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, · · · , K}. Let x = x1x2 · · · xl(xi ∈ S) denote a sequence of l system calls.

LSTM Based Model :

1. At the Input Layer, the call at each time step xi 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 xi.

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]:

```
# -*- coding: utf-8 -*-
"""
Created on Thu Aug  1 13:52:35 2019

@author: kuna
"""

#!/usr/bin/env python
# -*- coding: utf-8 -*-


import pickle
import sys

# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
# ignore all user warnings
simplefilter(action='ignore', category=UserWarning)

def saveintopickle(obj, filename):
    with open(filename, 'wb') as handle:
        pickle.dump(obj, handle, protocol=pickle.HIGHEST_PROTOCOL)

    print ("[Pickle]: save object into {}".format(filename))
    return



def loadfrompickle(filename):
    with open(filename, 'rb') as handle:
        b = pickle.load(handle)
    return b



#draw the  process bar
def drawProgressBar(percent, barLen = 20):
    sys.stdout.write("\r")
    progress = ""
    for i in range(barLen):
        if i < int(barLen * percent):
            progress += "="
        else:
            progress += " "
    sys.stdout.write("[ %s ] %.2f%%" % (progress, percent * 100))
    sys.stdout.flush()
```

In [25]:

```
import numpy as np
#import io_helper


random_data_dup = 10  # each sample randomly duplicated between 0 and 9 times, see drop

def dropin(X, y):
    """
    The name suggests the inverse of dropout, i.e. adding more samples. See Data Augmer
    http://simaaron.github.io/Estimating-rainfall-from-weather-radar-readings-using-rec
    :param X: Each row is a training sequence
    :param y: Tne target we train and will later predict
    :return: new augmented X, y
    """
    print("X shape:", X.shape)
    print("y shape:", y.shape)
    X_hat = []
    y_hat = []
    for i in range(0, len(X)):
        for j in range(0, np.random.random_integers(0, random_data_dup)):
            X_hat.append(X[i, :])
            y_hat.append(y[i])
    return np.asarray(X_hat), np.asarray(y_hat)



def preprocess():

    arrayfile = "./array_test.pickle"
    array = loadfrompickle(arrayfile)
    #print(type(array))
    #print(array)
    x_train = array[:,:-1]
    y_train = array[:,-1]

    print ("The train data size is that ")
    print (x_train.shape)
    print (y_train.shape)
    return (x_train,y_train)

def preprocess_val():

    arrayfile = "./array_val.pickle"
    array = loadfrompickle(arrayfile)
    #print(type(array))
    #print(array)
    x_test = array[:,:-1]
    y_test = array[:,-1]

    print ("The train data size is that ")
    print (x_test.shape)
    print (y_test.shape)
    return (x_test,y_test)

#if __name__ =="__main__":
#   preprocess()
```

In [26]:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-


import os
import sys
import numpy as np

#import io_helper

def readfilesfromAdir(dataset):
    #read a list of files
    files = os.listdir(dataset)
    files_absolute_paths = []
    for i in files:
        files_absolute_paths.append(dataset+str(i))
    return files_absolute_paths


file = "ADFA-LD/Training_Data_Master/UTD-0001.txt"
#this is used to read a char sequence from
def readCharsFromFile(file):
    channel_values = open(file).read().split()
    #print (len(channel_values))
    #channel_values is a list
    return channel_values
    #print (channel_values[800:819])

def get_attack_subdir(path):
    subdirectories = os.listdir(path)
    for i in range(0,len(subdirectories)):
        subdirectories[i] = path + subdirectories[i]

    print (subdirectories)
    return (subdirectories)


def get_all_call_sequences(dire):
    files = readfilesfromAdir(dire)
    allthelist = []
    print (len(files))

    for eachfile in files:
        if not eachfile.endswith("DS_Store"):
            allthelist.append(readCharsFromFile(eachfile))
        else:
            print ("Skip the file "+ str(eachfile))

    elements = []
    for item in allthelist:
        for key in item:
            if key not in elements:
                elements.append(key)

    elements = map(int,elements)
    elements = sorted(elements)

    print ("The total unique elements:")
```

```
    print (elements)

    print ("The maximum number of elements:")
    print (max(elements))

    #print ("The length elements:")
    #print (len(elements))
    print (len(allthelist))

    #clean the all list data set
    _max = 0
    for i in range(0,len(allthelist)):
        _max = max(_max,len(allthelist[i]))
        allthelist[i] = list(map(int,allthelist[i]))
        #print(allthelist[i])


    print ("The maximum length of a sequence is that {}".format(_max))

    return (allthelist)

## shift the data for analysis
def shift(seq, n):
    n = n % len(seq)
    return seq[n:] + seq[:n]


def convertToOneHot(vector, num_classes=None):
    """
    Converts an input 1-D vector of integers into an output
    2-D array of one-hot vectors, where an i'th input value
    of j will set a '1' in the i'th row, j'th column of the
    output array.

    Example:
        v = np.array((1, 0, 4))
        one_hot_v = convertToOneHot(v)
        print one_hot_v

        [[0 1 0 0 0]
         [1 0 0 0 0]
         [0 0 0 0 1]]
    """

    assert isinstance(vector, np.ndarray)
    assert len(vector) > 0

    if num_classes is None:
        num_classes = np.max(vector)+1
    else:
        assert num_classes > 0
        assert num_classes >= np.max(vector)

    result = np.zeros(shape=(len(vector), num_classes))
    result[np.arange(len(vector)), vector] = 1
    return result.astype(int)

"""
The num_class here is set as 341
"""

```

```
#one function do one thing
def sequence_n_gram_parsing(alist,n_gram=20,num_class=341):
    if len(alist) <= n_gram:
        return alist

    ans = []
    for i in range(0,len(alist)-n_gram+1,1):
        tmp = alist[i:i+n_gram]
        oneHot = convertToOneHot(np.asarray(tmp), num_class)
        #print(tmp)
        #print(np.asarray(tmp))
        #print(oneHot)
        ans.append(oneHot)

    #transform into nmup arrray
    ans = np.array(ans)
    return (ans)


def lists_of_list_into_big_matrix(allthelist,n_gram=20):

    print("lists_of_list_into_big_matrix")
    print(len(allthelist))
    array = sequence_n_gram_parsing(allthelist[0])
    #print(len(allthelist[0]))
    #print(allthelist[0])
    #print(len(array))
    #print(array)

    for i in range(1,len(allthelist),1):

        tmp = sequence_n_gram_parsing(allthelist[i])

        #print ("tmp shape")
        #print(tmp)
        #print (len(tmp))

        array = np.concatenate((array, tmp), axis=0)
        #print(allthelist[i])
        #print(array)

        percent = (i+0.0)/len(allthelist)
        #io_helper.drawProgressBar(percent)
        drawProgressBar(percent)

        if (len(array)> 20000):
            break
        #print ("array shape")
        #print (array.shape)
        #print(len(allthelist[1]))
        #print(allthelist[1])
        #print(len(array))
        #print(array)
        #break

    print (array.shape)
    print ("done")
    #io_helper.saveintopickle(array,"array_test.pickle")
    saveintopickle(array,"array_test.pickle")


```

```
def lists_of_list_into_big_matrix_val(allthelist,n_gram=20):

    array = sequence_n_gram_parsing(allthelist[0])

    for i in range(1,len(allthelist),1):
        tmp = sequence_n_gram_parsing(allthelist[i])

       # print ("tmp shape")
       # print (tmp.shape)

        array = np.concatenate((array, tmp), axis=0)


        percent = (i+0.0)/len(allthelist)
        #io_helper.drawProgressBar(percent)
        drawProgressBar(percent)

        if (len(array)> 20000):
            break
        #print ("array shape")
        #print (array.shape)


    print (array.shape)
    print ("done")
    #io_helper.saveintopickle(array,"array_test.pickle")
    saveintopickle(array,"array_val.pickle")


if __name__ == "__main__":
    dirc = "ADFA-LD/Training_Data_Master/"
    dirc_val = "ADFA-LD/Validation_Data_Master/"
    dic_attack ="ADFA-LD/Attack_Data_Master/"
    #train1 = get_all_call_sequences(dirc)

    #test = [i for i in range(0,300)]
    #array = sequence_n_gram_parsing(test)
    #print (type(array))
    #print (array.shape)

    #get_attack_subdir(dic_attack)
    #print ("XxxxxxxXXXXXXXXXXX")
    #val1 = get_all_call_sequences(dirc_val)

    #dirc_test = "Test/"
    #att_test = get_all_call_sequences(dirc_test)
    #lists_of_list_into_big_matrix(att_test)

    att = get_all_call_sequences(dirc)
    lists_of_list_into_big_matrix(att)

    att_val = get_all_call_sequences(dirc_val)
    lists_of_list_into_big_matrix_val(att_val)

```

```
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]:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import matplotlib.pyplot as plt
import numpy as np
import time
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM
from keras.models import Sequential
from keras.models import model_from_json
from keras.layers.embeddings import Embedding

#import preprocess

# Global hyper-parameters
sequence_length = 19
epochs = 1
batch_size = 50
feature_dimension = 341
top_words = 5000

def save_model_weight_into_file(model, modelname="model.json", weight="model.h5"):
    model_json = model.to_json()
    with open(modelname, "w") as json_file:
        json_file.write(model_json)
    # serialize weights to HDF5
    model.save_weights(weight)
    print("Saved model to disk in {} and {}".format(modelname,weight))


def load_model_and_wieght_from_file(modelname="model.json", weight="model.h5"):

    json_file = open(modelname, 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    # load weights into new model
    loaded_model.load_weights(weight)
    print("Loaded model from disk, you can do more analysis more")

    pass


def build_model():
    model = Sequential()
    layers = {'input': feature_dimension, 'hidden1': 64, 'hidden2': 256, 'hidden3': 10

    model.add(LSTM(
            input_length=sequence_length,
            input_dim=layers['input'],
            output_dim=layers['hidden1'],
            return_sequences=True))
    model.add(Dropout(0.2))

    model.add(LSTM(
            layers['hidden2'],
            return_sequences=True))
    model.add(Dropout(0.2))

```

```
    model.add(LSTM(
            layers['hidden3'],
            return_sequences=False))
    model.add(Dropout(0.2))

    model.add(Dense(
            output_dim=layers['output'],activation='softmax'))
    #model.add(Activation("linear"))

    start = time.time()

    model.compile(loss="categorical_crossentropy", optimizer='rmsprop',  metrics=['acc
    #model.compile(loss="mse", optimizer="rmsprop")

    #print ("Compilation Time : "%(time.time() - start))
    return model

from keras.callbacks import EarlyStopping

def run_network(model=None, data=None):

    global_start_time = time.time()

    if data is None:
        print ('Loading data... ')
        # train on first 700 samples and test on next 300 samples (has anomaly)
        X_train, y_train  = preprocess()
    else:
        X_train, y_train = data

    print ("X_train, y_train,shape")
    print (X_train.shape)
    print (y_train.shape)
    print ('\nData Loaded. Compiling...\n')

    if model is None:
        model = build_model()
        #model = build_model_2()
        print("Training...")
        model.fit(
                X_train, y_train,
                batch_size=batch_size,
                epochs=epochs,
                validation_split=0.3)
        model.summary()
        print("Done Training...")

    #predicted = model.predict(X_test)
    #print("Reshaping predicted")
    #predicted = np.reshape(predicted, (predicted.size,))




    """
    except KeyboardInterrupt:
        print("prediction exception")
        print 'Training duration (s) : ', time.time() - global_start_time
        return model, y_test, 0

    try:
```

```
        plt.figure(1)
        plt.subplot(311)
        plt.title("Actual Test Signal w/Anomalies")
        plt.plot(y_test[:len(y_test)], 'b')
        plt.subplot(312)
        plt.title("Predicted Signal")
        plt.plot(predicted[:len(y_test)], 'g')
        plt.subplot(313)
        plt.title("Squared Error")
        mse = ((y_test - predicted) ** 2)
        plt.plot(mse, 'r')
        plt.show()
    except Exception as e:
        print("plotting exception")
        print (str(e))
    print ('Training duration (s) : '% (time.time() - global_start_time))

    return model, y_test, predicted
   """

#if __name__ == "__main__":
# run_network()
```

```
Using TensorFlow backend.
```

Train LSTM Model

In [79]:

```
global_start_time = time.time()

model=None

print ('Loading data... ')
# train on first 700 samples and test on next 300 samples (has anomaly)
X_train, y_train  = preprocess()

print ("X_train, y_train,shape")
print (X_train.shape)
print (y_train.shape)
print ('\nData Loaded. Compiling...\n')

if model is None:
    model = build_model()
    print("Training...")
    history = model.fit(
            X_train, y_train,
            batch_size=batch_size,
            epochs=epochs,
            validation_split=0.3,
            callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)
    model.summary()
    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]:

```
#import pandas as pd

#def loadData(file):
    # for reading also binary mode is important
#    dbfile = open(file, 'rb')
#    db = pickle.load(dbfile)
#    for keys in db:
#        print(keys, '=>', db[keys])
#    dbfile.close()

#if __name__ == '__main__':
#    loadData("./array_test.pickle")
#df_val = pd.read_pickle("./array_val.pickle")
#df_val.head()
```

Run model on Validation Data

In [8]:

```
# https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd1

X_test, y_test = preprocess_val()

print ("X_test, y_test,shape")
print (X_test.shape)
print (y_test.shape)

print("Validating...")
predicted = model.predict(X_test)
print("Done Validating...")
print(predicted)

```

```
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]:

```

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

```
Score : 3.01
Validation Accuracy : 0.28
```

In [80]:

```
#plt.title('Loss')
#plt.plot(history.history['loss'], label='train')
#plt.plot(history.history['val_loss'], label='test')
#plt.legend()
#plt.show();
```

In [37]:

```
history.history
```

Out[37]:

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

In [81]:

```
#plt.title('Accuracy')
#plt.plot(history.history['acc'], label='train')
#plt.plot(history.history['val_acc'], label='test')
#plt.legend()
#plt.show();
```

How to Test with new systemcall sequence ??

In [ ]:

```

```

Train LSTM simpler model

In [82]:

```
# https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-

word_vec_length = 19
char_vec_length = 341
output_labels = 341


hidden_nodes = 4000 # int(2/3 * (word_vec_length * char_vec_length))
print(f"The number of hidden nodes is {hidden_nodes}.")

def build_model_2():
    # Build the model
    print('Build model...')
    model = Sequential()
    model.add(LSTM(hidden_nodes, return_sequences=False, input_shape=(word_vec_length,
    model.add(Dropout(0.2))
    model.add(Dense(units=output_labels))
    model.add(Activation('softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
    #print ("Compilation Time : "%(time.time() - start))
    return model
```

```
The number of hidden nodes is 4000.
```

In [83]:

```
global_start_time = time.time()

model=None

print ('Loading data... ')
# train on first 700 samples and test on next 300 samples (has anomaly)
X_train, y_train  = preprocess()

print ("X_train, y_train,shape")
print (X_train.shape)
print (y_train.shape)
print ('\nData Loaded. Compiling...\n')

batch_size=32
model = build_model_2()
print("Training...")
model.fit(X_train, y_train, batch_size=batch_size, epochs=10, validation_data=(X_test,
model.summary()
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 [==============================] - 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]:

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

```
Score : 2.39
Validation Accuracy : 0.51
```

In [93]:

```
## k-fold validation
from sklearn.model_selection import StratifiedKFold
import numpy

# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)

# split into input (X) and output (Y) variables
X = X_train
Y = y_train
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]:

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

In [80]:

```
# https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-

word_vec_length = 19
char_vec_length = 341
output_labels = 341


hidden_nodes = 100 # int(2/3 * (word_vec_length * char_vec_length))
print(f"The number of hidden nodes is {hidden_nodes}.")

def build_model_3():
    # Build the model
    print('Build model...')
    model = Sequential()
    model.add(LSTM(hidden_nodes, return_sequences=False, input_shape=(word_vec_length,
    model.add(Dropout(0.5))
    model.add(Dense(units=output_labels))
    model.add(Activation('softmax'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
    #print ("Compilation Time : "%(time.time() - start))
    return model

global_start_time = time.time()

model=None

print ('Loading data... ')
# train on first 700 samples and test on next 300 samples (has anomaly)
X_train, y_train  = preprocess()

print ("X_train, y_train,shape")
print (X_train.shape)
print (y_train.shape)
print ('\nData Loaded. Compiling...\n')

batch_size=32
model = build_model_3()
print("Training...")
model.fit(X_train, y_train, batch_size=batch_size, epochs=10, validation_data=(X_test,
model.summary()
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]:

```
def build_model_4():
    # Build the model
    print('Build model...')
    model = Sequential()
    model.add(LSTM(hidden_nodes, return_sequences=False, input_shape=(word_vec_length,
    model.add(Dropout(0.2))
    model.add(Dense(units=output_labels))
    model.add(Activation('softmax'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
    #print ("Compilation Time : "%(time.time() - start))
    return model

global_start_time = time.time()

model=None

print ('Loading data... ')
# train on first 700 samples and test on next 300 samples (has anomaly)
X_train, y_train  = preprocess()

print ("X_train, y_train,shape")
print (X_train.shape)
print (y_train.shape)
print ('\nData Loaded. Compiling...\n')

batch_size=32
model = build_model_4()
print("Training...")
model.fit(X_train, y_train, batch_size=batch_size, epochs=10, validation_data=(X_test,
model.summary()
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]:

```
def build_model_5():
    # Build the model
    print('Build model...')
    model = Sequential()
    model.add(LSTM(hidden_nodes, return_sequences=False, input_shape=(word_vec_length,
    #model.add(Dropout(0.2))
    model.add(Dense(units=output_labels))
    model.add(Activation('softmax'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
    #print ("Compilation Time : "%(time.time() - start))
    return model

global_start_time = time.time()

model=None

print ('Loading data... ')
# train on first 700 samples and test on next 300 samples (has anomaly)
X_train, y_train  = preprocess()

print ("X_train, y_train,shape")
print (X_train.shape)
print (y_train.shape)
print ('\nData Loaded. Compiling...\n')

batch_size=32
model = build_model_5()
print("Training...")
model.fit(X_train, y_train, batch_size=batch_size, epochs=10, validation_data=(X_test,
model.summary()
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]:

```
def preprocess():

    arrayfile = "./array_test.pickle"
    array = loadfrompickle(arrayfile)
    #print(type(array))
    #print(array)
    x_train = array[:,:-1]
    y_train = array[:,-1]

    print ("The train data size is that ")
    print (x_train.shape)
    print (y_train.shape)
    return (x_train,y_train)

def preprocess_val():

    arrayfile = "./array_val.pickle"
    array = loadfrompickle(arrayfile)
    #print(type(array))
    #print(array)
    x_test = array[:,:-1]
    y_test = array[:,-1]

    print ("The validation data size is that ")
    print (x_test.shape)
    print (y_test.shape)
    return (x_test,y_test)

def preprocess_attack():

    arrayfile = "./array_attack.pickle"
    array = loadfrompickle(arrayfile)
    #print(type(array))
    #print(array)
    x_attack = array[:,:-1]
    y_attack = array[:,-1]

    print ("The attack data size is that ")
    print (x_attack.shape)
    print (y_attack.shape)
    return (x_attack,y_test)



def convertToOneHot(vector, num_classes=None):
    """
    Converts an input 1-D vector of integers into an output
    2-D array of one-hot vectors, where an i'th input value
    of j will set a '1' in the i'th row, j'th column of the
    output array.

    Example:
        v = np.array((1, 0, 4))
        one_hot_v = convertToOneHot(v)
        print one_hot_v

        [[0 1 0 0 0]
         [1 0 0 0 0]
         [0 0 0 0 1]]
```

```
    """

    assert isinstance(vector, np.ndarray)
    assert len(vector) > 0

    if num_classes is None:
        num_classes = np.max(vector)+1
    else:
        assert num_classes > 0
        assert num_classes >= np.max(vector)

    result = np.zeros(shape=(len(vector), num_classes))
    result[np.arange(len(vector)), vector] = 1
    return result.astype(int)

"""
The num_class here is set as 341
"""

#one function do one thing
def sequence_n_gram_parsing(alist,n_gram=10,num_class=341):
    if len(alist) <= n_gram:
        return alist

    ans = []
    for i in range(0,len(alist)-n_gram+1,1):
        tmp = alist[i:i+n_gram]
        oneHot = convertToOneHot(np.asarray(tmp), num_class)
        #print(tmp)
        #print(np.asarray(tmp))
        #print(oneHot)
        ans.append(oneHot)

    #transform into nmup arrray
    ans = np.array(ans)
    return (ans)


def lists_of_list_into_big_matrix(allthelist,n_gram=10):

    print("lists_of_list_into_big_matrix train")
    print(len(allthelist))
    array = sequence_n_gram_parsing(allthelist[0])
    print(len(allthelist[0]))
    print(allthelist[0])
    print(len(array))
    print(array)

    for i in range(1,len(allthelist),1):

        tmp = sequence_n_gram_parsing(allthelist[i])

        #print ("tmp shape")
        #print(tmp)
        #print (len(tmp))

        array = np.concatenate((array, tmp), axis=0)
        #print(allthelist[i])
        #print(array)

        percent = (i+0.0)/len(allthelist)
```

```
        #io_helper.drawProgressBar(percent)
        drawProgressBar(percent)

        if (len(array)> 20000):
            break
        #print ("array shape")
        #print (array.shape)
        #print(len(allthelist[1]))
        #print(allthelist[1])
        #print(len(array))
        #print(array)
        #break

    print (array.shape)
    print ("done")
    #io_helper.saveintopickle(array,"array_test.pickle")
    saveintopickle(array,"array_test.pickle")


def lists_of_list_into_big_matrix_val(allthelist,n_gram=10):

    print("lists_of_list_into_big_matrix validation")
    print(len(allthelist))
    array = sequence_n_gram_parsing(allthelist[0])
    print(len(allthelist[0]))
    print(allthelist[0])
    print(len(array))
    print(array)

    for i in range(1,len(allthelist),1):
        tmp = sequence_n_gram_parsing(allthelist[i])

       # print ("tmp shape")
       # print (tmp.shape)

        array = np.concatenate((array, tmp), axis=0)


        percent = (i+0.0)/len(allthelist)
        #io_helper.drawProgressBar(percent)
        drawProgressBar(percent)

        if (len(array)> 20000):
            break
        #print ("array shape")
        #print (array.shape)


    print (array.shape)
    print ("done")
    #io_helper.saveintopickle(array,"array_test.pickle")
    saveintopickle(array,"array_val.pickle")


def lists_of_list_into_big_matrix_attack(allthelist,n_gram=10):

    array = sequence_n_gram_parsing(allthelist[0])

    for i in range(1,len(allthelist),1):
        tmp = sequence_n_gram_parsing(allthelist[i])

```

```
        # print ("tmp shape")
        # print (tmp.shape)

         array = np.concatenate((array, tmp), axis=0)


         percent = (i+0.0)/len(allthelist)
         #io_helper.drawProgressBar(percent)
         drawProgressBar(percent)

         if (len(array)> 20000):
             break
         #print ("array shape")
         #print (array.shape)


    print (array.shape)
    print ("done")
    #io_helper.saveintopickle(array,"array_test.pickle")
    saveintopickle(array,"array_attack.pickle")

if __name__ == "__main__":
    dirc = "ADFA-LD/Training_Data_Master/"
    dirc_val = "ADFA-LD/Validation_Data_Master/"
    dic_attack ="ADFA-LD/Attack_Data_Master_All/"
    #train1 = get_all_call_sequences(dirc)

    #test = [i for i in range(0,300)]
    #array = sequence_n_gram_parsing(test)
    #print (type(array))
    #print (array.shape)

    #get_attack_subdir(dic_attack)
    #print ("XxxxxxxXXXXXXXXXXX")
    #val1 = get_all_call_sequences(dirc_val)

    #dirc_test = "Test/"
    #att_test = get_all_call_sequences(dirc_test)
    #lists_of_list_into_big_matrix(att_test)

    att = get_all_call_sequences(dirc)
    lists_of_list_into_big_matrix(att)

    att_val = get_all_call_sequences(dirc_val)
    lists_of_list_into_big_matrix_val(att_val)

    #att_attack = get_all_call_sequences(dic_attack)
    #lists_of_list_into_big_matrix_attack(att_attack)

```

In [50]:

```
word_vec_length = 9
char_vec_length = 341
output_labels = 341
hidden_nodes = 100 # int(2/3 * (word_vec_length * char_vec_length))
epochs = 10
batch_size = 10

def build_model_6():
    # Build the model
    print('Build model...')
    model = Sequential()
    model.add(LSTM(hidden_nodes, return_sequences=False, input_shape=(word_vec_length,
    #model.add(Dropout(0.2))
    model.add(Dense(units=output_labels))
    model.add(Activation('softmax'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
    #print ("Compilation Time : "%(time.time() - start))
    return model




global_start_time = time.time()

model=None

print ('Loading data... ')
# train on first 700 samples and test on next 300 samples (has anomaly)
X_train, y_train  = preprocess()

print ("X_train, y_train,shape")
print (X_train.shape)
print (y_train.shape)
print ('\nData Loaded. Compiling...\n')



batch_size=32
model = build_model_6()
print("Training...")
history = model.fit(
            X_train, y_train,
            batch_size=batch_size,
            epochs=epochs,
            validation_split=0.3,
            callbacks=[EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)
model.summary()
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]:

```
X_test, y_test = preprocess_val()

print ("X_test, y_test,shape")
print (X_test.shape)
print (y_test.shape)


print("Validating...")
predicted = model.predict(X_test)
print("Done Validating...")
print(predicted)

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

```
#https://stackabuse.com/solving-sequence-problems-with-lstm-in-keras/
import numpy
numpy.set_printoptions(threshold=numpy.nan)

def int_to_onehot(n, n_classes):
    v = [0] * n_classes
    v[n] = 1
    return v

def onehot_to_int(v):
    return v.index(1)

X_train, y_train, X_test, y_test
import pprint

pprint.pprint(X_train[:1,:,:])

# systemcall trace-1 length = 819,
# [6, 6, 63, 6, 42, 120, 6, 195, 120, 6, 6, 114, 114, 1, 1, 252, 252,
# 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 252, 252, 252, 252, 252, 252,
# 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 1, 1, 252, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 1, 1, 1, 1, 1, 1, 252,
# 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 1, 1, 1, 1, 1, 1,
# 1, 1, 1, 1, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
# 1, 252, 1, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252,
# 252, 252, 252, 252, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 252, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 252, 252, 1, 252, 1, 1, 1,
# 1, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 252, 252, 252, 252, 1, 1, 1, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 252, 252, 252, 252, 252,
# 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 1, 1, 1, 1, 1,
# 252, 252, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 1, 1, 1, 1, 1, 1, 1, 1,
# 1, 1, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
# 1, 1, 1, 1, 1, 1, 252, 252, 252, 252, 252, 1, 1, 252, 1, 252, 252, 252,
# 252, 252, 1, 1, 252, 252, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 1, 1, 1, 1, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
# 1, 1, 1, 1, 252, 1, 1, 252, 1, 1, 252, 1, 1, 252, 252, 1, 1, 1, 1, 1, 1,
# 1, 1, 1, 252, 1, 1, 1, 1, 1, 1, 252, 252, 252, 1, 1, 1, 1, 1, 1, 1, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 252, 1, 1, 1, 1, 1, 252, 1, 1, 1, 1, 1, 1, 1,
# 252, 1, 1, 1, 1, 1, 1, 252, 252, 1, 1, 1, 1, 1, 252, 252, 252, 252, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 252, 252, 252, 252, 252, 252,
# 252, 252, 252, 252, 252, 252, 252, 252, 252, 1, 252, 252, 252, 252, 252,
# 252, 252, 252, 1, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252,
# 252, 252, 252, 252, 252, 1, 252, 252, 1, 252, 252, 1, 1, 252, 252, 252,
# 1, 1, 252, 252, 252, 252, 1, 1, 1, 1, 1, 1, 1, 1, 252, 252, 252, 252,
# 252, 252, 252, 1, 252, 252, 252, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252,
# 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252,
# 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 1, 252, 252, 1,
# 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 1,
# 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 252, 1, 1, 1, 1, 252, 252, 252, 252,
# 252, 252, 252, 1, 252, 1, 1, 252, 1, 1, 252, 1, 252, 252, 252, 252, 252,
# 252, 252, 252, 252, 252, 1, 252, 1, 1, 252, 1, 252, 252, 252, 1, 252,
```

```
# 252, 252, 1, 1, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252, 252,
# 252, 252, 252, 1, 1, 252]

```

```
array([[[0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0],
        [0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0],
        [0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0],
       [0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
         0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0]]])
```

In [37]:

```
# Sequence [6, 6, 63, 6, 42, 120, 6, 195, 120, 6]
# [X -> 6, 6, 63, 6, 42, 120, 6, 195, 120, Y-> 6]
pprint.pprint(y_train[:1,:])
```

```
array([[0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

In [ ]:

```

# Sequence [114 ,162, 114, 114 ,162, 114, 162, 162]
# [X ->114, 162 ,114, 114 ,162, 114, 162  Y-> 162]

test_input = array([[[0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0],
         [0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0],
         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0],
         [0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
 60          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 61          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 62          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 63          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 64          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 65          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 66          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 67          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 68          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 69          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 70          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 71          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 72          0, 0, 0, 0, 0],
 73         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 74          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 75          1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 76          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 77          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 78          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 79          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 80          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 81          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 82          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 83          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 84          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 85          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 86          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 87          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 88          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 89          0, 0, 0, 0, 0],
 90         [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 91          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 92          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 93          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 94          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 95          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
 96          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 97          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 98          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 99          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
100          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
101          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
102          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
103          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
104          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
105          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
106          0, 0, 0, 0, 0],
107         [0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
108          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
109          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
110          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
111          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
112          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
113          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
114          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
115          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
116          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
117          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
118          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
119          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
120          0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 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, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0]]])

test_input = test_input.reshape((1, 19, 341))
test_output = model.predict(test_input, verbose=0)
print(test_output)
```