NLP From Scratch: Generating Names with a Character-Level RNN

Cidade por País

```
%matplotlib inline
```

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

```
Mounted at /content/drive
```

```
import csv
import unicodedata
import string

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

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

csv_file_path = '/content/drive/MyDrive/Colab Notebooks/Introdução à Aprendizagem Profunda/data/world
category_lines = {}

with open(csv_file_path, mode='r', encoding='utf-8') as csv_file:
    csv_reader = csv.reader(csv_file)
    next(csv_reader, None)

    all_categories = []
    for row in csv_reader:
        cidade = unicodeToAscii(row[0].strip())
        pais = unicodeToAscii(row[4].strip())

        if pais in category_lines:
            category_lines[pais].append(cidade)
        else:
            category_lines[pais] = [cidade]
            all_categories.append(pais)

n_categories = len(all_categories)
```

Creating the Network

```
import torch
import torch.nn as nn

class LSTMGenerator(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(LSTMGenerator, self).__init__()
        self.hidden_size = hidden_size
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)
        self.fc = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=2)

    def forward(self, input, hidden):
        output, hidden = self.lstm(input, hidden)
        output = self.fc(output)
        output = self.softmax(output)
        return output, hidden

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

+ Code + Text

Training

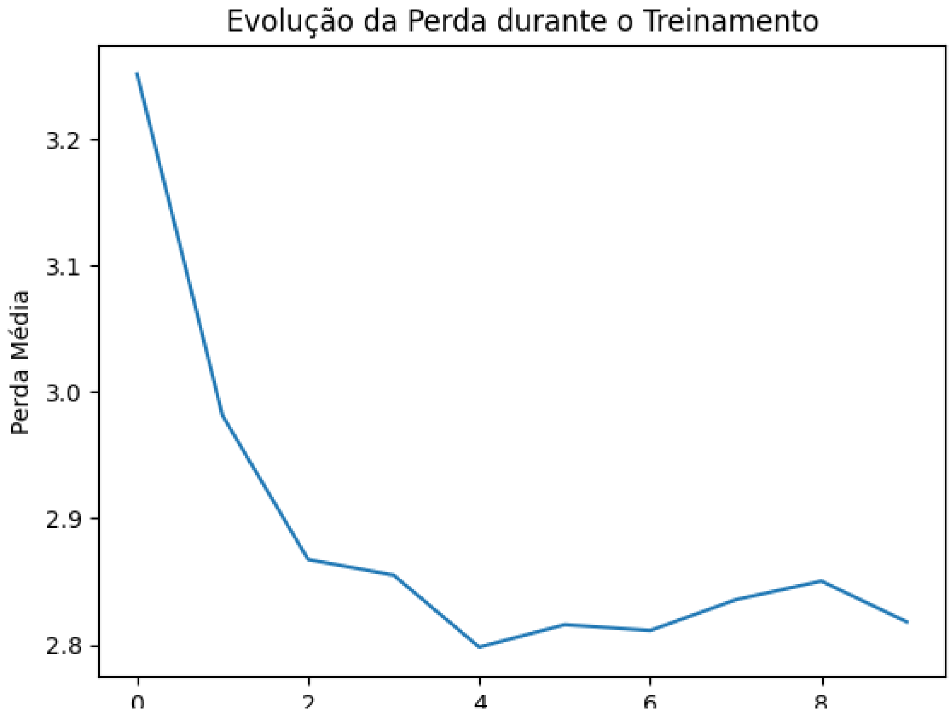
```
import random
import time
import math

def train(category_tensor, input_line_tensor, target_line_tensor):
    target_line_tensor.unsqueeze_(-1)
    hidden = rnn.initHidden(input_line_tensor.size(0))

    rnn.zero_grad()
    loss_function = nn.NLLLoss()
    output, hidden = rnn(input_line_tensor, hidden)
    loss = loss_function(output.view(-1, n_letters), target_line_tensor.view(-1))

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

    return output, loss.item()

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

```
from tqdm import tqdm

hidden_size = 256
learning_rate = 0.0005
n_iters = 10000
print_every = 5000
plot_every = 1000

rnn = LSTMGenerator(n_letters, hidden_size, n_letters)

all_losses = []
total_loss = 0
start = time.time()

for iter in tqdm(range(1, n_iters + 1)):
    output, loss = train(*randomTrainingExample())
    total_loss += loss

    if iter % print_every == 0:
        print('%s (%d %d%%) %.4f' % (timeSince(start), iter, iter / n_iters * 100, loss))

    if iter % plot_every == 0:
        all_losses.append(total_loss / plot_every)
        total_loss = 0
```

```
 50%|█████     | 5018/10000 [00:57<00:41, 118.70it/s]0m 56s (5000 50%) 2.8328
100%|██████████| 10000/10000 [01:44<00:00, 95.73it/s]1m 44s (10000 100%) 3.1851
```

Plotting the Losses

```
import matplotlib.pyplot as plt

plt.figure()
plt.plot(all_losses)
plt.xlabel('Época')
plt.ylabel('Perda Média')
plt.title('Evolução da Perda durante o Treinamento')
plt.show()
```



0s completed at 6:45 PM