/14315 [=====] - 7s 498us/step - loss: 0.0064
- acc: 0.9978 - val_loss: 0.0081 - val_acc: 0.9976

Epoch 5/10

14315/14315 [=====] - 7s 501us/step - loss: 0.0059
- acc: 0.9980 - val_loss: 0.0079 - val_acc: 0.9976

Epoch 6/10

14315/14315 [=====] - 7s 494us/step - loss: 0.0056
- acc: 0.9981 - val_loss: 0.0076 - val_acc: 0.9977

Epoch 7/10

14315/14315 [=====] - 7s 493us/step - loss: 0.0054
- acc: 0.9982 - val_loss: 0.0074 - val_acc: 0.9977

Epoch 8/10

14315/14315 [=====] - 7s 466us/step - loss: 0.0052
- acc: 0.9982 - val_loss: 0.0075 - val_acc: 0.9977

Epoch 9/10

14315/14315 [=====] - 7s 484us/step - loss: 0.0050
- acc: 0.9983 - val_loss: 0.0074 - val_acc: 0.9977

Epoch 10/10

14315/14315 [=====] - 7s 486us/step - loss: 0.0049
- acc: 0.9983 - val_loss: 0.0074 - val_acc: 0.9977

| Layer (type) | Output Shape | Param # |
|---------------------------|--------------|---------|
| ===== | | |
| lstm_5 (LSTM) | (None, 100) | 176800 |
| ===== | | |
| dense_5 (Dense) | (None, 341) | 34441 |
| ===== | | |
| activation_5 (Activation) | (None, 341) | 0 |
| ===== | | |

Total params: 211,241

Trainable params: 211,241

Non-trainable params: 0

Done Training...

In [51]:

```
1 X_test, y_test = preprocess_val()
2
3 print ("X_test, y_test,shape")
4 print (X_test.shape)
5 print (y_test.shape)
6
7
8 print("Validating...")
9 predicted = model.predict(X_test)
10 print("Done Validating...")
11 print(predicted)
12
13 score, accuracy = model.evaluate(X_test, y_test, verbose=2, batch_size=batch_size)
14 print('Score : %.2f'%(score))
15 print('Validation Accuracy : %.2f'%(accuracy))
```

The validation data size is that

(20157, 9, 341)

(20157, 341)

X_test, y_test,shape

(20157, 9, 341)

(20157, 341)

Validating...

Done Validating...

IOPub data rate exceeded.

The notebook server will temporarily stop sending output to the client in order to avoid crashing it.

To change this limit, set the config variable

`--NotebookApp.iopub_data_rate_limit`.

Current values:

NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)

NotebookApp.rate_limit_window=3.0 (secs)

Score : 0.01

Validation Accuracy : 1.00

Input/Output Data to LSTM

In [44]:

```

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[:,1,:,:])
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, 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, 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, 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, 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, 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, 1, 252, 252, 252, 252, 252, 252,
48 # 252, 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, 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, 252, 252, 252, 252,
52 # 252, 252, 252, 1, 252, 252, 252, 1, 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]

[illegible]

In [37]:

[illegible]

In []:

```

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,

```

38/39

```

121     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
122     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
123     0, 0, 0, 0, 0],
124     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
125     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
126     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
127     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
128     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
129     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
130     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
131     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
132     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
133     0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
134     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
135     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
136     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
137     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
138     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
139     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
140     0, 0, 0, 0, 0],
141     [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
142     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
143     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
144     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
145     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
146     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
147     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
148     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
149     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
150     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
151     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
152     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
153     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
154     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
155     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
156     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
157     0, 0, 0, 0, 0]])
158
159 test_input = test_input.reshape((1, 19, 341))
160 test_output = model.predict(test_input, verbose=0)
161 print(test_output)

```