In [1]:

```
from google.colab import drive
drive.mount('/content/gdrive')
import os
os.chdir("gdrive/My Drive/mp4")
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call
drive.mount("/content/gdrive", force_remount=True).
```

In [0]:

```
#!pip install torch
#!pip install unidecode
```

In [0]:

```
#!python download_language_data.py
```

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 [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 [6]:

```
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 [8]:

```
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

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 = 2000
hidden_size = 150
n_layers = 2
learning_rate = 0.001
model_type = 'gru'

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_tensor: 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

    """

    ###################################
    #          YOUR CODE HERE         #
    ###################################
    rnn.zero_grad()
    loss = 0
    hidden = rnn.init_hidden(batch_size, device=device)

    for c in range(chunk_len):
        output, hidden = rnn(data_tensor[:,c], hidden)
    loss = criterion(output, target_tensor.squeeze())

    loss.backward()
    optimizer.step()


    ##########       END      ##########
```

```
    return output, loss.item()
```

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()
```

In [20]:

```
n_iters = 5000 #2000 #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):
    if iter == 2000:
        optimizer = torch.optim.Adam(rnn.parameters(), lr=learning_rate/10)
    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

    _, test_loss = eval_test(rnn, target_category, input_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

    # 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 1% (0m 13s) 2.0228 1.9427 n hadde? Gi oss eiendom blandt var fars brodre! Og / danish ✗ (norweg
```

ian)
Train accuracy: 0.2012
100 2% (0m 26s) 1.0023 0.9473 aomi ferje, es marada o es az o ket fia. A kik Moa / esperanto ✗ (hu
ngarian)
Train accuracy: 0.5494
150 3% (0m 40s) 0.8101 0.8161 dora e tij rri e shtrire. Ai do te ngrere nje flam / norwegian ✗ (al
banian)
Train accuracy: 0.694
200 4% (0m 53s) 0.6724 0.6228 avendo ella da mangiare, Gesu, chiamati a se i dis / romanian ✗ (ita
lian)
Train accuracy: 0.7636
250 5% (1m 7s) 0.5867 0.5006 ears to hear, let him hear.] And when he was ente / english ✓
Train accuracy: 0.815
300 6% (1m 20s) 0.4167 0.3944 min nod, och han svarade mig; fran dodsrikets buk / swedish ✓
Train accuracy: 0.8404
350 7% (1m 34s) 0.4248 0.3638 tren mat dat. Day la dong doi cua Sem: Cach hai na / vietnamese ✓
Train accuracy: 0.8644
400 8% (1m 47s) 0.2850 0.2610 su Juo, kaip Istatymas reikalauja, Simeonas paeme / lithuanian ✓
Train accuracy: 0.8754
450 9% (2m 1s) 0.4301 0.3709 ses durys buvo is dvieju daliu ir kitos puses dury / lithuanian ✓
Train accuracy: 0.8882
500 10% (2m 14s) 0.2447 0.2252 ta tyhman nielee hanen oma suunsa. Hanen sanojensa / finnish ✓
Train accuracy: 0.9102
550 11% (2m 27s) 0.2924 0.2532 pfer, Speisopfer und fur das Fett der Dankopfer. S / danish ✗ (germ
an)
Train accuracy: 0.9182
600 12% (2m 41s) 0.3452 0.3012 re sul capo e sulla barba; poi prenditi una bilanc / romanian ✗ (it
alian)
Train accuracy: 0.9304
650 13% (2m 54s) 0.2068 0.1638 ht, dass die Philister uber uns Herrschen? Warum h / german ✓
Train accuracy: 0.932
700 14% (3m 8s) 0.1484 0.1284 at eders Glaede ma blive fuldkommen. Dette har jeg / norwegian ✗ (da
nish)
Train accuracy: 0.9296
750 15% (3m 21s) 0.2021 0.1727 hans Arme; og Israel sagde til Josef: "Lad mig nu / danish ✓
Train accuracy: 0.9422
800 16% (3m 35s) 0.2004 0.1539 er Geringste zu einem starken Volk; ich, der HERR, / german ✓
Train accuracy: 0.9484
850 17% (3m 48s) 0.2269 0.1927 orer ej Fogedens Rost; sma og store er lige der og / danish ✓
Train accuracy: 0.948
900 18% (4m 2s) 0.1722 0.1401 Miaj sklavoj, kiujn Mi elkondukis el la lando Egip / esperanto ✓
Train accuracy: 0.9448
950 19% (4m 15s) 0.1139 0.0951 leurs meres: Ou est le froment et le vin? lorsqu'i / french ✓
Train accuracy: 0.9534
1000 20% (4m 29s) 0.1212 0.1011 cuida bem dos teus rebanhos; porque as riquezas na / portuguese ✓
Train accuracy: 0.9578
1050 21% (4m 42s) 0.0995 0.0762 oa noku, he hoariri ia ki ahau: ko te tangata kaho / maori ✓
Train accuracy: 0.9578
1100 22% (4m 56s) 0.0950 0.0779 erna? Ed egli gli disse: Nella legge che sta scrit / italian ✓
Train accuracy: 0.9586
1150 23% (5m 10s) 0.1529 0.1459 ellasi? Han on naet tanaan lahtenyt kaupungista ja / finnish ✓
Train accuracy: 0.9616
1200 24% (5m 23s) 0.1990 0.1823 a te mahi pakeke a tou papa, me tana ioka taimaha / maori ✓
Train accuracy: 0.9558
1250 25% (5m 37s) 0.0433 0.0312 ose vreshtaret dhe vreshtin do t'ua jape te tjerev / albanian ✓
Train accuracy: 0.9582
1300 26% (5m 50s) 0.0765 0.0632 bras de la al Entonces mando el rey al sumo sacerd / spanish ✓
Train accuracy: 0.961
1350 27% (6m 4s) 0.1121 0.0723 na kuni ukudinisa abantu, ukuda oku nidinise noTh / finnish ✗ (xho
sa)
Train accuracy: 0.9684
1400 28% (6m 17s) 0.1646 0.1248 ovstvi od domu Davidova a dal jsem je tobe. Ty vsa / czech ✓
Train accuracy: 0.961
1450 28% (6m 31s) 0.0508 0.0449 mnul cu arfa, laudati -L cu alauta cu zece coarde. / romanian ✓
Train accuracy: 0.963
1500 30% (6m 44s) 0.0935 0.0751 iscatto, fallo; ma se non lo vuoi far valere, dimm / italian ✓
Train accuracy: 0.9678
1550 31% (6m 58s) 0.0217 0.0168 e vyrok Hospodinuv. Rekabejci plni slova Jonadaba, / czech ✓
Train accuracy: 0.9696
1600 32% (7m 11s) 0.0675 0.0517 va sinua, ettet saa puhua minulle muuta kuin totuu / finnish ✓
Train accuracy: 0.968
1650 33% (7m 25s) 0.0807 0.0622 and largeness of heart, even as the sand that is o / english ✓
Train accuracy: 0.9738
1700 34% (7m 38s) 0.1167 0.0590 intelegem; nu ne vorbi in limba iudaica, in auzul / romanian ✓
Train accuracy: 0.9694
1750 35% (7m 51s) 0.1523 0.0890 tai bus sugriauti. Etiopija, Libija, Lidija ir kit / lithuanian ✓
Train accuracy: 0.9752

Train accuracy: 0.9722
1800 36% (8m 5s) 0.0435 0.0354 uret karjalaumat, hopeaa, kultaa, pronssia ja raut / finnish ✓
Train accuracy: 0.9722
1850 37% (8m 19s) 0.0929 0.0585 v Tachpanchesu! Povezte: Postav se, priprav se, me / czech ✓
Train accuracy: 0.9716
1900 38% (8m 32s) 0.0807 0.0689 which Solomon had made. And king Rehoboam made in / english ✓
Train accuracy: 0.9726
1950 39% (8m 45s) 0.0544 0.0308 Ba biet rang nuoc von thuoc ve toi, va ca Y-so-ra- / vietnamese ✓
Train accuracy: 0.9782
2000 40% (8m 59s) 0.0282 0.0201 re? Da, se va usca in straturile unde a odraslit.` / romanian ✓
Train accuracy: 0.9786
2050 41% (9m 12s) 0.0630 0.0599 jo; sumanymai ir mano sirdies siekiai suduzo. Jie / lithuanian ✓
Train accuracy: 0.9768
2100 42% (9m 26s) 0.0701 0.0686 kaj ili sekvos Miajn decidojn, kaj Miajn legxojn / esperanto ✓
Train accuracy: 0.979
2150 43% (9m 39s) 0.0543 0.0527 skal han rense sig med vannet,* sa blir han ren; / norwegian ✓
Train accuracy: 0.9822
2200 44% (9m 53s) 0.0318 0.0311 Datter af Zibons Son Ana; hun fodte for Esau: Jeus / danish ✓
Train accuracy: 0.982
2250 45% (10m 6s) 0.0084 0.0076 perpara Aaronit dhe para bijve te tij dhe do t'i p / albanian ✓
Train accuracy: 0.984
2300 46% (10m 20s) 0.1104 0.1063 rne; hver mann hadde sitt vaben hos sig og vann. M / norwegian ✓
Train accuracy: 0.9816
2350 47% (10m 33s) 0.0560 0.0547 us vainikas savo tautos likuciui; teisingumo dvasi / lithuanian ✓
Train accuracy: 0.9814
2400 48% (10m 47s) 0.0169 0.0166 Haus zuruck. Und das Weib ward schwanger und sand / german ✓
Train accuracy: 0.985
2450 49% (11m 0s) 0.0151 0.0149 o direktu vin kaj venu tien, kaj tien alportu viaj / esperanto ✓
Train accuracy: 0.98
2500 50% (11m 14s) 0.0319 0.0306 brast i Grad; og han fortalte hende; at han var he / danish ✓
Train accuracy: 0.9826
2550 51% (11m 27s) 0.0991 0.0886 ik az en nepemnek husat, es lenyuzzak roluk boruke / hungarian ✓
Train accuracy: 0.9832
2600 52% (11m 41s) 0.0335 0.0325 heben! Wir sind abtrunnig und widerspenstig gewese / german ✓
Train accuracy: 0.9864
2650 53% (11m 54s) 0.0531 0.0494 gi tetahi tangata, he Hurai, ko tona ingoa ko Moro / maori ✓
Train accuracy: 0.9844
2700 54% (12m 7s) 0.0323 0.0316 k, mint az oreg emberek, mert vigyazok a te hataro / hungarian ✓
Train accuracy: 0.9834
2750 55% (12m 21s) 0.0682 0.0661 Eluvukweni ngoko, ungumfazi wawuphi na kubo? Kuba / xhosa ✓
Train accuracy: 0.9822
2800 56% (12m 34s) 0.0435 0.0392 s. Havendo eles estado muito tempo sem comer, Paul / portuguese ✓
Train accuracy: 0.9864
2850 56% (12m 48s) 0.0219 0.0209 Han gjorde icke vad ratt var i HERRENS ogon, sasom / swedish ✓
Train accuracy: 0.9854
2900 57% (13m 1s) 0.0302 0.0257 hode, mens han satt til bords. Men da disiplene s / norwegian ✓
Train accuracy: 0.985
2950 59% (13m 14s) 0.0905 0.0863 eitais, e nao vos julgais dignos da vida eterna, e / portuguese ✓
Train accuracy: 0.9848
3000 60% (13m 28s) 0.0170 0.0151 eo min sendis; kiom da forto mi tiam havis, tiom m / swedish ✗ (e
speranto)
Train accuracy: 0.9846
3050 61% (13m 41s) 0.0609 0.0585 Jerusalem. Han gjorde, hvad der var ret i HERRENs / danish ✓
Train accuracy: 0.985
3100 62% (13m 55s) 0.0105 0.0103 aaseen. Anna kansallesi Israelille anteeksi sen sy / finnish ✓
Train accuracy: 0.9868
3150 63% (14m 8s) 0.1126 0.1018 n he fought against Hazael king of Syria. And Azar / english ✓
Train accuracy: 0.9842
3200 64% (14m 22s) 0.0600 0.0575 mpre. El dia que estando tu delante, llevaban extr / spanish ✓
Train accuracy: 0.9836
3250 65% (14m 35s) 0.0303 0.0289 ovan nasemu Bohu, stane se v Judsku pohlavarem a E / czech ✓
Train accuracy: 0.987
3300 66% (14m 48s) 0.0246 0.0233 i spune seceratorilor: ,Smulgeti intii neghina, si / romanian ✓
Train accuracy: 0.9854
3350 67% (15m 2s) 0.0159 0.0156 Midian papja, Mozes ipa, mindazt a mit Isten Mozes / hungarian ✓
Train accuracy: 0.9858
3400 68% (15m 15s) 0.0701 0.0654 det: "Aldrig i Evighed skal nogen mere spise Frug / danish ✓
Train accuracy: 0.9874
3450 69% (15m 29s) 0.0204 0.0196 i ne versxu sur gxin pluvon. La vinbergxardeno de / esperanto ✓
Train accuracy: 0.9876
3500 70% (15m 43s) 0.0178 0.0177 ael. Uzavrou s tebou smlouvu a ty budes nade vsim / czech ✓
Train accuracy: 0.9848
3550 71% (15m 56s) 0.0323 0.0307 dayanmak zorunda kalacaksiniz.>> Vay halime benim! / turkish ✓
Train accuracy: 0.9844
3600 72% (16m 10s) 0.0431 0.0416 lysel Zebul, velitel mesta, rec Gaala, syna Ebedov / czech ✓
Train accuracy: 0.9882
3650 73% (16m 23s) 0.0733 0.0628 projevis jako sluzebnik a poslouzis jim, jestlize / czech ✓
Train accuracy: 0.9858

```
3700 74% (16m 37s) 0.0854 0.0778  hanen voimansa on suuri, ei han jata rankaisematt / finnish ✓
Train accuracy: 0.9836
3750 75% (16m 50s) 0.0902 0.0881 de ce qu'il lui avait dit pour la troisieme fois:  / french ✓
Train accuracy: 0.9856
3800 76% (17m 3s) 0.0700 0.0690 e, pjelljet e lopeve te tua dhe fryti i deleve te  / albanian ✓
Train accuracy: 0.9844
3850 77% (17m 17s) 0.0243 0.0231 athu. Wathi, akuba engasenako ukumfihla, wamthabat / xhosa ✓
Train accuracy: 0.9872
3900 78% (17m 30s) 0.0366 0.0341 tretti. Pallus sonn var Eliab, og Eliabs barn var  / norwegian ✓
Train accuracy: 0.9862
3950 79% (17m 44s) 0.0590 0.0569 kal prise dig, HERRE, nar de horer din Munds Ord,  / danish ✓
Train accuracy: 0.9858
4000 80% (17m 57s) 0.0119 0.0119 rete sur Damas; - car l'Eternel a l'oeil sur les h / french ✓
Train accuracy: 0.9862
4050 81% (18m 10s) 0.1038 0.1029 Raara papa o Mareha, me nga hapu o te whare o nga  / maori ✓
Train accuracy: 0.9882
4100 82% (18m 24s) 0.0112 0.0114  njezet e pese kubite gjeresi. Dritaret e saj, har / albanian ✓
Train accuracy: 0.9856
4150 83% (18m 37s) 0.0289 0.0276 i kova. Benjaminai nezinojo, kad ju laukia nelaime / lithuanian ✓
Train accuracy: 0.9866
4200 84% (18m 51s) 0.0434 0.0369 Love, det er ude med dig! Du var som en Drage i Ha / danish ✓
Train accuracy: 0.9858
4250 85% (19m 4s) 0.0087 0.0084 es. Men alle rene Fugle ma I spise. I ma ikke spis / danish ✓
Train accuracy: 0.9856
4300 86% (19m 17s) 0.0095 0.0091 abatheni nina, nimgwebe ngesiko lenu. Athi ngoko a / xhosa ✓
Train accuracy: 0.99
4350 87% (19m 31s) 0.0090 0.0090 eance me ta ne Hebron para Zotit, dhe ata e vajose / albanian ✓
Train accuracy: 0.9886
4400 88% (19m 44s) 0.0101 0.0099 v, k vyucovani, kdyz valcil s Aramejci z Dvojrici  / czech ✓
Train accuracy: 0.9862
4450 89% (19m 58s) 0.0063 0.0062 i -o voi arata.`` El a iesit atunci din tara Halde / romanian ✓
Train accuracy: 0.9886
4500 90% (20m 11s) 0.0364 0.0245 iuoli di Beniamino: Bela, Beker e Jediael: tre in  / italian ✓
Train accuracy: 0.99
4550 91% (20m 24s) 0.0643 0.0569 e kurores se kryelartesise te te dehurve, te Efrai / albanian ✓
Train accuracy: 0.9852
4600 92% (20m 38s) 0.0425 0.0417 jai atmete, tapo kertiniu akmeniu. Tai Viespats pa / lithuanian ✓
Train accuracy: 0.9828
4650 93% (20m 51s) 0.0298 0.0272 a ka haere atu ia i reira, ka tomo ki to ratou wha / maori ✓
Train accuracy: 0.9876
4700 94% (21m 4s) 0.0242 0.0195 raendt, sa ingen bor der. Selv Nofs og Takpankes's / danish ✓
Train accuracy: 0.9876
4750 95% (21m 18s) 0.0056 0.0056 ro: Certamente tu es um deles; pois es tambem gali / portuguese ✓
Train accuracy: 0.9866
4800 96% (21m 31s) 0.0823 0.0748 lu zezindlu zooyise zakoonyana bakaSirayeli, eYeru / xhosa ✓
Train accuracy: 0.9866
4850 97% (21m 44s) 0.0373 0.0329 ficistoj laborgvidantoj en la domo de la Eternulo; / esperanto ✓
Train accuracy: 0.9882
4900 98% (21m 58s) 0.0443 0.0412 asarit care are o intindere de patru mii cinci sut / romanian ✓
Train accuracy: 0.9878
4950 99% (22m 11s) 0.0458 0.0444 mai i te hope o Hakopa, e whitu tekau wairua: i Ih / maori ✓
Train accuracy: 0.9886
5000 100% (22m 24s) 0.0238 0.0233 erbimi qe Zoti ka shqiptuar kunder ketij populli". / albanian ✓
Train accuracy: 0.9864
```

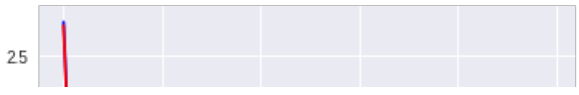
Plot loss functions

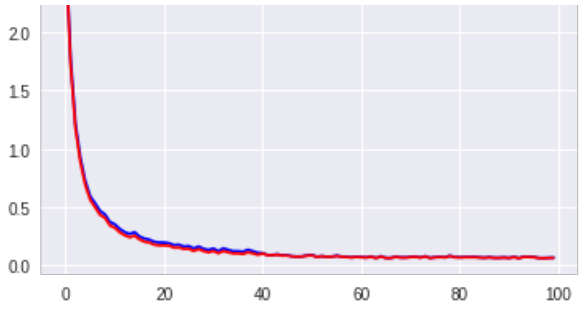
In [15]:

```
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[15]:

```
[<matplotlib.lines.Line2D at 0x7f229ea76048>]
```





Evaluate results

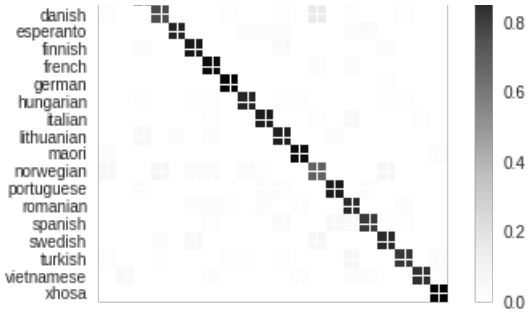
We now vizualize the performance of our model by creating a confusion matrix. The ground truth languages of samples are represented by rows in the matrix while the predicted languages are represented by columns.

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

In [21]:

```
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:  0.876
```



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 [17]:

```
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('This is a phrase to test the model on user input')
```

```
> This is a phrase to test the model on user input
(10.92) english
(3.46) albanian
(3.39) hungarian
(2.77) norwegian
(1.72) german
```

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:
```

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