.::به نام خدا::.

تمرین دوم تکلیف سری چهارم

نام-نام خانوادگی

شماره دانشجویی

در این تمرین یک پایگاه داده از نام‌های مختلف داریم، و می‌خواهیم با استفاده از یک مدل rnn یک شبکه مولد اسم جدید داشته باشیم. در حالت ساده یک مدل تولید کننده اسم به این صورت عمل می‌کند که ابتدا یک حرف الفبا به عنوان حالت اولیه می‌گیرد و سپس با شروع از آن حالت، حالت‌های بعد که دنباله‌ای از حروف هستند را تولید می‌کند. این دنباله حروف می‌تواند بجای یک نام جدید استفاده شوند. در اینجا در دو حالت مختلف به مسأله نگاه شده است.

حالت اول- می‌توان دنباله نام‌ها را به عنوان یک دنباله از متن و واژگان یک داستان فرض کرد و با داشتن یک دنباله اولیه از متن، با استفاده از یک شبکه بازگشتی ادامه متن را تولید کرد. در این حالت شبکه کاراکتر خط جدید یا همان \n به عنوان جداکننده واژگان می‌شناسد و در سطح حرف واژه تولید می‌کند. این واژه‌های تولیدی با توجه به این که با الگوهای پایگاه داده تولید می‌شوند، بنابراین می‌توانند به عنوان اسم جدید بکار روند.

حالت دوم- داده‌های آموزشی را به صورت اسم جدا کرده و هر ورودی دنباله کوچکی از حروف سازنده یک نام باشند.

در اینجا هر دو حالت بالا بررسی می‌شوند. البته هر دو مدل شباهت ساختاری زیادی به یکدیگر دارند، و فقط در داده‌های ورودی متفاوت هستند.

In [1]:
```
from __future__ import print_function
from keras.callbacks import LambdaCallback
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import LSTM, GRU, SimpleRNN, TimeDistributed
from keras.utils.np_utils import to_categorical
from keras.optimizers import RMSprop
import numpy as np
import random
import sys
import io
```

```
Using TensorFlow backend.
```

ابتدا فایل متنی داده‌ها را باز می‌کنیم. تمام حروف موجود درون فایل را به صورت lower case تبدیل می‌کنیم تا مجموعه کاراکترهای مورد نیاز کم باشد.

In [2]:
```
data = open('Names', 'r').read().lower()
data
```

Out[2]: 'abbas\nabtin\nadel\nafareen\nafarin\nafsane\nafsaneh\nafsar\nafshar\nafshin\
nafsoon\nafsun\naghigh\nahmad\nahoo\nahvaz\nahwaz\nakbar\nakram\nalborz\nali\
nalireza\namin\namir\nanahita\nanna\nanoosheh\nanusheh\nara\narad\narash\nard
alan\nardavan\nardeshir\naref\narezoo\narezou\narghavan\narmaghan\narman\narm
in\narsalan\narshia\narya\naryan\nasa\nasad\nasal\nasghar\nashkan\nashraf\nas
sieh\nassiyeh\natash\natefe\natefeh\natoof\natoosa\natusa\nava\navizeh\nazad\
nazadeh\nazadi\nazar\nazin\nazita\naziz\nbabak\nbaby\nbahador\nbahamin\nbaha
r\nbaharak\nbahare\nbahareh\nbahman\nbahram\nbamdad\nbanafshe\nbanafsheh\nban
oo\nbanou\nbardia\nbehbaha\nbehdad\nbehnam\nbehnaz\nbehrang\nbehrokh\nbehroo
z\nbehruz\nbehzad\nbijan\nbita\nbolour\nbooseh\nborna\nbouseh\nbousseh\nbozor
gmehr\ncaspian\nchalipa\nchangeez\nchief\ncyrus\ndadbeh\ndanush\ndara\ndarab\
ndariush\ndavood\ndavoud\ndelaram\ndelbar\ndelkash\ndina\ndinah\ndj\ndonya\nd
orri\nebi\nebrahim\nehsan\nehteram\neifel\nelaheh\nelham\nelnaz\nemad\nesfaha
n\nesfandiar\nesfehan\nesmaeel\nfakhri\nfarah\nfarahnaz\nfaramarz\nfarana\nfa
rangis\nfaraz\nfarbod\nfardad\nfardin\nfareed\nfarhad\nfarhang\nfariborz\nfar
jad\nfarkhondeh\nfarokh\nfarokhzad\nfarrin\nfarrokh\nfarrokhzad\nfars\nfarsha
d\nfarshid\nfarsi\nfarzad\nfarzam\nfarzan\nfarzaneh\nfarzin\nfatemeh\nferdow
s\nfereshteh\nfereydoon\nfirouz\nfirouzeh\nflower\nfojan\nforoohar\nforoud\nf
orough\nforouzan\nforouzandeh\ngamzan\ngeesou\nghassedak\nghazal\nghazaleh\ng
hobad\nghodsi\nghoncheh\ngita\ngiti\ngiv\ngolbahar\ngoli\ngolnar\ngolnaz\ngol
naz5785\ngolnessa\ngolshan\ngordia\ngoshtasb\ngoudarz\nhabib\nhadi\nhafez\nha
ideh\nhaleh\nhamed\nhami\nhamid\nhamideh\nhassan\nhasti\nhedayat\nhediyeh\nhe
ngameh\nhesam\nheydar\nhighclass\nhirad\nhoda\nhoma\nhomayoon\nhomeira\nhooma
n\nhooshang\nhooshmand\nhooshyar\nhootan\nhormat\nhormoz\nhosein\nhossein\nho
uri\niman\niraj\niran\nirani\niranian\njack\njafar\njahandar\njahangir\njahan
shah\njakjoon\njalal\njalil\njamileh\njamshid\njannat\njavad\njavaneh\njavee
d\njigar\njoon\nkkhorsheed\nkaiser\nkamal\nkambiz\nkamdin\nkamran\nkamshad\nk
amyar\nkarim\nkasra\nkatayoon\nkaveh\nkavoos\nkayvan\nkeyghobad\nkeykhosrow\n
khandan\nkhashayar\nkhodadad\nkhojassteh\nkhosrow\nkia\nkian\nkiana\nkianoos
h\nkiarash\nkimiya\nkir\nkish\nkiumars\nkobra\nkoloft\nkoohyar\nkoon\nkoosha\
nkos\nkouros\nkourosh\nladan\nlaleh\nleila\nleily\nleyla\nlila\nlili\nmahast
i\nmahbod\nmahdokht\nmahin\nmahkameh\nmahla\nmahlegha\nmahmood\nmahnaz\nmahno
osh\nmahsa\nmahta\nmahtab\nmahvash\nmahyar\nmajid\nmakan\nmalakeh\nmaliheh\nm
aloos\nmana\nmani\nmanizheh\nmanouchehr\nmansoor\nmariam\nmarjan\nmarjane\nma
rjaneh\nmarmar\nmaryam\nmarzieh\nmashhad\nmasood\nmasoud\nmasoumeh\nmassoud\n
mastaneh\nmastoureh\nmazdak\nmaziar\nmehdi\nmehrangiz\nmehra\nmehrab\nmehran\
nmehrang\nmehrani\nmehrdad\nmehrnaz\nmehrnoosh\nmehry\nmehrzad\nmeshia\nmicro
soft\nmilad\nmina\nminoo\nmitra\nmohammad\nmohsen\nmojtaba\nmona\nmonir\nmoni
reh\nmorad\nmorteza\nmorvareed\nmosaken\nmozhdeh\nmozhgan\nmsn\nmustafa\nnade
r\nnadereh\nnaghmeh\nnahal\nnahid\nnamdar\nnamvar\nnanaz\nnargess\nnariman\nn
aser\nnasim\nnasrin\nnastaran\nnava\nnavid\nnayyer\nnazafarin\nnazanin\nnazgo
l\nnazhin\nnazi\nnazilla\nnaznazi\nneda\nnegah\nnegar\nnegin\nneshat\nniki\nn
ikoo\nniloufar\nnima\nnini\nniyoosha\nnoor\nnouri\nnoushafarin\nnoushin\nnous
hzad\noldooz\nomid\norang\noranous\norkideh\npadideh\npanteha\nparand\nparast
oo\nparee\npareechehr\npareerou\npareesa\npareevash\ncorel\ngolnaz5785\ngolna
z\nparvin\npegah\npeymaneh\npeyvand\npooneh\npoupak\npouran\npouri\nparham\np
arisa\nparishad\npars\nparsa\nparsi\nparvaneh\nparviz\npasha\npayam\npejman\n
persia\npersian\npeyman\npirooz\npouriya\npouya\npujman\nraha\nrahim\nrakhsha
n\nrambod\nramesh\nramin\nramtin\nrana\nrasa\nrasheed\nravan\nrayhaneh\nreza\
nrobabeh\nrokhsana\nroozbeh\nrose\nroshanak\nrostam\nroudabeh\nroxana\nroya\n
saba\nsadaf\nsadegh\nsadri\nsaeed\nsaeedeh\nsafi\nsahar\nsahba\nsalar\nsalma\
nsalman\nsalomeh\nsam\nsaman\nsami\nsamila\nsamin\nsamira\nsamireh\nsanam\nsa
naz\nsanjar\nsara\nsarah\nsarvenaz\nsasan\nsasha\nsattar\nsaviz\nsayareh\nsay
eh\nsepehr\nsepideh\nsetareh\nshabnam\nshadan\nshadi\nshaghayegh\nshahab\nsha
hbaz\nshahin\nshahkam\nshahla\nshahnaz\nshahram\nshahrdad\nshahriar\nshahrna
z\nshahrokh\nshahrooz\nshahrzad\nshahyar\nshahzadeh\nshalizeh\nshams\nshapou
r\nsharareh\nshaya\nshayan\nsheefteh\nshervin\nsheyda\nshideh\nshima\nshiraz\
nshireen\nshirin\nshiva\nshohreh\nshokoufeh\nshokouh\nsholeh\nshouka\nsiamak\
nsiavash\nsima\nsimin\nsina\nsita\nslim\nsoheil\nsoheila\nsohrab\nsoraya\nsor
oush\nsoudabeh\nsoulmaz\nsouzan\nsunny\nsunshine\nsuper\nsuri\nsussan\ntabri
z\ntaher\ntahereh\ntahmaseb\ntahmineh\ntahmouress\ntala\ntalayeh\ntannaz\ntar
a\ntaraneh\ntarsa\ntayyebeh\ntehran\nteymour\ntina\ntirdad\ntooba\ntooraj\nto
uca\ntouran\nvadood\nvafa\nvahid\nvanda\nvida\nyaghoub\nyahoo\nyahya\nyalda\n
yasaman\nyashar\nyass\nyeganeh\nyekta\nyouness\nyousef\nzahra\nzakaria\nzal\n
zand\nzari\nzarrin\nzartosht\nzhaleh\nzia\nziba\nzohreh\n\n'

حالت اول- در حالت اول فرض می‌کنیم که مجموعه داده‌های بالا یک متن بلند می‌باشد. پیش از هر کاری نیاز داریم که واژه‌نامه‌ای از حروف و کاراکترهای بکار رفته در متن ایجاد نماییم.

In [3]:
```
text = data
print('text length:', len(text))

chars = sorted(list(set(text)))
print('total chars:', len(chars))
char_indices = dict((c, i) for i, c in enumerate(chars))
indices_char = dict((i, c) for i, c in enumerate(chars))
print(indices_char)
```

```
text length: 4360
total chars: 29
{0: '\n', 1: '5', 2: '7', 3: '8', 4: 'a', 5: 'b', 6: 'c', 7: 'd', 8: 'e', 9:
'f', 10: 'g', 11: 'h', 12: 'i', 13: 'j', 14: 'k', 15: 'l', 16: 'm', 17: 'n',
18: 'o', 19: 'p', 20: 'r', 21: 's', 22: 't', 23: 'u', 24: 'v', 25: 'w', 26: '
x', 27: 'y', 28: 'z'}
```

حال متن بالا را به مجموعه‌هایی از دنباله حروف با طول ثابت جدا می‌کنیم. برای این کار طول ۴۰ انتخاب شده است. همچنین هر مجموعه نسبت به مجموعه پیش از خود ۳ کاراکتر در ابتدا و ۳ کاراکتر در انتها اختلاف دارد.

In [4]:
```
# cut text into sets of chars sequences by length 40 and some step size:
maxlen = 40
step = 3
sentences = []
next_chars = []
for i in range(0, len(text) - maxlen, step):
    sentences.append(text[i: i + maxlen])
    next_chars.append(text[i + maxlen])
print('number of sequences:', len(sentences))
```

```
number of sequences: 1440
```

با استفاده از واژه‌نامه‌ای که پیشتر ساختیم، دنباله‌های ورودی را بر حسب کاراکترها به صورت one-hot vector تبدیل می‌کنیم. داده‌های ورودی که دنباله‌ای از ۴۰ کاراکتر پشت سر هم در متن هستند، اما برچسب آموزشی یک کاراکتر بلافاصله بعد از دنباله ورودی می‌باشد.

In [6]:
```
x = np.zeros((len(sentences), maxlen, len(chars)), dtype=np.bool)
y = np.zeros((len(sentences), len(chars)), dtype=np.bool)
for i, sentence in enumerate(sentences):
    for t, char in enumerate(sentence):
        x[i, t, char_indices[char]] = 1
    y[i, char_indices[next_chars[i]]] = 1
print('input size:', x.shape)
print('target size:', y.shape)
```

```
input size: (1440, 40, 29)
target size: (1440, 29)
```

مدلی که در حالت اول در نظر گرفتیم یک مدل ساده و با یک لایه بازگشتی و یک لایه fully connected می‌باشد. دقت شود که طبقه بند فقط روی آخرین خروجی دنباله متصل شده است. یعنی شبکه بکار رفته در اینجا مدل many to one می‌باشد.

In [8]:
```
# build the model: a single LSTM, feel free to alter this by GRU and SimpleRNN
print('Build model...')
model = Sequential()
model.add(LSTM(128, input_shape=(maxlen, len(chars))))
model.add(Dense(len(chars), activation='softmax'))

optimizer = RMSprop(lr=0.01)
model.compile(loss='categorical_crossentropy', optimizer=optimizer)
```

```
Build model...
```

نکته‌ای که در اینجا استفاده کردیم، این است که می‌خواهیم یک مدل احتمالاتی داشته باشیم، بنابراین برای پیشبینی کاراکتر بعدی صرفا از بیشینه مقدار خروجی softmax استفاده نمی‌کنیم. بلکه کل خروجی softmax را یک تابع درست نمایی یا likelihood میان کاراکتر فعلی و کاراکتر بعدی در نظر می‌گیریم و سپس با استفاده از این تابع چگالی، کاراکتر بعدی را انتخاب می‌کنیم. بدیهی است که در این حالت نیز کاراکتری که مستقیم با خروجی argmax(softmax) بدست می‌آید همچنان شانس بیشتری در انتخاب شدن دارد.

شبکه‌ای که در این تمرین بررسی می‌کنیم از نظر رفتار بسیار شبیه یک مدل زنجیره مارکوف می‌باشد، که در هر حالت، با یک توزیع احتمال به حالت بعد منتقل می‌شود. این توزیع احتمال انتقال در حقیقت با استفاده از شبکه تخمین زده می‌شود.

In [9]:
```
def sample(preds, temperature=1.0):
    # helper function to sample an index from a probability array
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    # draw one sample from multinomial dist. to select next character:
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
```

تابع زیر یک تابع کمکی برای نمایش اسم‌های پیشنهادی می‌باشد. این تابع یک دنباله اولیه ۴۰ کاراکتری (شامل عبارت فاصله) به عنوان حالت اولیه می‌گیرد و سپس هر بار یک کاراکتر تخمین می‌زند. و با توجه به تابع احتمالاتی بالا در ۴ واریانس مختلف مدل خروجی ارزیابی می‌شود. از این تابع در خروجی هر دوره زمانی آموزش استفاده می‌شود که بتواند دید معنایی نسبت به کاهش loss در حین فرآیند آموزش بدهد.

In [10]:
```
def on_epoch_end(epoch, _):
    # Function invoked at end of each epoch. Prints generated text.
    print()
    print('----- Generating text after Epoch: %d' % epoch)

    start_index = random.randint(0, len(text) - maxlen - 1)
    for diversity in [0.2, 0.5, 1.0, 1.2]:
        print('----- diversity:', diversity)

        generated = ''
        sentence = text[start_index: start_index + maxlen]
        generated += sentence

        for i in range(12):
            x_pred = np.zeros((1, maxlen, len(chars)))
            for t, char in enumerate(sentence):
                x_pred[0, t, char_indices[char]] = 1.

            preds = model.predict(x_pred, verbose=0)[0]
            next_index = sample(preds, diversity)
            next_char = indices_char[next_index]

            generated += next_char
            sentence = sentence[1:] + next_char

            sys.stdout.write(next_char)
            sys.stdout.flush()
        print('\nname = ', generated.split('\n')[-2])
        print()
```

In [11]:
```
print_callback = LambdaCallback(on_epoch_end=on_epoch_end)
```

حال با استفاده از داده‌های آماده شده مدل را آموزش می‌دهیم. همانطور که در خروجی تابع مشاهده می‌شود در دوره‌های اولی آموزش نام‌های تولیدی مناسب نیستند و صرفا یک دنباله تصادفی از کاراکترها می‌باشند. اما هر چه فرآیند آموزش جلوتر می‌رود نام‌های بهتری پیشنهاد داده می‌شود.

In [ ]:
```
model.load_weights('lstm_1_next_char')
```

In [12]:
```
model.fit(x, y,
          batch_size=128,
          epochs=60,
          callbacks=[print_callback])
```

```
Epoch 1/60
1440/1440 [==============================] - 4s 3ms/step - loss: 3.2335

----- Generating text after Epoch: 0
----- diversity: 0.2









a


a

name =  a

----- diversity: 0.5

ao
a
a

a
a
name =  a

----- diversity: 1.0
oaia
j

nr
g
name =  nr

----- diversity: 1.2
og
a
aalnoah
name =  a

Epoch 2/60
1440/1440 [==============================] - 2s 2ms/step - loss: 2.7283

----- Generating text after Epoch: 1
----- diversity: 0.2
a
ria


r
ma
name =  r

----- diversity: 0.5
mam

amnmraa
name =

----- diversity: 1.0
m
tmazia
hiz
name =  tmazia

----- diversity: 1.2
```

Out[12]: <keras.callbacks.History at 0x7f4321b0d5c0>

In [15]:
```
model.save_weights('lstm_1_next_char')
```

پس از آموزش شبکه، می‌خواهیم با استفاده از مدل و یک دنباله اولیه تصادفی از پایگاه داده به عنوان حالت اولیه مدل، ادامه متن ورودی را بازتولید می‌کنیم. طول دنباله خروجی که قرار است پیش‌بینی شود را ۴۰ کاراکتر شامل کاراکتر جدا کننده در نظر گرفتیم و به این علت که حالت اولیه شبکه نیز شامل ۴۰ کاراکتر می‌باشد، و می‌خواهیم در کاراکترهای نهایی اثری از حالت اولیه نباشد و صرفا بر اساس پیش‌بینی خود شبکه باشند. سپس دنباله خروجی را بر اساس کاراکتر جدا کننده یا همان کاراکتر \n جدا می‌کنیم و عبارت یکی مانده به آخر را به عنوان اسم پیشنهادی در نظر می‌گیریم. به دو دلیل عبارت یکی مانده به آخر را در نظر گرفتیم: اول این که این عبارت حتما کاراکتر جدا کننده را داشته و یعنی یک نام کامل است. دوم این که این پیش بینی کمترین وابستگی را نسبت به حالت اولیه شبکه دارد.

In [30]:
```
def name_gen():
    start_index = random.randint(0, len(text) - maxlen - 1)
    for diversity in [0.2, 0.5, 1.0, 1.2]:
        print('----- diversity:', diversity)

        generated = ''
        sentence = text[start_index: start_index + maxlen]
        generated += sentence

        for i in range(40):
            x_pred = np.zeros((1, maxlen, len(chars)))
            for t, char in enumerate(sentence):
                x_pred[0, t, char_indices[char]] = 1.

            preds = model.predict(x_pred, verbose=0)[0]
            next_index = sample(preds, diversity)
            next_char = indices_char[next_index]

            generated += next_char
            sentence = sentence[1:] + next_char

        print('name = ', generated.split('\n')[-2])
```

In [34]:
```
name_gen()
```

```
----- diversity: 0.2
name =  kosh
----- diversity: 0.5
name =  mehran
----- diversity: 1.0
name =  mehran
----- diversity: 1.2
name =  perizan
```

حالت دوم- در این حالت شبکه را به صورت یک مدل many to many در نظر می‌گیریم . به این صورت که ورودی شبکه یک دنباله‌ای از حروف است و برای هر لحظه از ورودی یک خروجی پیش‌بینی می‌شود، خروجی این لحظه به عنوان ورودی حالت بعد در نظر گرفته می‌شود. بنابراین داده‌های ورودی و برچسب‌ها مشابه هستند ولی داده‌های برچسب یک تأخیر زمانی نسبت به ورودی دارند. اگر کد توقف را در این حالت کاراکتر \n در نظر بگیریم، فرض کنید دنباله ورودی برای آموزش به صورت زیر باشد:

> a b b a s \n

در این صورت دنباله برچسب به صورت زیر خواهد بود:

> b b a s \n \n

بنابراین داده‌های ورودی را به همین صورت تبدیل می‌کنیم:

In [3]:
```
label = list(set(data))
label.sort()
print(label)
```

```
['\n', '5', '7', '8', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k',
'l', 'm', 'n', 'o', 'p', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

In [4]:
```
num_classes = len(label)
```

سلول پایین یک تابع تعریف شده است که دنباله‌ای از کاراکترها را می‌گیرد و به صورت دنباله‌ای از اعداد با مرتبه‌ای که از label می‌گیرید مرتب می‌کند.

In [5]:
```
def character2num(x):
    n = len(x)
    t = [label.index(x[i]) for i in range(n)]
    return np.array(t)
print(character2num('abbas'))
```

```
[ 4  5  5  4 21]
```

با توجه به این که می‌خواهیم داده‌های ورودی را در یک ماتریس ذخیره کنیم و به صورت دنباله‌هایی با طول یکسان به شبکه آموزش دهیم نیاز است که بزرگ‌ترین طول نام در دنباله‌ها مشخص شود:

In [6]:
```
Names = data.splitlines()[0:-1]
maxlen = max([len(Names[i]) for i in range(len(Names))])
maxlen
```

Out[6]: 11

حال داده‌های ورودی و برچسب را با استفاده از کد گذاری one-hot vector روی حروف و بر اساس آن چه گفته شد تولید می‌کنیم. توجه شود که در این حالت یک کاراکتر توقف نیز به انتهای هر واژه اضافه شده است.

In [7]:
```
X = np.zeros((len(Names), maxlen+1, num_classes))
Y = np.zeros_like(X)
for i in range(len(Names)):
    t = Names[i] + '\n'
    t = character2num(t)
    X[i,0:t.shape[0]] =  to_categorical(t, num_classes=num_classes)
    t = Names[i][1::] + '\n\n'
    t = character2num(t)
    Y[i,0:t.shape[0]] = to_categorical(t, num_classes=num_classes)
X.shape, Y.shape
```

Out[7]:
```
((616, 12, 29), (616, 12, 29))
```

حال مدل را به صورت many to many طراحی می‌کنیم. برای این منظور خروجی همه لحظه‌های آخرین لایه بازگشتی را به یک دسته بند می‌دهیم. در کتابخانه keras این کار با استفاده از فعال بودن return_sequences لایه TimeDistributed پیش از لایه fully connected انجام می‌شود. اینجا تفاوت این مدل با مدل حالت اول است.

In [252]:
```
LAYER_NUM = 2
HIDDEN_DIM = 16
model = Sequential()
model.add(LSTM(HIDDEN_DIM, input_shape=(None, num_classes), return_sequence
s=True))
for i in range(LAYER_NUM - 1):
    model.add(LSTM(HIDDEN_DIM, return_sequences=True))
model.add(TimeDistributed(Dense(num_classes)))
model.add(Activation('softmax'))
model.compile(loss="categorical_crossentropy", optimizer="rmsprop", metric
s=['acc'])
model.load_weights('NameWfixed-2-16')
```

In [15]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, None, 16)          2944
_________________________________________________________________
lstm_6 (LSTM)                (None, None, 16)          2112
_________________________________________________________________
time_distributed_3 (TimeDist (None, None, 29)          493
_________________________________________________________________
activation_3 (Activation)    (None, None, 29)          0
=================================================================
Total params: 5,549
Trainable params: 5,549
Non-trainable params: 0
_________________________________________________________________
```

در اینجا با توجه به این که مدل مولد داریم، بنابراین ارزیابی تخمین معنا ندارد و مدل را صرفا با همان داده‌های آموزشی آموزش می‌دهیم.

In [17]:
```
model.fit(X, Y, epochs=2000)
```

```
Epoch 1/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.7201 - acc:
0.7835
Epoch 2/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.7145 - acc:
0.7858
Epoch 3/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.7102 - acc:
0.7842
Epoch 4/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.7073 - acc:
0.7842
Epoch 5/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.7037 - acc:
0.7834
Epoch 6/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.7016 - acc:
0.7854
Epoch 7/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6995 - acc:
0.7867
Epoch 8/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6973 - acc:
0.7853
Epoch 9/2000
616/616 [==============================] - 1s 2ms/step - loss: 0.6951 - acc:
0.7868
Epoch 10/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6939 - acc:
0.7856
Epoch 11/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6926 - acc:
0.7860
Epoch 12/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6914 - acc:
0.7861
Epoch 13/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6894 - acc:
0.7868
Epoch 14/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6886 - acc:
0.7865
Epoch 15/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6869 - acc:
0.7863
Epoch 16/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6859 - acc:
0.7854
Epoch 17/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6848 - acc:
0.7872
Epoch 18/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6839 - acc:
0.7891
Epoch 19/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6826 - acc:
0.7886
Epoch 20/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6818 - acc:
0.7892
Epoch 21/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6799 - acc:
0.7894
Epoch 22/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6794 - acc:
0.7902
Epoch 23/2000
616/616 [==============================] - 1s 1ms/step - loss: 0.6784 - acc:
```

Out[17]: <keras.callbacks.History at 0x7feef55257b8>

In [18]:
```
model.save_weights('NameWfixed-2-16')
```

حال برای تولید نام جدید، ابتدا باید یک کاراکتر به عنوان حالت اولیه در نظر گرفت، و با پیشبینی کاراکتر بعدی و اعمال آن به ورودی شبکه در لحظه بعد، و تا رسیدن به کد توقف یعنی \n این کار را تکرار می‌کنیم. در اینجا نیز برای این که مدل احتمالاتی باشد، از خروجی softmax به عنوان یک تابع چگالی برای انتخاب کاراکتر بعدی استفاده می‌کنیم.

In [38]:
```
def sample(preds, temperature=1.0):
    # helper function to sample an index from a probability array
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    # draw one sample from multinomial dist. to select next character:
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
```

In [175]:
```
def generate_name(maxlen=12):
    # start from index 4 to avoid generate names started with numbers
    n = [np.random.randint(4, num_classes)]
    name = [label[n[-1]]]
    x = np.zeros((1,maxlen,num_classes))
    for i in range(maxlen):
        x[0,i, :][n[-1]] = 1.0
        n.append(sample(model.predict(x[:,0:i+1, :])[0,-1,:], 1.0))
        #print(n)
        #print(name)
        name.append(label[n[-1]])
        if name[-1] == '\n':
            break
    return ''.join(name)
```

In [272]:
```
for i in range(10): print(generate_name()[0:-1], end='||')
```

```
kiumars||salar||touran||ramton||leily||merrkhokamal||naznazi||leila||geesou||
mozhdeh||
```

جمع بندی:

برای حل مسأله تولید نام دو مدل بررسی شد، مدل اول صورت یک مدل many to one طراحی شده است که یک دنباله از حروف و نشانه‌ها را به عنوان حالت اولیه می‌گیرد و سپس به صورت دنباله‌ای و یک کاراکتری هر بار یک حرف بعد را مشخص می‌کند. در این حالت از آنجایی که داده‌های آموزشی بیشتر هستند و این که الگوهای قرار گیری حروف پس از یکدیگر در حالت‌های بیشتری قرار می‌گیرند بنابراین شبکه نام‌های متنوع‌تری را از ترکیب حالت‌های مختلف ایجاد می‌کند.

مدل دوم یک مدل many to many است اما با این تفاوت که دنباله خروجی در هر لحظه به دنباله ورودی در لحظه بعد منتقل می‌شود. در این حالت فقط نام‌های موجود در متن را به صورت دنباله‌ای از کاراکترها تبدیل کردیم، هر چند که مشابه حالت قبل روی داده‌های دنباله‌ای هم آموزش انجام شود. اما با توجه به این که داده‌ها همچنان کم هستند، مدل می‌تواند به صورت یک تبدیل همانی صرفا همان دنباله‌های پایگاه داده را تولید نماید و به این معنی که نام جدید تولید نمی‌کند.

در هر دو مدل برای پیش‌بینی حرف بعد، بجای استفاده مستقیم از خروجی softmax از آن به عنوان یک تابع چگالی برای انتخاب حرف بعد انتخاب کردیم که خروجی به صورت مولد و احتمالاتی باشد.

همچنین در تعریف شبکه می‌توانیم بجای LSTM از لایه‌های GRUو SimpleRNN نیز استفاده کنیم. دو لایه اول نتیجه مشابه دارند و شبکه در تعداد دوره زمانی کمتری به loss پایین‌تر می‌رسد، اما در حالت سوم شبکه در زمان بیشتر و به هزینه بیشتری می‌رسد و نتیجه نهایی با لایه‌های LSTM و GRU بهتر می‌باشد.

نتیجه شبکه با لایه‌های GRU:

In [250]:
```
LAYER_NUM = 2
HIDDEN_DIM = 16
layer = GRU
model = Sequential()
model.add(layer(HIDDEN_DIM, input_shape=(None, num_classes), return_sequence
s=True))
for i in range(LAYER_NUM - 1):
    model.add(layer(HIDDEN_DIM, return_sequences=True))
model.add(TimeDistributed(Dense(num_classes)))
model.add(Activation('softmax'))
model.compile(loss="categorical_crossentropy", optimizer="rmsprop", metric
s=['acc'])
model.load_weights('NameWfixed-2-16_GRU')
```

In [209]:
```
model.fit(X, Y, epochs=2000)
model.save_weights('NameWfixed-2-16_GRU')
```

```
Epoch 1/2000
616/616 [==============================] - 4s 6ms/step - loss: 1.9530 - acc:
0.4677
Epoch 2/2000
616/616 [==============================] - 1s 888us/step - loss: 1.8232 - ac
c: 0.5772
Epoch 3/2000
616/616 [==============================] - 1s 891us/step - loss: 1.6150 - ac
c: 0.5770
Epoch 4/2000
616/616 [==============================] - 1s 894us/step - loss: 1.5135 - ac
c: 0.5779
Epoch 5/2000
616/616 [==============================] - 1s 887us/step - loss: 1.4755 - ac
c: 0.6017
Epoch 6/2000
616/616 [==============================] - 1s 894us/step - loss: 1.4478 - ac
c: 0.6178
Epoch 7/2000
616/616 [==============================] - 1s 937us/step - loss: 1.4240 - ac
c: 0.6281
Epoch 8/2000
616/616 [==============================] - 1s 938us/step - loss: 1.4042 - ac
c: 0.6374
Epoch 9/2000
616/616 [==============================] - 1s 899us/step - loss: 1.3867 - ac
c: 0.6404
Epoch 10/2000
616/616 [==============================] - 1s 985us/step - loss: 1.3696 - ac
c: 0.6456
Epoch 11/2000
616/616 [==============================] - 1s 956us/step - loss: 1.3545 - ac
c: 0.6472
Epoch 12/2000
616/616 [==============================] - 1s 1ms/step - loss: 1.3397 - acc:
0.6487
Epoch 13/2000
616/616 [==============================] - 1s 923us/step - loss: 1.3254 - ac
c: 0.6500
Epoch 14/2000
616/616 [==============================] - 1s 950us/step - loss: 1.3130 - ac
c: 0.6507
Epoch 15/2000
616/616 [==============================] - 1s 957us/step - loss: 1.2996 - ac
c: 0.6515
Epoch 16/2000
616/616 [==============================] - 1s 965us/step - loss: 1.2860 - ac
c: 0.6517
Epoch 17/2000
616/616 [==============================] - 1s 912us/step - loss: 1.2725 - ac
c: 0.6521
Epoch 18/2000
616/616 [==============================] - 1s 924us/step - loss: 1.2577 - ac
c: 0.6517
Epoch 19/2000
616/616 [==============================] - 1s 913us/step - loss: 1.2434 - ac
c: 0.6540
Epoch 20/2000
616/616 [==============================] - 1s 977us/step - loss: 1.2288 - ac
c: 0.6636
Epoch 21/2000
616/616 [==============================] - 1s 925us/step - loss: 1.2144 - ac
c: 0.6638
Epoch 22/2000
616/616 [==============================] - 1s 988us/step - loss: 1.1993 - ac
c: 0.6675
Epoch 23/2000
616/616 [==============================] - 1s 969us/step - loss: 1.1858 - ac
```

In [251]:
```
for i in range(10): print(generate_name()[0:-1], end='||')
```

```
fargeldozt||ormar||behrang||giti||parisood||afsun||piroor||nava||leila||bahy
n||
```

نتیجه شبکه با لایه‌های SimpleRNN:

In [241]:
```
LAYER_NUM = 2
HIDDEN_DIM = 16
layer = SimpleRNN
model = Sequential()
model.add(layer(HIDDEN_DIM, input_shape=(None, num_classes), return_sequence
s=True))
for i in range(LAYER_NUM - 1):
    model.add(layer(HIDDEN_DIM, return_sequences=True))
model.add(TimeDistributed(Dense(num_classes)))
model.add(Activation('softmax'))
model.compile(loss="categorical_crossentropy", optimizer="rmsprop", metric
s=['acc'])
model.load_weights('NameWfixed-2-16_RNN')
```

In [223]:
```
model.fit(X, Y, epochs=2000)
model.save_weights('NameWfixed-2-16_RNN')
```

```
Epoch 1/2000
616/616 [==============================] - 3s 5ms/step - loss: 1.9010 - acc:
0.1810
Epoch 2/2000
616/616 [==============================] - 0s 426us/step - loss: 1.6990 - ac
c: 0.5084
Epoch 3/2000
616/616 [==============================] - 0s 431us/step - loss: 1.5770 - ac
c: 0.6025
Epoch 4/2000
616/616 [==============================] - 0s 407us/step - loss: 1.5098 - ac
c: 0.6057
Epoch 5/2000
616/616 [==============================] - 0s 395us/step - loss: 1.4661 - ac
c: 0.6151
Epoch 6/2000
616/616 [==============================] - 0s 457us/step - loss: 1.4314 - ac
c: 0.6289
Epoch 7/2000
616/616 [==============================] - 0s 424us/step - loss: 1.4016 - ac
c: 0.6364
Epoch 8/2000
616/616 [==============================] - 0s 452us/step - loss: 1.3755 - ac
c: 0.6446
Epoch 9/2000
616/616 [==============================] - 0s 434us/step - loss: 1.3518 - ac
c: 0.6514
Epoch 10/2000
616/616 [==============================] - 0s 468us/step - loss: 1.3297 - ac
c: 0.6579
Epoch 11/2000
616/616 [==============================] - 0s 410us/step - loss: 1.3087 - ac
c: 0.6592
Epoch 12/2000
616/616 [==============================] - 0s 404us/step - loss: 1.2889 - ac
c: 0.6642
Epoch 13/2000
616/616 [==============================] - 0s 429us/step - loss: 1.2698 - ac
c: 0.6669
Epoch 14/2000
616/616 [==============================] - 0s 437us/step - loss: 1.2513 - ac
c: 0.6683
Epoch 15/2000
616/616 [==============================] - 0s 443us/step - loss: 1.2334 - ac
c: 0.6706
Epoch 16/2000
616/616 [==============================] - 0s 447us/step - loss: 1.2162 - ac
c: 0.6698
Epoch 17/2000
616/616 [==============================] - 0s 489us/step - loss: 1.2006 - ac
c: 0.6729
Epoch 18/2000
616/616 [==============================] - 0s 443us/step - loss: 1.1857 - ac
c: 0.6729
Epoch 19/2000
616/616 [==============================] - 0s 505us/step - loss: 1.1717 - ac
c: 0.6718
Epoch 20/2000
616/616 [==============================] - 0s 453us/step - loss: 1.1591 - ac
c: 0.6733
Epoch 21/2000
616/616 [==============================] - 0s 464us/step - loss: 1.1468 - ac
c: 0.6741
Epoch 22/2000
616/616 [==============================] - 0s 524us/step - loss: 1.1351 - ac
c: 0.6768
Epoch 23/2000
616/616 [==============================] - 0s 541us/step - loss: 1.1229 - ac
```

In [249]:
```
for i in range(10): print(generate_name()[0:-1], end='||')
```

```
shorgherah||wehiyad||parmi||fares||vordid||delaheh||sanash||yara||warmiyara
n||xinan||
```