HW5 - Samet Oymak

Introduction to Deep Learning (University of California Riverside)

In [2]:
```
!unzip /data.zip
```

```
Archive:  /data.zip
   creating: data/
  inflating: data/eng-fra.txt
   creating: data/names/
  inflating: data/names/Arabic.txt
  inflating: data/names/Chinese.txt
  inflating: data/names/Czech.txt
  inflating: data/names/Dutch.txt
  inflating: data/names/English.txt
  inflating: data/names/French.txt
  inflating: data/names/German.txt
  inflating: data/names/Greek.txt
  inflating: data/names/Irish.txt
  inflating: data/names/Italian.txt
  inflating: data/names/Japanese.txt
  inflating: data/names/Korean.txt
  inflating: data/names/Polish.txt
  inflating: data/names/Portuguese.txt
  inflating: data/names/Russian.txt
  inflating: data/names/Scottish.txt
  inflating: data/names/Spanish.txt
  inflating: data/names/Vietnamese.txt
```

In [3]:
```
from __future__ import unicode_literals, print_function, division
from io import open
import glob
import os

def findFiles(path): return glob.glob(path)

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

import unicodedata
import string

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

# Turn a Unicode string to plain ASCII, thanks to https://stackoverflow.com/a/518232/28
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'))

# 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)
```

```
['data/names/Japanese.txt', 'data/names/Korean.txt', 'data/names/Irish.txt', 'data/name
s/Spanish.txt', 'data/names/Scottish.txt', 'data/names/Dutch.txt', 'data/names/Chinese.t
xt', 'data/names/French.txt', 'data/names/English.txt', 'data/names/Greek.txt', 'data/na
mes/Polish.txt', 'data/names/Vietnamese.txt', 'data/names/Arabic.txt', 'data/names/Czec
h.txt', 'data/names/Portuguese.txt', 'data/names/Italian.txt', 'data/names/German.txt',
'data/names/Russian.txt']
Slusarski
```

In [4]:

```
print(category_lines['Italian'][:5])
```

```
['Abandonato', 'Abatangelo', 'Abatantuono', 'Abate', 'Abategiovanni']
```

In [5]:

```
import torch

# Find letter index from all_letters, e.g. "a" = 0
def letterToIndex(letter):
    return all_letters.find(letter)

# Just 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>,
# or 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., 1.,
         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])
```

In [6]:

```
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.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)
```

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

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

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```

In [7]:

```
input = letterToTensor('A')
hidden = torch.zeros(1, n_hidden)

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

In [8]:

```
input = lineToTensor('Albert')
hidden = torch.zeros(1, n_hidden)

output, next_hidden = rnn(input[0], hidden)
print(output)
```

```
tensor([[-2.8500, -2.8957, -2.8686, -2.9587, -2.9786, -2.8103, -2.9236, -2.8152,
         -2.8647, -2.9089, -2.8848, -2.9006, -2.8629, -2.8931, -2.9785, -2.9269,
         -2.7911, -2.9401]], grad_fn=<LogSoftmaxBackward>)
```

In [9]:

```
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))
```

```
('German', 16)
```

In [10]:

```
import random

def randomChoice(l):
    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=torch.long)
    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 = Arabic / line = Maloof
category = Spanish / line = Salcedo
category = Vietnamese / line = Pham
```

```
category = English / line = Orbell
category = Japanese / line = Hiyama
category = French / line = Fabian
category = Spanish / line = Mingo
category = Portuguese / line = Madeira
category = Polish / line = Zabek
category = Korean / line = Chi
```

In [11]:

```
criterion = nn.NLLLoss()
```

In [12]:

```
learning_rate = 0.005 # If you set this too high, it might explode. If too low, it migh

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_(p.grad.data, alpha=-learning_rate)

    return output, loss.item()
```

In [13]:

```
import time
import math

n_iters = 100000
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, timeSince(s

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

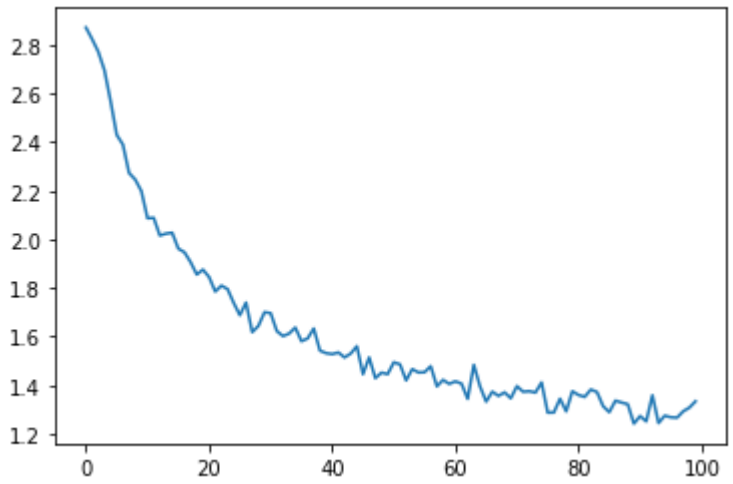
```
5000 5% (0m 5s) 2.9169 Bleskan / Irish ✗ (Czech)
10000 10% (0m 11s) 0.6307 Serjantov / Russian ✓
15000 15% (0m 16s) 0.4081 Rogashkov / Russian ✓
20000 20% (0m 22s) 1.4747 Danilyan / Russian ✓
25000 25% (0m 27s) 0.4260 Wojewodka / Polish ✓
30000 30% (0m 32s) 2.7829 Molcan / Irish ✗ (Czech)
35000 35% (0m 38s) 0.5677 Prosdocimi / Italian ✓
40000 40% (0m 43s) 1.3489 Fuhrmann / Dutch ✗ (German)
45000 45% (0m 49s) 0.8936 O'Kelly / Irish ✓
50000 50% (0m 54s) 1.0823 Pelaez / Spanish ✓
55000 55% (1m 0s) 0.2186 Tchekmenev / Russian ✓
60000 60% (1m 5s) 2.9117 Shaw / Chinese ✗ (Scottish)
65000 65% (1m 11s) 0.3800 Jeon / Korean ✓
70000 70% (1m 16s) 1.9341 Bowen / Dutch ✗ (English)
75000 75% (1m 22s) 0.1747 Leontarakis / Greek ✓
80000 80% (1m 27s) 2.4544 Coulson / Scottish ✗ (English)
85000 85% (1m 33s) 3.9484 Auttenberg / German ✗ (Polish)
90000 90% (1m 38s) 0.9130 Nurkaev / Dutch ✗ (Russian)
95000 95% (1m 44s) 1.4419 Smith / Scottish ✓
100000 100% (1m 49s) 0.0302 Niijima / Japanese ✓
```

In [14]:

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

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

Out[14]:

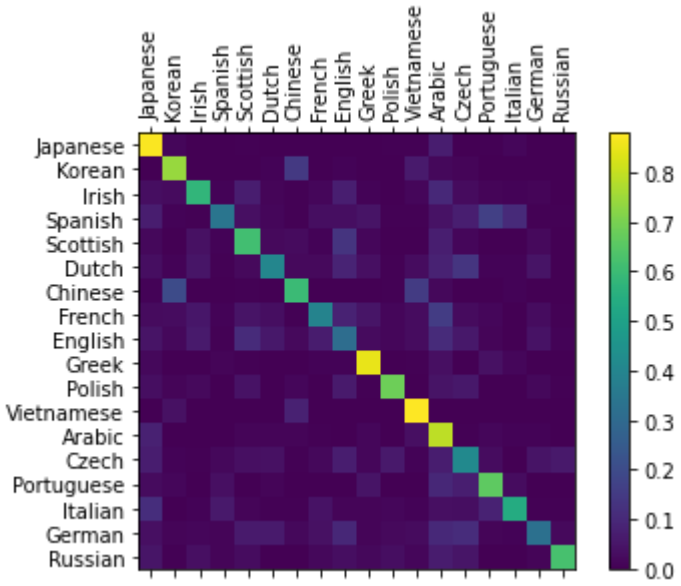
```
[<matplotlib.lines.Line2D at 0x7efda0ce80d0>]
```



In [15]:

```
# 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()
```



In [16]:

```
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')
```

```
> Dovesky
(-0.73) Czech
(-0.83) Russian
(-3.44) Polish

> Jackson
(-0.74) English
(-1.79) Scottish
(-2.10) Greek

> Satoshi
(-0.95) Japanese
(-1.16) Arabic
(-2.25) Polish
```