2.2. (10 points) Modify the implementation from 2.1 to support arbitrary mini-batch sizes.

Submitted by:

Aradhya Mathur

Lakshmi Nikhil Goduguluri

In this case, instead of padding to a unique sequence length, adaptively pad the length of the mini batch to the length of the longest sample in the mini batch itself. Report the accuracy number (on the full training set) yielded by this approach on mini batch sizes of 1000, 2000, 3000 after five epochs of training.

Note that since these problems only ask you to train for five epochs it won't be graded based on performance (unless you get significantly smaller numbers than what's reasonable for five epochs of training).

In [1]:

```
from __future__ import unicode_literals, print_function, division
from io import open
import glob
import os
import numpy as np
import pandas as pd
import unicodedata
import string
import torch
import torch.nn as nn
import random
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker


# In[5]:
def findFiles(path):
    return glob.glob(path)

all_letters = string.ascii_letters + " .,;'"
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
    )


# In[6]:q


names = {}
languages = []


def readLines(filename):
    lines = open(filename, encoding='utf-8').read().strip().split('\n')
```

```
    return [unicodeToAscii(line) for line in lines]

# (TO DO:) CHANGE FILE PATH AS NECESSARY
for filename in findFiles(r"C:\Users\aradh\Desktop\Fall 22\TSA\Project 3.1\data\da
    category = os.path.splitext(os.path.basename(filename))[0]
    languages.append(category)
    lines = readLines(filename)
    names[category] = lines


# In[7]:

def letterToIndex(letter):
    return all_letters.find(letter)


def nameToTensor(name):
    tensor = torch.zeros(len(name), 1, n_letters)
    for li, letter in enumerate(name):
        tensor[li][0][letterToIndex(letter)] = 1
    return tensor


# In[54]:


class RNN(nn.Module):
    def __init__(self, INPUT_SIZE, HIDDEN_SIZE, N_LAYERS,OUTPUT_SIZE):
        super(RNN, self).__init__()
        self.rnn = nn.RNN(
            input_size = INPUT_SIZE,
            hidden_size = HIDDEN_SIZE, # number of hidden units
            num_layers = N_LAYERS, # number of layers
            batch_first = True)
        self.out = nn.Linear(HIDDEN_SIZE, OUTPUT_SIZE)

    def forward(self, x):
        r_out, h = self.rnn(x, None) # None represents zero initial hidden state
        out = self.out(r_out[:, -1, :])
        return out


# In[8]:

#list comprehension:
# list_data=[]
# for category in languages:
#     for name in names[category]:
#         list_data.append((name, category))

n_hidden = 128

allnames = [] # Create list of all names and corresponding output language
for language in list(names.keys()):
    for name in names[language]:
        allnames.append([name, language])

random.shuffle(allnames)
n = 1000
x = [allnames[i:i + n] for i in range(0, len(allnames), n)]
#print(x)

# for category in list_data:
#     for name in names[category]:
```

```
#         allnames.append([name, category])

## (TO DO:) Determine Padding length (this is the length of the longest string)

# maxlen = ..... # Add code here to compute the maximum length of string
n_letters = len(all_letters)
n_categories = len(languages)

def categoryFromOutput(output):
    top_n, top_i = output.topk(1)
    category_i = top_i.item()
    return languages[category_i], category_i



learning_rate = 0.005
rnn = RNN(n_letters, 128, 1, n_categories)
optimizer = torch.optim.Adam(rnn.parameters(), lr=learning_rate)   # optimize all
loss_func = nn.CrossEntropyLoss()
accuracies = []
for epoch in range(5):
    for batch_in_all_names in x:
        batch_size = len(batch_in_all_names)

        maxlen = max(len(x[0]) for x in batch_in_all_names)
        b_in = torch.zeros(batch_size, maxlen, n_letters)  # (TO DO:) Initialize "
        b_out = torch.zeros(batch_size, n_categories, dtype=torch.long)  # (TO DO:

        def get(charachter):
            return [z for z in charachter]
        for i in batch_in_all_names:
            j=batch_in_all_names.index(i)
            k=get(i[0])
            for l in range(len(i[0])):
                b_in[j][l][letterToIndex(k[l])]=1
            m=i[1]
            l=languages.index(m)
            b_out[j][l]=1
        max_b_out=torch.max(b_out,1)[1]
        output = rnn(b_in)                               # rnn output
        #(TO DO:)
        loss = loss_func(output, max_b_out)   # (TO DO:) Fill "...." to calculate
        optimizer.zero_grad()                           # clear gradients for this
        loss.backward()                                 # backpropagation, compute
        optimizer.step()                                # apply gradients

        # Print accuracy
        test_output = rnn(b_in)                   #
        pred_y = torch.max(test_output, 1)[1].data.numpy().squeeze()
        test_y = torch.max(b_out, 1)[1].data.numpy().squeeze()
        accuracy = sum(pred_y == test_y)/(batch_size)
        print("Epoch: ", epoch, "| train loss: %.4f" % loss.item(), '| accuracy: %
        accuracies.append(round(accuracy,3))

print("Average accuracy is", np.mean(accuracies))
```

```
Epoch:  0 | train loss: 2.9527 | accuracy: 0.484
Epoch:  0 | train loss: 2.7494 | accuracy: 0.492
Epoch:  0 | train loss: 2.1975 | accuracy: 0.453
Epoch:  0 | train loss: 1.9759 | accuracy: 0.460
Epoch:  0 | train loss: 1.9487 | accuracy: 0.462
Epoch:  0 | train loss: 1.9491 | accuracy: 0.468
Epoch:  0 | train loss: 1.8701 | accuracy: 0.464
Epoch:  0 | train loss: 1.9240 | accuracy: 0.443
Epoch:  0 | train loss: 1.8639 | accuracy: 0.474
Epoch:  0 | train loss: 1.9230 | accuracy: 0.462
Epoch:  0 | train loss: 1.8599 | accuracy: 0.480
Epoch:  0 | train loss: 1.9042 | accuracy: 0.461
Epoch:  0 | train loss: 1.7810 | accuracy: 0.499
Epoch:  0 | train loss: 1.8760 | accuracy: 0.459
Epoch:  0 | train loss: 1.8938 | accuracy: 0.450
Epoch:  0 | train loss: 1.8182 | accuracy: 0.487
Epoch:  0 | train loss: 1.8589 | accuracy: 0.460
Epoch:  0 | train loss: 1.8336 | accuracy: 0.471
Epoch:  0 | train loss: 1.9328 | accuracy: 0.442
Epoch:  0 | train loss: 1.8108 | accuracy: 0.485
Epoch:  0 | train loss: 1.8901 | accuracy: 0.527
Epoch:  1 | train loss: 1.7890 | accuracy: 0.484
Epoch:  1 | train loss: 1.8144 | accuracy: 0.492
Epoch:  1 | train loss: 1.9304 | accuracy: 0.453
Epoch:  1 | train loss: 1.9041 | accuracy: 0.460
Epoch:  1 | train loss: 1.8772 | accuracy: 0.462
Epoch:  1 | train loss: 1.8703 | accuracy: 0.468
Epoch:  1 | train loss: 1.8316 | accuracy: 0.464
Epoch:  1 | train loss: 1.9191 | accuracy: 0.443
Epoch:  1 | train loss: 1.8366 | accuracy: 0.474
Epoch:  1 | train loss: 1.8894 | accuracy: 0.462
Epoch:  1 | train loss: 1.8109 | accuracy: 0.480
Epoch:  1 | train loss: 1.8959 | accuracy: 0.461
Epoch:  1 | train loss: 1.7770 | accuracy: 0.499
Epoch:  1 | train loss: 1.8699 | accuracy: 0.459
Epoch:  1 | train loss: 1.8811 | accuracy: 0.450
Epoch:  1 | train loss: 1.8020 | accuracy: 0.487
Epoch:  1 | train loss: 1.8513 | accuracy: 0.460
Epoch:  1 | train loss: 1.8351 | accuracy: 0.471
Epoch:  1 | train loss: 1.9144 | accuracy: 0.442
Epoch:  1 | train loss: 1.8024 | accuracy: 0.485
Epoch:  1 | train loss: 1.8814 | accuracy: 0.527
Epoch:  2 | train loss: 1.7901 | accuracy: 0.484
Epoch:  2 | train loss: 1.8160 | accuracy: 0.492
Epoch:  2 | train loss: 1.9282 | accuracy: 0.453
Epoch:  2 | train loss: 1.9004 | accuracy: 0.460
Epoch:  2 | train loss: 1.8778 | accuracy: 0.462
Epoch:  2 | train loss: 1.8754 | accuracy: 0.468
Epoch:  2 | train loss: 1.8371 | accuracy: 0.464
Epoch:  2 | train loss: 1.9246 | accuracy: 0.443
Epoch:  2 | train loss: 1.8467 | accuracy: 0.474
Epoch:  2 | train loss: 1.8930 | accuracy: 0.462
Epoch:  2 | train loss: 1.8190 | accuracy: 0.480
Epoch:  2 | train loss: 1.8902 | accuracy: 0.461
Epoch:  2 | train loss: 1.7739 | accuracy: 0.499
Epoch:  2 | train loss: 1.8679 | accuracy: 0.459
Epoch:  2 | train loss: 1.8852 | accuracy: 0.450
Epoch:  2 | train loss: 1.8076 | accuracy: 0.487
Epoch:  2 | train loss: 1.8543 | accuracy: 0.460
Epoch:  2 | train loss: 1.8272 | accuracy: 0.471
Epoch:  2 | train loss: 1.9137 | accuracy: 0.442
Epoch:  2 | train loss: 1.8052 | accuracy: 0.485
Epoch:  2 | train loss: 1.8949 | accuracy: 0.527
Epoch:  3 | train loss: 1.8005 | accuracy: 0.484
```

```
Epoch:  3 | train loss: 1.8206 | accuracy: 0.492
Epoch:  3 | train loss: 1.9188 | accuracy: 0.453
Epoch:  3 | train loss: 1.8945 | accuracy: 0.460
Epoch:  3 | train loss: 1.8742 | accuracy: 0.462
Epoch:  3 | train loss: 1.8831 | accuracy: 0.468
Epoch:  3 | train loss: 1.8492 | accuracy: 0.464
Epoch:  3 | train loss: 1.9171 | accuracy: 0.443
Epoch:  3 | train loss: 1.8379 | accuracy: 0.474
Epoch:  3 | train loss: 1.8858 | accuracy: 0.462
Epoch:  3 | train loss: 1.8198 | accuracy: 0.480
Epoch:  3 | train loss: 1.8992 | accuracy: 0.461
Epoch:  3 | train loss: 1.7747 | accuracy: 0.499
Epoch:  3 | train loss: 1.8657 | accuracy: 0.459
Epoch:  3 | train loss: 1.8767 | accuracy: 0.450
Epoch:  3 | train loss: 1.8034 | accuracy: 0.487
Epoch:  3 | train loss: 1.8562 | accuracy: 0.460
Epoch:  3 | train loss: 1.8338 | accuracy: 0.471
Epoch:  3 | train loss: 1.9197 | accuracy: 0.442
Epoch:  3 | train loss: 1.8061 | accuracy: 0.485
Epoch:  3 | train loss: 1.8763 | accuracy: 0.527
Epoch:  4 | train loss: 1.7956 | accuracy: 0.484
Epoch:  4 | train loss: 1.8220 | accuracy: 0.492
Epoch:  4 | train loss: 1.9156 | accuracy: 0.453
Epoch:  4 | train loss: 1.8913 | accuracy: 0.460
Epoch:  4 | train loss: 1.8599 | accuracy: 0.462
Epoch:  4 | train loss: 1.8720 | accuracy: 0.468
Epoch:  4 | train loss: 1.8398 | accuracy: 0.464
Epoch:  4 | train loss: 1.9186 | accuracy: 0.443
Epoch:  4 | train loss: 1.8412 | accuracy: 0.474
Epoch:  4 | train loss: 1.8864 | accuracy: 0.462
Epoch:  4 | train loss: 1.8085 | accuracy: 0.480
Epoch:  4 | train loss: 1.8916 | accuracy: 0.461
Epoch:  4 | train loss: 1.7725 | accuracy: 0.499
Epoch:  4 | train loss: 1.8652 | accuracy: 0.459
Epoch:  4 | train loss: 1.8768 | accuracy: 0.450
Epoch:  4 | train loss: 1.7997 | accuracy: 0.487
Epoch:  4 | train loss: 1.8497 | accuracy: 0.460
Epoch:  4 | train loss: 1.8318 | accuracy: 0.471
Epoch:  4 | train loss: 1.9160 | accuracy: 0.442
Epoch:  4 | train loss: 1.8053 | accuracy: 0.485
Epoch:  4 | train loss: 1.8712 | accuracy: 0.527
Average accuracy is 0.47061904761904766
```

In [2]:

```
from __future__ import unicode_literals, print_function, division
from io import open
import glob
import os
import numpy as np
import pandas as pd
import unicodedata
import string
import torch
import torch.nn as nn
import random
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker


# In[5]:
def findFiles(path):
    return glob.glob(path)

all_letters = string.ascii_letters + " .,;'"
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
    )


# In[6]:q


names = {}
languages = []


def readLines(filename):
    lines = open(filename, encoding='utf-8').read().strip().split('\n')
    return [unicodeToAscii(line) for line in lines]

# (TO DO:) CHANGE FILE PATH AS NECESSARY
for filename in findFiles(r"C:\Users\aradh\Desktop\Fall 22\TSA\Project 3.1\data\da
    category = os.path.splitext(os.path.basename(filename))[0]
    languages.append(category)
    lines = readLines(filename)
    names[category] = lines


# In[7]:

def letterToIndex(letter):
    return all_letters.find(letter)


def nameToTensor(name):
    tensor = torch.zeros(len(name), 1, n_letters)
    for li, letter in enumerate(name):
        tensor[li][0][letterToIndex(letter)] = 1
    return tensor


# In[54]:


class RNN(nn.Module):
    def __init__(self, INPUT_SIZE, HIDDEN_SIZE, N_LAYERS,OUTPUT_SIZE):
        super(RNN, self).__init__()
        self.rnn = nn.RNN(
            input_size = INPUT_SIZE,
            hidden_size = HIDDEN_SIZE, # number of hidden units
            num_layers = N_LAYERS, # number of layers
            batch_first = True)
        self.out = nn.Linear(HIDDEN_SIZE, OUTPUT_SIZE)

    def forward(self, x):
        r_out, h = self.rnn(x, None) # None represents zero initial hidden state
        out = self.out(r_out[:, -1, :])
        return out


# In[8]:

#list comprehension:
# list_data=[]
```

```
# for category in languages:
#     for name in names[category]:
#         list_data.append((name, category))

n_hidden = 128

allnames = [] # Create list of all names and corresponding output language
for language in list(names.keys()):
    for name in names[language]:
        allnames.append([name, language])

random.shuffle(allnames)
n = 2000
x = [allnames[i:i + n] for i in range(0, len(allnames), n)]
#print(x)

# for category in list_data:
#     for name in names[category]:
#         allnames.append([name, category])

## (TO DO:) Determine Padding length (this is the length of the longest string)

# maxlen = ..... # Add code here to compute the maximum length of string
n_letters = len(all_letters)
n_categories = len(languages)

def categoryFromOutput(output):
    top_n, top_i = output.topk(1)
    category_i = top_i.item()
    return languages[category_i], category_i



learning_rate = 0.005
rnn = RNN(n_letters, 128, 1, n_categories)
optimizer = torch.optim.Adam(rnn.parameters(), lr=learning_rate)   # optimize all 
loss_func = nn.CrossEntropyLoss()
accuracies = []
for epoch in range(5):
    for batch_in_all_names in x:
        batch_size = len(batch_in_all_names)

        maxlen = max(len(x[0]) for x in batch_in_all_names)
        b_in = torch.zeros(batch_size, maxlen, n_letters)  # (TO DO:) Initialize "
        b_out = torch.zeros(batch_size, n_categories, dtype=torch.long)  # (TO DO:

        def get(charachter):
            return [z for z in charachter]
        for i in batch_in_all_names:
            j=batch_in_all_names.index(i)
            k=get(i[0])
            for l in range(len(i[0])):
                b_in[j][l][letterToIndex(k[l])]=1
            m=i[1]
            l=languages.index(m)
            b_out[j][l]=1
        max_b_out=torch.max(b_out,1)[1]
        output = rnn(b_in)                               # rnn output
        #(TO DO:)
        loss = loss_func(output, max_b_out)   # (TO DO:) Fill "...." to calculate 
        optimizer.zero_grad()                           # clear gradients for this
        loss.backward()                                 # backpropagation, compute
        optimizer.step()                                # apply gradients
```

```
        # Print accuracy
        test_output = rnn(b_in)                   #
        pred_y = torch.max(test_output, 1)[1].data.numpy().squeeze()
        test_y = torch.max(b_out, 1)[1].data.numpy().squeeze()
        accuracy = sum(pred_y == test_y)/(batch_size)
        print("Epoch: ", epoch, "| train loss: %.4f" % loss.item(), '| accuracy: %
        accuracies.append(round(accuracy,3))

print("Average accuracy is", np.mean(accuracies))
```

```
Epoch:  0 | train loss: 2.9143 | accuracy: 0.484
Epoch:  0 | train loss: 2.7362 | accuracy: 0.482
Epoch:  0 | train loss: 2.3330 | accuracy: 0.465
Epoch:  0 | train loss: 2.0021 | accuracy: 0.454
Epoch:  0 | train loss: 1.9107 | accuracy: 0.465
Epoch:  0 | train loss: 1.9316 | accuracy: 0.465
Epoch:  0 | train loss: 1.9261 | accuracy: 0.456
Epoch:  0 | train loss: 1.8570 | accuracy: 0.469
Epoch:  0 | train loss: 1.8987 | accuracy: 0.464
Epoch:  0 | train loss: 1.9000 | accuracy: 0.470
Epoch:  0 | train loss: 1.6729 | accuracy: 0.514
Epoch:  1 | train loss: 1.8462 | accuracy: 0.484
Epoch:  1 | train loss: 1.7928 | accuracy: 0.482
Epoch:  1 | train loss: 1.8641 | accuracy: 0.465
Epoch:  1 | train loss: 1.9122 | accuracy: 0.454
Epoch:  1 | train loss: 1.8689 | accuracy: 0.465
Epoch:  1 | train loss: 1.8910 | accuracy: 0.465
Epoch:  1 | train loss: 1.8837 | accuracy: 0.456
Epoch:  1 | train loss: 1.8304 | accuracy: 0.469
Epoch:  1 | train loss: 1.8824 | accuracy: 0.464
Epoch:  1 | train loss: 1.8809 | accuracy: 0.470
Epoch:  1 | train loss: 1.6824 | accuracy: 0.514
Epoch:  2 | train loss: 1.8184 | accuracy: 0.484
Epoch:  2 | train loss: 1.7934 | accuracy: 0.482
Epoch:  2 | train loss: 1.8586 | accuracy: 0.465
Epoch:  2 | train loss: 1.8977 | accuracy: 0.454
Epoch:  2 | train loss: 1.8450 | accuracy: 0.465
Epoch:  2 | train loss: 1.8863 | accuracy: 0.465
Epoch:  2 | train loss: 1.8950 | accuracy: 0.456
Epoch:  2 | train loss: 1.8313 | accuracy: 0.469
Epoch:  2 | train loss: 1.8709 | accuracy: 0.464
Epoch:  2 | train loss: 1.8715 | accuracy: 0.470
Epoch:  2 | train loss: 1.6504 | accuracy: 0.514
Epoch:  3 | train loss: 1.8246 | accuracy: 0.484
Epoch:  3 | train loss: 1.7986 | accuracy: 0.482
Epoch:  3 | train loss: 1.8634 | accuracy: 0.465
Epoch:  3 | train loss: 1.8940 | accuracy: 0.454
Epoch:  3 | train loss: 1.8426 | accuracy: 0.465
Epoch:  3 | train loss: 1.8752 | accuracy: 0.465
Epoch:  3 | train loss: 1.8861 | accuracy: 0.456
Epoch:  3 | train loss: 1.8275 | accuracy: 0.469
Epoch:  3 | train loss: 1.8756 | accuracy: 0.464
Epoch:  3 | train loss: 1.8714 | accuracy: 0.470
Epoch:  3 | train loss: 1.6499 | accuracy: 0.514
Epoch:  4 | train loss: 1.8260 | accuracy: 0.484
Epoch:  4 | train loss: 1.8012 | accuracy: 0.482
Epoch:  4 | train loss: 1.8659 | accuracy: 0.465
Epoch:  4 | train loss: 1.9004 | accuracy: 0.454
Epoch:  4 | train loss: 1.8442 | accuracy: 0.465
Epoch:  4 | train loss: 1.8773 | accuracy: 0.465
Epoch:  4 | train loss: 1.8850 | accuracy: 0.456
Epoch:  4 | train loss: 1.8289 | accuracy: 0.469
Epoch:  4 | train loss: 1.8806 | accuracy: 0.464
Epoch:  4 | train loss: 1.8766 | accuracy: 0.470
Epoch:  4 | train loss: 1.6750 | accuracy: 0.514
Average accuracy is 0.47154545454545443
```

In [3]:

```
from __future__ import unicode_literals, print_function, division
from io import open
import glob
import os
import numpy as np
import pandas as pd
import unicodedata
```

```
import string
import torch
import torch.nn as nn
import random
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker


# In[5]:
def findFiles(path):
    return glob.glob(path)

all_letters = string.ascii_letters + " .,;'"
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
    )


# In[6]:q


names = {}
languages = []


def readLines(filename):
    lines = open(filename, encoding='utf-8').read().strip().split('\n')
    return [unicodeToAscii(line) for line in lines]

# (TO DO:) CHANGE FILE PATH AS NECESSARY
for filename in findFiles(r"C:\Users\aradh\Desktop\Fall 22\TSA\Project 3.1\data\da
    category = os.path.splitext(os.path.basename(filename))[0]
    languages.append(category)
    lines = readLines(filename)
    names[category] = lines


# In[7]:

def letterToIndex(letter):
    return all_letters.find(letter)


def nameToTensor(name):
    tensor = torch.zeros(len(name), 1, n_letters)
    for li, letter in enumerate(name):
        tensor[li][0][letterToIndex(letter)] = 1
    return tensor


# In[54]:


class RNN(nn.Module):
    def __init__(self, INPUT_SIZE, HIDDEN_SIZE, N_LAYERS,OUTPUT_SIZE):
        super(RNN, self).__init__()
        self.rnn = nn.RNN(
            input_size = INPUT_SIZE,
            hidden_size = HIDDEN_SIZE, # number of hidden units
```

```
            num_layers = N_LAYERS, # number of layers
            batch_first = True)
        self.out = nn.Linear(HIDDEN_SIZE, OUTPUT_SIZE)

    def forward(self, x):
        r_out, h = self.rnn(x, None) # None represents zero initial hidden state
        out = self.out(r_out[:, -1, :])
        return out


# In[8]:

#list comprehension:
# list_data=[]
# for category in languages:
#     for name in names[category]:
#         list_data.append((name, category))

n_hidden = 128

allnames = [] # Create list of all names and corresponding output language
for language in list(names.keys()):
    for name in names[language]:
        allnames.append([name, language])

random.shuffle(allnames)
n = 3000
x = [allnames[i:i + n] for i in range(0, len(allnames), n)]
#print(x)

# for category in list_data:
#     for name in names[category]:
#         allnames.append([name, category])

## (TO DO:) Determine Padding length (this is the length of the longest string)

# maxlen = ..... # Add code here to compute the maximum length of string
n_letters = len(all_letters)
n_categories = len(languages)

def categoryFromOutput(output):
    top_n, top_i = output.topk(1)
    category_i = top_i.item()
    return languages[category_i], category_i



learning_rate = 0.005
rnn = RNN(n_letters, 128, 1, n_categories)
optimizer = torch.optim.Adam(rnn.parameters(), lr=learning_rate)   # optimize all 
loss_func = nn.CrossEntropyLoss()
accuracies = []
for epoch in range(5):
    for batch_in_all_names in x:
        batch_size = len(batch_in_all_names)

        maxlen = max(len(x[0]) for x in batch_in_all_names)
        b_in = torch.zeros(batch_size, maxlen, n_letters)  # (TO DO:) Initialize "
        b_out = torch.zeros(batch_size, n_categories, dtype=torch.long)  # (TO DO:

        def get(charachter):
            return [z for z in charachter]
        for i in batch_in_all_names:
            j=batch_in_all_names.index(i)
```

```
            k=get(i[0])
            for l in range(len(i[0])):
                b_in[j][l][letterToIndex(k[l])]=1
            m=i[1]
            l=languages.index(m)
            b_out[j][l]=1
        max_b_out=torch.max(b_out,1)[1]
        output = rnn(b_in)                                    # rnn output
        #(TO DO:)
        loss = loss_func(output, max_b_out)   # (TO DO:) Fill "...." to calculate 
        optimizer.zero_grad()                                # clear gradients for this
        loss.backward()                                      # backpropagation, compute
        optimizer.step()                                     # apply gradients

        # Print accuracy
        test_output = rnn(b_in)                    #
        pred_y = torch.max(test_output, 1)[1].data.numpy().squeeze()
        test_y = torch.max(b_out, 1)[1].data.numpy().squeeze()
        accuracy = sum(pred_y == test_y)/(batch_size)
        print("Epoch: ", epoch, "| train loss: %.4f" % loss.item(), '| accuracy: %
        accuracies.append(round(accuracy,3))

print("Average accuracy is", np.mean(accuracies))
```

```
Epoch:  0 | train loss: 2.8482 | accuracy: 0.472
Epoch:  0 | train loss: 2.6521 | accuracy: 0.469
Epoch:  0 | train loss: 2.1873 | accuracy: 0.464
Epoch:  0 | train loss: 1.9353 | accuracy: 0.480
Epoch:  0 | train loss: 1.9598 | accuracy: 0.463
Epoch:  0 | train loss: 1.9508 | accuracy: 0.465
Epoch:  0 | train loss: 1.8793 | accuracy: 0.456
Epoch:  1 | train loss: 1.8638 | accuracy: 0.472
Epoch:  1 | train loss: 1.8731 | accuracy: 0.469
Epoch:  1 | train loss: 1.9056 | accuracy: 0.464
Epoch:  1 | train loss: 1.8671 | accuracy: 0.480
Epoch:  1 | train loss: 1.8795 | accuracy: 0.463
Epoch:  1 | train loss: 1.8695 | accuracy: 0.465
Epoch:  1 | train loss: 1.8463 | accuracy: 0.456
Epoch:  2 | train loss: 1.8714 | accuracy: 0.472
Epoch:  2 | train loss: 1.8596 | accuracy: 0.469
Epoch:  2 | train loss: 1.8758 | accuracy: 0.464
Epoch:  2 | train loss: 1.8486 | accuracy: 0.480
Epoch:  2 | train loss: 1.8644 | accuracy: 0.463
Epoch:  2 | train loss: 1.8664 | accuracy: 0.465
Epoch:  2 | train loss: 1.8535 | accuracy: 0.456
Epoch:  3 | train loss: 1.8556 | accuracy: 0.472
Epoch:  3 | train loss: 1.8451 | accuracy: 0.469
Epoch:  3 | train loss: 1.8607 | accuracy: 0.464
Epoch:  3 | train loss: 1.8403 | accuracy: 0.480
Epoch:  3 | train loss: 1.8548 | accuracy: 0.463
Epoch:  3 | train loss: 1.8584 | accuracy: 0.465
Epoch:  3 | train loss: 1.8418 | accuracy: 0.456
Epoch:  4 | train loss: 1.8510 | accuracy: 0.472
Epoch:  4 | train loss: 1.8378 | accuracy: 0.469
Epoch:  4 | train loss: 1.8532 | accuracy: 0.464
Epoch:  4 | train loss: 1.8363 | accuracy: 0.480
Epoch:  4 | train loss: 1.8546 | accuracy: 0.463
Epoch:  4 | train loss: 1.8607 | accuracy: 0.465
Epoch:  4 | train loss: 1.8421 | accuracy: 0.456
Average accuracy is 0.46699999999999997
```

In [ ]:

```
In [ ]:
```

```
In [ ]:
```