In [0]:

```
import os
import time
import math
import glob
import string
import random

import torch
import torch.nn as nn

from rnn.helpers import time_since

%matplotlib inline
```

In [2]:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6
qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Awg%3Aoauth%3A2.0%
b&scope=email%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdocs.test%20https%3A%2F%2Fwww.googleapis.
2Fauth%2Fdrive%20https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fdrive.photos.readonly%20https%3A%2F%2Fww
ogleapis.com%2Fauth%2Fpeopleapi.readonly&response_type=code

Enter your authorization code:
··········
Mounted at /content/gdrive
```

In [0]:

```
import os
os.chdir("gdrive/My Drive/MP-1/Assignment4")
```

In [0]:

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

Language recognition with an RNN

If you've ever used an online translator you've probably seen a feature that automatically detects the input language. While this might be easy to do if you input unicode characters that are unique to one or a small group of languages (like "你好" or "γεια σας"), this problem is more challenging if the input only uses the available ASCII characters. In this case, something like "těší mě" would beome "tesi me" in the ascii form. This is a more challenging problem in which the language must be recognized purely by the pattern of characters rather than unique unicode characters.

We will train an RNN to solve this problem for a small set of languages thta can be converted to romanized ASCII form. For training data it would be ideal to have a large and varied dataset in different language styles. However, it is easy to find copies of the Bible which is a large text translated to different languages but in the same easily parsable format, so we will use 20 different copies of the Bible as training data. Using the same book for all of the different languages will hopefully prevent minor overfitting that might arise if we used different books for each language (fitting to common characteristics of the individual books rather than the language).

In [8]:

```
from unidecode import unidecode as unicodeToAscii

all_characters = string.printable
n_letters = len(all_characters)

print(unicodeToAscii('těší mě'))
```

```
tesi me
```

In [0]:

```
# Read a file and split into lines
def readFile(filename):
    data = open(filename, encoding='utf-8').read().strip()
    return unicodeToAscii(data)

def get_category_data(data_path):
    # Build the category_data dictionary, a list of names per language
    category_data = {}
    all_categories = []
    for filename in glob.glob(data_path):
        category = os.path.splitext(os.path.basename(filename))[0].split('_')[0]
        all_categories.append(category)
        data = readFile(filename)
        category_data[category] = data

    return category_data, all_categories
```

The original text is split into two parts, train and test, so that we can make sure that the model is not simply memorizing the train data.

In [10]:

```
train_data_path = 'language_data/train/*_train.txt'
test_data_path = 'language_data/test/*_test.txt'

train_category_data, all_categories = get_category_data(train_data_path)
test_category_data, test_all_categories = get_category_data(test_data_path)

n_languages = len(all_categories)

print(len(all_categories))
print(all_categories)
```

```
20
['albanian', 'english', 'czech', 'danish', 'esperanto', 'finnish', 'french', 'german',
'hungarian', 'italian', 'lithuanian', 'maori', 'norwegian', 'portuguese', 'romanian', 'spanish', '
swedish', 'turkish', 'vietnamese', 'xhosa']
```

Data processing

In [0]:

```
def categoryFromOutput(output):
    top_n, top_i = output.topk(1, dim=1)
    category_i = top_i[:, 0]
    return category_i

# Turn string into long tensor
def stringToTensor(string):
    tensor = torch.zeros(len(string), requires_grad=True).long()
    for c in range(len(string)):
        tensor[c] = all_characters.index(string[c])
    return tensor

def load_random_batch(text, chunk_len, batch_size):
    input_data = torch.zeros(batch_size, chunk_len).long().to(device)
    target = torch.zeros(batch_size, 1).long().to(device)
    input_text = []
    for i in range(batch_size):
        category = all_categories[random.randint(0, len(all_categories) - 1)]
        line_start = random.randint(0, len(text[category])-chunk_len)
        category_tensor = torch.tensor([all_categories.index(category)], dtype=torch.long)
        line = text[category][line_start:line_start+chunk_len]
        input_text.append(line)
        input_data[i] = stringToTensor(line)
        target[i] = category_tensor
    return input_data, target, input_text
```

Implement Model

For this classification task, we can use the same model we implement for the generation task which is located in `rnn/model.py`. See the `MP4_P2_generation.ipynb` notebook for more instructions. In this case each output vector of our RNN will have the dimension of the number of possible languages (i.e. `n_languages`). We will use this vector to predict a distribution over the languages.

In the generation task, we used the output of the RNN at every time step to predict the next letter and our loss included the output from each of these predictions. However, in this task we use the output of the RNN at the end of the sequence to predict the language, so our loss function will use only the predicted output from the last time step.

Train RNN

In [0]:

```
from rnn.model import RNN
```

In [0]:

```
chunk_len = 50

BATCH_SIZE = 100
n_epochs = 7000
hidden_size = 150
n_layers = 3
learning_rate = 0.0001
model_type = 'lstm'

criterion = nn.CrossEntropyLoss()
rnn = RNN(n_letters, hidden_size, n_languages, model_type=model_type, n_layers=n_layers).to(device)
```

TODO: Fill in the train function. You should initialize a hidden layer representation using your RNN's `init_hidden` function, set the model gradients to zero, and loop over each time step (character) in the input tensor. For each time step compute the output of the of the RNN and the next hidden layer representation. The cross entropy loss should be computed over the last RNN output scores from the end of the sequence and the target classification tensor. Lastly, call backward on the loss and take an optimizer step.

In [0]:

```
def train(rnn, target_tensor, data_tensor, optimizer, criterion, batch_size=BATCH_SIZE):
    """
    Inputs:
    - rnn: model
    - target_target: target character data tensor of shape (batch_size, 1)
    - data_tensor: input character data tensor of shape (batch_size, chunk_len)
    - optimizer: rnn model optimizer
    - criterion: loss function
    - batch_size: data batch size

    Returns:
    - output: output from RNN from end of sequence
    - loss: computed loss value as python float

    """

    output, loss = None, None
    hidden=rnn.init_hidden(data_tensor.size(0))
    rnn.zero_grad()
    for i in range(data_tensor.size(1)):
      output,hidden= rnn(data_tensor[:,i], hidden)

    #loss += criterion(output.reshape(input.size(0), -1), target[:,i])

    #one_hot=torch.LongTensor(output.size(0),output.size(1)).zero_()
    #target = one_hot.scatter_(1, target_tensor.data, 1)
    #print(target)
    loss= criterion(output, target_tensor.squeeze(1))
    #print(loss)
    loss.backward()
    optimizer.step()
    #print(loss)
```

```
    #loss=loss.data[0]/data_tensor.size(0)
    loss=loss.item()
    ####################################
    #          YOUR CODE HERE          #
    ####################################


    ##########       END      ##########

    return output, loss
```

In [0]:

```
def evaluate(rnn, data_tensor, seq_len=chunk_len, batch_size=BATCH_SIZE):
    with torch.no_grad():
        data_tensor = data_tensor.to(device)
        hidden = rnn.init_hidden(batch_size, device=device)
        for i in range(seq_len):
            output, hidden = rnn(data_tensor[:,i], hidden)

        return output

def eval_test(rnn, category_tensor, data_tensor):
    with torch.no_grad():
        output = evaluate(rnn, data_tensor)
        loss = criterion(output, category_tensor.squeeze())
        return output, loss.item()
```

I have saved the model after a few epochs. Again run it on previously saved model.

In [19]:

```
n_iters = 5500 #100000
print_every = 50
plot_every = 50


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

start = time.time()

optimizer = torch.optim.Adam(rnn.parameters(), lr=learning_rate)


number_correct = 0
for iter in range(1, n_iters + 1):
    input_data, target_category, text_data = load_random_batch(train_category_data, chunk_len, BATC
H_SIZE)
    output, loss = train(rnn, target_category, input_data, optimizer, criterion)
    current_loss += loss
    input_test_data, target_test_category,_ = load_random_batch(test_category_data, chunk_len, BATCH
_SIZE)
    _, test_loss = eval_test(rnn, target_test_category, input_test_data)
    current_test_loss += test_loss

    guess_i = categoryFromOutput(output)
    number_correct += (target_category.squeeze()==guess_i.squeeze()).long().sum()

    # Print iter number, loss, name and guess
    if iter % print_every == 0:
        sample_idx = 0
        guess = all_categories[guess_i[sample_idx]]

        category = all_categories[int(target_category[sample_idx])]

        correct = '✓' if guess == category else '✗ (%s)' % category
        print('%d %d%% (%s) %.4f %.4f %s / %s %s' % (iter, iter / n_iters * 100, time_since(start),
loss, test_loss, text_data[sample_idx], guess, correct))
        print('Train accuracy: {}'.format(float(number_correct)/float(print_every*BATCH_SIZE)))
        number_correct = 0
```

```python
        number_correct = 0

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

```
50 0% (0m 53s) 0.1952 0.1132 qu'elle enfanta a Jacob, a Paddan-Aram, avec Dina  / spanish ✗ (frenc
h)
Train accuracy: 0.932
100 1% (1m 45s) 0.0970 0.2166 tri mil kvarcent. La filoj de Naftali laux iliaj f / esperanto ✓
Train accuracy: 0.9362
150 2% (2m 38s) 0.1005 0.1488 asa a fost. Dumnezeu a facut fiarele pamintului du / romanian ✓
Train accuracy: 0.9414
200 3% (3m 31s) 0.1907 0.2242 ore] have we afflicted our soul, and thou takest n / english ✓
Train accuracy: 0.9376
250 4% (4m 24s) 0.3100 0.3393 r flydler med Maelk og Honning, det dejligste af a / danish ✓
Train accuracy: 0.9482
300 5% (5m 17s) 0.0992 0.1768  schlief im Tempel des HERRN, wo die Lade Gottes w / german ✓
Train accuracy: 0.9458
350 6% (6m 10s) 0.1265 0.2171 ai scoale!`` Despre Beniamin a zis: ,,El este prea / romanian ✓
Train accuracy: 0.9388
400 7% (7m 3s) 0.2040 0.2594  muchos pueblos; y volveran sus espadas en rejas d / spanish ✓
Train accuracy: 0.9384
450 8% (7m 55s) 0.1517 0.2021 d og tilbad. Derpa bod Kong Ezekias og Oversterne  / danish ✓
Train accuracy: 0.9444
500 9% (8m 48s) 0.1388 0.1214 ratou, a kahore e rongo: kahore ano he manawa i o  / maori ✓
Train accuracy: 0.944
550 10% (9m 40s) 0.1437 0.2078 nd falle. Und die Agypter will ich unter die Heide / german ✓
Train accuracy: 0.9474
600 10% (10m 33s) 0.1715 0.1954 sacak,<br />Onun onunde krallarin agizlari kapanac / turkish ✓
Train accuracy: 0.9444
650 11% (11m 26s) 0.3335 0.2010 e lucru impotriva voastra.` David si tot poporul c / romanian ✓
Train accuracy: 0.945
700 12% (12m 19s) 0.1916 0.1772 d'oro il pavimento della casa. All'ingresso del sa / italian ✓
Train accuracy: 0.9446
750 13% (13m 11s) 0.1049 0.1928 y speech distilled upon them. And they waited for  / english ✓
Train accuracy: 0.9432
800 14% (14m 3s) 0.1269 0.3408 sprit immonde. Ses freres et sa mere arriverent do / french ✓
Train accuracy: 0.9442
850 15% (14m 56s) 0.2349 0.2925 ed Johxanan, filo de Kareahx, kaj cxiuj militestro / esperanto ✓
Train accuracy: 0.9428
900 16% (15m 48s) 0.0731 0.2384 e seapte ori.`` Si Domnul a hotarit un semn pentru / romanian ✓
Train accuracy: 0.9526
950 17% (16m 41s) 0.0826 0.1383 kiujn Moseo faris antaux la okuloj de la tuta Izra / esperanto ✓
Train accuracy: 0.9532
1000 18% (17m 33s) 0.1767 0.2967 efter sitt billede; og han kalte ham Set. Og efter / danish ✗ (no
rwegian)
Train accuracy: 0.9492
1050 19% (18m 25s) 0.1200 0.2147 sta paha mellakka. Kaupungissa oli Demetrios- nimi / finnish ✓
Train accuracy: 0.9486
1100 20% (19m 18s) 0.1047 0.0988 ke strax till jorden?  Jag skulle da garna hava gi / swedish ✓
Train accuracy: 0.954
1150 20% (20m 11s) 0.1748 0.1308 uj homoj ektimis per granda timo, kaj diris al li: / esperanto ✓
Train accuracy: 0.95
1200 21% (21m 3s) 0.1912 0.0837 a, va raminea incremenit, si va zice: ,Pentruce a  / romanian ✓
Train accuracy: 0.9478
1250 22% (21m 56s) 0.1717 0.1110 i moabitai verks ir raudos. Jie verks ir liudes de / lithuanian ✓
Train accuracy: 0.9502
1300 23% (22m 48s) 0.0957 0.1678 vetett kegyetlenseg [buntetese,] es szalljon az o  / hungarian ✓
Train accuracy: 0.9536
1350 24% (23m 41s) 0.1628 0.2272  And Shimei the son of Gera fell down before the k / english ✓
Train accuracy: 0.9454
1400 25% (24m 33s) 0.0851 0.0659 ch hade med sig tio man, och de slogo ihjal Gedalj / swedish ✓
Train accuracy: 0.958
1450 26% (25m 26s) 0.0751 0.2687  Og veg sa vaeldige Konger; thi hans Miskundhed va / danish ✓
Train accuracy: 0.9518
1500 27% (26m 19s) 0.1432 0.1111 onsulu, zydai visi kaip vienas sukilo pries Pauliu / lithuanian ✓
Train accuracy: 0.9522
1550 28% (27m 11s) 0.1171 0.1699 oudatti varoitusta, ja myos sina pelastat elamasi. / finnish ✓
Train accuracy: 0.9498
1600 29% (28m 3s) 0.1457 0.1844 anoah: "Anche se tu mi trattenessi, non mangerei d / italian ✓
Train accuracy: 0.9486
1650 30% (28m 55s) 0.1403 0.1424 !" Mutta mina, Herra, kayn oikeutta sinua vastaan, / finnish ✓
Train accuracy: 0.952
```

```
1700 30% (29m 48s) 0.1742 0.0842 ini bile anmiyordun. Simdi sen de Edom kizlariyla  / turkish ✓
Train accuracy: 0.9576
1750 31% (30m 41s) 0.0938 0.2853 o Viespaties namus, karaliaus namus ir visus didel / lithuanian ✓
Train accuracy: 0.9504
1800 32% (31m 33s) 0.1912 0.1547 kaara ki runga ki oku turi, ki runga hoki i nga ka / maori ✓
Train accuracy: 0.9554
1850 33% (32m 25s) 0.1754 0.0630 rete lo stesso giorno col suo parto. Quando offrir / italian ✓
Train accuracy: 0.9562
1900 34% (33m 18s) 0.0850 0.1650  szive, es fuleikkel nehezen hallanak, es szemeike / hungarian ✓
Train accuracy: 0.9608
1950 35% (34m 10s) 0.1076 0.1300 mo dia el se lo declaro, porque le constrino; y el / spanish ✓
Train accuracy: 0.9576
2000 36% (35m 2s) 0.1791 0.2052 ye umfazi. Ngoko eluvukweni, uya kuba ngumfazi waw / xhosa ✓
Train accuracy: 0.9564
2050 37% (35m 54s) 0.1443 0.1507 tada jis apiples jo namus. Kas ne su manimi, tas p / lithuanian ✓
Train accuracy: 0.9536
2100 38% (36m 47s) 0.0752 0.1389 zda ctyri kridla. Nohy mely rovne, ale chodidla by / czech ✓
Train accuracy: 0.9604
2150 39% (37m 40s) 0.1425 0.1444 en estambre, o en trama, o en cualquiera obra de p / portuguese ✗
(spanish)
Train accuracy: 0.9538
2200 40% (38m 33s) 0.1313 0.1551 ati un sanctuaire pour ton nom, en disant: S'il no / french ✓
Train accuracy: 0.9616
2250 40% (39m 25s) 0.1414 0.1793 o bem, o Senhor, aos bons e aos que sao retos de c / portuguese ✓
Train accuracy: 0.9604
2300 41% (40m 17s) 0.1623 0.1829 egou nas nossas maos a tropa que vinha contra nos. / portuguese ✓
Train accuracy: 0.9534
2350 42% (41m 10s) 0.0516 0.1667 bet tutardi. Toplam 172 kisiydiler. Israillilerin  / turkish ✓
Train accuracy: 0.956
2400 43% (42m 2s) 0.1804 0.1466 vsemi pronarody, a prijdou s tim nejvzacnejsim, co / czech ✓
Train accuracy: 0.956
2450 44% (42m 55s) 0.1126 0.2124 nd sprach: Also hat mir der Herr getan in den Tage / german ✓
Train accuracy: 0.9578
2500 45% (43m 47s) 0.0481 0.0747 nga mare linga Pi-Hahirot, fata in fata cu Baal-Te / romanian ✓
Train accuracy: 0.9524
2550 46% (44m 39s) 0.1050 0.1703 i geri donmez,<br />Yasam yollarina erismez. Bu ne / turkish ✓
Train accuracy: 0.9608
2600 47% (45m 32s) 0.0718 0.0293 oti e goditi Nabalin dhe ai vdiq. Kur mesoi qe Nab / albanian ✓
Train accuracy: 0.9622
2650 48% (46m 24s) 0.1362 0.1012 hova ngokubhekisele kukumkani waseAsiriya, ukuthi, / xhosa ✓
Train accuracy: 0.962
2700 49% (47m 17s) 0.1063 0.0843 c pour lui ces terres-la, toi et tes fils et tes s / french ✓
Train accuracy: 0.9574
2750 50% (48m 9s) 0.2365 0.1258 ene du tar for folk. Men Ga'al sa atter: Jo, det k / norwegian ✓
Train accuracy: 0.9594
2800 50% (49m 2s) 0.1558 0.1031 r seinen Bund aufrechterhalte, den er deinen Vater / german ✓
Train accuracy: 0.9626
2850 51% (49m 54s) 0.0712 0.1234 ttigini krala anlatirken, oglu diriltilen kadin ev / turkish ✓
Train accuracy: 0.9626
2900 52% (50m 46s) 0.0775 0.1486 Leben lang in Zelten wohnen, auf dass ihr lange le / german ✓
Train accuracy: 0.9614
2950 53% (51m 38s) 0.0445 0.1275 ir mano tevo namus". Ta diena Gadas atejo pas Dovy / lithuanian ✓
Train accuracy: 0.9586
3000 54% (52m 30s) 0.0299 0.1400 herche la loi; car il est le messager de l'Eternel / french ✓
Train accuracy: 0.9632
3050 55% (53m 23s) 0.1137 0.1240 escendio a lavarse al rio, y paseandose sus doncel / spanish ✓
Train accuracy: 0.9666
3100 56% (54m 15s) 0.0755 0.0857 uoi cho ta mot mieng banh nua. Nang dap: Toi chi m / vietnamese ✓
Train accuracy: 0.9554
3150 57% (55m 8s) 0.0952 0.1319 sman med femtio man.  Och nar denne kom upp till h / swedish ✓
Train accuracy: 0.9602
3200 58% (56m 1s) 0.0480 0.2877 n Israel sprach: Einen Propheten wird euch der Her / german ✓
Train accuracy: 0.9614
3250 59% (56m 53s) 0.1233 0.2315 th-Moab fiai, Jesua es Joab fiaitol: ketezernyolcz / hungarian ✓
Train accuracy: 0.9632
3300 60% (57m 46s) 0.0971 0.0544 ere la Eternulo savis HXizkijan kaj la logxantojn  / esperanto ✓
Train accuracy: 0.9602
3350 60% (58m 38s) 0.1518 0.2424 a zonke; vulani amaqonga alo;  lifumbeni ngokwezid / xhosa ✓
Train accuracy: 0.957
3400 61% (59m 31s) 0.0614 0.0526 Nop bi lua doi; nhung nguoi lam hon da goc cua cac / vietnamese ✓
Train accuracy: 0.9664
3450 62% (60m 23s) 0.0376 0.1434 yo ngemizi, Ndiya kusukela phezulu kubo, utsho uYe / xhosa ✓
Train accuracy: 0.9734
3500 63% (61m 16s) 0.0914 0.0795  ke mi havis la forton, por oferi tiom? de Vi esta / esperanto ✓
Train accuracy: 0.9592
3550 64% (62m 9s) 0.0873 0.1095 fa soning for det blod som utoses der, uten ved de / norwegian ✓
Train accuracy: 0.9672
```

```
3600 65% (63m 1s) 0.1153 0.1548 rde og sagde, at Ovnen skulde gores syv Gange hede / danish ✓
Train accuracy: 0.967
3650 66% (63m 54s) 0.0631 0.0942 oveho dreva, aby se daly postavit. Kazda deska byl / czech ✓
Train accuracy: 0.9668
3700 67% (64m 46s) 0.1652 0.1830 ur, et y etant entre, il s'assit avec les valets p / french ✓
Train accuracy: 0.9618
3750 68% (65m 39s) 0.1315 0.1421  at svardet, sager HERREN. Sa sager HERREN Sebaot: / swedish ✓
Train accuracy: 0.9654
3800 69% (66m 31s) 0.0998 0.1144 os mundur te gjeje ne copat e tij qofte edhe nje c / albanian ✓
Train accuracy: 0.964
3850 70% (67m 24s) 0.0547 0.2267 sios dienos. Skrynioje buvo tik dvi akmenines plok / lithuanian ✓
Train accuracy: 0.972
3900 70% (68m 17s) 0.1375 0.0570 ofet, sa vi kunde sporre Herren til rads gjennem h / norwegian ✓
Train accuracy: 0.9654
3950 71% (69m 9s) 0.1997 0.0776 apaea a ratou whakahere ki a Ihowa, i te koraha o  / maori ✓
Train accuracy: 0.954
4000 72% (70m 2s) 0.1135 0.0817  de kom til Bileam, overbragte de ham Balaks Ord.  / danish ✓
Train accuracy: 0.9656
4050 73% (70m 55s) 0.0813 0.1148 helyt ez halla, felkele hamar es hozza mene. Jezus / hungarian ✓
Train accuracy: 0.964
4100 74% (71m 48s) 0.1400 0.0876 i vart sarskilt hovdingdome ett pabud, oppet for a / norwegian ✗
(swedish)
Train accuracy: 0.9672
4150 75% (72m 41s) 0.1655 0.0785 imali ezisiweyo endlwini kaYehova, abayilulanganis / xhosa ✓
Train accuracy: 0.9624
4200 76% (73m 33s) 0.1407 0.0850 siu tikrus namus, kaip pastaciau Dovydui, ir tau d / lithuanian ✓
Train accuracy: 0.9656
4250 77% (74m 26s) 0.2012 0.1528 ana, kihai hoki i nohinohi nga utu i tika mai i ta / maori ✓
Train accuracy: 0.9684
4300 78% (75m 19s) 0.0745 0.1264 ek? Ott a gonoszok megszunnek a fenyegetestol, es  / hungarian ✓
Train accuracy: 0.9688
4350 79% (76m 11s) 0.1141 0.1004 umkhiwane untshwenyile, nomrharnate,  kwanamasundu / xhosa ✓
Train accuracy: 0.9698
4400 80% (77m 4s) 0.0778 0.1737 ovi celkovy soucet lidu: Vseho Izraele bylo jeden  / czech ✓
Train accuracy: 0.969
4450 80% (77m 56s) 0.0347 0.0943 si kaikkea syotavaa ja talletti sen kaupunkeihin.  / finnish ✓
Train accuracy: 0.969
4500 81% (78m 49s) 0.0507 0.1436 e de son heritage, a Thimnath-Serach, qui est dans / french ✓
Train accuracy: 0.97
4550 82% (79m 42s) 0.0226 0.0986  trage de multimea pestilor. Atunci ucenicul, pe c / romanian ✓
Train accuracy: 0.9728
4600 83% (80m 34s) 0.0333 0.2147 unga e haere tahi nei matou, e hemo ana hoki ratou / maori ✓
Train accuracy: 0.9738
4650 84% (81m 26s) 0.0308 0.1208 leyen kisidir. Ogul babasinin sucundan sorumlu tut / turkish ✓
Train accuracy: 0.967
4700 85% (82m 19s) 0.2039 0.0594 helomiti ngunyana kaYosifiya, enamadoda alikhulu e / xhosa ✓
Train accuracy: 0.9678
4750 86% (83m 12s) 0.1735 0.1757 uien mi fortaleza? Haga conmigo paz, si, haga paz  / italian ✗ (s
panish)
Train accuracy: 0.9712
4800 87% (84m 4s) 0.0424 0.1541 t us shut the doors of the temple: for they will c / english ✓
Train accuracy: 0.9692
4850 88% (84m 57s) 0.0651 0.1889 och fortara dess tornen och dess tistlar, allt pa  / swedish ✓
Train accuracy: 0.971
4900 89% (85m 50s) 0.0597 0.1148  pecheurs d'hommes. Et eux, laissant aussitot leur / french ✓
Train accuracy: 0.9718
4950 90% (86m 42s) 0.0821 0.1839 ta ngoi tren ngai Duc Chua Troi, o giua cac bien;  / vietnamese ✓
Train accuracy: 0.9716
5000 90% (87m 35s) 0.1257 0.0781 at Moabiterne og ammoniterne sammen med Folk fra M / norwegian ✗
(danish)
Train accuracy: 0.9694
5050 91% (88m 28s) 0.1167 0.0481 nku onlar kilictan, yalin kilictan,<br />Gerilmis  / turkish ✓
Train accuracy: 0.9716
5100 92% (89m 21s) 0.0481 0.0963 adan cikin,<br />Murdara dokunmayin.<br />Oradan c / turkish ✓
Train accuracy: 0.974
5150 93% (90m 13s) 0.0958 0.1275 jt si te ishin nje send shume i vogel. Libani nuk  / albanian ✓
Train accuracy: 0.9702
5200 94% (91m 6s) 0.0854 0.0568 do cung duoc xem cong viec thay lam. Khi nao nguoi / vietnamese ✓
Train accuracy: 0.9712
5250 95% (91m 59s) 0.0680 0.0395 ecause of his uncleanness: and afterward he shall  / english ✓
Train accuracy: 0.9734
5300 96% (92m 52s) 0.1201 0.1584 aye iyimingxuma yodwa yebhitumene;  basaba ookumka / xhosa ✓
Train accuracy: 0.975
5350 97% (93m 45s) 0.0816 0.2950 ci uslyseli, ze synove presidlencu buduji chram Ho / czech ✓
Train accuracy: 0.9666
5400 98% (94m 37s) 0.0536 0.1130 uch that the dumb man spake and saw. And all the m / english ✓
Train accuracy: 0.9748
```

```
5450 99% (95m 30s) 0.0601 0.0784 nde. De gudlose lurer pa at laegge mig ode, dine V / danish ✓
Train accuracy: 0.9656
5500 100% (96m 22s) 0.1074 0.1121 la pastro lavu siajn vestojn kaj banu sian korpon  / esperanto ✓
Train accuracy: 0.972
```

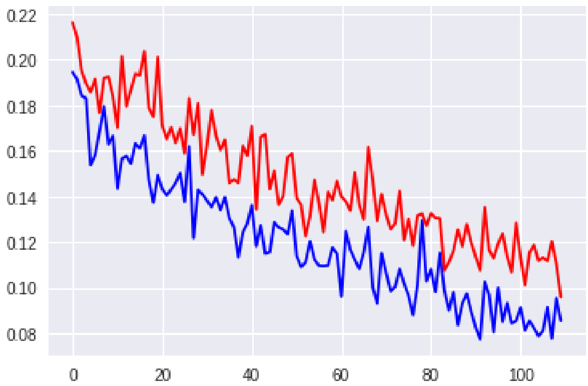
Plot loss functions

In [20]:

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

plt.figure()
plt.plot(all_losses, color='b')
plt.plot(all_test_losses, color='r')
```

Out[20]:

```
[<matplotlib.lines.Line2D at 0x7fa986443c50>]
```



Evaluate results

We now vizualize the performance of our model by creating a confusion matrix. The ground truth languages of samples are represented by rows in the matrix while the predicted languages are represented by columns.

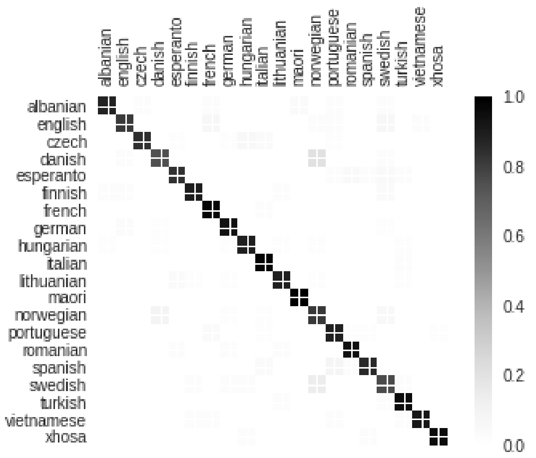
In this evaluation we consider sequences of variable sizes rather than the fixed length sequences we used for training.

In [0]:

```
eval_batch_size = 1  # needs to be set to 1 for evaluating different sequence lengths

# Keep track of correct guesses in a confusion matrix
confusion = torch.zeros(n_languages, n_languages)
n_confusion = 1000
num_correct = 0
total = 0

for i in range(n_confusion):
    eval_chunk_len = random.randint(10, 50) # in evaluation we will look at sequences of variable s
izes
    input_data, target_category, text_data = load_random_batch(test_category_data, chunk_len=eval_c
hunk_len, batch_size=eval_batch_size)
    output = evaluate(rnn, input_data, seq_len=eval_chunk_len, batch_size=eval_batch_size)

    guess_i = categoryFromOutput(output)
    category_i = [int(target_category[idx]) for idx in range(len(target_category))]
    for j in range(eval_batch_size):
        category = all_categories[category_i[j]]
        confusion[category_i[j]][guess_i[j]] += 1
        num_correct += int(guess_i[j]==category_i[j])
        total += 1

print('Test accuracy: ', float(num_correct)/float(n_confusion*eval_batch_size))

# Normalize by dividing every row by its sum
for i in range(n_languages):
```

```
    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))

plt.show()
```

```
Test accuracy:  0.892
```



You can pick out bright spots off the main axis that show which languages it guesses incorrectly.

Run on User Input

Now you can test your model on your own input.

In [0]:

```
def predict(input_line, n_predictions=5):
    print('\n> %s' % input_line)
    with torch.no_grad():
        input_data = stringToTensor(input_line).long().unsqueeze(0).to(device)
        output = evaluate(rnn, input_data, seq_len=len(input_line), batch_size=1)

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

    for i in range(n_predictions):
        topv.shape
        topi.shape
        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('Ich sehe ihn nicht.')
```

```
> Ich sehe ihn nicht.
(10.82) german
(4.61) french
(2.91) english
(1.05) czech
(0.75) danish
```

Output Kaggle submission file

Once you have found a good set of hyperparameters submit the output of your model on the Kaggle test file.

In [0]:

```
### DO NOT CHANGE KAGGLE SUBMISSION CODE ####
import csv

kaggle_test_file_path = 'language_data/kaggle_rnn_language_classification_test.txt'
with open(kaggle_test_file_path, 'r') as f:
    lines = f.readlines()

output_rows = []
for i, line in enumerate(lines):
    sample = line.rstrip()
    sample_chunk_len = len(sample)
    input_data = stringToTensor(sample).unsqueeze(0)
    output = evaluate(rnn, input_data, seq_len=sample_chunk_len, batch_size=1)
    guess_i = categoryFromOutput(output)
    output_rows.append((str(i+1), all_categories[guess_i]))

submission_file_path = 'kaggle_rnn_submission.txt'
with open(submission_file_path, 'w') as f:
    output_rows = [('id', 'category')] + output_rows
    writer = csv.writer(f)
    writer.writerows(output_rows)
```

In [0]:

```
torch.save(rnn.state_dict(), './rnn_generator.pth')
```

In [0]:

```
rnn.load_state_dict(torch.load('./rnn_generator.pth'))
```