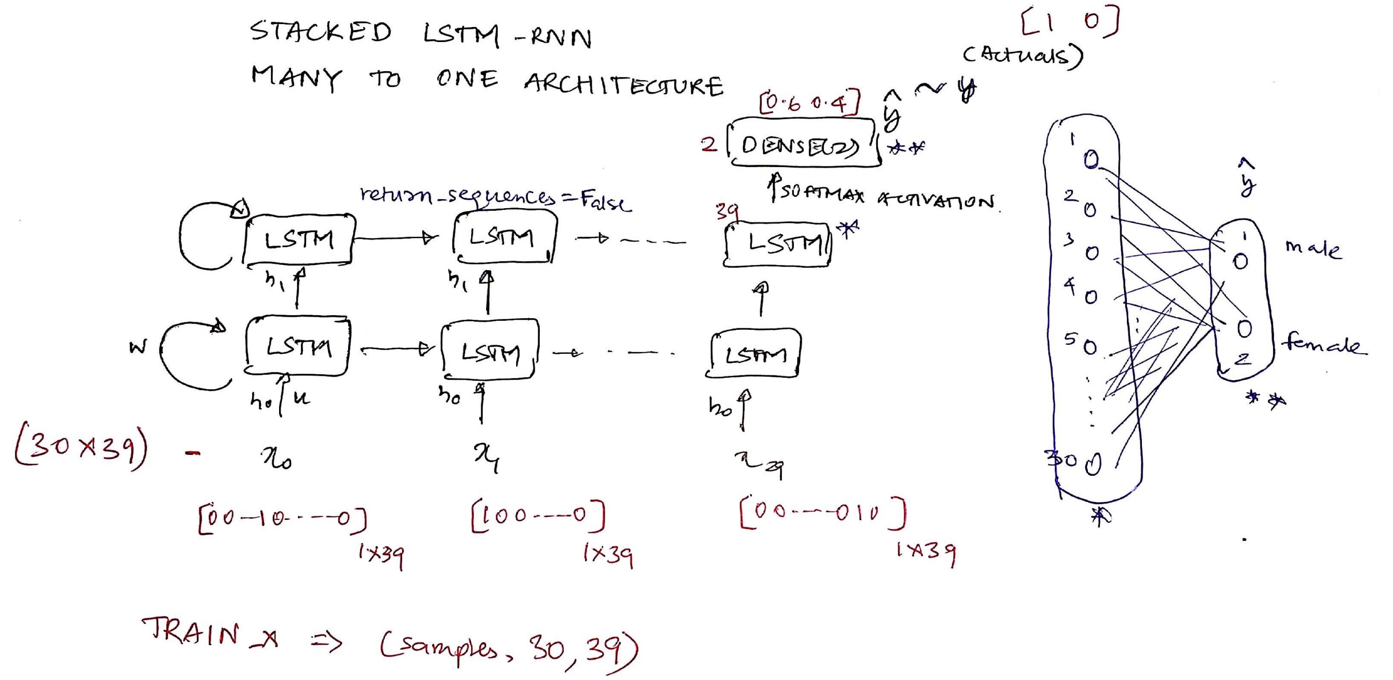
Deep Learning gender from name - RNN LSTMs

we will use an LSTM RNN to learn gender as f(name). we will use a stacked LSTM with many-to-one architecture feeding charecter inputs and predicting a binary outcome M/F. loss function used will be binary_crossentropy (a special case of categorical_crossentropy with m=2) and using adam optimizer (modified SGD) sample input /output would like this
['r','a','k','e','s','h',' '] - male
['p','r','a','d','e','e','p'] - male
['g','a','n','g','a',' '] - female
and so on...



regexp applied [^a-zA-Z0-9 ,.\r\n] = remove [ ]+ = ' ' [^a-zA-Z ,.\r\n] = remove [ ]{3}+ - regex to check where 3 consecutive space occurs.

In [199]:
```
from __future__ import print_function

from sklearn.preprocessing import OneHotEncoder
from keras.layers.core import Dense, Activation, Dropout
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense, Embedding
from keras.layers import LSTM
from keras.datasets import imdb
import pandas as pd
import numpy as np
import os
```

In [86]:
```
#parameters
maxlen = 30
labels = 2
```

In [158]:
```
input = pd.read_csv("gender_data.csv",header=None)
input.columns = ['name','m_or_f']
input['namelen']= [len(str(i)) for i in input['name']]
input1 = input[(input['namelen'] >= 2) ]
```

In [159]:
```
input1.groupby('m_or_f')['name'].count()
```

Out[159]:
```
m_or_f
f    6705
m    8475
Name: name, dtype: int64
```

In [160]:
```
names = input['name']
gender = input['m_or_f']
vocab = set(' '.join([str(i) for i in names]))
vocab.add('END')
len_vocab = len(vocab)
```

In [161]:
```
print(vocab)
print("vocab length is ",len_vocab)
print ("length of input is ",len(input1))
```

```
set([' ', '.', '1', '0', '3', '2', '5', '4', '7', '6', '9', '8', 'END', 'a',
 'c', 'b', 'e', 'd', 'g', 'f', 'i', 'h', 'k', 'j', 'm', 'l', 'o', 'n', 'q',
 'p', 's', 'r', 'u', 't', 'w', 'v', 'y', 'x', 'z'])
vocab length is  39
length of input is  15226
```

In [162]:
```
char_index = dict((c, i) for i, c in enumerate(vocab))
```

```
In [163]: print(char_index)
```

```
{' ': 0, '.': 1, '1': 2, '0': 3, '3': 4, '2': 5, '5': 6, '4': 7, '7': 8, '6':
9, '9': 10, '8': 11, 'END': 12, 'a': 13, 'c': 14, 'b': 15, 'e': 16, 'd': 17,
 'g': 18, 'f': 19, 'i': 20, 'h': 21, 'k': 22, 'j': 23, 'm': 24, 'l': 25, 'o':
26, 'n': 27, 'q': 28, 'p': 29, 's': 30, 'r': 31, 'u': 32, 't': 33, 'w': 34,
 'v': 35, 'y': 36, 'x': 37, 'z': 38}
```

```
In [164]: #train test split
msk = np.random.rand(len(input1)) < 0.8
train = input1[msk]
test = input1[~msk]
```

```
In [165]: #take input upto max and truncate rest
#encode to vector space(one hot encoding)
#padd 'END' to shorter sequences
train_X = []
trunc_train_name = [str(i)[0:30] for i in train.name]
for i in trunc_train_name:
    tmp = [char_index[j] for j in str(i)]
    for k in range(0,maxlen - len(str(i))):
        tmp.append(char_index["END"])
    train_X.append(tmp)
```

```
In [166]: np.asarray(train_X).shape
```

```
Out[166]: (12198, 30)
```

```
In [179]: def set_flag(i):
    tmp = np.zeros(39);
    tmp[i] = 1
    return(tmp)
```

```
In [184]: set_flag(3)
```

```
Out[184]: array([ 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.])
```

modify the code above to also convert each index to one-hot encoded representation

In [195]:

```
#take input upto max and truncate rest
#encode to vector space(one hot encoding)
#padd 'END' to shorter sequences
#also convert each index to one-hot encoding
train_X = []
train_Y = []
trunc_train_name = [str(i)[0:maxlen] for i in train.name]
for i in trunc_train_name:
    tmp = [set_flag(char_index[j]) for j in str(i)]
    for k in range(0,maxlen - len(str(i))):
        tmp.append(set_flag(char_index["END"]))
    train_X.append(tmp)
for i in train.m_or_f:
    if i == 'm':
        train_Y.append([1,0])
    else:
        train_Y.append([0,1])
```

In [196]:

```
np.asarray(train_X).shape
```

Out[196]:

```
(12198, 30, 39)
```

In [197]:

```
np.asarray(train_Y).shape
```

Out[197]:

```
(12198, 2)
```

build model in keras (a stacked LSTM model with many-to-one arch) here 30 sequence and 2 output each for one category(m/f)

In [212]:

```
#build the model: 2 stacked LSTM
print('Build model...')
model = Sequential()
model.add(LSTM(512, return_sequences=True, input_shape=(maxlen,len_vocab)))
model.add(Dropout(0.2))
model.add(LSTM(512, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(2))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam',metrics=['accu
racy'])
```

```
Build model...
```

In [206]:

```
test_X = []
test_Y = []
trunc_test_name = [str(i)[0:maxlen] for i in test.name]
for i in trunc_test_name:
    tmp = [set_flag(char_index[j]) for j in str(i)]
    for k in range(0,maxlen - len(str(i))):
        tmp.append(set_flag(char_index["END"]))
    test_X.append(tmp)
for i in test.m_or_f:
    if i == 'm':
        test_Y.append([1,0])
    else:
        test_Y.append([0,1])
```

In [207]:

```
print(np.asarray(test_X).shape)
print(np.asarray(test_Y).shape)
```

```
(3028, 30, 39)
(3028, 2)
```

In [215]:

```
batch_size=1000
model.fit(train_X, train_Y,batch_size=batch_size,nb_epoch=10,validation_data=
(test_X, test_Y))
```

```
Train on 12198 samples, validate on 3028 samples
Epoch 1/10
12198/12198 [==============================] - 146s - loss: 0.5867 - acc: 0.6
849 - val_loss: 0.5630 - val_acc: 0.7081
Epoch 2/10
12198/12198 [==============================] - 145s - loss: 0.5312 - acc: 0.7
336 - val_loss: 0.5880 - val_acc: 0.6909
Epoch 3/10
12198/12198 [==============================] - 145s - loss: 0.5217 - acc: 0.7
395 - val_loss: 0.4982 - val_acc: 0.7576
Epoch 4/10
12198/12198 [==============================] - 145s - loss: 0.4866 - acc: 0.7
620 - val_loss: 0.4823 - val_acc: 0.7652
Epoch 5/10
12198/12198 [==============================] - 145s - loss: 0.4682 - acc: 0.7
791 - val_loss: 0.4918 - val_acc: 0.7632
Epoch 6/10
12198/12198 [==============================] - 145s - loss: 0.4583 - acc: 0.7
878 - val_loss: 0.4771 - val_acc: 0.7678
Epoch 7/10
12198/12198 [==============================] - 144s - loss: 0.4525 - acc: 0.7
862 - val_loss: 0.4926 - val_acc: 0.7632
Epoch 8/10
12198/12198 [==============================] - 145s - loss: 0.4492 - acc: 0.7
919 - val_loss: 0.4677 - val_acc: 0.7794
Epoch 9/10
12198/12198 [==============================] - 144s - loss: 0.4228 - acc: 0.8
058 - val_loss: 0.4745 - val_acc: 0.7797
Epoch 10/10
12198/12198 [==============================] - 145s - loss: 0.4085 - acc: 0.8
154 - val_loss: 0.4534 - val_acc: 0.7893
```

Out[215]:

```
<keras.callbacks.History at 0x7f5ff409ba10>
```

In [216]:

```
score, acc = model.evaluate(test_X, test_Y)
print('Test score:', score)
print('Test accuracy:', acc)
```

```
3028/3028 [==============================] - 16s
Test score: 0.453434576998
Test accuracy: 0.789299867978
```

In [288]:

```
name=["sandhya","jaspreet","rajesh"]
X=[]
trunc_name = [i[0:maxlen] for i in name]
for i in trunc_name:
    tmp = [set_flag(char_index[j]) for j in str(i)]
    for k in range(0,maxlen - len(str(i))):
        tmp.append(set_flag(char_index["END"]))
    X.append(tmp)
pred=model.predict(np.asarray(X))
```

```
In [289]: pred
```

```
Out[289]: array([[ 0.62356585,  0.37643418],
                 [ 0.72094178,  0.27905828],
                 [ 0.90337974,  0.09662029]], dtype=float32)
```

Lets train more, clearly some very simple female names it doesnt get right like mentioned above (inspite it exists in training data)

In [290]:

```
batch_size=1000
model.fit(train_X, train_Y,batch_size=batch_size,nb_epoch=50,validation_data=
(test_X, test_Y))
```

```
Train on 12198 samples, validate on 3028 samples
Epoch 1/50
12198/12198 [==============================] - 145s - loss: 0.4107 - acc: 0.8
137 - val_loss: 0.4408 - val_acc: 0.7966
Epoch 2/50
12198/12198 [==============================] - 144s - loss: 0.3912 - acc: 0.8
254 - val_loss: 0.4479 - val_acc: 0.7936
Epoch 3/50
12198/12198 [==============================] - 145s - loss: 0.3927 - acc: 0.8
228 - val_loss: 0.4511 - val_acc: 0.7982
Epoch 4/50
12198/12198 [==============================] - 145s - loss: 0.3730 - acc: 0.8
344 - val_loss: 0.4253 - val_acc: 0.8071
Epoch 5/50
12198/12198 [==============================] - 145s - loss: 0.3640 - acc: 0.8
396 - val_loss: 0.4240 - val_acc: 0.8164
Epoch 6/50
12198/12198 [==============================] - 145s - loss: 0.3490 - acc: 0.8
505 - val_loss: 0.4183 - val_acc: 0.8180
Epoch 7/50
12198/12198 [==============================] - 145s - loss: 0.3411 - acc: 0.8
542 - val_loss: 0.4089 - val_acc: 0.8243
Epoch 8/50
12198/12198 [==============================] - 145s - loss: 0.3396 - acc: 0.8
529 - val_loss: 0.4026 - val_acc: 0.8184
Epoch 9/50
12198/12198 [==============================] - 146s - loss: 0.3294 - acc: 0.8
590 - val_loss: 0.3781 - val_acc: 0.8451
Epoch 10/50
12198/12198 [==============================] - 145s - loss: 0.3159 - acc: 0.8
684 - val_loss: 0.3935 - val_acc: 0.8332
Epoch 11/50
12198/12198 [==============================] - 144s - loss: 0.3025 - acc: 0.8
736 - val_loss: 0.3912 - val_acc: 0.8425
Epoch 12/50
12198/12198 [==============================] - 145s - loss: 0.2921 - acc: 0.8
808 - val_loss: 0.3981 - val_acc: 0.8408
Epoch 13/50
12198/12198 [==============================] - 144s - loss: 0.2885 - acc: 0.8
800 - val_loss: 0.4018 - val_acc: 0.8336
Epoch 14/50
12198/12198 [==============================] - 145s - loss: 0.2823 - acc: 0.8
853 - val_loss: 0.3687 - val_acc: 0.8464
Epoch 15/50
12198/12198 [==============================] - 145s - loss: 0.2763 - acc: 0.8
879 - val_loss: 0.3866 - val_acc: 0.8606
Epoch 16/50
12198/12198 [==============================] - 145s - loss: 0.2653 - acc: 0.8
933 - val_loss: 0.3820 - val_acc: 0.8554
Epoch 17/50
12198/12198 [==============================] - 145s - loss: 0.2573 - acc: 0.8
970 - val_loss: 0.3962 - val_acc: 0.8471
Epoch 18/50
12198/12198 [==============================] - 145s - loss: 0.2651 - acc: 0.8
919 - val_loss: 0.3756 - val_acc: 0.8501
Epoch 19/50
12198/12198 [==============================] - 145s - loss: 0.2539 - acc: 0.8
```

```
982 - val_loss: 0.3837 - val_acc: 0.8491
Epoch 20/50
12198/12198 [==============================] - 144s - loss: 0.2576 - acc: 0.8
970 - val_loss: 0.4036 - val_acc: 0.8494
Epoch 21/50
12198/12198 [==============================] - 146s - loss: 0.2476 - acc: 0.9
010 - val_loss: 0.4013 - val_acc: 0.8471
Epoch 22/50
12198/12198 [==============================] - 144s - loss: 0.2485 - acc: 0.8
985 - val_loss: 0.3795 - val_acc: 0.8587
Epoch 23/50
12198/12198 [==============================] - 144s - loss: 0.2273 - acc: 0.9
087 - val_loss: 0.3745 - val_acc: 0.8583
Epoch 24/50
12198/12198 [==============================] - 145s - loss: 0.2287 - acc: 0.9
074 - val_loss: 0.3830 - val_acc: 0.8570
Epoch 25/50
12198/12198 [==============================] - 145s - loss: 0.2205 - acc: 0.9
097 - val_loss: 0.3870 - val_acc: 0.8563
Epoch 26/50
12198/12198 [==============================] - 145s - loss: 0.2133 - acc: 0.9
150 - val_loss: 0.3778 - val_acc: 0.8656
Epoch 27/50
12198/12198 [==============================] - 145s - loss: 0.2024 - acc: 0.9
225 - val_loss: 0.3910 - val_acc: 0.8646
Epoch 28/50
12198/12198 [==============================] - 145s - loss: 0.1911 - acc: 0.9
244 - val_loss: 0.4067 - val_acc: 0.8570
Epoch 29/50
12198/12198 [==============================] - 144s - loss: 0.1869 - acc: 0.9
241 - val_loss: 0.4113 - val_acc: 0.8633
Epoch 30/50
12198/12198 [==============================] - 147s - loss: 0.1897 - acc: 0.9
229 - val_loss: 0.3766 - val_acc: 0.8662
Epoch 31/50
12198/12198 [==============================] - 145s - loss: 0.1786 - acc: 0.9
279 - val_loss: 0.4527 - val_acc: 0.8623
Epoch 32/50
12198/12198 [==============================] - 145s - loss: 0.1728 - acc: 0.9
311 - val_loss: 0.4064 - val_acc: 0.8633
Epoch 33/50
12198/12198 [==============================] - 144s - loss: 0.1893 - acc: 0.9
250 - val_loss: 0.3870 - val_acc: 0.8613
Epoch 34/50
12198/12198 [==============================] - 146s - loss: 0.1880 - acc: 0.9
253 - val_loss: 0.3886 - val_acc: 0.8692
Epoch 35/50
12198/12198 [==============================] - 145s - loss: 0.1672 - acc: 0.9
344 - val_loss: 0.4596 - val_acc: 0.8504
Epoch 36/50
12198/12198 [==============================] - 144s - loss: 0.1610 - acc: 0.9
329 - val_loss: 0.4256 - val_acc: 0.8669
Epoch 37/50
12198/12198 [==============================] - 145s - loss: 0.1596 - acc: 0.9
344 - val_loss: 0.4235 - val_acc: 0.8705
Epoch 38/50
12198/12198 [==============================] - 144s - loss: 0.1651 - acc: 0.9
```

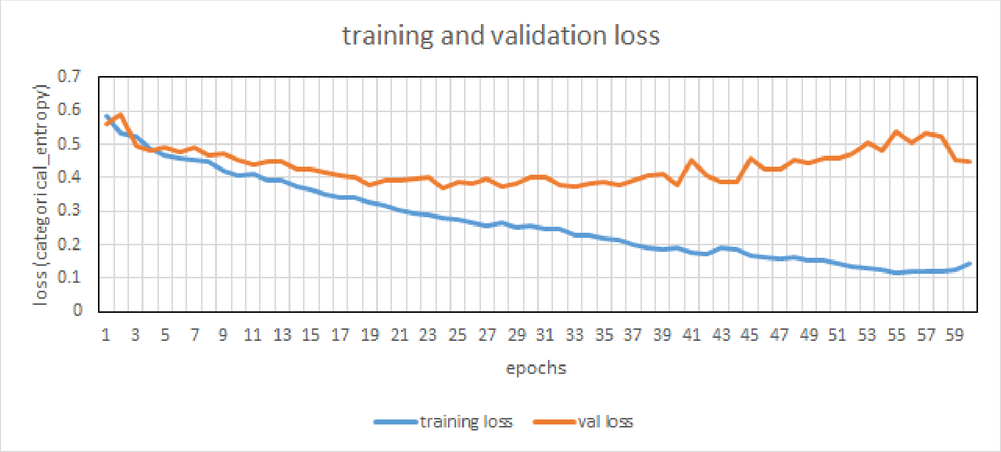
```
333 - val_loss: 0.4543 - val_acc: 0.8596
Epoch 39/50
12198/12198 [==============================] - 145s - loss: 0.1557 - acc: 0.9
382 - val_loss: 0.4427 - val_acc: 0.8662
Epoch 40/50
12198/12198 [==============================] - 144s - loss: 0.1558 - acc: 0.9
371 - val_loss: 0.4607 - val_acc: 0.8530
Epoch 41/50
12198/12198 [==============================] - 145s - loss: 0.1461 - acc: 0.9
410 - val_loss: 0.4565 - val_acc: 0.8633
Epoch 42/50
12198/12198 [==============================] - 145s - loss: 0.1365 - acc: 0.9
444 - val_loss: 0.4703 - val_acc: 0.8600
Epoch 43/50
12198/12198 [==============================] - 145s - loss: 0.1310 - acc: 0.9
461 - val_loss: 0.5031 - val_acc: 0.8705
Epoch 44/50
12198/12198 [==============================] - 145s - loss: 0.1247 - acc: 0.9
480 - val_loss: 0.4818 - val_acc: 0.8643
Epoch 45/50
12198/12198 [==============================] - 145s - loss: 0.1173 - acc: 0.9
520 - val_loss: 0.5398 - val_acc: 0.8662
Epoch 46/50
12198/12198 [==============================] - 146s - loss: 0.1194 - acc: 0.9
508 - val_loss: 0.5055 - val_acc: 0.8649
Epoch 47/50
12198/12198 [==============================] - 145s - loss: 0.1230 - acc: 0.9
512 - val_loss: 0.5328 - val_acc: 0.8656
Epoch 48/50
12198/12198 [==============================] - 145s - loss: 0.1219 - acc: 0.9
491 - val_loss: 0.5247 - val_acc: 0.8696
Epoch 49/50
12198/12198 [==============================] - 145s - loss: 0.1245 - acc: 0.9
492 - val_loss: 0.4557 - val_acc: 0.8676
Epoch 50/50
12198/12198 [==============================] - 145s - loss: 0.1437 - acc: 0.9
427 - val_loss: 0.4484 - val_acc: 0.8643
```

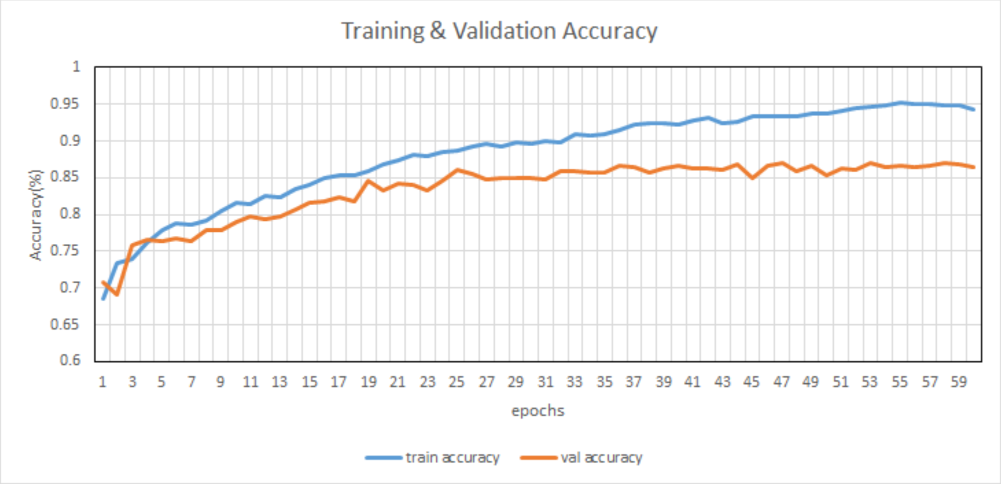
```
Out[290]: <keras.callbacks.History at 0x7f5fe98ba8d0>
```

```
In [460]: score, acc = model.evaluate(test_X, test_Y)
          print('Test score:', score)
          print('Test accuracy:', acc)
```

```
3028/3028 [==============================] - 16s
Test score: 0.448404541104
Test accuracy: 0.864266842879
```

lets look at the loss and accuracy chart as a function of epochs





```
In [342]: name=["sandhya","jaspreet","rajesh","kaveri","aditi deepak","arihant","sasikala","aditi","ragini rajaram"]
          X=[]
          trunc_name = [i[0:maxlen] for i in name]
          for i in trunc_name:
              tmp = [set_flag(char_index[j]) for j in str(i)]
              for k in range(0,maxlen - len(str(i))):
                  tmp.append(set_flag(char_index["END"]))
              X.append(tmp)
          pred=model.predict(np.asarray(X))
          pred
```

```
Out[342]: array([[ 0.0859881 ,  0.91401184],
                 [ 0.96310365,  0.03689628],
                 [ 0.7148453 ,  0.28515476],
                 [ 0.02246205,  0.97753793],
                 [ 0.13607673,  0.86392319],
                 [ 0.99559009,  0.00440993],
                 [ 0.05380283,  0.94619709],
                 [ 0.55060732,  0.44939268],
                 [ 0.10676169,  0.89323831]], dtype=float32)
```

In [345]:

```
name=["abhi","abhi deepak","mr. abhi"]
X=[]
trunc_name = [i[0:maxlen] for i in name]
for i in trunc_name:
    tmp = [set_flag(char_index[j]) for j in str(i)]
    for k in range(0,maxlen - len(str(i))):
        tmp.append(set_flag(char_index["END"]))
    X.append(tmp)
pred=model.predict(np.asarray(X))
pred
```

Out[345]:

```
array([[ 0.15557961,  0.84442037],
       [ 0.25342518,  0.74657482],
       [ 0.8618474 ,  0.13815261]], dtype=float32)
```

In [502]:

```
name=["rajini","rajinikanth","mr. rajini"]
X=[]
trunc_name = [i[0:maxlen] for i in name]
for i in trunc_name:
    tmp = [set_flag(char_index[j]) for j in str(i)]
    for k in range(0,maxlen - len(str(i))):
        tmp.append(set_flag(char_index["END"]))
    X.append(tmp)
pred=model.predict(np.asarray(X))
pred
```

Out[502]:

```
array([[ 0.33718896,  0.66281104],
       [ 0.99896383,  0.00103616],
       [ 0.99664474,  0.00335527]], dtype=float32)
```

In [450]:

```
#save our model and data
model.save_weights('gender_model',overwrite=True)
train.to_csv("train_split.csv")
test.to_csv("test_split.csv")
```

In [464]:

```
evals = model.predict(test_X)
prob_m = [i[0] for i in evals]
```

In [479]:

```
out = pd.DataFrame(prob_m)
out['name'] = test.name.reset_index()['name']
out['m_or_f']=test.m_or_f.reset_index()['m_or_f']
```

In [483]:

```
out.head(10)
out.columns = ['prob_m','name','actual']
out.head(10)
out.to_csv("gender_pred_out.csv")
```

In [ ]:

```

```