Lab10

Pranisaa Charnparttaravanit st121720

This lab consists of three parts which are as follows:

1. Change the structure to be identical to Goodfellow's Figure 10.3 with tanh activation functions and see if you get different results.
2. Explore methods for batching patterns of different length prior to presentation to a RNN and implement them. See how much speedup you can get from the GPU with minibatch training.
3. Do a bit of research on similar problems such as named entity recognition, find a dataset, train a model, and report your results.

Task1: Change the structure to be identical to Goodfellow's Figure 10.3 with tanh activation functions and see if you get different results.

Here's code directly from the tutorial to read the names into a dictionary of the form { language1: [name1, name2, ...], language2: ... }

In [1]:
```
from __future__ import unicode_literals, print_function, division
from io import open
import glob
import os
import unicodedata
import string
```

1. Load data

In [2]:
```
def findFiles(path):
    return glob.glob(path)

print(findFiles('data/names/*.txt'))

all_letters = string.ascii_letters + " .,;'"
n_letters = len(all_letters)
```

```
['data/names/Arabic.txt', 'data/names/Chinese.txt', 'data/names/Czech.txt', 'd
ata/names/Dutch.txt', 'data/names/English.txt', 'data/names/French.txt', 'dat
a/names/German.txt', 'data/names/Greek.txt', 'data/names/Irish.txt', 'data/nam
es/Italian.txt', 'data/names/Japanese.txt', 'data/names/Korean.txt', 'data/nam
es/Polish.txt', 'data/names/Portuguese.txt', 'data/names/Russian.txt', 'data/n
ames/Scottish.txt', 'data/names/Spanish.txt', 'data/names/Vietnamese.txt']
```

1.1 Turn a Unicode string to plain ASCII, thanks to https://stackoverflow.com/a/518232/2809427

In [3]:
```
# Turn a Unicode string to plain ASCII, thanks to https://stackoverflow.com/a

def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
        and c in all_letters
    )

print(unicodeToAscii('Ślusàrski'))
```

```
Slusarski
```

1.2 Build the category_lines dictionary, a list of names per language

In [4]:

```
category_lines = {}
all_categories = []

# Read a file and split into lines

def readLines(filename):
    lines = open(filename, encoding='utf-8').read().strip().split('\n')
    return [unicodeToAscii(line) for line in lines]

for filename in findFiles('data/names/*.txt'):
    category = os.path.splitext(os.path.basename(filename))[0]
    all_categories.append(category)
    lines = readLines(filename)
    category_lines[category] = lines

n_categories = len(all_categories)
print('Number of categories: ', n_categories)
```

```
Number of categories:  18
```

In [5]:

```
# Check that it worked

for c in all_categories[:2]:
    print(c)
    print(category_lines[c])
```

```
Arabic
['Khoury', 'Nahas', 'Daher', 'Gerges', 'Nazari', 'Maalouf', 'Gerges', 'Naife
h', 'Guirguis', 'Baba', 'Sabbagh', 'Attia', 'Tahan', 'Haddad', 'Aswad', 'Najja
r', 'Dagher', 'Maloof', 'Isa', 'Asghar', 'Nader', 'Gaber', 'Abboud', 'Maalou
f', 'Zogby', 'Srour', 'Bahar', 'Mustafa', 'Hanania', 'Daher', 'Tuma', 'Nahas',
'Saliba', 'Shamoon', 'Handal', 'Baba', 'Amari', 'Bahar', 'Atiyeh', 'Said', 'Kh
ouri', 'Tahan', 'Baba', 'Mustafa', 'Guirguis', 'Sleiman', 'Seif', 'Dagher', 'B
ahar', 'Gaber', 'Harb', 'Seif', 'Asker', 'Nader', 'Antar', 'Awad', 'Srour', 'S
hadid', 'Hajjar', 'Hanania', 'Kalb', 'Shadid', 'Bazzi', 'Mustafa', 'Masih', 'G
hanem', 'Haddad', 'Isa', 'Antoun', 'Sarraf', 'Sleiman', 'Dagher', 'Najjar', 'M
alouf', 'Nahas', 'Naser', 'Saliba', 'Shamon', 'Malouf', 'Kalb', 'Daher', 'Maal
ouf', 'Wasem', 'Kanaan', 'Naifeh', 'Boutros', 'Moghadam', 'Masih', 'Sleiman',
'Aswad', 'Cham', 'Assaf', 'Quraishi', 'Shalhoub', 'Sabbag', 'Mifsud', 'Gaber',
'Shammas', 'Tannous', 'Sleiman', 'Bazzi', 'Quraishi', 'Rahal', 'Cham', 'Ghane
m', 'Ghanem', 'Naser', 'Baba', 'Shamon', 'Almasi', 'Basara', 'Quraishi', 'Bat
a', 'Wasem', 'Shamoun', 'Deeb', 'Touma', 'Asfour', 'Deeb', 'Hadad', 'Naifeh',
'Touma', 'Bazzi', 'Shamoun', 'Nahas', 'Haddad', 'Arian', 'Kouri', 'Deeb', 'Tom
a', 'Halabi', 'Nazari', 'Saliba', 'Fakhoury', 'Hadad', 'Baba', 'Mansour', 'Say
egh', 'Antar', 'Deeb', 'Morcos', 'Shalhoub', 'Sarraf', 'Amari', 'Wasem', 'Gani
m', 'Tuma', 'Fakhoury', 'Hadad', 'Hakimi', 'Nader', 'Said', 'Ganim', 'Daher',
'Ganem', 'Tuma', 'Boutros', 'Aswad', 'Sarkis', 'Daher', 'Toma', 'Boutros', 'Ka
naan', 'Antar', 'Gerges', 'Kouri', 'Maroun', 'Wasem', 'Dagher', 'Naifeh', 'Bis
hara', 'Ba', 'Cham', 'Kalb', 'Bazzi', 'Bitar', 'Hadad', 'Moghadam', 'Sleiman',
'Shamoun', 'Antar', 'Atiyeh', 'Koury', 'Nahas', 'Kouri', 'Maroun', 'Nassar',
'Sayegh', 'Haik', 'Ghanem', 'Sayegh', 'Salib', 'Cham', 'Bata', 'Touma', 'Antou
n', 'Antar', 'Bata', 'Botros', 'Shammas', 'Ganim', 'Sleiman', 'Seif', 'Moghada
m', 'Ba', 'Tannous', 'Bazzi', 'Seif', 'Salib', 'Hadad', 'Quraishi', 'Halabi',
'Essa', 'Bahar', 'Kattan', 'Boutros', 'Nahas', 'Sabbagh', 'Kanaan', 'Sayegh',
'Said', 'Botros', 'Najjar', 'Toma', 'Bata', 'Atiyeh', 'Halabi', 'Tannous', 'Ko
uri', 'Shamoon', 'Kassis', 'Haddad', 'Tuma', 'Mansour', 'Antar', 'Kassis', 'Ka
```

```
lb', 'Basara', 'Rahal', 'Mansour', 'Handal', 'Morcos', 'Fakhoury', 'Hadad', 'M
orcos', 'Kouri', 'Quraishi', 'Almasi', 'Awad', 'Naifeh', 'Koury', 'Asker', 'Ma
roun', 'Fakhoury', 'Sabbag', 'Sarraf', 'Shamon', 'Assaf', 'Boutros', 'Malouf',
'Nassar', 'Qureshi', 'Ghanem', 'Srour', 'Almasi', 'Qureshi', 'Ghannam', 'Musta
fa', 'Najjar', 'Kassab', 'Shadid', 'Shamoon', 'Morcos', 'Atiyeh', 'Isa', 'Ba',
'Baz', 'Asker', 'Seif', 'Asghar', 'Hajjar', 'Deeb', 'Essa', 'Qureshi', 'Abbou
d', 'Ganem', 'Haddad', 'Koury', 'Nassar', 'Abadi', 'Toma', 'Tannous', 'Harb',
'Issa', 'Khouri', 'Mifsud', 'Kalb', 'Gaber', 'Ganim', 'Boulos', 'Samaha', 'Had
dad', 'Sabbag', 'Wasem', 'Dagher', 'Rahal', 'Atiyeh', 'Antar', 'Asghar', 'Mans
our', 'Awad', 'Boulos', 'Sarraf', 'Deeb', 'Abadi', 'Nazari', 'Daher', 'Gerge
s', 'Shamoon', 'Gaber', 'Amari', 'Sarraf', 'Nazari', 'Saliba', 'Naifeh', 'Naza
ri', 'Hakimi', 'Shamon', 'Abboud', 'Quraishi', 'Tahan', 'Safar', 'Hajjar', 'Sr
our', 'Gaber', 'Shalhoub', 'Attia', 'Safar', 'Said', 'Ganem', 'Nader', 'Asgha
r', 'Mustafa', 'Said', 'Antar', 'Botros', 'Nader', 'Ghannam', 'Asfour', 'Taha
n', 'Mansour', 'Attia', 'Touma', 'Najjar', 'Kassis', 'Abboud', 'Bishara', 'Baz
zi', 'Shalhoub', 'Shalhoub', 'Safar', 'Khoury', 'Nazari', 'Sabbag', 'Sleiman',
'Atiyeh', 'Kouri', 'Bitar', 'Zogby', 'Ghanem', 'Assaf', 'Abadi', 'Arian', 'Sha
lhoub', 'Khoury', 'Morcos', 'Shamon', 'Wasem', 'Abadi', 'Antoun', 'Baz', 'Nase
r', 'Assaf', 'Saliba', 'Nader', 'Mikhail', 'Naser', 'Daher', 'Morcos', 'Awad',
'Nahas', 'Sarkis', 'Malouf', 'Mustafa', 'Fakhoury', 'Ghannam', 'Shadid', 'Gabe
r', 'Koury', 'Atiyeh', 'Shamon', 'Boutros', 'Sarraf', 'Arian', 'Fakhoury', 'Ab
adi', 'Kassab', 'Nahas', 'Quraishi', 'Mansour', 'Samaha', 'Wasem', 'Seif', 'Fa
khoury', 'Saliba', 'Cham', 'Bahar', 'Shamoun', 'Essa', 'Shamon', 'Asfour', 'Bi
tar', 'Cham', 'Tahan', 'Tannous', 'Daher', 'Khoury', 'Shamon', 'Bahar', 'Qurai
shi', 'Ghannam', 'Kassab', 'Zogby', 'Basara', 'Shammas', 'Arian', 'Sayegh', 'N
aifeh', 'Mifsud', 'Sleiman', 'Arian', 'Kassis', 'Shamoun', 'Kassis', 'Harb',
'Mustafa', 'Boulos', 'Asghar', 'Shamon', 'Kanaan', 'Atiyeh', 'Kassab', 'Taha
n', 'Bazzi', 'Kassis', 'Qureshi', 'Basara', 'Shalhoub', 'Sayegh', 'Haik', 'Att
ia', 'Maroun', 'Kassis', 'Sarkis', 'Harb', 'Assaf', 'Kattan', 'Antar', 'Sleima
n', 'Touma', 'Sarraf', 'Bazzi', 'Boulos', 'Baz', 'Issa', 'Shamon', 'Shadid',
'Deeb', 'Sabbag', 'Wasem', 'Awad', 'Mansour', 'Saliba', 'Fakhoury', 'Arian',
'Bishara', 'Dagher', 'Bishara', 'Koury', 'Fakhoury', 'Naser', 'Nader', 'Anta
r', 'Gerges', 'Handal', 'Hanania', 'Shadid', 'Gerges', 'Kassis', 'Essa', 'Assa
f', 'Shadid', 'Seif', 'Shalhoub', 'Shamoun', 'Hajjar', 'Baba', 'Sayegh', 'Must
afa', 'Sabbagh', 'Isa', 'Najjar', 'Tannous', 'Hanania', 'Ganem', 'Gerges', 'Fa
khoury', 'Mifsud', 'Nahas', 'Bishara', 'Bishara', 'Abadi', 'Sarkis', 'Masih',
'Isa', 'Attia', 'Kalb', 'Essa', 'Boulos', 'Basara', 'Halabi', 'Halabi', 'Daghe
r', 'Attia', 'Kassis', 'Tuma', 'Gerges', 'Ghannam', 'Toma', 'Baz', 'Asghar',
'Zogby', 'Aswad', 'Hadad', 'Dagher', 'Naser', 'Shadid', 'Atiyeh', 'Zogby', 'Ab
boud', 'Tannous', 'Khouri', 'Atiyeh', 'Ganem', 'Maalouf', 'Isa', 'Maroun', 'Is
sa', 'Khouri', 'Harb', 'Nader', 'Awad', 'Nahas', 'Said', 'Baba', 'Totah', 'Gan
im', 'Handal', 'Mansour', 'Basara', 'Malouf', 'Said', 'Botros', 'Samaha', 'Saf
ar', 'Tahan', 'Botros', 'Shamoun', 'Handal', 'Sarraf', 'Malouf', 'Bishara', 'A
swad', 'Khouri', 'Baz', 'Asker', 'Toma', 'Koury', 'Gerges', 'Bishara', 'Boulo
s', 'Najjar', 'Aswad', 'Shamon', 'Kouri', 'Srour', 'Assaf', 'Tannous', 'Atti
a', 'Mustafa', 'Kattan', 'Asghar', 'Amari', 'Shadid', 'Said', 'Bazzi', 'Masi
h', 'Antar', 'Fakhoury', 'Shadid', 'Masih', 'Handal', 'Sarraf', 'Kassis', 'Sal
ib', 'Hajjar', 'Totah', 'Koury', 'Totah', 'Mustafa', 'Sabbagh', 'Moghadam', 'T
oma', 'Srour', 'Almasi', 'Totah', 'Maroun', 'Kattan', 'Naifeh', 'Sarkis', 'Mik
hail', 'Nazari', 'Boutros', 'Guirguis', 'Gaber', 'Kassis', 'Masih', 'Hanania',
'Maloof', 'Quraishi', 'Cham', 'Hadad', 'Tahan', 'Bitar', 'Arian', 'Gaber', 'Ba
z', 'Mansour', 'Kalb', 'Sarkis', 'Attia', 'Antar', 'Asfour', 'Said', 'Essa',
'Koury', 'Hadad', 'Tuma', 'Moghadam', 'Sabbagh', 'Amari', 'Dagher', 'Srour',
'Antoun', 'Sleiman', 'Maroun', 'Tuma', 'Nahas', 'Hanania', 'Sayegh', 'Amari',
'Sabbagh', 'Said', 'Cham', 'Asker', 'Nassar', 'Bitar', 'Said', 'Dagher', 'Safa
r', 'Khouri', 'Totah', 'Khoury', 'Salib', 'Basara', 'Abboud', 'Baz', 'Isa', 'C
ham', 'Amari', 'Mifsud', 'Hadad', 'Rahal', 'Khoury', 'Bazzi', 'Basara', 'Tota
h', 'Ghannam', 'Koury', 'Malouf', 'Zogby', 'Zogby', 'Boutros', 'Nassar', 'Hand
al', 'Hajjar', 'Maloof', 'Abadi', 'Maroun', 'Mifsud', 'Kalb', 'Amari', 'Hakim
i', 'Boutros', 'Masih', 'Kattan', 'Haddad', 'Arian', 'Nazari', 'Assaf', 'Atti
a', 'Wasem', 'Gerges', 'Asker', 'Tahan', 'Fakhoury', 'Shadid', 'Sarraf', 'Atti
a', 'Naifeh', 'Aswad', 'Deeb', 'Tannous', 'Totah', 'Cham', 'Baba', 'Najjar',
'Hajjar', 'Shamoon', 'Handal', 'Awad', 'Guirguis', 'Awad', 'Ganem', 'Naifeh',
'Khoury', 'Hajjar', 'Moghadam', 'Mikhail', 'Ghannam', 'Guirguis', 'Tannous',
'Kanaan', 'Handal', 'Khoury', 'Kalb', 'Qureshi', 'Najjar', 'Atiyeh', 'Gerges',
'Nassar', 'Tahan', 'Hadad', 'Fakhoury', 'Salib', 'Wasem', 'Bitar', 'Fakhoury',
'Attia', 'Awad', 'Totah', 'Deeb', 'Touma', 'Botros', 'Nazari', 'Nahas', 'Kour
i', 'Ghannam', 'Assaf', 'Asfour', 'Sarraf', 'Naifeh', 'Toma', 'Asghar', 'Abbou
d', 'Issa', 'Sabbag', 'Sabbagh', 'Isa', 'Koury', 'Kattan', 'Shamoon', 'Rahal',
'Kalb', 'Naser', 'Masih', 'Sayegh', 'Dagher', 'Asker', 'Maroun', 'Dagher', 'Sl
eiman', 'Botros', 'Sleiman', 'Harb', 'Tahan', 'Tuma', 'Said', 'Hadad', 'Samah
a', 'Harb', 'Cham', 'Atiyeh', 'Haik', 'Malouf', 'Bazzi', 'Harb', 'Malouf', 'Gh
anem', 'Cham', 'Asghar', 'Samaha', 'Khouri', 'Nassar', 'Rahal', 'Baz', 'Kalb',
'Rahal', 'Gerges', 'Cham', 'Sayegh', 'Shadid', 'Morcos', 'Shamoon', 'Hakimi',
'Shamoon', 'Qureshi', 'Ganim', 'Shadid', 'Khoury', 'Boutros', 'Hanania', 'Anto
un', 'Naifeh', 'Deeb', 'Samaha', 'Awad', 'Asghar', 'Awad', 'Saliba', 'Shamou
n', 'Mikhail', 'Hakimi', 'Mikhail', 'Cham', 'Halabi', 'Sarkis', 'Kattan', 'Naz
ari', 'Safar', 'Morcos', 'Khoury', 'Essa', 'Nassar', 'Haik', 'Shadid', 'Fakhou
ry', 'Najjar', 'Arian', 'Botros', 'Daher', 'Saliba', 'Saliba', 'Kattan', 'Hajj
ar', 'Nader', 'Daher', 'Nassar', 'Maroun', 'Harb', 'Nassar', 'Antar', 'Shamma
s', 'Toma', 'Antar', 'Koury', 'Nader', 'Botros', 'Bahar', 'Najjar', 'Maloof',
'Salib', 'Malouf', 'Mansour', 'Bazzi', 'Atiyeh', 'Kanaan', 'Bishara', 'Hakim
i', 'Saliba', 'Tuma', 'Mifsud', 'Hakimi', 'Assaf', 'Nassar', 'Sarkis', 'Bita
r', 'Isa', 'Halabi', 'Shamon', 'Qureshi', 'Bishara', 'Maalouf', 'Srour', 'Boul
os', 'Safar', 'Shamoun', 'Ganim', 'Abadi', 'Koury', 'Shadid', 'Zogby', 'Boutro
s', 'Shadid', 'Hakimi', 'Bazzi', 'Isa', 'Totah', 'Salib', 'Shamoon', 'Gaber',
'Antar', 'Antar', 'Najjar', 'Fakhoury', 'Malouf', 'Salib', 'Rahal', 'Boulos',
'Attia', 'Said', 'Kassis', 'Bahar', 'Bazzi', 'Srour', 'Antar', 'Nahas', 'Kassi
s', 'Samaha', 'Quraishi', 'Asghar', 'Asker', 'Antar', 'Totah', 'Haddad', 'Malo
of', 'Kouri', 'Basara', 'Bata', 'Antar', 'Shammas', 'Arian', 'Gerges', 'Seif',
'Almasi', 'Tuma', 'Shamoon', 'Khoury', 'Hakimi', 'Abboud', 'Baz', 'Seif', 'Iss
a', 'Nazari', 'Harb', 'Shammas', 'Amari', 'Totah', 'Malouf', 'Sarkis', 'Nase
r', 'Zogby', 'Handal', 'Naifeh', 'Cham', 'Hadad', 'Gerges', 'Kalb', 'Shalhou
b', 'Saliba', 'Tannous', 'Tahan', 'Tannous', 'Kassis', 'Shadid', 'Sabbag', 'Ta
han', 'Abboud', 'Nahas', 'Shamoun', 'Dagher', 'Botros', 'Amari', 'Maalouf', 'A
wad', 'Gerges', 'Shamoon', 'Haddad', 'Salib', 'Attia', 'Kassis', 'Sleiman', 'M
aloof', 'Maroun', 'Koury', 'Asghar', 'Kalb', 'Asghar', 'Touma', 'Ganim', 'Raha
l', 'Haddad', 'Zogby', 'Mansour', 'Guirguis', 'Touma', 'Maroun', 'Tannous', 'H
akimi', 'Baba', 'Toma', 'Botros', 'Sarraf', 'Koury', 'Sarraf', 'Nassar', 'Bout
ros', 'Guirguis', 'Qureshi', 'Aswad', 'Basara', 'Toma', 'Tuma', 'Mansour', 'B
a', 'Naifeh', 'Mikhail', 'Amari', 'Shamon', 'Malouf', 'Boutros', 'Hakimi', 'Sr
our', 'Morcos', 'Halabi', 'Bazzi', 'Abadi', 'Shamoun', 'Haddad', 'Baz', 'Bab
a', 'Hadad', 'Saliba', 'Haddad', 'Maalouf', 'Bitar', 'Shammas', 'Totah', 'Sai
d', 'Najjar', 'Mikhail', 'Samaha', 'Boulos', 'Kalb', 'Shamon', 'Shamoun', 'Sei
f', 'Touma', 'Hajjar', 'Hadad', 'Atiyeh', 'Totah', 'Mansour', 'Nazari', 'Qurai
shi', 'Ba', 'Sarkis', 'Gerges', 'Shalhoub', 'Nazari', 'Issa', 'Salib', 'Shalho
ub', 'Nassar', 'Guirguis', 'Daher', 'Hakimi', 'Attia', 'Cham', 'Isa', 'Hakim
i', 'Amari', 'Boutros', 'Sarraf', 'Antoun', 'Botros', 'Haddad', 'Tahan', 'Bish
ara', 'Shalhoub', 'Safar', 'Haik', 'Tahan', 'Seif', 'Awad', 'Antoun', 'Atiye
h', 'Samaha', 'Assaf', 'Guirguis', 'Hadad', 'Sayegh', 'Khouri', 'Asghar', 'Tan
nous', 'Maalouf', 'Khouri', 'Hajjar', 'Abadi', 'Ghanem', 'Salib', 'Botros', 'B
itar', 'Bishara', 'Quraishi', 'Boutros', 'Aswad', 'Srour', 'Shamon', 'Abboud',
'Almasi', 'Baba', 'Tahan', 'Essa', 'Sabbag', 'Issa', 'Abadi', 'Abboud', 'Bazz
i', 'Nader', 'Bahar', 'Ghannam', 'Asghar', 'Gaber', 'Sayegh', 'Guirguis', 'Sro
ur', 'Asghar', 'Quraishi', 'Sayegh', 'Rahal', 'Tahan', 'Morcos', 'Cham', 'Kana
an', 'Nahas', 'Essa', 'Mifsud', 'Kouri', 'Isa', 'Saliba', 'Asfour', 'Guirgui
s', 'Isa', 'Bishara', 'Assaf', 'Naser', 'Moghadam', 'Kalb', 'Baba', 'Guirgui
s', 'Naifeh', 'Bitar', 'Samaha', 'Abboud', 'Hadad', 'Ghannam', 'Hanania', 'Sha
did', 'Totah', 'Tahan', 'Toma', 'Maloof', 'Botros', 'Issa', 'Deeb', 'Nahas',
'Khoury', 'Sayegh', 'Harb', 'Said', 'Guirguis', 'Nader', 'Harb', 'Atiyeh', 'Zo
gby', 'Basara', 'Nassar', 'Kalb', 'Khoury', 'Mifsud', 'Wasem', 'Handal', 'Gani
m', 'Harb', 'Ganim', 'Malouf', 'Sayegh', 'Khoury', 'Sabbag', 'Sabbag', 'Boulo
s', 'Malouf', 'Gaber', 'Shammas', 'Fakhoury', 'Halabi', 'Haddad', 'Asker', 'Mo
rcos', 'Hanania', 'Amari', 'Kassab', 'Malouf', 'Khouri', 'Moghadam', 'Totah',
'Maloof', 'Atiyeh', 'Abadi', 'Baz', 'Khoury', 'Arian', 'Handal', 'Dagher', 'Aw
ad', 'Atiyeh', 'Arian', 'Khoury', 'Amari', 'Attia', 'Ganim', 'Nader', 'Daghe
r', 'Sabbag', 'Halabi', 'Khouri', 'Khouri', 'Saliba', 'Mifsud', 'Koury', 'Awa
d', 'Bahar', 'Mustafa', 'Kassis', 'Gaber', 'Mifsud', 'Bishara', 'Asker', 'Naha
s', 'Wasem', 'Sleiman', 'Bata', 'Daher', 'Antar', 'Isa', 'Ganim', 'Rahal', 'To
ma', 'Rahal', 'Shamoun', 'Maloof', 'Hakimi', 'Safar', 'Gerges', 'Hanania', 'Ko
ury', 'Assaf', 'Safar', 'Gerges', 'Ganim', 'Morcos', 'Awad', 'Arian', 'Tahan',
'Sleiman', 'Asker', 'Boulos', 'Koury', 'Mifsud', 'Sabbag', 'Dagher', 'Bazzi',
'Mustafa', 'Almasi', 'Handal', 'Isa', 'Guirguis', 'Sayegh', 'Ganim', 'Ghanem',
'Toma', 'Mustafa', 'Basara', 'Bitar', 'Samaha', 'Mifsud', 'Tahan', 'Issa', 'Sa
lib', 'Khoury', 'Hadad', 'Haik', 'Gaber', 'Mansour', 'Hakimi', 'Ba', 'Mustaf
a', 'Gaber', 'Kattan', 'Koury', 'Awad', 'Maalouf', 'Masih', 'Harb', 'Atiyeh',
'Zogby', 'Nahas', 'Assaf', 'Morcos', 'Ganem', 'Ganem', 'Wasem', 'Fakhoury', 'G
hanem', 'Salib', 'Khouri', 'Maloof', 'Khouri', 'Shalhoub', 'Issa', 'Najjar',
'Kassis', 'Mustafa', 'Sayegh', 'Kassis', 'Hajjar', 'Nader', 'Sarkis', 'Tahan',
'Haddad', 'Antar', 'Sayegh', 'Zogby', 'Mifsud', 'Kassab', 'Hanania', 'Bishar
a', 'Shamoun', 'Abboud', 'Mustafa', 'Sleiman', 'Abadi', 'Sarraf', 'Zogby', 'Da
her', 'Issa', 'Nazari', 'Shamon', 'Tuma', 'Asghar', 'Morcos', 'Mifsud', 'Cha
m', 'Sarraf', 'Antar', 'Ba', 'Aswad', 'Mikhail', 'Kouri', 'Mikhail', 'Awad',
'Halabi', 'Moghadam', 'Mikhail', 'Naifeh', 'Kattan', 'Shammas', 'Malouf', 'Naj
jar', 'Srour', 'Masih', 'Fakhoury', 'Khouri', 'Assaf', 'Mifsud', 'Malouf', 'Ab
boud', 'Shamoon', 'Mansour', 'Halabi', 'Ganem', 'Deeb', 'Wasem', 'Kalb', 'Safa
r', 'Tuma', 'Fakhoury', 'Toma', 'Guirguis', 'Kassab', 'Nader', 'Handal', 'Bab
a', 'Fakhoury', 'Haik', 'Guirguis', 'Seif', 'Almasi', 'Shamon', 'Ba', 'Salib',
'Zogby', 'Koury', 'Najjar', 'Atiyeh', 'Morcos', 'Antar', 'Awad', 'Hadad', 'Mar
oun', 'Touma', 'Almasi', 'Kassis', 'Arian', 'Malouf', 'Koury', 'Sarraf', 'Hada
d', 'Bata', 'Tuma', 'Sarkis', 'Quraishi', 'Gaber', 'Abadi', 'Nader', 'Bazzi',
'Ghannam', 'Botros', 'Deeb', 'Awad', 'Kattan', 'Kanaan', 'Sarraf', 'Nahas', 'A
ssaf', 'Shadid', 'Gaber', 'Samaha', 'Harb', 'Samaha', 'Zogby', 'Atiyeh', 'Must
afa', 'Hanania', 'Isa', 'Almasi', 'Bitar', 'Fakhoury', 'Moghadam', 'Handal',
'Seif', 'Mustafa', 'Rahal', 'Antoun', 'Kassab', 'Bazzi', 'Hadad', 'Nader', 'Tu
ma', 'Basara', 'Totah', 'Nassar', 'Seif', 'Nassar', 'Daher', 'Daher', 'Maalou
f', 'Rahal', 'Quraishi', 'Hadad', 'Bahar', 'Sabbag', 'Halabi', 'Tuma', 'Antou
n', 'Boutros', 'Gerges', 'Bishara', 'Baba', 'Zogby', 'Nahas', 'Atiyeh', 'Raha
l', 'Sabbagh', 'Bitar', 'Botros', 'Tuma', 'Ganim', 'Handal', 'Daher', 'Boutro
s', 'Khouri', 'Maroun', 'Mifsud', 'Arian', 'Safar', 'Koury', 'Deeb', 'Shamou
n', 'Cham', 'Asghar', 'Morcos', 'Tahan', 'Salib', 'Aswad', 'Shadid', 'Saliba',
'Ganim', 'Haik', 'Kattan', 'Antoun', 'Hajjar', 'Toma', 'Toma', 'Antoun', 'Taha
n', 'Haik', 'Kassis', 'Shamoun', 'Shammas', 'Kassis', 'Shadid', 'Samaha', 'Sar
raf', 'Nader', 'Ganem', 'Zogby', 'Maloof', 'Kalb', 'Gerges', 'Seif', 'Nahas',
'Arian', 'Asfour', 'Hakimi', 'Ba', 'Handal', 'Abadi', 'Harb', 'Nader', 'Asgha
r', 'Sabbag', 'Touma', 'Amari', 'Kanaan', 'Hajjar', 'Said', 'Sarraf', 'Hadda
d', 'Mifsud', 'Shammas', 'Sleiman', 'Asfour', 'Deeb', 'Kattan', 'Naser', 'Sai
d', 'Bishara', 'Harb', 'Morcos', 'Sayegh', 'Said', 'Naser', 'Aswad', 'Seif',
'Kouri', 'Dagher', 'Shamon', 'Hadad', 'Handal', 'Tuma', 'Shamon', 'Hakimi', 'R
ahal', 'Hadad', 'Ghannam', 'Almasi', 'Daher', 'Handal', 'Malouf', 'Mansour',
'Sabbagh', 'Sabbag', 'Saliba', 'Haddad', 'Tahan', 'Khoury', 'Harb', 'Ganim',
'Mansour', 'Ganem', 'Handal', 'Handal', 'Antar', 'Asfour', 'Kouri', 'Cham', 'M
asih', 'Saliba', 'Qureshi', 'Daher', 'Safar', 'Assaf', 'Harb', 'Abboud', 'Hai
k', 'Ghannam', 'Maalouf', 'Daher', 'Najjar', 'Mifsud', 'Daher', 'Amari', 'Sali
ba', 'Kanaan', 'Guirguis', 'Atiyeh', 'Sleiman', 'Mikhail', 'Arian', 'Wasem',
'Attia', 'Nassar', 'Cham', 'Koury', 'Baba', 'Guirguis', 'Morcos', 'Quraishi',
'Seif', 'Sarkis', 'Moghadam', 'Ba', 'Boutros', 'Nader', 'Gerges', 'Salib', 'Sa
lib', 'Guirguis', 'Essa', 'Guirguis', 'Antoun', 'Kassis', 'Abboud', 'Najjar',
'Aswad', 'Srour', 'Mifsud', 'Ghanem', 'Bitar', 'Ghannam', 'Asghar', 'Deeb', 'K
alb', 'Nader', 'Srour', 'Attia', 'Shamon', 'Bata', 'Nahas', 'Gerges', 'Kanaa
n', 'Kassis', 'Sarkis', 'Maloof', 'Almasi', 'Nassar', 'Saliba', 'Arian', 'Ghan
em', 'Awad', 'Naifeh', 'Boutros', 'Fakhoury', 'Sabbag', 'Antar', 'Tahan', 'Mus
tafa', 'Almasi', 'Shammas', 'Totah', 'Boutros', 'Cham', 'Shamon', 'Ganim', 'Gh
anem', 'Assaf', 'Khoury', 'Naifeh', 'Bahar', 'Quraishi', 'Bishara', 'Cham', 'A
sfour', 'Ghannam', 'Khoury', 'Sayegh', 'Hanania', 'Maroun', 'Kouri', 'Sarkis',
'Haik', 'Basara', 'Salib', 'Shammas', 'Fakhoury', 'Nahas', 'Ganim', 'Botros',
'Arian', 'Shalhoub', 'Hadad', 'Mustafa', 'Shalhoub', 'Kassab', 'Asker', 'Botro
s', 'Kanaan', 'Gaber', 'Bazzi', 'Sayegh', 'Nassar', 'Kassis', 'Fakhoury', 'Kas
sis', 'Amari', 'Sarraf', 'Mifsud', 'Salib', 'Samaha', 'Mustafa', 'Asfour', 'Na
jjar', 'Essa', 'Naifeh', 'Cham', 'Sarraf', 'Moghadam', 'Fakhoury', 'Assaf', 'A
lmasi', 'Asghar', 'Nader', 'Kalb', 'Shamoun', 'Gerges', 'Wasem', 'Morcos', 'Na
der', 'Said', 'Safar', 'Quraishi', 'Samaha', 'Kassab', 'Deeb', 'Sarraf', 'Raha
l', 'Naifeh', 'Ba', 'Nazari', 'Ganim', 'Arian', 'Asker', 'Touma', 'Kassab', 'T
ahan', 'Mansour', 'Morcos', 'Shammas', 'Baba', 'Morcos', 'Isa', 'Moghadam', 'G
anem', 'Baz', 'Totah', 'Nader', 'Kouri', 'Guirguis', 'Koury', 'Zogby', 'Basar
a', 'Baz', 'Deeb', 'Mustafa', 'Shadid', 'Awad', 'Sarraf', 'Quraishi', 'Kanaa
n', 'Tahan', 'Ghannam', 'Shammas', 'Abboud', 'Najjar', 'Bishara', 'Tuma', 'Sro
ur', 'Mifsud', 'Srour', 'Hajjar', 'Qureshi', 'Bitar', 'Hadad', 'Almasi', 'Wase
m', 'Abadi', 'Maroun', 'Baz', 'Koury', 'Ganem', 'Awad', 'Maalouf', 'Mifsud',
'Haik', 'Sleiman', 'Arian', 'Seif', 'Mansour', 'Koury', 'Kattan', 'Koury', 'As
wad', 'Ba', 'Rahal', 'Zogby', 'Bahar', 'Fakhoury', 'Samaha', 'Sarraf', 'Mifsu
d', 'Antar', 'Moghadam', 'Botros', 'Srour', 'Sabbag', 'Sayegh', 'Rahal', 'Atti
a', 'Naifeh', 'Saliba', 'Mustafa', 'Amari', 'Issa', 'Masih', 'Khouri', 'Hadda
d', 'Kalb', 'Bazzi', 'Salib', 'Hanania', 'Shamoon', 'Tuma', 'Cham', 'Antoun',
'Wasem', 'Kouri', 'Ghanem', 'Wasem', 'Khoury', 'Assaf', 'Ganem', 'Seif', 'Nade
r', 'Essa', 'Shadid', 'Botros', 'Sleiman', 'Bishara', 'Basara', 'Maalouf', 'Is
sa', 'Nassar', 'Moghadam', 'Ganim', 'Kassis', 'Antoun', 'Said', 'Khouri', 'Sal
```

```
ib', 'Baz', 'Sarkis', 'Tuma', 'Naifeh', 'Najjar', 'Asker', 'Khouri', 'Mustaf
a', 'Najjar', 'Sabbag', 'Malouf', 'Wasem', 'Maalouf', 'Gaber', 'Said', 'Zogb
y', 'Bahar', 'Hanania', 'Shalhoub', 'Abadi', 'Handal', 'Qureshi', 'Kanaan', 'A
bboud', 'Mifsud', 'Touma', 'Ganim', 'Bishara', 'Bazzi', 'Gaber', 'Haik', 'Ghan
em', 'Sarraf', 'Sarkis', 'Mustafa', 'Baz', 'Kanaan', 'Nazari', 'Bahar', 'Malou
f', 'Quraishi', 'Kattan', 'Arian', 'Shadid', 'Tuma', 'Nader', 'Khoury', 'Safa
r', 'Wasem', 'Toma', 'Haddad', 'Quraishi', 'Nassar', 'Kanaan', 'Gaber', 'Hadda
d', 'Rahal', 'Koury', 'Harb', 'Mikhail', 'Dagher', 'Shadid', 'Boutros', 'Mikha
il', 'Khouri', 'Nader', 'Issa', 'Harb', 'Dagher', 'Gerges', 'Morcos', 'Essa',
'Fakhoury', 'Tuma', 'Kattan', 'Totah', 'Qureshi', 'Nahas', 'Bitar', 'Tahan',
'Daher', 'Shammas', 'Kouri', 'Ganim', 'Daher', 'Awad', 'Malouf', 'Mustafa', 'A
swad']
Chinese
['Ang', 'AuYong', 'Bai', 'Ban', 'Bao', 'Bei', 'Bian', 'Bui', 'Cai', 'Cao', 'Ce
n', 'Chai', 'Chaim', 'Chan', 'Chang', 'Chao', 'Che', 'Chen', 'Cheng', 'Cheun
g', 'Chew', 'Chieu', 'Chin', 'Chong', 'Chou', 'Chu', 'Cui', 'Dai', 'Deng', 'Di
ng', 'Dong', 'Dou', 'Duan', 'Eng', 'Fan', 'Fei', 'Feng', 'Foong', 'Fung', 'Ga
n', 'Gauk', 'Geng', 'Gim', 'Gok', 'Gong', 'Guan', 'Guang', 'Guo', 'Gwock', 'Ha
n', 'Hang', 'Hao', 'Hew', 'Hiu', 'Hong', 'Hor', 'Hsiao', 'Hua', 'Huan', 'Huan
g', 'Hui', 'Huie', 'Huo', 'Jia', 'Jiang', 'Jin', 'Jing', 'Joe', 'Kang', 'Kau',
'Khoo', 'Khu', 'Kong', 'Koo', 'Kwan', 'Kwei', 'Kwong', 'Lai', 'Lam', 'Lang',
'Lau', 'Law', 'Lew', 'Lian', 'Liao', 'Lim', 'Lin', 'Ling', 'Liu', 'Loh', 'Lon
g', 'Loong', 'Luo', 'Mah', 'Mai', 'Mak', 'Mao', 'Mar', 'Mei', 'Meng', 'Miao',
'Min', 'Ming', 'Moy', 'Mui', 'Nie', 'Niu', 'OuYang', 'OwYang', 'Pan', 'Pang',
'Pei', 'Peng', 'Ping', 'Qian', 'Qin', 'Qiu', 'Quan', 'Que', 'Ran', 'Rao', 'Ron
g', 'Ruan', 'Sam', 'Seah', 'See ', 'Seow', 'Seto', 'Sha', 'Shan', 'Shang', 'Sh
ao', 'Shaw', 'She', 'Shen', 'Sheng', 'Shi', 'Shu', 'Shuai', 'Shui', 'Shum', 'S
iew', 'Siu', 'Song', 'Sum', 'Sun', 'Sze ', 'Tan', 'Tang', 'Tao', 'Teng', 'Teo
h', 'Thean', 'Thian', 'Thien', 'Tian', 'Tong', 'Tow', 'Tsang', 'Tse', 'Tsen',
'Tso', 'Tze', 'Wan', 'Wang', 'Wei', 'Wen', 'Weng', 'Won', 'Wong', 'Woo', 'Xian
g', 'Xiao', 'Xie', 'Xing', 'Xue', 'Xun', 'Yan', 'Yang', 'Yao', 'Yap', 'Yau',
'Yee', 'Yep', 'Yim', 'Yin', 'Ying', 'Yong', 'You', 'Yuan', 'Zang', 'Zeng', 'Zh
a', 'Zhan', 'Zhang', 'Zhao', 'Zhen', 'Zheng', 'Zhong', 'Zhou', 'Zhu', 'Zhuo',
'Zong', 'Zou', 'Bing', 'Chi', 'Chu', 'Cong', 'Cuan', 'Dan', 'Fei', 'Feng', 'Ga
i', 'Gao', 'Gou', 'Guan', 'Gui', 'Guo', 'Hong', 'Hou', 'Huan', 'Jian', 'Jiao',
'Jin', 'Jiu', 'Juan', 'Jue', 'Kan', 'Kuai', 'Kuang', 'Kui', 'Lao', 'Liang', 'L
u', 'Luo', 'Man', 'Nao', 'Pian', 'Qiao', 'Qing', 'Qiu', 'Rang', 'Rui', 'She',
'Shi', 'Shuo', 'Sui', 'Tai', 'Wan', 'Wei', 'Xian', 'Xie', 'Xin', 'Xing', 'Xion
g', 'Xuan', 'Yan', 'Yin', 'Ying', 'Yuan', 'Yue', 'Yun', 'Zha', 'Zhai', 'Zhan
g', 'Zhi', 'Zhuan', 'Zhui']
```

In [6]:
```python
print('5 examples of italian names: ',category_lines['Italian'][:5])
```

```
5 examples of italian names:  ['Abandonato', 'Abatangelo', 'Abatantuono', 'Aba
te', 'Abategiovanni']
```

2. Encode

Method1: One-hot encoding of a word vocabulary using scikit-learn’s OneHotEncoder

In [7]:
```python
from sklearn.preprocessing import OneHotEncoder,LabelEncoder
labelencoder_X = LabelEncoder()
X = [['red'], ['green'], ['blue']]
X = labelencoder_X.fit_transform(X)
# print(X)
# print(X.shape)
encoder = OneHotEncoder(sparse=False)
print(encoder.fit_transform(X.reshape(-1,1)))
```

```
[[0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
/usr/lib/python3/dist-packages/sklearn/preprocessing/label.py:111: DataConvers
ionWarning: A column-vector y was passed when a 1d array was expected. Please
```

```
change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Method2: One-hot encoding of a word using numpy

In [8]:
```
import numpy as np

arr = [2, 1, 0]
max = np.max(arr) + 1
print(np.eye(max)[arr])
```

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

2.1 Encode names

In [9]:
```
import torch

# Find letter index from all_letters, e.g. "a" -> 0

def letterToIndex(letter):
    return all_letters.find(letter)

# (For demonstration) turn a letter into a <1 x n_letters> tensor

def letterToTensor(letter):
    tensor = torch.zeros(1, n_letters)
    tensor[0][letterToIndex(letter)] = 1
    return tensor

# Turn a line into a <line_length x 1 x n_letters> tensor
# (an array of one-hot letter vectors)

def lineToTensor(line):
    tensor = torch.zeros(len(line), 1, n_letters)
    for li, letter in enumerate(line):
        tensor[li][0][letterToIndex(letter)] = 1
    return tensor

print(letterToTensor('J'))
print(lineToTensor('Jones').size())
```

```
tensor([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 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.]])
torch.Size([5, 1, 57])
```

3. Define RNN model

In [10]:
```
import torch.nn as nn

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Sequential(nn.Linear(input_size + hidden_size, hidden_s
                                    nn.Tanh())
        self.i2o = nn.Linear(hidden_size, output_size) #add tanh
        self.softmax = nn.LogSoftmax(dim=1)  #logsoftmax thats why -2.XXX to

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(hidden) #take from hidden
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, self.hidden_size)
```

4. Inspection

Below is the implementation of one time step for the model. The forward function takes an input and a previous hidden state, returning the output and the new hidden state.

In [11]:
```
n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```

In [12]:
```
input = letterToTensor('A')
hidden = torch.zeros(1, n_hidden)

output, next_hidden = rnn(input, hidden)
output
```

Out[12]:
```
tensor([[-2.9738, -2.8447, -2.8066, -2.9305, -2.9357, -2.9382, -2.7940, -2.889
8,
         -2.9467, -2.9419, -2.9330, -2.8231, -2.9231, -2.9241, -2.8375, -2.937
7,
         -2.9117, -2.7680]], grad_fn=<LogSoftmaxBackward>)
```

In [13]:
```
input = lineToTensor('Albert')
hidden = torch.zeros(1, n_hidden)

next_hidden = hidden
for i in range(input.shape[0]):
    output, next_hidden = rnn(input[i], next_hidden)
    print(output)
```

```
tensor([[-2.9738, -2.8447, -2.8066, -2.9305, -2.9357, -2.9382, -2.7940, -2.889
8,
         -2.9467, -2.9419, -2.9330, -2.8231, -2.9231, -2.9241, -2.8375, -2.937
7,
         -2.9117, -2.7680]], grad_fn=<LogSoftmaxBackward>)
tensor([[-3.0347, -2.8348, -2.7844, -2.8709, -2.9712, -2.9585, -2.8427, -2.871
6,
         -2.8743, -3.0025, -2.9169, -2.8201, -2.8660, -2.9132, -2.8825, -2.914
8,
         -2.8741, -2.8295]], grad_fn=<LogSoftmaxBackward>)
tensor([[-2.9754, -2.8221, -2.7778, -2.8936, -2.9847, -2.9128, -2.8968, -2.843
2,
         -2.9296, -2.9746, -2.9330, -2.8319, -2.8799, -2.9170, -2.8585, -2.929
7,
         -2.8681, -2.8270]], grad_fn=<LogSoftmaxBackward>)
tensor([[-2.9539, -2.8573, -2.7907, -2.8806, -2.9682, -2.8994, -2.8989, -2.850
1,
         -2.9395, -2.9991, -2.9068, -2.8357, -2.9317, -2.8629, -2.8050, -2.969
1,
         -2.9076, -2.8024]], grad_fn=<LogSoftmaxBackward>)
tensor([[-2.9556, -2.8555, -2.7242, -2.8951, -3.0164, -2.9001, -2.8933, -2.889
0,
         -2.8710, -3.0030, -2.9369, -2.7928, -2.9378, -2.8680, -2.8344, -2.945
0,
         -2.9383, -2.8159]], grad_fn=<LogSoftmaxBackward>)
tensor([[-3.0049, -2.8325, -2.7827, -2.9133, -2.9707, -2.8779, -2.8922, -2.851
6,
         -2.9302, -2.9302, -2.9271, -2.8975, -2.8944, -2.9234, -2.8125, -2.909
8,
         -2.9046, -2.7998]], grad_fn=<LogSoftmaxBackward>)
```

5. Training

5.1 Converter

To get started with training, we need some helper functions. This one converts an output vector to a category:

In [14]:
```python
def categoryFromOutput(output):
    top_n, top_i = output.topk(1)
    category_i = top_i[0].item()
    return all_categories[category_i], category_i

print(categoryFromOutput(output))
```

```
('Czech', 2)
```

5.2 Get a random element of our training set

In [15]:
```python
import random

def randomChoice(l):
    # random.randint range is inclusive thus len(l)-1
    return l[random.randint(0, len(l) - 1)]

def randomTrainingExample():
    category = randomChoice(all_categories)
    line = randomChoice(category_lines[category])
    category_tensor = torch.tensor([all_categories.index(category)], dtype=to
    line_tensor = lineToTensor(line)
    return category, line, category_tensor, line_tensor

for i in range(10):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    print('category =', category, '/ line =', line)
```

```
category = Vietnamese / line = Than
category = Italian / line = Amoretto
category = Chinese / line = Kwong
category = Russian / line = Jelezny
category = Dutch / line = Heel
category = Vietnamese / line = Ngo
category = Irish / line = Mooney
category = Italian / line = Borghi
category = Portuguese / line = Melo
category = Polish / line = Salomon
```

5.3 Define loss function and learning rate

In [16]:
```python
criterion = nn.NLLLoss()
learning_rate = 0.005 # If you set this too high, it might explode. If too lo
```

5.4 Define train function

In [17]:
```
def train(category_tensor, line_tensor):
    hidden = rnn.initHidden()

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by learning rate
    for p in rnn.parameters():
        p.data.add_(-learning_rate, p.grad.data)

    return output, loss.item()
```

5.5 Actual training

In [18]:
```
import time
import math

n_iters = 200000
print_every = 5000
plot_every = 1000

# Keep track of losses for plotting
current_loss = 0
all_losses = []

def timeSince(since):
    now = time.time()
    s = now - since
    m = math.floor(s / 60)
    s -= m * 60
    return '%dm %ds' % (m, s)

start = time.time()

for iter in range(1, n_iters + 1):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    output, loss = train(category_tensor, line_tensor)
    current_loss += loss

    # Print iter number, loss, name and guess
    if iter % print_every == 0:
        guess, guess_i = categoryFromOutput(output)
        correct = '✓' if guess == category else '✗ (%s)' % category
        print('%d %d%% (%s) %.4f %s / %s %s' % (iter, iter / n_iters * 100, t

    # Add current loss avg to list of losses
    if iter % plot_every == 0:
        all_losses.append(current_loss / plot_every)
        current_loss = 0
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:14: UserWarning:
This overload of add_ is deprecated:
        add_(Number alpha, Tensor other)
Consider using one of the following signatures instead:
        add_(Tensor other, *, Number alpha) (Triggered internally at  /pytorc
h/torch/csrc/utils/python_arg_parser.cpp:882.)

5000 2% (0m 21s) 2.8723 Clineburg / Polish ✗ (Czech)
10000 5% (0m 42s) 1.6201 Hwang / Chinese ✗ (Korean)
15000 7% (1m 3s) 1.6307 Johov / Arabic ✗ (Russian)
20000 10% (1m 24s) 2.4282 Holzknecht / Russian ✗ (German)
25000 12% (1m 46s) 0.7306 Trang / Vietnamese ✓
30000 15% (2m 8s) 0.6126 Pispinis / Greek ✓
35000 17% (2m 29s) 0.9138 Santos / Portuguese ✓
40000 20% (2m 50s) 3.3509 Janshole / Scottish ✗ (Russian)
45000 22% (3m 11s) 1.0383 Koziol / Polish ✓
50000 25% (3m 32s) 0.0242 Nguyen / Vietnamese ✓
55000 27% (3m 53s) 3.7001 Yanzhul / Czech ✗ (Russian)
60000 30% (4m 14s) 3.6702 Ino / Vietnamese ✗ (Japanese)
65000 32% (4m 35s) 0.0351 Akrivopoulos / Greek ✓
70000 35% (4m 56s) 2.3189 Gok / Korean ✗ (Chinese)
75000 37% (5m 17s) 0.2074 Matsuda / Japanese ✓
80000 40% (5m 38s) 0.0226 Bassanelli / Italian ✓
85000 42% (5m 59s) 0.1631 Mustafa / Arabic ✓
90000 45% (6m 20s) 0.9200 Severin / French ✓
95000 47% (6m 41s) 0.5036 Sanchez / Spanish ✓
100000 50% (7m 2s) 0.0279 Snijders / Dutch ✓
105000 52% (7m 24s) 2.1675 Vozab / Arabic ✗ (Czech)
110000 55% (7m 45s) 0.1696 Shintaro / Japanese ✓
115000 57% (8m 6s) 0.8909 Hung / Korean ✓
120000 60% (8m 27s) 0.0028 Maceachthighearna / Irish ✓
125000 62% (8m 49s) 0.2182 Santana / Portuguese ✓
130000 65% (9m 10s) 1.7204 Gottlieb / English ✗ (German)
135000 67% (9m 32s) 0.1314 Abzyaparoff / Russian ✓
140000 70% (9m 53s) 0.1265 Tzehansky / Russian ✓
145000 72% (10m 16s) 2.9839 Salazar / Spanish ✗ (Portuguese)
150000 75% (10m 37s) 0.0277 Koeman / Dutch ✓
155000 77% (10m 58s) 0.0115 Szewc / Polish ✓
160000 80% (11m 19s) 0.1010 Milne / Scottish ✓
165000 82% (11m 40s) 1.9017 Sanna / Czech ✗ (Italian)
170000 85% (12m 1s) 0.1751 Johnston / Scottish ✓
175000 87% (12m 22s) 1.2844 Martz / Spanish ✗ (German)
180000 90% (12m 43s) 0.2304 Rapallino / Italian ✓
185000 92% (13m 4s) 3.1653 Bilonog / Korean ✗ (Russian)
190000 95% (13m 26s) 0.9521 Irving / English ✓
195000 97% (13m 47s) 0.0089 O'Doherty / Irish ✓
200000 100% (14m 8s) 1.1829 Dale / Dutch ✓
```

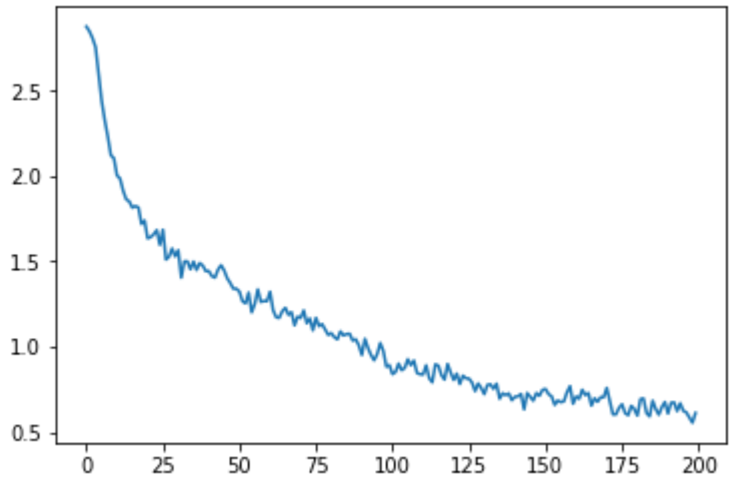
6. Plot

In [22]:

```python
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

plt.figure()
plt.plot(all_losses)
```

Out[22]:

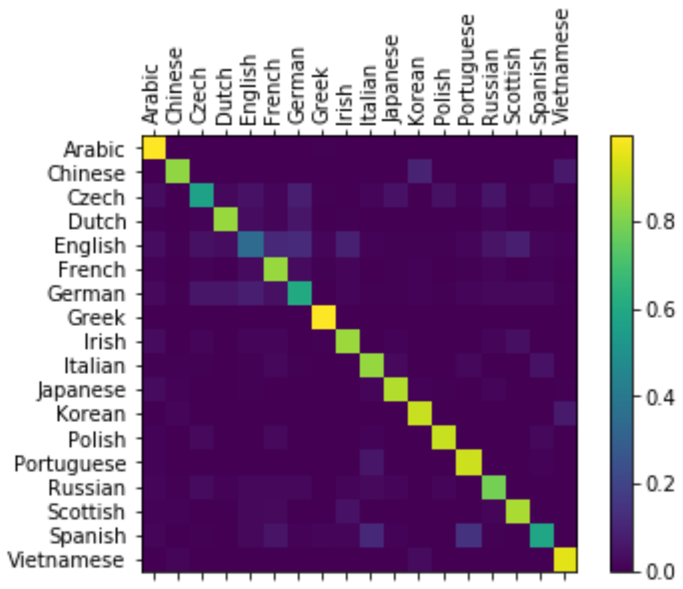
```
[<matplotlib.lines.Line2D at 0x7f7b75600cc0>]
```



7. Confusion matrix

In [20]:

```
# Keep track of correct guesses in a confusion matrix
confusion = torch.zeros(n_categories, n_categories)
n_confusion = 10000

# Just return an output given a line
def evaluate(line_tensor):
    hidden = rnn.initHidden()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    return output

# Go through a bunch of examples and record which are correctly guessed
for i in range(n_confusion):
    category, line, category_tensor, line_tensor = randomTrainingExample()
    output = evaluate(line_tensor)
    guess, guess_i = categoryFromOutput(output)
    category_i = all_categories.index(category)
    confusion[category_i][guess_i] += 1

# Normalize by dividing every row by its sum
for i in range(n_categories):
    confusion[i] = confusion[i] / confusion[i].sum()

# Set up plot
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(confusion.numpy())
fig.colorbar(cax)

# Set up axes
ax.set_xticklabels([''] + all_categories, rotation=90)
ax.set_yticklabels([''] + all_categories)

# Force label at every tick
ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

# sphinx_gallery_thumbnail_number = 2
plt.show()
```



8. Prediction

In [21]:
```
def predict(input_line, n_predictions=3):
    print('\n> %s' % input_line)
    with torch.no_grad():
        output = evaluate(lineToTensor(input_line))

        # Get top N categories
        topv, topi = output.topk(n_predictions, 1, True)
        predictions = []

        for i in range(n_predictions):
            value = topv[0][i].item()
            category_index = topi[0][i].item()
            print('(%.2f) %s' % (value, all_categories[category_index]))
            predictions.append([value, all_categories[category_index]])

predict('Dovesky')
predict('Jackson')
predict('Satoshi')
predict('Charnparttaravanit')
```

```
> Dovesky
(-0.14) Russian
(-2.44) Czech
(-3.22) English

> Jackson
(-0.21) Scottish
(-2.26) English
(-2.84) Russian

> Satoshi
(-0.10) Japanese
(-3.10) Arabic
(-3.44) Italian

> Charnparttaravanit
(-0.06) Russian
(-4.02) Czech
(-4.06) Italian
```

As shown above, I have tested with my last name, Charnparttaravanit, and apparently it is a russian name!

TASK2: Explore methods for batching patterns of different length prior to presentation to a RNN and implement them. See how much speedup you can get from the GPU with minibatch training.

With minibatching, the training is expected to be faster as it enables the training on GPU.

The code for batching is taken from the following link:

https://github.com/Niranjankumar-c/DeepLearning-PadhAI/blob/master/DeepLearning_Materials/8_RNN_LSTM_Model/BatchingSeqModels.ipynb

The results reported are as follows:

Top-1 Accuracy: 0.7599003735990038 Top-2 Accuracy: 0.862266500622665
CPU times: user 33min 58s, sys: 49.4 s, total: 34min 47s
Wall time: 6min 27s

In [7]:
```
from __future__ import unicode_literals, print_function, division
from io import open
import torch
import glob
import os
import unicodedata
import string
import numpy as np
import torch.optim as optim
from IPython.display import clear_output
```

In [8]:
```
from chosen_gpu import get_freer_gpu
device = torch.device(get_freer_gpu())
print("Configured device: ", device)
```

```
Configured device:  cuda:1
```

1. Load data

In [9]:
```
def findFiles(path):
    return glob.glob(path)

print(findFiles('data/names/*.txt'))

all_letters = string.ascii_letters + " .,;'"
n_letters = len(all_letters)
```

```
['data/names/Arabic.txt', 'data/names/Chinese.txt', 'data/names/Czech.txt', 'd
ata/names/Dutch.txt', 'data/names/English.txt', 'data/names/French.txt', 'dat
a/names/German.txt', 'data/names/Greek.txt', 'data/names/Irish.txt', 'data/nam
es/Italian.txt', 'data/names/Japanese.txt', 'data/names/Korean.txt', 'data/nam
es/Polish.txt', 'data/names/Portuguese.txt', 'data/names/Russian.txt', 'data/n
ames/Scottish.txt', 'data/names/Spanish.txt', 'data/names/Vietnamese.txt']
```

1.1 Turn a Unicode string to plain ASCII, thanks to https://stackoverflow.com/a/518232/2809427

In [10]:
```
def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
        and c in all_letters
    )

print(unicodeToAscii('Ślusàrski'))
```

```
Slusarski
```

1.2 Build the category_lines dictionary, a list of names per language

In [11]:

```
# Build the category_lines dictionary, a list of names per language

category_lines = {}
all_categories = []

# Read a file and split into lines

def readLines(filename):
    lines = open(filename, encoding='utf-8').read().strip().split('\n')
    return [unicodeToAscii(line) for line in lines]


for filename in findFiles('data/names/*.txt'):
    category = os.path.splitext(os.path.basename(filename))[0]
    all_categories.append(category)
    lines = readLines(filename)
    category_lines[category] = lines

n_categories = len(all_categories)

# Check that it worked

for c in all_categories[:2]:
    print(c)
    print(category_lines[c])
```

```
Arabic
['Khoury', 'Nahas', 'Daher', 'Gerges', 'Nazari', 'Maalouf', 'Gerges', 'Naife
h', 'Guirguis', 'Baba', 'Sabbagh', 'Attia', 'Tahan', 'Haddad', 'Aswad', 'Najja
r', 'Dagher', 'Maloof', 'Isa', 'Asghar', 'Nader', 'Gaber', 'Abboud', 'Maalou
f', 'Zogby', 'Srour', 'Bahar', 'Mustafa', 'Hanania', 'Daher', 'Tuma', 'Nahas',
'Saliba', 'Shamoon', 'Handal', 'Baba', 'Amari', 'Bahar', 'Atiyeh', 'Said', 'Kh
ouri', 'Tahan', 'Baba', 'Mustafa', 'Guirguis', 'Sleiman', 'Seif', 'Dagher', 'B
ahar', 'Gaber', 'Harb', 'Seif', 'Asker', 'Nader', 'Antar', 'Awad', 'Srour', 'S
hadid', 'Hajjar', 'Hanania', 'Kalb', 'Shadid', 'Bazzi', 'Mustafa', 'Masih', 'G
hanem', 'Haddad', 'Isa', 'Antoun', 'Sarraf', 'Sleiman', 'Dagher', 'Najjar', 'M
alouf', 'Nahas', 'Naser', 'Saliba', 'Shamon', 'Malouf', 'Kalb', 'Daher', 'Maal
ouf', 'Wasem', 'Kanaan', 'Naifeh', 'Boutros', 'Moghadam', 'Masih', 'Sleiman',
'Aswad', 'Cham', 'Assaf', 'Quraishi', 'Shalhoub', 'Sabbag', 'Mifsud', 'Gaber',
'Shammas', 'Tannous', 'Sleiman', 'Bazzi', 'Quraishi', 'Rahal', 'Cham', 'Ghane
m', 'Ghanem', 'Naser', 'Baba', 'Shamon', 'Almasi', 'Basara', 'Quraishi', 'Bat
a', 'Wasem', 'Shamoun', 'Deeb', 'Touma', 'Asfour', 'Deeb', 'Hadad', 'Naifeh',
'Touma', 'Bazzi', 'Shamoun', 'Nahas', 'Haddad', 'Arian', 'Kouri', 'Deeb', 'Tom
a', 'Halabi', 'Nazari', 'Saliba', 'Fakhoury', 'Hadad', 'Baba', 'Mansour', 'Say
egh', 'Antar', 'Deeb', 'Morcos', 'Shalhoub', 'Sarraf', 'Amari', 'Wasem', 'Gani
m', 'Tuma', 'Fakhoury', 'Hadad', 'Hakimi', 'Nader', 'Said', 'Ganim', 'Daher',
'Ganem', 'Tuma', 'Boutros', 'Aswad', 'Sarkis', 'Daher', 'Toma', 'Boutros', 'Ka
naan', 'Antar', 'Gerges', 'Kouri', 'Maroun', 'Wasem', 'Dagher', 'Naifeh', 'Bis
hara', 'Ba', 'Cham', 'Kalb', 'Bazzi', 'Bitar', 'Hadad', 'Moghadam', 'Sleiman',
'Shamoun', 'Antar', 'Atiyeh', 'Koury', 'Nahas', 'Kouri', 'Maroun', 'Nassar',
'Sayegh', 'Haik', 'Ghanem', 'Sayegh', 'Salib', 'Cham', 'Bata', 'Touma', 'Antou
n', 'Antar', 'Bata', 'Botros', 'Shammas', 'Ganim', 'Sleiman', 'Seif', 'Moghada
m', 'Ba', 'Tannous', 'Bazzi', 'Seif', 'Salib', 'Hadad', 'Quraishi', 'Halabi',
'Essa', 'Bahar', 'Kattan', 'Boutros', 'Nahas', 'Sabbagh', 'Kanaan', 'Sayegh',
'Said', 'Botros', 'Najjar', 'Toma', 'Bata', 'Atiyeh', 'Halabi', 'Tannous', 'Ko
uri', 'Shamoon', 'Kassis', 'Haddad', 'Tuma', 'Mansour', 'Antar', 'Kassis', 'Ka
```

```
lb', 'Basara', 'Rahal', 'Mansour', 'Handal', 'Morcos', 'Fakhoury', 'Hadad', 'M
orcos', 'Kouri', 'Quraishi', 'Almasi', 'Awad', 'Naifeh', 'Koury', 'Asker', 'Ma
roun', 'Fakhoury', 'Sabbag', 'Sarraf', 'Shamon', 'Assaf', 'Boutros', 'Malouf',
'Nassar', 'Qureshi', 'Ghanem', 'Srour', 'Almasi', 'Qureshi', 'Ghannam', 'Musta
fa', 'Najjar', 'Kassab', 'Shadid', 'Shamoon', 'Morcos', 'Atiyeh', 'Isa', 'Ba',
'Baz', 'Asker', 'Seif', 'Asghar', 'Hajjar', 'Deeb', 'Essa', 'Qureshi', 'Abbou
d', 'Ganem', 'Haddad', 'Koury', 'Nassar', 'Abadi', 'Toma', 'Tannous', 'Harb',
'Issa', 'Khouri', 'Mifsud', 'Kalb', 'Gaber', 'Ganim', 'Boulos', 'Samaha', 'Had
dad', 'Sabbag', 'Wasem', 'Dagher', 'Rahal', 'Atiyeh', 'Antar', 'Asghar', 'Mans
our', 'Awad', 'Boulos', 'Sarraf', 'Deeb', 'Abadi', 'Nazari', 'Daher', 'Gerge
s', 'Shamoon', 'Gaber', 'Amari', 'Sarraf', 'Nazari', 'Saliba', 'Naifeh', 'Naza
ri', 'Hakimi', 'Shamon', 'Abboud', 'Quraishi', 'Tahan', 'Safar', 'Hajjar', 'Sr
our', 'Gaber', 'Shalhoub', 'Attia', 'Safar', 'Said', 'Ganem', 'Nader', 'Asgha
r', 'Mustafa', 'Said', 'Antar', 'Botros', 'Nader', 'Ghannam', 'Asfour', 'Taha
n', 'Mansour', 'Attia', 'Touma', 'Najjar', 'Kassis', 'Abboud', 'Bishara', 'Baz
zi', 'Shalhoub', 'Shalhoub', 'Safar', 'Khoury', 'Nazari', 'Sabbag', 'Sleiman',
'Atiyeh', 'Kouri', 'Bitar', 'Zogby', 'Ghanem', 'Assaf', 'Abadi', 'Arian', 'Sha
lhoub', 'Khoury', 'Morcos', 'Shamon', 'Wasem', 'Abadi', 'Antoun', 'Baz', 'Nase
r', 'Assaf', 'Saliba', 'Nader', 'Mikhail', 'Naser', 'Daher', 'Morcos', 'Awad',
'Nahas', 'Sarkis', 'Malouf', 'Mustafa', 'Fakhoury', 'Ghannam', 'Shadid', 'Gabe
r', 'Koury', 'Atiyeh', 'Shamon', 'Boutros', 'Sarraf', 'Arian', 'Fakhoury', 'Ab
adi', 'Kassab', 'Nahas', 'Quraishi', 'Mansour', 'Samaha', 'Wasem', 'Seif', 'Fa
khoury', 'Saliba', 'Cham', 'Bahar', 'Shamoun', 'Essa', 'Shamon', 'Asfour', 'Bi
tar', 'Cham', 'Tahan', 'Tannous', 'Daher', 'Khoury', 'Shamon', 'Bahar', 'Qurai
shi', 'Ghannam', 'Kassab', 'Zogby', 'Basara', 'Shammas', 'Arian', 'Sayegh', 'N
aifeh', 'Mifsud', 'Sleiman', 'Arian', 'Kassis', 'Shamoun', 'Kassis', 'Harb',
'Mustafa', 'Boulos', 'Asghar', 'Shamon', 'Kanaan', 'Atiyeh', 'Kassab', 'Taha
n', 'Bazzi', 'Kassis', 'Qureshi', 'Basara', 'Shalhoub', 'Sayegh', 'Haik', 'Att
ia', 'Maroun', 'Kassis', 'Sarkis', 'Harb', 'Assaf', 'Kattan', 'Antar', 'Sleima
n', 'Touma', 'Sarraf', 'Bazzi', 'Boulos', 'Baz', 'Issa', 'Shamon', 'Shadid',
'Deeb', 'Sabbag', 'Wasem', 'Awad', 'Mansour', 'Saliba', 'Fakhoury', 'Arian',
'Bishara', 'Dagher', 'Bishara', 'Koury', 'Fakhoury', 'Naser', 'Nader', 'Anta
r', 'Gerges', 'Handal', 'Hanania', 'Shadid', 'Gerges', 'Kassis', 'Essa', 'Assa
f', 'Shadid', 'Seif', 'Shalhoub', 'Shamoun', 'Hajjar', 'Baba', 'Sayegh', 'Must
afa', 'Sabbagh', 'Isa', 'Najjar', 'Tannous', 'Hanania', 'Ganem', 'Gerges', 'Fa
khoury', 'Mifsud', 'Nahas', 'Bishara', 'Bishara', 'Abadi', 'Sarkis', 'Masih',
'Isa', 'Attia', 'Kalb', 'Essa', 'Boulos', 'Basara', 'Halabi', 'Halabi', 'Daghe
r', 'Attia', 'Kassis', 'Tuma', 'Gerges', 'Ghannam', 'Toma', 'Baz', 'Asghar',
'Zogby', 'Aswad', 'Hadad', 'Dagher', 'Naser', 'Shadid', 'Atiyeh', 'Zogby', 'Ab
boud', 'Tannous', 'Khouri', 'Atiyeh', 'Ganem', 'Maalouf', 'Isa', 'Maroun', 'Is
sa', 'Khouri', 'Harb', 'Nader', 'Awad', 'Nahas', 'Said', 'Baba', 'Totah', 'Gan
im', 'Handal', 'Mansour', 'Basara', 'Malouf', 'Said', 'Botros', 'Samaha', 'Saf
ar', 'Tahan', 'Botros', 'Shamoun', 'Handal', 'Sarraf', 'Malouf', 'Bishara', 'A
swad', 'Khouri', 'Baz', 'Asker', 'Toma', 'Koury', 'Gerges', 'Bishara', 'Boulo
s', 'Najjar', 'Aswad', 'Shamon', 'Kouri', 'Srour', 'Assaf', 'Tannous', 'Atti
a', 'Mustafa', 'Kattan', 'Asghar', 'Amari', 'Shadid', 'Said', 'Bazzi', 'Masi
h', 'Antar', 'Fakhoury', 'Shadid', 'Masih', 'Handal', 'Sarraf', 'Kassis', 'Sal
ib', 'Hajjar', 'Totah', 'Koury', 'Totah', 'Mustafa', 'Sabbagh', 'Moghadam', 'T
oma', 'Srour', 'Almasi', 'Totah', 'Maroun', 'Kattan', 'Naifeh', 'Sarkis', 'Mik
hail', 'Nazari', 'Boutros', 'Guirguis', 'Gaber', 'Kassis', 'Masih', 'Hanania',
'Maloof', 'Quraishi', 'Cham', 'Hadad', 'Tahan', 'Bitar', 'Arian', 'Gaber', 'Ba
z', 'Mansour', 'Kalb', 'Sarkis', 'Attia', 'Antar', 'Asfour', 'Said', 'Essa',
'Koury', 'Hadad', 'Tuma', 'Moghadam', 'Sabbagh', 'Amari', 'Dagher', 'Srour',
'Antoun', 'Sleiman', 'Maroun', 'Tuma', 'Nahas', 'Hanania', 'Sayegh', 'Amari',
'Sabbagh', 'Said', 'Cham', 'Asker', 'Nassar', 'Bitar', 'Said', 'Dagher', 'Safa
r', 'Khouri', 'Totah', 'Khoury', 'Salib', 'Basara', 'Abboud', 'Baz', 'Isa', 'C
ham', 'Amari', 'Mifsud', 'Hadad', 'Rahal', 'Khoury', 'Bazzi', 'Basara', 'Tota
h', 'Ghannam', 'Koury', 'Malouf', 'Zogby', 'Zogby', 'Boutros', 'Nassar', 'Hand
al', 'Hajjar', 'Maloof', 'Abadi', 'Maroun', 'Mifsud', 'Kalb', 'Amari', 'Hakim
i', 'Boutros', 'Masih', 'Kattan', 'Haddad', 'Arian', 'Nazari', 'Assaf', 'Atti
a', 'Wasem', 'Gerges', 'Asker', 'Tahan', 'Fakhoury', 'Shadid', 'Sarraf', 'Atti
a', 'Naifeh', 'Aswad', 'Deeb', 'Tannous', 'Totah', 'Cham', 'Baba', 'Najjar',
'Hajjar', 'Shamoon', 'Handal', 'Awad', 'Guirguis', 'Awad', 'Ganem', 'Naifeh',
'Khoury', 'Hajjar', 'Moghadam', 'Mikhail', 'Ghannam', 'Guirguis', 'Tannous',
'Kanaan', 'Handal', 'Khoury', 'Kalb', 'Qureshi', 'Najjar', 'Atiyeh', 'Gerges',
'Nassar', 'Tahan', 'Hadad', 'Fakhoury', 'Salib', 'Wasem', 'Bitar', 'Fakhoury',
'Attia', 'Awad', 'Totah', 'Deeb', 'Touma', 'Botros', 'Nazari', 'Nahas', 'Kour
i', 'Ghannam', 'Assaf', 'Asfour', 'Sarraf', 'Naifeh', 'Toma', 'Asghar', 'Abbou
d', 'Issa', 'Sabbag', 'Sabbagh', 'Isa', 'Koury', 'Kattan', 'Shamoon', 'Rahal',
'Kalb', 'Naser', 'Masih', 'Sayegh', 'Dagher', 'Asker', 'Maroun', 'Dagher', 'Sl
eiman', 'Botros', 'Sleiman', 'Harb', 'Tahan', 'Tuma', 'Said', 'Hadad', 'Samah
a', 'Harb', 'Cham', 'Atiyeh', 'Haik', 'Malouf', 'Bazzi', 'Harb', 'Malouf', 'Gh
anem', 'Cham', 'Asghar', 'Samaha', 'Khouri', 'Nassar', 'Rahal', 'Baz', 'Kalb',
'Rahal', 'Gerges', 'Cham', 'Sayegh', 'Shadid', 'Morcos', 'Shamoon', 'Hakimi',
'Shamoon', 'Qureshi', 'Ganim', 'Shadid', 'Khoury', 'Boutros', 'Hanania', 'Anto
un', 'Naifeh', 'Deeb', 'Samaha', 'Awad', 'Asghar', 'Awad', 'Saliba', 'Shamou
n', 'Mikhail', 'Hakimi', 'Mikhail', 'Cham', 'Halabi', 'Sarkis', 'Kattan', 'Naz
ari', 'Safar', 'Morcos', 'Khoury', 'Essa', 'Nassar', 'Haik', 'Shadid', 'Fakhou
ry', 'Najjar', 'Arian', 'Botros', 'Daher', 'Saliba', 'Saliba', 'Kattan', 'Hajj
ar', 'Nader', 'Daher', 'Nassar', 'Maroun', 'Harb', 'Nassar', 'Antar', 'Shamma
s', 'Toma', 'Antar', 'Koury', 'Nader', 'Botros', 'Bahar', 'Najjar', 'Maloof',
'Salib', 'Malouf', 'Mansour', 'Bazzi', 'Atiyeh', 'Kanaan', 'Bishara', 'Hakim
i', 'Saliba', 'Tuma', 'Mifsud', 'Hakimi', 'Assaf', 'Nassar', 'Sarkis', 'Bita
r', 'Isa', 'Halabi', 'Shamon', 'Qureshi', 'Bishara', 'Maalouf', 'Srour', 'Boul
os', 'Safar', 'Shamoun', 'Ganim', 'Abadi', 'Koury', 'Shadid', 'Zogby', 'Boutro
s', 'Shadid', 'Hakimi', 'Bazzi', 'Isa', 'Totah', 'Salib', 'Shamoon', 'Gaber',
'Antar', 'Antar', 'Najjar', 'Fakhoury', 'Malouf', 'Salib', 'Rahal', 'Boulos',
'Attia', 'Said', 'Kassis', 'Bahar', 'Bazzi', 'Srour', 'Antar', 'Nahas', 'Kassi
s', 'Samaha', 'Quraishi', 'Asghar', 'Asker', 'Antar', 'Totah', 'Haddad', 'Malo
of', 'Kouri', 'Basara', 'Bata', 'Antar', 'Shammas', 'Arian', 'Gerges', 'Seif',
'Almasi', 'Tuma', 'Shamoon', 'Khoury', 'Hakimi', 'Abboud', 'Baz', 'Seif', 'Iss
a', 'Nazari', 'Harb', 'Shammas', 'Amari', 'Totah', 'Malouf', 'Sarkis', 'Nase
r', 'Zogby', 'Handal', 'Naifeh', 'Cham', 'Hadad', 'Gerges', 'Kalb', 'Shalhou
b', 'Saliba', 'Tannous', 'Tahan', 'Tannous', 'Kassis', 'Shadid', 'Sabbag', 'Ta
han', 'Abboud', 'Nahas', 'Shamoun', 'Dagher', 'Botros', 'Amari', 'Maalouf', 'A
wad', 'Gerges', 'Shamoon', 'Haddad', 'Salib', 'Attia', 'Kassis', 'Sleiman', 'M
aloof', 'Maroun', 'Koury', 'Asghar', 'Kalb', 'Asghar', 'Touma', 'Ganim', 'Raha
l', 'Haddad', 'Zogby', 'Mansour', 'Guirguis', 'Touma', 'Maroun', 'Tannous', 'H
akimi', 'Baba', 'Toma', 'Botros', 'Sarraf', 'Koury', 'Sarraf', 'Nassar', 'Bout
ros', 'Guirguis', 'Qureshi', 'Aswad', 'Basara', 'Toma', 'Tuma', 'Mansour', 'B
a', 'Naifeh', 'Mikhail', 'Amari', 'Shamon', 'Malouf', 'Boutros', 'Hakimi', 'Sr
our', 'Morcos', 'Halabi', 'Bazzi', 'Abadi', 'Shamoun', 'Haddad', 'Baz', 'Bab
a', 'Hadad', 'Saliba', 'Haddad', 'Maalouf', 'Bitar', 'Shammas', 'Totah', 'Sai
d', 'Najjar', 'Mikhail', 'Samaha', 'Boulos', 'Kalb', 'Shamon', 'Shamoun', 'Sei
f', 'Touma', 'Hajjar', 'Hadad', 'Atiyeh', 'Totah', 'Mansour', 'Nazari', 'Qurai
shi', 'Ba', 'Sarkis', 'Gerges', 'Shalhoub', 'Nazari', 'Issa', 'Salib', 'Shalho
ub', 'Nassar', 'Guirguis', 'Daher', 'Hakimi', 'Attia', 'Cham', 'Isa', 'Hakim
i', 'Amari', 'Boutros', 'Sarraf', 'Antoun', 'Botros', 'Haddad', 'Tahan', 'Bish
ara', 'Shalhoub', 'Safar', 'Haik', 'Tahan', 'Seif', 'Awad', 'Antoun', 'Atiye
h', 'Samaha', 'Assaf', 'Guirguis', 'Hadad', 'Sayegh', 'Khouri', 'Asghar', 'Tan
nous', 'Maalouf', 'Khouri', 'Hajjar', 'Abadi', 'Ghanem', 'Salib', 'Botros', 'B
itar', 'Bishara', 'Quraishi', 'Boutros', 'Aswad', 'Srour', 'Shamon', 'Abboud',
'Almasi', 'Baba', 'Tahan', 'Essa', 'Sabbag', 'Issa', 'Abadi', 'Abboud', 'Bazz
i', 'Nader', 'Bahar', 'Ghannam', 'Asghar', 'Gaber', 'Sayegh', 'Guirguis', 'Sro
ur', 'Asghar', 'Quraishi', 'Sayegh', 'Rahal', 'Tahan', 'Morcos', 'Cham', 'Kana
an', 'Nahas', 'Essa', 'Mifsud', 'Kouri', 'Isa', 'Saliba', 'Asfour', 'Guirgui
s', 'Isa', 'Bishara', 'Assaf', 'Naser', 'Moghadam', 'Kalb', 'Baba', 'Guirgui
s', 'Naifeh', 'Bitar', 'Samaha', 'Abboud', 'Hadad', 'Ghannam', 'Hanania', 'Sha
did', 'Totah', 'Tahan', 'Toma', 'Maloof', 'Botros', 'Issa', 'Deeb', 'Nahas',
'Khoury', 'Sayegh', 'Harb', 'Said', 'Guirguis', 'Nader', 'Harb', 'Atiyeh', 'Zo
gby', 'Basara', 'Nassar', 'Kalb', 'Khoury', 'Mifsud', 'Wasem', 'Handal', 'Gani
m', 'Harb', 'Ganim', 'Malouf', 'Sayegh', 'Khoury', 'Sabbag', 'Sabbag', 'Boulo
s', 'Malouf', 'Gaber', 'Shammas', 'Fakhoury', 'Halabi', 'Haddad', 'Asker', 'Mo
rcos', 'Hanania', 'Amari', 'Kassab', 'Malouf', 'Khouri', 'Moghadam', 'Totah',
'Maloof', 'Atiyeh', 'Abadi', 'Baz', 'Khoury', 'Arian', 'Handal', 'Dagher', 'Aw
ad', 'Atiyeh', 'Arian', 'Khoury', 'Amari', 'Attia', 'Ganim', 'Nader', 'Daghe
r', 'Sabbag', 'Halabi', 'Khouri', 'Khouri', 'Saliba', 'Mifsud', 'Koury', 'Awa
d', 'Bahar', 'Mustafa', 'Kassis', 'Gaber', 'Mifsud', 'Bishara', 'Asker', 'Naha
s', 'Wasem', 'Sleiman', 'Bata', 'Daher', 'Antar', 'Isa', 'Ganim', 'Rahal', 'To
ma', 'Rahal', 'Shamoun', 'Maloof', 'Hakimi', 'Safar', 'Gerges', 'Hanania', 'Ko
ury', 'Assaf', 'Safar', 'Gerges', 'Ganim', 'Morcos', 'Awad', 'Arian', 'Tahan',
'Sleiman', 'Asker', 'Boulos', 'Koury', 'Mifsud', 'Sabbag', 'Dagher', 'Bazzi',
'Mustafa', 'Almasi', 'Handal', 'Isa', 'Guirguis', 'Sayegh', 'Ganim', 'Ghanem',
'Toma', 'Mustafa', 'Basara', 'Bitar', 'Samaha', 'Mifsud', 'Tahan', 'Issa', 'Sa
lib', 'Khoury', 'Hadad', 'Haik', 'Gaber', 'Mansour', 'Hakimi', 'Ba', 'Mustaf
a', 'Gaber', 'Kattan', 'Koury', 'Awad', 'Maalouf', 'Masih', 'Harb', 'Atiyeh',
'Zogby', 'Nahas', 'Assaf', 'Morcos', 'Ganem', 'Ganem', 'Wasem', 'Fakhoury', 'G
hanem', 'Salib', 'Khouri', 'Maloof', 'Khouri', 'Shalhoub', 'Issa', 'Najjar',
'Kassis', 'Mustafa', 'Sayegh', 'Kassis', 'Hajjar', 'Nader', 'Sarkis', 'Tahan',
'Haddad', 'Antar', 'Sayegh', 'Zogby', 'Mifsud', 'Kassab', 'Hanania', 'Bishar
a', 'Shamoun', 'Abboud', 'Mustafa', 'Sleiman', 'Abadi', 'Sarraf', 'Zogby', 'Da
her', 'Issa', 'Nazari', 'Shamon', 'Tuma', 'Asghar', 'Morcos', 'Mifsud', 'Cha
m', 'Sarraf', 'Antar', 'Ba', 'Aswad', 'Mikhail', 'Kouri', 'Mikhail', 'Awad',
'Halabi', 'Moghadam', 'Mikhail', 'Naifeh', 'Kattan', 'Shammas', 'Malouf', 'Naj
jar', 'Srour', 'Masih', 'Fakhoury', 'Khouri', 'Assaf', 'Mifsud', 'Malouf', 'Ab
boud', 'Shamoon', 'Mansour', 'Halabi', 'Ganem', 'Deeb', 'Wasem', 'Kalb', 'Safa
r', 'Tuma', 'Fakhoury', 'Toma', 'Guirguis', 'Kassab', 'Nader', 'Handal', 'Bab
a', 'Fakhoury', 'Haik', 'Guirguis', 'Seif', 'Almasi', 'Shamon', 'Ba', 'Salib',
'Zogby', 'Koury', 'Najjar', 'Atiyeh', 'Morcos', 'Antar', 'Awad', 'Hadad', 'Mar
oun', 'Touma', 'Almasi', 'Kassis', 'Arian', 'Malouf', 'Koury', 'Sarraf', 'Hada
d', 'Bata', 'Tuma', 'Sarkis', 'Quraishi', 'Gaber', 'Abadi', 'Nader', 'Bazzi',
'Ghannam', 'Botros', 'Deeb', 'Awad', 'Kattan', 'Kanaan', 'Sarraf', 'Nahas', 'A
ssaf', 'Shadid', 'Gaber', 'Samaha', 'Harb', 'Samaha', 'Zogby', 'Atiyeh', 'Must
afa', 'Hanania', 'Isa', 'Almasi', 'Bitar', 'Fakhoury', 'Moghadam', 'Handal',
'Seif', 'Mustafa', 'Rahal', 'Antoun', 'Kassab', 'Bazzi', 'Hadad', 'Nader', 'Tu
ma', 'Basara', 'Totah', 'Nassar', 'Seif', 'Nassar', 'Daher', 'Daher', 'Maalou
f', 'Rahal', 'Quraishi', 'Hadad', 'Bahar', 'Sabbag', 'Halabi', 'Tuma', 'Antou
n', 'Boutros', 'Gerges', 'Bishara', 'Baba', 'Zogby', 'Nahas', 'Atiyeh', 'Raha
l', 'Sabbagh', 'Bitar', 'Botros', 'Tuma', 'Ganim', 'Handal', 'Daher', 'Boutro
s', 'Khouri', 'Maroun', 'Mifsud', 'Arian', 'Safar', 'Koury', 'Deeb', 'Shamou
n', 'Cham', 'Asghar', 'Morcos', 'Tahan', 'Salib', 'Aswad', 'Shadid', 'Saliba',
'Ganim', 'Haik', 'Kattan', 'Antoun', 'Hajjar', 'Toma', 'Toma', 'Antoun', 'Taha
n', 'Haik', 'Kassis', 'Shamoun', 'Shammas', 'Kassis', 'Shadid', 'Samaha', 'Sar
raf', 'Nader', 'Ganem', 'Zogby', 'Maloof', 'Kalb', 'Gerges', 'Seif', 'Nahas',
'Arian', 'Asfour', 'Hakimi', 'Ba', 'Handal', 'Abadi', 'Harb', 'Nader', 'Asgha
r', 'Sabbag', 'Touma', 'Amari', 'Kanaan', 'Hajjar', 'Said', 'Sarraf', 'Hadda
d', 'Mifsud', 'Shammas', 'Sleiman', 'Asfour', 'Deeb', 'Kattan', 'Naser', 'Sai
d', 'Bishara', 'Harb', 'Morcos', 'Sayegh', 'Said', 'Naser', 'Aswad', 'Seif',
'Kouri', 'Dagher', 'Shamon', 'Hadad', 'Handal', 'Tuma', 'Shamon', 'Hakimi', 'R
ahal', 'Hadad', 'Ghannam', 'Almasi', 'Daher', 'Handal', 'Malouf', 'Mansour',
'Sabbagh', 'Sabbag', 'Saliba', 'Haddad', 'Tahan', 'Khoury', 'Harb', 'Ganim',
'Mansour', 'Ganem', 'Handal', 'Handal', 'Antar', 'Asfour', 'Kouri', 'Cham', 'M
asih', 'Saliba', 'Qureshi', 'Daher', 'Safar', 'Assaf', 'Harb', 'Abboud', 'Hai
k', 'Ghannam', 'Maalouf', 'Daher', 'Najjar', 'Mifsud', 'Daher', 'Amari', 'Sali
ba', 'Kanaan', 'Guirguis', 'Atiyeh', 'Sleiman', 'Mikhail', 'Arian', 'Wasem',
'Attia', 'Nassar', 'Cham', 'Koury', 'Baba', 'Guirguis', 'Morcos', 'Quraishi',
'Seif', 'Sarkis', 'Moghadam', 'Ba', 'Boutros', 'Nader', 'Gerges', 'Salib', 'Sa
lib', 'Guirguis', 'Essa', 'Guirguis', 'Antoun', 'Kassis', 'Abboud', 'Najjar',
'Aswad', 'Srour', 'Mifsud', 'Ghanem', 'Bitar', 'Ghannam', 'Asghar', 'Deeb', 'K
alb', 'Nader', 'Srour', 'Attia', 'Shamon', 'Bata', 'Nahas', 'Gerges', 'Kanaa
n', 'Kassis', 'Sarkis', 'Maloof', 'Almasi', 'Nassar', 'Saliba', 'Arian', 'Ghan
em', 'Awad', 'Naifeh', 'Boutros', 'Fakhoury', 'Sabbag', 'Antar', 'Tahan', 'Mus
tafa', 'Almasi', 'Shammas', 'Totah', 'Boutros', 'Cham', 'Shamon', 'Ganim', 'Gh
anem', 'Assaf', 'Khoury', 'Naifeh', 'Bahar', 'Quraishi', 'Bishara', 'Cham', 'A
sfour', 'Ghannam', 'Khoury', 'Sayegh', 'Hanania', 'Maroun', 'Kouri', 'Sarkis',
'Haik', 'Basara', 'Salib', 'Shammas', 'Fakhoury', 'Nahas', 'Ganim', 'Botros',
'Arian', 'Shalhoub', 'Hadad', 'Mustafa', 'Shalhoub', 'Kassab', 'Asker', 'Botro
s', 'Kanaan', 'Gaber', 'Bazzi', 'Sayegh', 'Nassar', 'Kassis', 'Fakhoury', 'Kas
sis', 'Amari', 'Sarraf', 'Mifsud', 'Salib', 'Samaha', 'Mustafa', 'Asfour', 'Na
jjar', 'Essa', 'Naifeh', 'Cham', 'Sarraf', 'Moghadam', 'Fakhoury', 'Assaf', 'A
lmasi', 'Asghar', 'Nader', 'Kalb', 'Shamoun', 'Gerges', 'Wasem', 'Morcos', 'Na
der', 'Said', 'Safar', 'Quraishi', 'Samaha', 'Kassab', 'Deeb', 'Sarraf', 'Raha
l', 'Naifeh', 'Ba', 'Nazari', 'Ganim', 'Arian', 'Asker', 'Touma', 'Kassab', 'T
ahan', 'Mansour', 'Morcos', 'Shammas', 'Baba', 'Morcos', 'Isa', 'Moghadam', 'G
anem', 'Baz', 'Totah', 'Nader', 'Kouri', 'Guirguis', 'Koury', 'Zogby', 'Basar
a', 'Baz', 'Deeb', 'Mustafa', 'Shadid', 'Awad', 'Sarraf', 'Quraishi', 'Kanaa
n', 'Tahan', 'Ghannam', 'Shammas', 'Abboud', 'Najjar', 'Bishara', 'Tuma', 'Sro
ur', 'Mifsud', 'Srour', 'Hajjar', 'Qureshi', 'Bitar', 'Hadad', 'Almasi', 'Wase
m', 'Abadi', 'Maroun', 'Baz', 'Koury', 'Ganem', 'Awad', 'Maalouf', 'Mifsud',
'Haik', 'Sleiman', 'Arian', 'Seif', 'Mansour', 'Koury', 'Kattan', 'Koury', 'As
wad', 'Ba', 'Rahal', 'Zogby', 'Bahar', 'Fakhoury', 'Samaha', 'Sarraf', 'Mifsu
d', 'Antar', 'Moghadam', 'Botros', 'Srour', 'Sabbag', 'Sayegh', 'Rahal', 'Atti
a', 'Naifeh', 'Saliba', 'Mustafa', 'Amari', 'Issa', 'Masih', 'Khouri', 'Hadda
d', 'Kalb', 'Bazzi', 'Salib', 'Hanania', 'Shamoon', 'Tuma', 'Cham', 'Antoun',
'Wasem', 'Kouri', 'Ghanem', 'Wasem', 'Khoury', 'Assaf', 'Ganem', 'Seif', 'Nade
r', 'Essa', 'Shadid', 'Botros', 'Sleiman', 'Bishara', 'Basara', 'Maalouf', 'Is
sa', 'Nassar', 'Moghadam', 'Ganim', 'Kassis', 'Antoun', 'Said', 'Khouri', 'Sal
```

```
ib', 'Baz', 'Sarkis', 'Tuma', 'Naifeh', 'Najjar', 'Asker', 'Khouri', 'Mustaf
a', 'Najjar', 'Sabbag', 'Malouf', 'Wasem', 'Maalouf', 'Gaber', 'Said', 'Zogb
y', 'Bahar', 'Hanania', 'Shalhoub', 'Abadi', 'Handal', 'Qureshi', 'Kanaan', 'A
bboud', 'Mifsud', 'Touma', 'Ganim', 'Bishara', 'Bazzi', 'Gaber', 'Haik', 'Ghan
em', 'Sarraf', 'Sarkis', 'Mustafa', 'Baz', 'Kanaan', 'Nazari', 'Bahar', 'Malou
f', 'Quraishi', 'Kattan', 'Arian', 'Shadid', 'Tuma', 'Nader', 'Khoury', 'Safa
r', 'Wasem', 'Toma', 'Haddad', 'Quraishi', 'Nassar', 'Kanaan', 'Gaber', 'Hadda
d', 'Rahal', 'Koury', 'Harb', 'Mikhail', 'Dagher', 'Shadid', 'Boutros', 'Mikha
il', 'Khouri', 'Nader', 'Issa', 'Harb', 'Dagher', 'Gerges', 'Morcos', 'Essa',
'Fakhoury', 'Tuma', 'Kattan', 'Totah', 'Qureshi', 'Nahas', 'Bitar', 'Tahan',
'Daher', 'Shammas', 'Kouri', 'Ganim', 'Daher', 'Awad', 'Malouf', 'Mustafa', 'A
swad']
Chinese
['Ang', 'AuYong', 'Bai', 'Ban', 'Bao', 'Bei', 'Bian', 'Bui', 'Cai', 'Cao', 'Ce
n', 'Chai', 'Chaim', 'Chan', 'Chang', 'Chao', 'Che', 'Chen', 'Cheng', 'Cheun
g', 'Chew', 'Chieu', 'Chin', 'Chong', 'Chou', 'Chu', 'Cui', 'Dai', 'Deng', 'Di
ng', 'Dong', 'Dou', 'Duan', 'Eng', 'Fan', 'Fei', 'Feng', 'Foong', 'Fung', 'Ga
n', 'Gauk', 'Geng', 'Gim', 'Gok', 'Gong', 'Guan', 'Guang', 'Guo', 'Gwock', 'Ha
n', 'Hang', 'Hao', 'Hew', 'Hiu', 'Hong', 'Hor', 'Hsiao', 'Hua', 'Huan', 'Huan
g', 'Hui', 'Huie', 'Huo', 'Jia', 'Jiang', 'Jin', 'Jing', 'Joe', 'Kang', 'Kau',
'Khoo', 'Khu', 'Kong', 'Koo', 'Kwan', 'Kwei', 'Kwong', 'Lai', 'Lam', 'Lang',
'Lau', 'Law', 'Lew', 'Lian', 'Liao', 'Lim', 'Lin', 'Ling', 'Liu', 'Loh', 'Lon
g', 'Loong', 'Luo', 'Mah', 'Mai', 'Mak', 'Mao', 'Mar', 'Mei', 'Meng', 'Miao',
'Min', 'Ming', 'Moy', 'Mui', 'Nie', 'Niu', 'OuYang', 'OwYang', 'Pan', 'Pang',
'Pei', 'Peng', 'Ping', 'Qian', 'Qin', 'Qiu', 'Quan', 'Que', 'Ran', 'Rao', 'Ron
g', 'Ruan', 'Sam', 'Seah', 'See ', 'Seow', 'Seto', 'Sha', 'Shan', 'Shang', 'Sh
ao', 'Shaw', 'She', 'Shen', 'Sheng', 'Shi', 'Shu', 'Shuai', 'Shui', 'Shum', 'S
iew', 'Siu', 'Song', 'Sum', 'Sun', 'Sze ', 'Tan', 'Tang', 'Tao', 'Teng', 'Teo
h', 'Thean', 'Thian', 'Thien', 'Tian', 'Tong', 'Tow', 'Tsang', 'Tse', 'Tsen',
'Tso', 'Tze', 'Wan', 'Wang', 'Wei', 'Wen', 'Weng', 'Won', 'Wong', 'Woo', 'Xian
g', 'Xiao', 'Xie', 'Xing', 'Xue', 'Xun', 'Yan', 'Yang', 'Yao', 'Yap', 'Yau',
'Yee', 'Yep', 'Yim', 'Yin', 'Ying', 'Yong', 'You', 'Yuan', 'Zang', 'Zeng', 'Zh
a', 'Zhan', 'Zhang', 'Zhao', 'Zhen', 'Zheng', 'Zhong', 'Zhou', 'Zhu', 'Zhuo',
'Zong', 'Zou', 'Bing', 'Chi', 'Chu', 'Cong', 'Cuan', 'Dan', 'Fei', 'Feng', 'Ga
i', 'Gao', 'Gou', 'Guan', 'Gui', 'Guo', 'Hong', 'Hou', 'Huan', 'Jian', 'Jiao',
'Jin', 'Jiu', 'Juan', 'Jue', 'Kan', 'Kuai', 'Kuang', 'Kui', 'Lao', 'Liang', 'L
u', 'Luo', 'Man', 'Nao', 'Pian', 'Qiao', 'Qing', 'Qiu', 'Rang', 'Rui', 'She',
'Shi', 'Shuo', 'Sui', 'Tai', 'Wan', 'Wei', 'Xian', 'Xie', 'Xin', 'Xing', 'Xion
g', 'Xuan', 'Yan', 'Yin', 'Ying', 'Yuan', 'Yue', 'Yun', 'Zha', 'Zhai', 'Zhan
g', 'Zhi', 'Zhuan', 'Zhui']
```

All countries

In [12]:
```
print(category_lines.keys())
```

```
dict_keys(['Arabic', 'Chinese', 'Czech', 'Dutch', 'English', 'French', 'Germa
n', 'Greek', 'Irish', 'Italian', 'Japanese', 'Korean', 'Polish', 'Portuguese',
'Russian', 'Scottish', 'Spanish', 'Vietnamese'])
```

Example: German names

In [13]:
```
print(category_lines['German'][:5])
```

```
['Abbing', 'Abel', 'Abeln', 'Abt', 'Achilles']
```

2. Train Test Split

In [14]:
```
names = []
targets = []

for k,v in category_lines.items():
    for name in v:
        names.append(name)
        targets.append(k)
```

```
print(len(names))
print(len(targets))
```

```
20074
20074
```

In [19]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(names, targets, test_size
```

In [20]:
```
print("The number of observations in the training data: ", len(X_train))
print("The number of observations in the test data: ", len(X_test))
```

```
The number of observations in the training data:  16059
The number of observations in the test data:  4015
```

3. Encode names

In [21]:
```
#function to create representation of the name
def name_rep(name):
    rep = torch.zeros(len(name), 1, n_letters) #Create a zeros tensor
    #iterate through all the characters in the name
    for index, letter in enumerate(name):
        pos = all_letters.find(letter)
        rep[index][0][pos] = 1 #Assign a value for each pos value
    return rep
```

In [22]:
```
#function to create vec representation of the language
def lang_rep(lang):
    return torch.tensor([all_categories.index(lang)], dtype = torch.long)
```

Example of name and language representation

In [23]:
```
#example of name representation
beau = name_rep("beau")
print(beau)
print(beau.shape)

lang = lang_rep("German")
print(lang)
print(lang.shape)
# print(all_categories)
```

```
tensor([[[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., 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.]],

        [[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., 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.]]])
torch.Size([4, 1, 57])
tensor([6])
torch.Size([1])
```

4. Define model

In [24]:
```
#create simple rnn network
import torch.nn as nn
class RNN_net(nn.Module):
    #Create a constructor
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN_net, self).__init__()
        self.hidden_size = hidden_size
        self.rnn_cell = nn.RNN(input_size, hidden_size)
        self.h2o = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim = 1)

    #create a forward pass function
    def forward(self, input_, hidden = None, batch_size = 1):
        out, hidden = self.rnn_cell(input_, hidden)
        output = self.h2o(hidden.view(-1, self.hidden_size))
        output = self.softmax(output)
        return output, hidden

    def init_hidden(self, batch_size = 1):
        #function to init the hidden layers
        return torch.zeros(1, batch_size, self.hidden_size)
```

5. Inference

In [25]:
```
#function to run interference
def infer(net, name, device = "cpu"):
    name_ohe = name_rep(name).to(device)

    #get the output
    output, hidden = net(name_ohe)

    if type(hidden) is tuple: #for lSTM
        hidden = hidden[0]
    index = torch.argmax(hidden)

    return output
```

In [26]:
```
#create hidden layers
n_hidden = 128 #hidden layers count

#number of languages
n_languages = len(category_lines.keys())
print("Total number of languages present: ", n_languages)

#initialize the network
net = RNN_net(input_size=n_letters, hidden_size=n_hidden, output_size=n_langu
```

```
Total number of languages present:  18
```

In [27]:
```
#check for inference
net = net.to(device)
infer(net, "kumar", device = device)
```

Out[27]:
```
tensor([[-2.9477, -3.0273, -2.9186, -2.9304, -2.8071, -2.7905, -2.8676, -2.919
2,
         -3.0071, -2.7959, -2.8193, -2.8625, -2.9019, -2.8879, -3.0303, -2.821
0,
         -2.9022, -2.8388]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
```

In [28]:
```
input = name_rep('A')

# put stuff on GPU
input = input.to(device)
hidden = torch.zeros((1,1, n_hidden)).to(device)

output, next_hidden = net(input, hidden)
output
```

Out[28]:
```
tensor([[-2.9957, -2.9867, -2.9632, -2.9747, -2.8603, -2.9040, -2.8793, -2.813
1,
         -2.9525, -2.7328, -2.7486, -2.8436, -2.9248, -2.8446, -3.0442, -2.843
3,
         -2.9362, -2.8418]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
```

In [29]:
```
input = name_rep('Albert')
hidden = torch.zeros((1,1, n_hidden))

# put stuff on GPU
input = input.to(device)
hidden = hidden.to(device)

next_hidden = hidden
for i in range(input.shape[0]):
    output, next_hidden = net(input[i].reshape(1,1,-1), next_hidden)
    print(output)
```

```
tensor([[-2.9957, -2.9867, -2.9632, -2.9747, -2.8603, -2.9040, -2.8793, -2.813
1,
         -2.9525, -2.7328, -2.7486, -2.8436, -2.9248, -2.8446, -3.0442, -2.843
3,
         -2.9362, -2.8418]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
tensor([[-3.0031, -2.9854, -2.9370, -3.0118, -2.7898, -2.8088, -2.7997, -2.848
3,
         -2.9793, -2.7378, -2.8377, -2.8630, -2.9877, -2.9383, -3.0378, -2.773
5,
         -2.9251, -2.8382]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
tensor([[-2.9733, -3.0077, -2.9775, -2.9582, -2.8080, -2.7824, -2.7881, -2.889
9,
         -2.9802, -2.7620, -2.8139, -2.8748, -2.8930, -2.9887, -3.0873, -2.849
3,
         -2.8759, -2.7918]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
tensor([[-2.9643, -2.9691, -2.8957, -2.9035, -2.8444, -2.7950, -2.8628, -2.906
4,
         -3.0787, -2.7819, -2.8322, -2.8551, -2.9770, -2.9503, -3.0153, -2.803
9,
         -2.8929, -2.7608]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
tensor([[-2.9662, -3.0093, -2.9411, -2.9419, -2.8299, -2.8497, -2.8673, -2.898
5,
         -2.9658, -2.7566, -2.8161, -2.8556, -2.9006, -2.8603, -2.9987, -2.827
1,
         -2.9602, -2.8252]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
tensor([[-2.9575, -3.0057, -2.9465, -2.9602, -2.8036, -2.7875, -2.8602, -2.933
8,
         -3.0264, -2.7572, -2.8098, -2.8025, -2.9254, -2.9394, -2.9843, -2.798
3,
         -2.9485, -2.8417]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
```

6. Visualization

In [30]:
```
#loading dataloader
# dataloader(2, X_train, y_train)
```

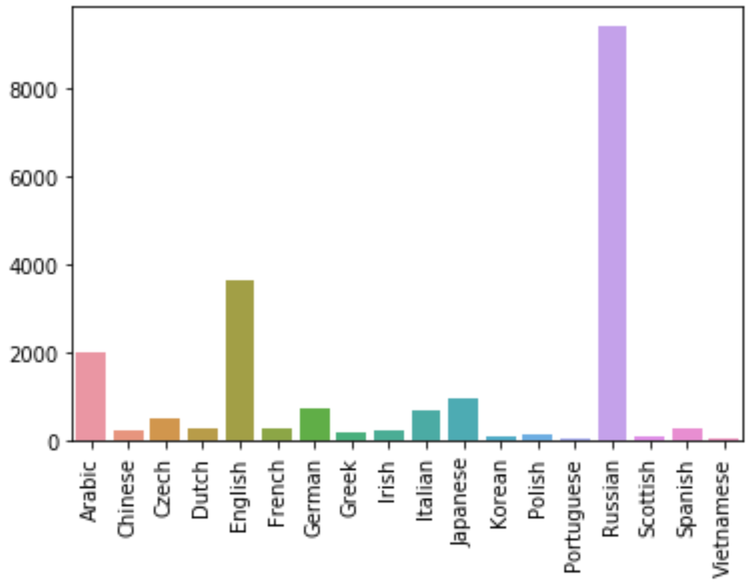
In [31]:
```
count = {}
for l in all_categories:
    count[l] = 0
for k,v in category_lines.items():
    count[k] += len(v)
```

In [32]:
```
print(count)
```

```
{'Arabic': 2000, 'Chinese': 268, 'Czech': 519, 'Dutch': 297, 'English': 3668,
'French': 277, 'German': 724, 'Greek': 203, 'Irish': 232, 'Italian': 709, 'Jap
anese': 991, 'Korean': 94, 'Polish': 139, 'Portuguese': 74, 'Russian': 9408,
'Scottish': 100, 'Spanish': 298, 'Vietnamese': 73}
```

In [36]:
```
import seaborn as sns
import matplotlib.pyplot as plt
plt_ = sns.barplot(list(count.keys()), list(count.values()))
plt_.set_xticklabels(plt_.get_xticklabels(), rotation=90)
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the
only valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



7. Dataloader & Evaluate Model

Check whether it works before training!

In [37]:
```
def dataloader(npoints, X_, y_):
    """Function to load the data"""
    to_ret = []
    for i in range(npoints):
        index_ = np.random.randint(len(X_))
        name, lang = X_[index_], y_[index_] #subset the data
        to_ret.append((name, lang, name_rep(name), lang_rep(lang)))

    return to_ret
```

In [38]:

```
#loading dataloader
dataloader(2, X_train, y_train)
```

Out[38]:

```
[('Molina',
  'Spanish',
  tensor([[[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 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., 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., 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., 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., 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.]],

          [[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.]]]),
  tensor([16])),
 ('Kerner',
  'Czech',
  tensor([[[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 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., 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.,
              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., 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., 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.,
              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.]]]]),
  tensor([2]))]
```

In [39]:

```python
def eval(net, n_points, topk, X_, y_, device = "cpu"):
    "Evaluation function"

    net = net.eval().to(device)
    data_ = dataloader(n_points, X_, y_)
    correct = 0

    #iterate
    for name, language, name_ohe, lang_rep in data_:

        #get the output
        output = infer(net, name, device)
        val, indices = output.topk(topk) #get the top k values
        indices = indices.to(device) #convert to devices

        if lang_rep in indices:
            correct += 1

    accuracy = correct/n_points
    return accuracy
```

In [44]:

```python
#test the evaluation function
eval(net, 1000, 1, X_test, y_test)
```

Out[44]:

```
0.068
```

8. Batching pytorch

In [45]:
```
#create a batched name rep

def batched_name_rep(names, max_word_size):
    rep = torch.zeros(max_word_size, len(names), n_letters)
    for name_index, name in enumerate(names):
        for letter_index, letter in enumerate(name):
            pos = all_letters.find(letter)
            rep[letter_index][name_index][pos] = 1
    return rep
```

In [46]:
```
def print_char(name_reps):
    name_reps = name_reps.view((-1, name_reps.size()[-1]))
    for t in name_reps:
        if torch.sum(t) == 0:
            print('<pad>')
        else:
            index = t.argmax()
            print(all_letters[index])
```

In [47]:
```
def batched_lang_rep(langs):
    rep = torch.zeros([len(langs)], dtype=torch.long)
    for index, lang in enumerate(langs):
        rep[index] = all_categories.index(lang)
    return rep
```

In [48]:
```
#create dataloader
def batched_dataloader(npoints, X_, y_, verbose=False, device = 'cpu'):
    names = []
    langs = []
    X_lengths = []

    for i in range(npoints):
        index_ = np.random.randint(len(X_))
        name, lang = X_[index_], y_[index_]
        X_lengths.append(len(name))
        names.append(name)
        langs.append(lang)
    max_length = max(X_lengths)

    names_rep = batched_name_rep(names, max_length).to(device)
    langs_rep = batched_lang_rep(langs).to(device)

    padded_names_rep = torch.nn.utils.rnn.pack_padded_sequence(names_rep, X_l

    if verbose:
        print(names_rep.shape, padded_names_rep.data.shape)
        print('--')

    if verbose:
        print(names)
        print_char(names_rep)
        print('--')

    if verbose:
        print_char(padded_names_rep.data)
        print('Lang Rep', langs_rep.data)
        print('Batch sizes', padded_names_rep.batch_sizes)


    return padded_names_rep.to(device), langs_rep
```

In [49]:
```
out_ = batched_name_rep(['Beau', 'Ivo'], 5)
print_char(out_)
```

```
B
I
e
v
a
o
u
<pad>
<pad>
<pad>
```

In [50]:
```
batched_dataloader(2, X_train, y_train, True)
```

```
torch.Size([7, 2, 57]) torch.Size([14, 57])
--
['Atlanov', 'Egleton']
A
E
t
g
l
l
a
e
n
t
o
o
v
n
--
A
E
t
g
l
l
a
e
n
t
o
o
v
n
Lang Rep tensor([14,  4])
Batch sizes tensor([2, 2, 2, 2, 2, 2, 2])
```

Out[50]:
```
(PackedSequence(data=tensor([[0., 0., 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., 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., 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., 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., 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., 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.],
         [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., 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., 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., 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., 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., 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., 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., 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.]]), batch_sizes=tensor([2, 2, 2, 2, 2, 2, 2]), sorted_ind
ices=tensor([0, 1]), unsorted_indices=tensor([0, 1])),
 tensor([14,  4]))
```

10. Training

10.1 Define train function

In [51]:
```python
def train(net, opt, criterion, n_points):

    opt.zero_grad()
    total_loss = 0

    data_ = dataloader(n_points, X_train, y_train)

    total_loss = 0

    for name, language, name_ohe, lang_rep in data_:

        hidden = net.init_hidden()

        for i in range(name_ohe.size()[0]):
            output, hidden = net(name_ohe[i:i+1], hidden)

        loss = criterion(output, lang_rep)
        loss.backward(retain_graph=True)

        total_loss += loss

    opt.step()
    return total_loss/n_points
```

In [54]:
```python
def train_batch(net, opt, criterion, n_points, device = 'cpu'):

    net.train().to(device)
    opt.zero_grad()

    batch_input, batch_groundtruth = batched_dataloader(n_points, X_train, y_

    output, hidden = net(batch_input)

    loss = criterion(output, batch_groundtruth)

    loss.backward()
    opt.step()
    return loss
```

10.2 Define loss and optimizer

```
In [58]: net = RNN_net(n_letters, n_hidden, n_categories)
         criterion = nn.NLLLoss().to(device)
         opt = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
```

10.4 Actual training

```
In [59]: %%time
         #time for normal training
         train(net, opt, criterion, 256)
```

```
CPU times: user 6.65 s, sys: 503 ms, total: 7.16 s
Wall time: 1.68 s
```

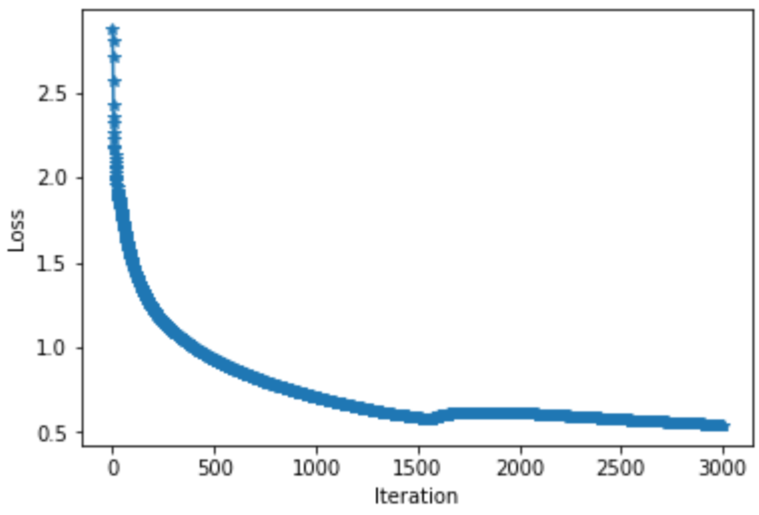
```
Out[59]: tensor(2.8584, grad_fn=<DivBackward0>)
```

11. Full training setup

```
In [60]: def train_setup(net, lr = 0.01, n_batches = 100, batch_size = 10, momentum = 
             net = net.to(device)
             criterion = nn.NLLLoss()
             opt = optim.SGD(net.parameters(), lr=lr, momentum=momentum)

             loss_arr = np.zeros(n_batches + 1)

             for i in range(n_batches):
                 loss_arr[i+1] = (loss_arr[i]*i + train_batch(net, opt, criterion, bat

                 if i%display_freq == display_freq-1:
                     clear_output(wait=True)

                     print('Iteration', i, 'Loss', loss_arr[i])
                     # print('Top-1:', eval(net, len(X_test), 1, X_test, y_test), 'Top
                     plt.figure()
                     plt.plot(loss_arr[1:i], '-*')
                     plt.xlabel('Iteration')
                     plt.ylabel('Loss')
                     plt.show()
                     print('\n\n')

             print('Top-1 Accuracy:', eval(net, len(X_test), 1, X_test, y_test, device
```

```
In [63]: %%time

         #training RNN using batch technique
         net = RNN_net(n_letters, 128, n_languages)
         train_setup(net, lr=0.15, n_batches=3200, batch_size = 512, display_freq=500)
```

```
Iteration 2999 Loss 0.5385945439338684
```



```
Top-1 Accuracy: 0.7599003735990038 Top-2 Accuracy: 0.862266500622665
CPU times: user 33min 58s, sys: 49.4 s, total: 34min 47s
Wall time: 6min 27s
```

TASK3: Do a bit of research on similar problems such as named entity recognition, find a dataset, train a model, and report your results.

The chosen data is taken from https://www.kaggle.com/abhinavwalia95/entity-annotated-corpus?select=ner_dataset.csv

And the data set is for name entity recodnition, the two columns selected are 'Word' and 'POS'
The model was trained on SGD optimizer with lr of 0.01 and a momentum of 0.9 and a loss function of NNLLoss() And the results are as follows:

Top-1 Accuracy: 0.917979162196314 Top-2 Accuracy: 0.9770784159454498
CPU times: user 55min 8s, sys: 1min, total: 56min 8s
Wall time: 10min 17s

In [1]:
```python
from __future__ import unicode_literals, print_function, division
from io import open
import torch
import glob
import os
import unicodedata
import string
import numpy as np
import torch.optim as optim
import pandas as pd
from IPython.display import clear_output
```

In [2]:
```python
from chosen_gpu import get_freer_gpu
device = torch.device(get_freer_gpu())
print("Configured device: ", device)
```

```
Configured device:  cuda:1
```

1. Load data

https://www.kaggle.com/abhinavwalia95/entity-annotated-corpus?select=ner_dataset.csv

In [3]:
```python
all_letters = string.ascii_letters + " .,;'"
n_letters = len(all_letters)

df = pd.read_csv('ner_dataset.csv', encoding= 'unicode_escape')
df = df.fillna(method="ffill")
df = df.drop(['Tag'], axis =1)
df = df.drop(['Sentence #'], axis =1)
# words = df['Word']
# tags = df['POS']
```

1.1 Build the cat_words dictionary, a list of tags per word and a list of all words

In [6]:
```python
cat_words = {}
all_tags = []
all_words = []
tags = list(set(df["POS"].values))
```

```
for i in tags:
    cat_words[i] = []

for i in range(len(df)):
    cat_words[df['POS'].iloc[i]].append(df['Word'].iloc[i])
    all_tags.append(df['POS'].iloc[i])
    all_words.append(df['Word'].iloc[i])
```

In [7]:

```
for i in tags:
    print(f'{i}:{len(cat_words[i])}')

print(len(all_words))
print(len(all_tags))
```

```
NNP:131426
``:3728
VBN:32328
PDT:147
RB:20252
WDT:3698
MD:6973
CC:23716
RBS:296
PRP:13318
EX:663
VBG:19125
NN:145807
RBR:1055
UH:24
JJR:2967
VBD:39379
.:47831
TO:23061
::795
NNPS:2521
WP:2542
,:32757
JJS:3034
WRB:2184
IN:120996
VBP:16158
WP$:99
DT:98454
;:214
VBZ:24960
FW:1
CD:24695
LRB:678
RRB:679
PRP$:8655
POS:11257
JJ:78412
$:1149
VB:24211
RP:2490
NNS:75840
1048575
1048575
```

All words

In [10]:

```
print(np.unique(df['Word']))
print(len(np.unique(df['Word'])))
```

```
['!' '"' '#' ... '\x96' '\x97' '°C']
35178
```

All tags

```
In [11]:
print(np.unique(tags))
print(len(tags))
```

```
['$' ',' '.' ':' ';' 'CC' 'CD' 'DT' 'EX' 'FW' 'IN' 'JJ' 'JJR' 'JJS' 'LRB'
 'MD' 'NN' 'NNP' 'NNPS' 'NNS' 'PDT' 'POS' 'PRP' 'PRP$' 'RB' 'RBR' 'RBS'
 'RP' 'RRB' 'TO' 'UH' 'VB' 'VBD' 'VBG' 'VBN' 'VBP' 'VBZ' 'WDT' 'WP' 'WP$'
 'WRB' '``']
42
```

2. Train Test Split

```
In [12]:
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(all_words, all_tags, test_
```

```
In [13]:
print("The number of observations in the training data: ", len(X_train))
print("The number of observations in the test data: ", len(X_test))
```

```
The number of observations in the training data:  838860
The number of observations in the test data:  209715
```

3. Encode words and tags

```
In [14]:
#function to create representation of the name
def word_rep(word):
    rep = torch.zeros(len(word), 1, n_letters) #Create a zeros tensor
    #iterate through all the characters in the name
    for index, letter in enumerate(word):
        pos = all_letters.find(letter)
        rep[index][0][pos] = 1 #Assign a value for each pos value
    return rep
```

```
In [49]:
#function to create vec representation of the language
def tag_rep(tag):
    return torch.tensor([tags.index(tag)], dtype = torch.long)
```

Example of word and tag representation

```
In [50]:
#example of name representation
beau = word_rep("beau")
print(beau)
print(beau.shape)

tag = tag_rep("DT")
print(tag)
```

```
tensor([[[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., 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.]],

        [[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., 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.]]])
torch.Size([4, 1, 57])
tensor([28])
```

4. Define model

In [51]:

```
#create simple rnn network
import torch.nn as nn
class RNN_net(nn.Module):
    #Create a constructor
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN_net, self).__init__()
        self.hidden_size = hidden_size
        self.rnn_cell = nn.RNN(input_size, hidden_size)
        self.h2o = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim = 1)

    #create a forward pass function
    def forward(self, input_, hidden = None, batch_size = 1):
        out, hidden = self.rnn_cell(input_, hidden)
        output = self.h2o(hidden.view(-1, self.hidden_size))
        output = self.softmax(output)
        return output, hidden

    def init_hidden(self, batch_size = 1):
        #function to init the hidden layers
        return torch.zeros(1, batch_size, self.hidden_size)
```

5. Inference

In [52]:

```
#function to run interference
def infer(net, name, device = "cpu"):
    name_ohe = word_rep(name).to(device)

    #get the output
    output, hidden = net(name_ohe)

    if type(hidden) is tuple: #for lSTM
        hidden = hidden[0]
    index = torch.argmax(hidden)

    return output
```

In [53]:

```
#create hidden layers
n_hidden = 128 #hidden layers count

#number of tags
n_tags= len(cat_words.keys())
print("Total number of tags present: ", n_tags)

#initialize the network
net = RNN_net(input_size=n_letters, hidden_size=n_hidden, output_size=n_tags)
```

```
Total number of tags present:  42
```

In [54]:
```
#check for inference
net = net.to(device)
infer(net, "kumar", device = device)
```

Out[54]:
```
tensor([[-3.7101, -3.7837, -3.7638, -3.6738, -3.7322, -3.8714, -3.8046, -3.740
7,
         -3.8662, -3.7178, -3.7269, -3.6650, -3.7664, -3.7766, -3.7517, -3.742
2,
         -3.8214, -3.6814, -3.7943, -3.7446, -3.6949, -3.7062, -3.8419, -3.637
1,
         -3.7452, -3.6865, -3.6187, -3.6388, -3.7371, -3.8577, -3.7310, -3.697
5,
         -3.7140, -3.7254, -3.7686, -3.6913, -3.7332, -3.7029, -3.7157, -3.775
6,
         -3.6985, -3.8031]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
```

In [55]:
```
input = word_rep('A')

# put stuff on GPU
input = input.to(device)
hidden = torch.zeros((1,1, n_hidden)).to(device)

output, next_hidden = net(input, hidden)
output
```

Out[55]:
```
tensor([[-3.7256, -3.7546, -3.7892, -3.6754, -3.6894, -3.8181, -3.7581, -3.720
2,
         -3.8300, -3.6722, -3.6850, -3.7424, -3.7674, -3.7925, -3.7973, -3.773
0,
         -3.7930, -3.6093, -3.8378, -3.7816, -3.6645, -3.7387, -3.8039, -3.678
6,
         -3.7691, -3.7115, -3.6383, -3.6441, -3.7763, -3.8337, -3.6468, -3.690
8,
         -3.6748, -3.7310, -3.8185, -3.6466, -3.7817, -3.7214, -3.7556, -3.821
9,
         -3.7053, -3.7970]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
```

In [56]:
```
input = word_rep('Albert')
hidden = torch.zeros((1,1, n_hidden))

# put stuff on GPU
input = input.to(device)
hidden = hidden.to(device)

next_hidden = hidden
for i in range(input.shape[0]):
    output, next_hidden = net(input[i].reshape(1,1,-1), next_hidden)
    print(output)
```

```
tensor([[-3.7256, -3.7546, -3.7892, -3.6754, -3.6894, -3.8181, -3.7581, -3.720
2,
         -3.8300, -3.6722, -3.6850, -3.7424, -3.7674, -3.7925, -3.7973, -3.773
0,
         -3.7930, -3.6093, -3.8378, -3.7816, -3.6645, -3.7387, -3.8039, -3.678
6,
         -3.7691, -3.7115, -3.6383, -3.6441, -3.7763, -3.8337, -3.6468, -3.690
8,
         -3.6748, -3.7310, -3.8185, -3.6466, -3.7817, -3.7214, -3.7556, -3.821
9,
         -3.7053, -3.7970]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
tensor([[-3.7417, -3.7391, -3.7758, -3.6708, -3.6568, -3.7514, -3.8263, -3.748
9,
         -3.8795, -3.7125, -3.7390, -3.6674, -3.7828, -3.6888, -3.7516, -3.747
6,
         -3.8114, -3.5825, -3.7930, -3.7743, -3.6775, -3.7027, -3.7602, -3.770
0,
         -3.7316, -3.7220, -3.6404, -3.6993, -3.7385, -3.8266, -3.7213, -3.668
2,
         -3.7103, -3.7382, -3.7721, -3.6905, -3.8237, -3.7307, -3.6579, -3.827
7,
         -3.7440, -3.8662]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
tensor([[-3.6819, -3.7936, -3.7914, -3.6230, -3.7738, -3.7930, -3.8007, -3.783
8,
         -3.8044, -3.6951, -3.6763, -3.6510, -3.8428, -3.7512, -3.7488, -3.757
0,
         -3.7820, -3.6198, -3.7406, -3.7853, -3.6886, -3.7357, -3.8209, -3.668
4,
         -3.7108, -3.6488, -3.6490, -3.6905, -3.6998, -3.8608, -3.6922, -3.728
4,
         -3.6912, -3.7314, -3.7218, -3.7582, -3.7952, -3.7664, -3.6890, -3.832
3,
         -3.7402, -3.8481]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
tensor([[-3.7646, -3.7730, -3.7986, -3.6864, -3.7179, -3.8664, -3.7660, -3.781
7,
         -3.7883, -3.7097, -3.6801, -3.6703, -3.7928, -3.7749, -3.8181, -3.781
0,
         -3.8945, -3.6499, -3.7539, -3.7415, -3.6094, -3.7148, -3.7963, -3.651
3,
         -3.7288, -3.6601, -3.6171, -3.5824, -3.7841, -3.8322, -3.6344, -3.718
8,
         -3.6641, -3.6999, -3.7786, -3.6543, -3.8588, -3.8002, -3.7411, -3.809
7,
         -3.7553, -3.7927]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
tensor([[-3.7373, -3.7632, -3.7761, -3.6725, -3.7190, -3.8795, -3.7819, -3.760
7,
         -3.8387, -3.6998, -3.7306, -3.6843, -3.7790, -3.7892, -3.7592, -3.740
2,
         -3.8419, -3.6711, -3.8186, -3.7352, -3.6813, -3.7269, -3.7961, -3.640
5,
         -3.7422, -3.6719, -3.6296, -3.5955, -3.7515, -3.8774, -3.7063, -3.712
4,
         -3.6973, -3.7212, -3.7552, -3.6494, -3.8172, -3.6979, -3.7412, -3.764
8,
         -3.7166, -3.7948]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
tensor([[-3.7370, -3.8110, -3.7710, -3.6596, -3.7193, -3.8272, -3.7731, -3.888
4,
         -3.7653, -3.7264, -3.7765, -3.6107, -3.7879, -3.7061, -3.7567, -3.750
2,
         -3.8029, -3.6482, -3.8015, -3.7090, -3.6209, -3.7321, -3.7687, -3.591
6,
         -3.6854, -3.6805, -3.7042, -3.6738, -3.7563, -3.9156, -3.6669, -3.700
9,
         -3.6472, -3.7012, -3.7883, -3.7199, -3.8495, -3.7716, -3.7169, -3.782
5,
         -3.7335, -3.8522]], device='cuda:1', grad_fn=<LogSoftmaxBackward>)
```

6. Visualization

In [57]:

```
count = {}
for l in all_tags:
    count[l] = 0
for k,v in cat_words.items():
    count[k] += len(v)
```

In [58]:

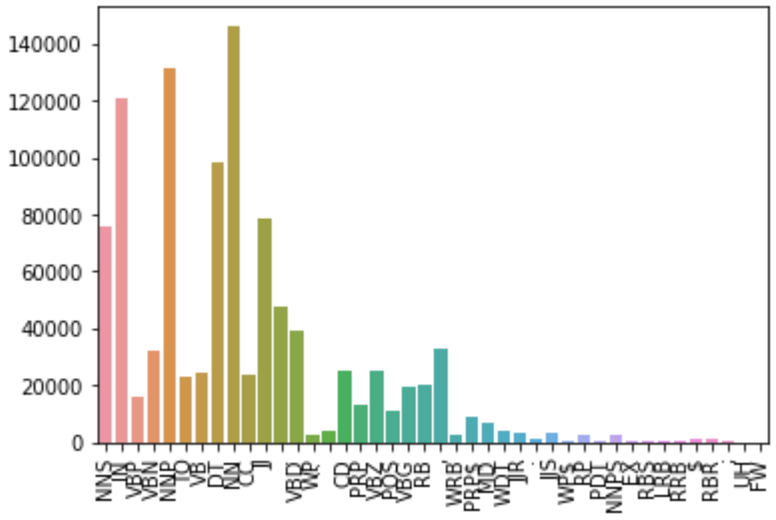
```
print(count)
```

```
{'NNS': 75840, 'IN': 120996, 'VBP': 16158, 'VBN': 32328, 'NNP': 131426, 'TO':
23061, 'VB': 24211, 'DT': 98454, 'NN': 145807, 'CC': 23716, 'JJ': 78412, '.':
47831, 'VBD': 39379, 'WP': 2542, '``': 3728, 'CD': 24695, 'PRP': 13318, 'VBZ':
24960, 'POS': 11257, 'VBG': 19125, 'RB': 20252, ',': 32757, 'WRB': 2184, 'PRP
$': 8655, 'MD': 6973, 'WDT': 3698, 'JJR': 2967, ':': 795, 'JJS': 3034, 'WP$':
```

```
99, 'RP': 2490, 'PDT': 147, 'NNPS': 2521, 'EX': 663, 'RBS': 296, 'LRB': 678,
'RRB': 679, '$': 1149, 'RBR': 1055, ';': 214, 'UH': 24, 'FW': 1}
```

In [59]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
plt_ = sns.barplot(list(count.keys()), list(count.values()))
plt_.set_xticklabels(plt_.get_xticklabels(), rotation=90)
plt.show()
```

```
/usr/local/lib/python3.6/dist-packages/seaborn/_decorators.py:43: FutureWarnin
g: Pass the following variables as keyword args: x, y. From version 0.12, the
only valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
  FutureWarning
```



7. Dataloader & Evaluate Model

Check whether it works before training!

In [60]:
```python
def dataloader(npoints, X_, y_):
    """Function to load the data"""
    to_ret = []
    for i in range(npoints):
        index_ = np.random.randint(len(X_))
        name, lang = X_[index_], y_[index_] #subset the data
        to_ret.append((name, lang, word_rep(name), tag_rep(lang)))

    return to_ret
```

In [61]:
```python
#loading dataloader
dataloader(2, X_train, y_train)
```

Out[61]:
```
[('Benshoofin',
  'NNP',
  tensor([[[0., 0., 0., 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., 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., 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., 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., 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., 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., 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., 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., 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., 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.]]]]),
  tensor([0])),
 ('Wednesday',
  'NNP',
  tensor([[[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0.,
            0., 0., 0., 0., 0., 0., 0., 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., 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., 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., 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., 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., 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., 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.]],

          [[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., 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.]]]),
  tensor([0]))]
```

In [62]:
```
def eval(net, n_points, topk, X_, y_, device = device):
    "Evaluation function"

    net = net.eval().to(device)
    data_ = dataloader(n_points, X_, y_)
    correct = 0

    #iterate
    for name, language, name_ohe, lang_rep in data_:

        name_ohe = name_ohe.to(device)
        lang_rep = lang_rep.to(device)



        #get the output
        output = infer(net, name, device)
        val, indices = output.topk(topk) #get the top k values
        indices = indices.to(device) #convert to devices

        if lang_rep in indices:
            correct += 1

    accuracy = correct/n_points
    return accuracy
```

In [63]:
```
#test the evaluation function
eval(net, 1000, 1, X_test, y_test)
```

Out[63]:
```
0.013
```

8. Batching pytorch

In [64]:
```
#create a batched name rep

def batched_name_rep(names, max_word_size):
    rep = torch.zeros(max_word_size, len(names), n_letters)
    for name_index, name in enumerate(names):
        for letter_index, letter in enumerate(name):
            pos = all_letters.find(letter)
            rep[letter_index][name_index][pos] = 1
    return rep
```

In [65]:
```
def print_char(name_reps):
    name_reps = name_reps.view((-1, name_reps.size()[-1]))
    for t in name_reps:
        if torch.sum(t) == 0:
            print('<pad>')
        else:
            index = t.argmax()
            print(all_letters[index])
```

In [66]:
```
def batched_lang_rep(langs):
    rep = torch.zeros([len(langs)], dtype=torch.long)
    for index, lang in enumerate(langs):
        rep[index] = tags.index(lang)
    return rep
```

In [67]:
```
#create dataloader
def batched_dataloader(npoints, X_, y_, verbose=False, device = device):
    names = []
    langs = []
    X_lengths = []

    for i in range(npoints):
        index_ = np.random.randint(len(X_))
        name, lang = X_[index_], y_[index_]
        X_lengths.append(len(name))
        names.append(name)
        langs.append(lang)
    max_length = max(X_lengths)

    names_rep = batched_name_rep(names, max_length).to(device)
    langs_rep = batched_lang_rep(langs).to(device)

    padded_names_rep = torch.nn.utils.rnn.pack_padded_sequence(names_rep, X_l

    if verbose:
        print(names_rep.shape, padded_names_rep.data.shape)
        print('--')

    if verbose:
        print(names)
        print_char(names_rep)
        print('--')

    if verbose:
        print_char(padded_names_rep.data)
        print('Lang Rep', langs_rep.data)
        print('Batch sizes', padded_names_rep.batch_sizes)


    return padded_names_rep.to(device), langs_rep
```

In [68]:
```
out_ = batched_name_rep(['Beau', 'Ivo'], 5)
print_char(out_)
```

```
B
I
e
v
a
o
u
<pad>
<pad>
<pad>
```

In [69]:
```
batched_dataloader(2, X_train, y_train, True)
```

```
torch.Size([7, 2, 57]) torch.Size([9, 57])
--
['to', 'billion']
t
b
```

```
o
i
<pad>
l
<pad>
l
<pad>
i
<pad>
o
<pad>
n
--
b
t
i
o
l
l
i
o
n
Lang Rep tensor([18, 32], device='cuda:1')
Batch sizes tensor([2, 2, 1, 1, 1, 1, 1])
```

Out[69]:

```
(PackedSequence(data=tensor([[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., 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., 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., 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., 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., 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., 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., 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., 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.]], device='cuda:1'), batch_sizes=tensor([2, 2, 1, 1, 1,
1, 1]), sorted_indices=tensor([1, 0], device='cuda:1'), unsorted_indices=tenso
r([1, 0], device='cuda:1')),
 tensor([18, 32], device='cuda:1'))
```

9. Training

9.1 Define train function

In [79]:

```
#basic train function

def train(net, opt, criterion, n_points):

    opt.zero_grad()
    total_loss = 0

    data_ = dataloader(n_points, X_train, y_train)

    total_loss = 0

    for name, language, name_ohe, lang_rep in data_:

        hidden = net.init_hidden()

        for i in range(name_ohe.size()[0]):
            output, hidden = net(name_ohe[i:i+1], hidden)
        loss = criterion(output, lang_rep)
        loss.backward(retain_graph=True)

        total_loss += loss

    opt.step()
    return total_loss/n_points
```

In [80]:

```
def train_batch(net, opt, criterion, n_points, device = device):

    net.train().to(device)
    opt.zero_grad()

    batch_input, batch_groundtruth = batched_dataloader(n_points, X_train, y_
    batch_input = batch_input.to(device)
    batch_groundtruth = batch_groundtruth.to(device)

    output, hidden = net(batch_input)

    loss = criterion(output, batch_groundtruth)
    loss.backward()
    opt.step()
    return loss
```

9.2 Define loss and optimizer

In [81]:
```
net = RNN_net(n_letters, n_hidden, n_tags) #.to(device)
criterion = nn.NLLLoss().to(device)
opt = optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
```

9.3 Actual training

In [82]:
```
%%time
#time for normal training
train(net, opt, criterion, 256)
```

```
CPU times: user 4.92 s, sys: 368 ms, total: 5.29 s
Wall time: 1.28 s
```

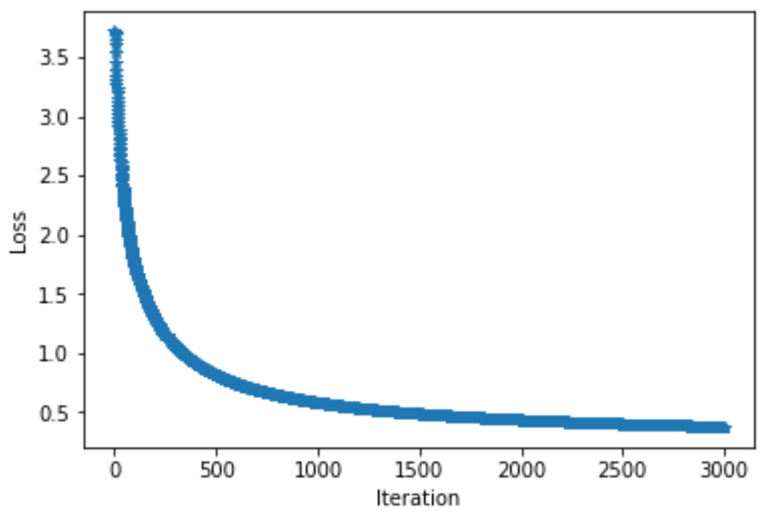
Out[82]:
```
tensor(3.6930, grad_fn=<DivBackward0>)
```

10. Full training setup

In [83]:
```
def train_setup(net, lr = 0.01, n_batches = 100, batch_size = 10, momentum =
    net = net.to(device)
    criterion = nn.NLLLoss()
    opt = optim.SGD(net.parameters(), lr=lr, momentum=momentum)

    loss_arr = np.zeros(n_batches + 1)

    for i in range(n_batches):
        loss_arr[i+1] = (loss_arr[i]*i + train_batch(net, opt, criterion, bat

        if i%display_freq == display_freq-1:
            clear_output(wait=True)

            print('Iteration', i, 'Loss', loss_arr[i])
            # print('Top-1:', eval(net, len(X_test), 1, X_test, y_test), 'Top
            plt.figure()
            plt.plot(loss_arr[1:i], '-*')
            plt.xlabel('Iteration')
            plt.ylabel('Loss')
            plt.show()
            print('\n\n')

    print('Top-1 Accuracy:', eval(net, len(X_test), 1, X_test, y_test, device
```

In [84]:
```
%%time

#training RNN using batch technique
net = RNN_net(n_letters, 128, n_tags)
train_setup(net, lr=0.15, n_batches=3200, batch_size = 512, display_freq=500)
```

```
Iteration 2999 Loss 0.3721696436405182
```



```
Top-1 Accuracy: 0.917979162196314 Top-2 Accuracy: 0.9770784159454498
CPU times: user 55min 8s, sys: 1min, total: 56min 8s
Wall time: 10min 17s
```

What I have learnt

One interesting point that I got from this lab is that training in batches is possible despite its difficulty. In order to do so, padding is required, since words come in different length. Traning in batches enables the model to be trained on the GPU which boosts the speed of the traning.