lab_1

April 2, 2021

1 RNN

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

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

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',
'data/names/Dutch.txt', 'data/names/English.txt', 'data/names/French.txt',
'data/names/German.txt', 'data/names/Greek.txt', 'data/names/Irish.txt',
'data/names/Italian.txt', 'data/names/Japanese.txt', 'data/names/Korean.txt',
'data/names/Polish.txt', 'data/names/Portuguese.txt', 'data/names/Russian.txt',
'data/names/Scottish.txt', 'data/names/Spanish.txt',
'data/names/Vietnamese.txt']
```

[2]:
```
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

2 Get text dataset

[3]:
```
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(n_categories)
```

```
18
```

3 One hot encoding

[4]:
```
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., 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])
```

4 RNN model

[5]:
```
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)
        self.tanh = nn.Tanh()

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        hidden = self.tanh(hidden)
        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)
```

[6]:
```
def categoryFromOutput(output):
    top_n, top_i = output.topk(1)
    category_i = top_i[0].item()
```

```
    return all_categories[category_i], category_i
```

5 Random training set

[7]:
```
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=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 = Japanese / line = Nakamoto
category = Italian / line = Traversi
category = Chinese / line = Zou
category = English / line = Rogerson
category = Czech / line = Jirovy
category = Scottish / line = Murphy
category = Scottish / line = Mcdonald
category = Vietnamese / line = Ho
category = Irish / line = Gorman
category = German / line = Kirchner
```

6 Training

[8]:
```
criterion = nn.NLLLoss()
learning_rate = 0.005 # If you set this too high, it might explode. If too low,␣
 ↪it might not learn

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()
```

[9]:
```
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(start), loss, line, guess, correct))

    # Add current loss avg to list of losses
    if iter % plot_every == 0:
```

```
        all_losses.append(current_loss / plot_every)
        current_loss = 0
```

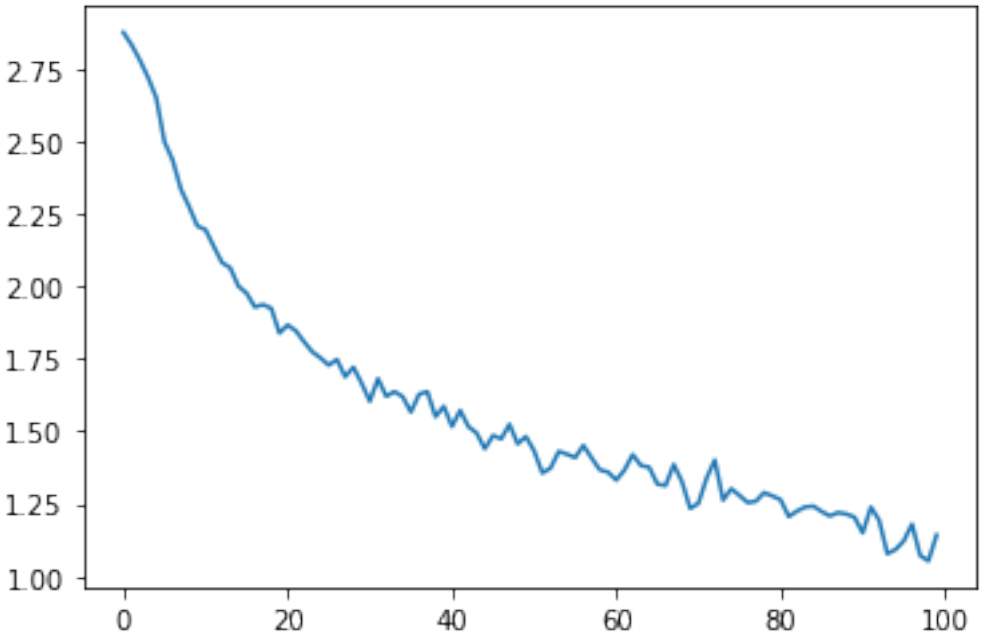
```
/usr/local/lib/python3.6/dist-packages/torch/autograd/__init__.py:147:
UserWarning: CUDA initialization: CUDA driver initialization failed, you might
not have a CUDA gpu. (Triggered internally at
/pytorch/c10/cuda/CUDAFunctions.cpp:109.)
  allow_unreachable=True, accumulate_grad=True)  # allow_unreachable flag
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:17: 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
/pytorch/torch/csrc/utils/python_arg_parser.cpp:1005.)
5000 5% (0m 5s) 2.7421 Nolan / Arabic  (Irish)
10000 10% (0m 11s) 1.8213 Martzenyuk / Polish  (Russian)
15000 15% (0m 16s) 2.7450 Scott / French  (Scottish)
20000 20% (0m 22s) 1.4909 Cardozo / Portuguese
25000 25% (0m 28s) 1.8020 Bianchi / Japanese  (Italian)
30000 30% (0m 34s) 1.3609 Schwarzenberg / Dutch
35000 35% (0m 39s) 1.1299 Ryoo / Korean
40000 40% (0m 45s) 2.1664 Kerr / German  (Scottish)
45000 45% (0m 50s) 0.7142 Makferson / Russian
50000 50% (0m 55s) 1.7720 Rutten / Scottish  (Dutch)
55000 55% (1m 1s) 0.3495 Byon / Korean
60000 60% (1m 6s) 2.3234 Baumbach / Vietnamese  (German)
65000 65% (1m 12s) 0.1906 Takara / Japanese
70000 70% (1m 18s) 0.9642 Zubizarreta / Spanish
75000 75% (1m 23s) 1.1456 Geroux / French
80000 80% (1m 28s) 2.4270 Delaney / Russian  (Irish)
85000 85% (1m 33s) 0.0174 Brisimitzakis / Greek
90000 90% (1m 39s) 4.0669 Fay / Chinese  (French)
95000 95% (1m 45s) 0.1495 Thao / Vietnamese
100000 100% (1m 50s) 2.9390 Chromy / Arabic  (Czech)
```

7 Loss

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

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

[10]:
```
[<matplotlib.lines.Line2D at 0x7f258f628208>]
```



8 Confusion matrix

[11]:
```
# 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
```

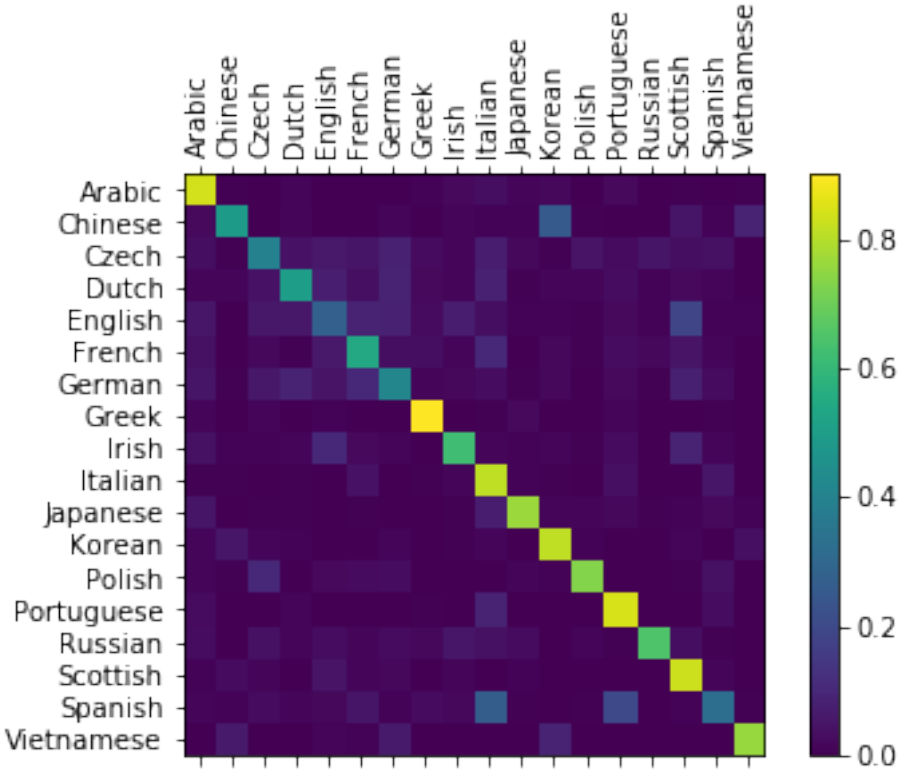
```
for i in range(n_categories):
    confusion[i] = confusion[i] / confusion[i].sum()

# 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()
```



9 Predict

[13]:
```
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('Higanbana')
predict('pongkorn')
```

```
> Dovesky
(-0.67) Czech
(-1.12) Russian
(-2.38) English

> Jackson
(-0.03) Scottish
(-3.89) English
(-6.52) Russian

> Higanbana
(-0.45) Japanese
(-2.57) German
(-2.87) Dutch

> pongkorn
(-0.36) Dutch
(-2.00) English
(-2.58) German
```