lab10 - report

April 2, 2021

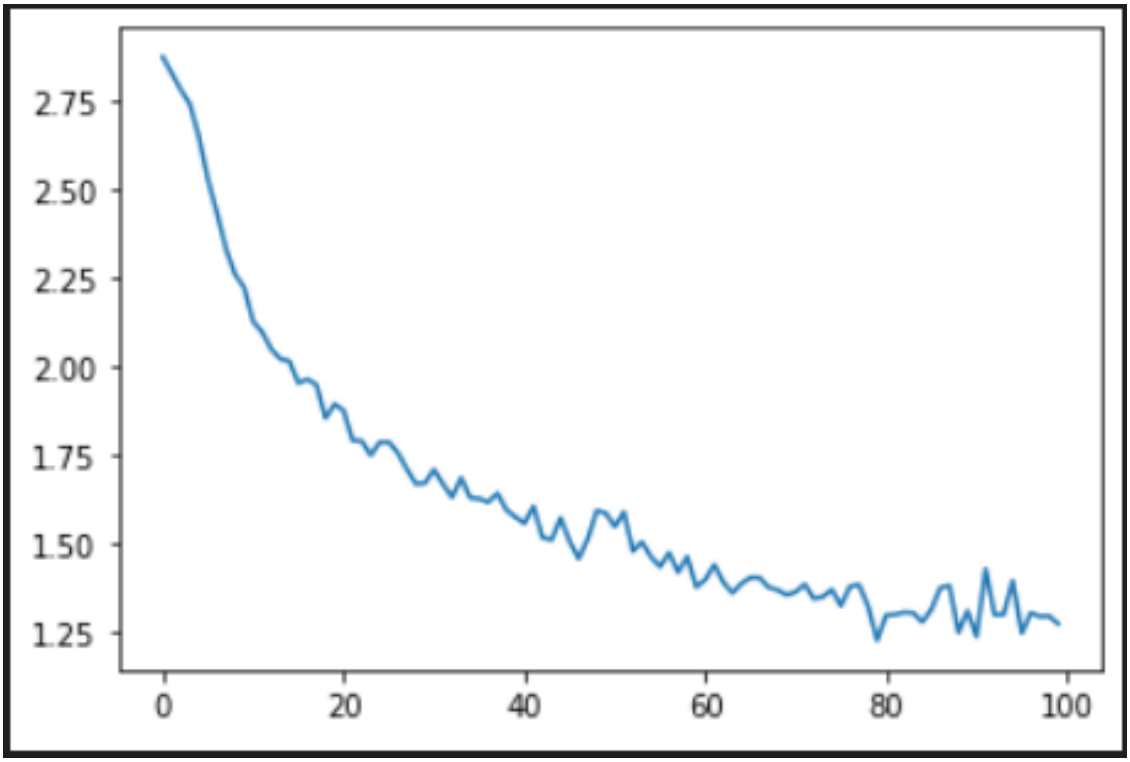
1 Lab 10 - st121413

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

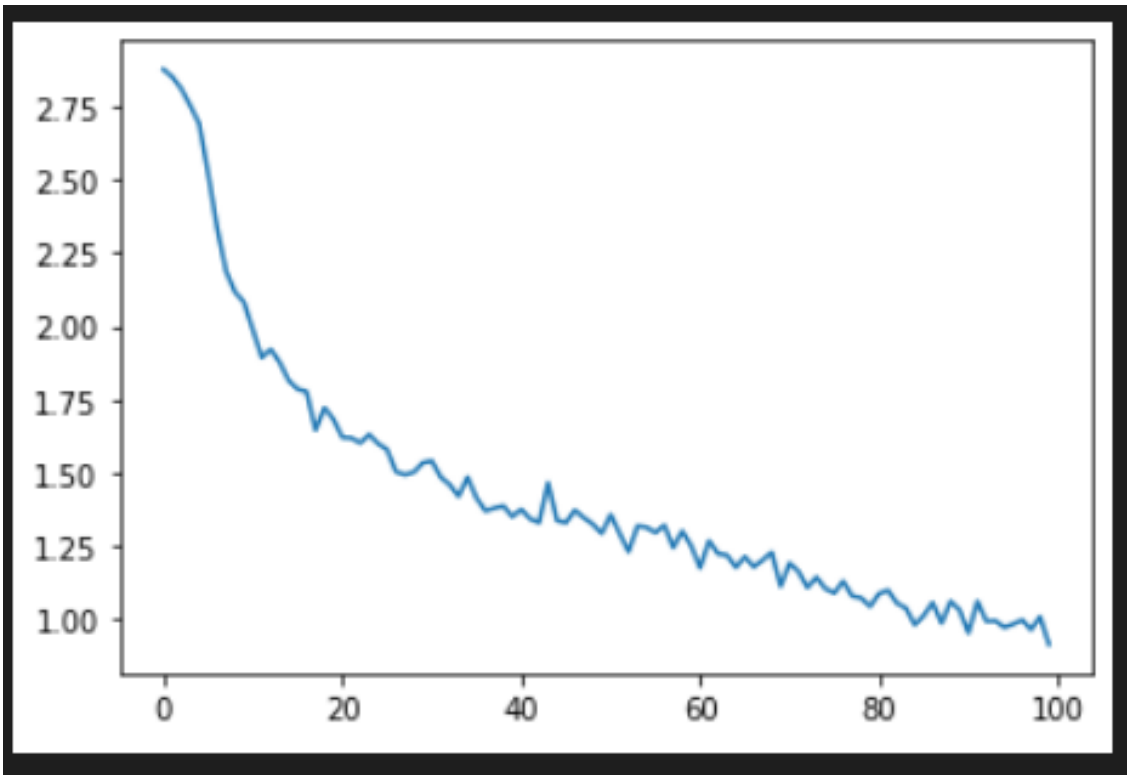
[ ]:
```
import torch.nn as nn

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.h2o = nn.Linear(hidden_size, output_size)
        # A bit more efficient than normal Softmax
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        a = self.i2h(combined)
        hidden = torch.tanh(a)
        o = self.h2o(hidden)
        y_hat = self.softmax(o)
        # hidden = self.i2h(combined)
        # output = self.i2o(combined)
        # output = self.softmax(output)
        return y_hat, hidden

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

The result of Goodfellow likes RNN performe similar or a bit better at best.

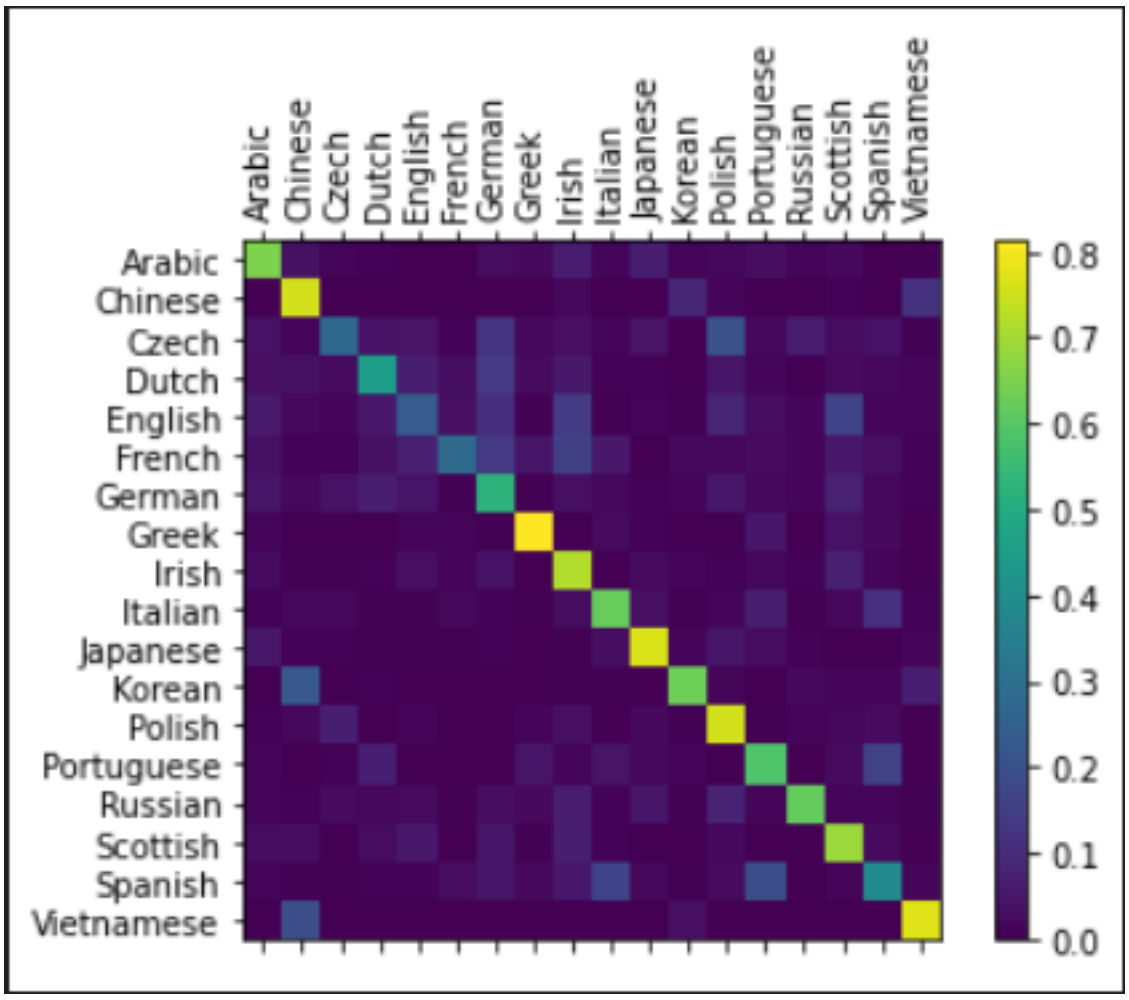
The given RNN loss



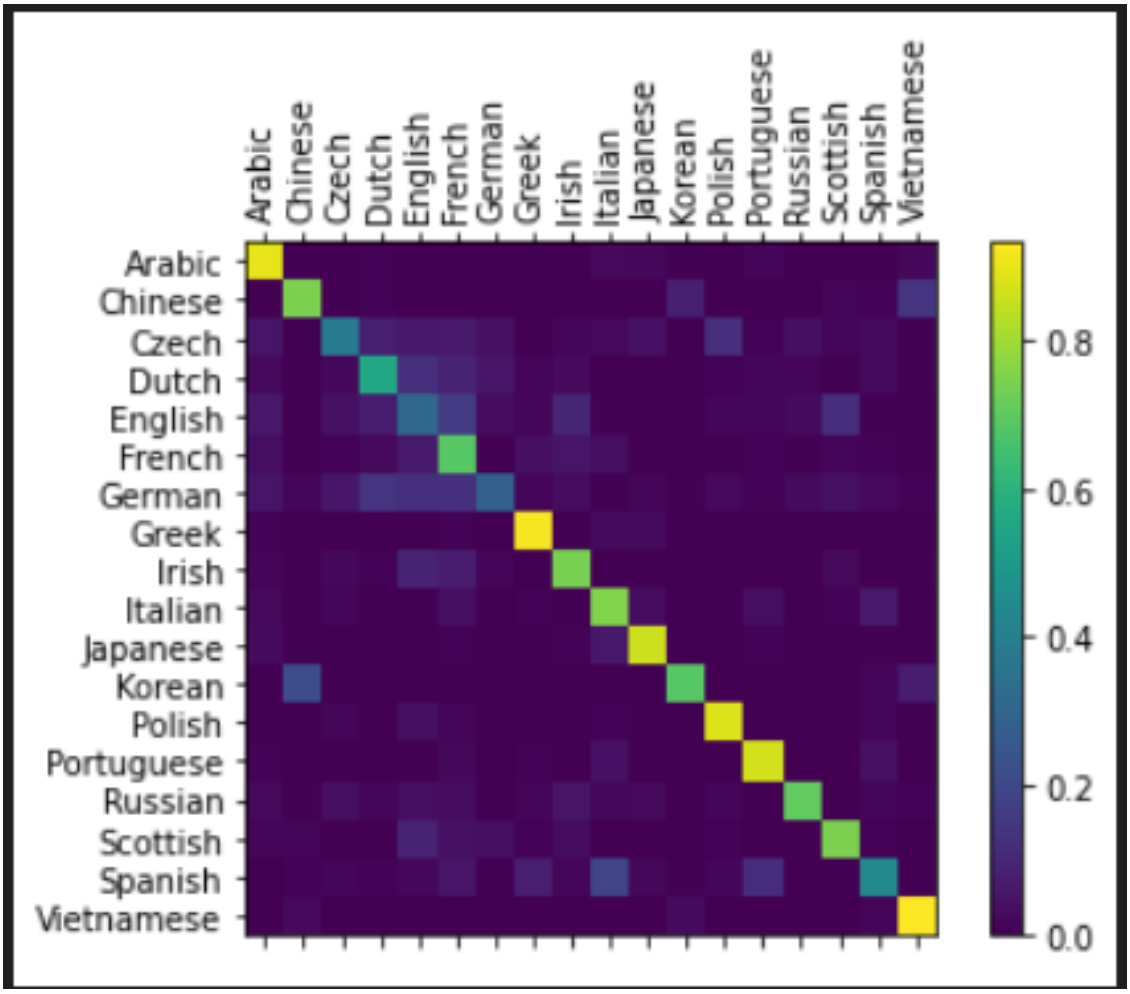
The Goodfellow likes RNN loss



The given RNN confusion martix



The Goodfellow likes RNN confusion matrix



1.2 2. Explore methods for batching patterns of different length prior to presentation to a RNN and implement them. See how much speedup you can get from the GPU with minibatch training.

This one requires a bit of modification.

First, I consult https://www.marktechpost.com/2020/04/12/implementing-batching-for-seq2seq-models-in-pytorch/ to see how to do batching.

It turns out that our previous model takes tensor of shape (word_lenght, 1, characters) and that second dimension means the size of the batch.

Idea is we want to increase the number of 1 but all the word must have the same lenght (tensor problem) so we need to pad the smaller word to equal to the biggest word in that batch.

[ ]:
```
import random

def randomChoice(l):
    # random.randint range is inclusive thus len(l)-1
    return l[random.randint(0, len(l) - 1)]
```

```
def randomTrainingExample(batch_size = 1):
    if(batch_size == 1):
        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
    else:
        max_length = 0
        categories = []
        lines = []
        lines_length = []
        # Randomly choose words from our data
        for i in range(batch_size):
            category = randomChoice(all_categories)
            line = randomChoice(category_lines[category])
            categories.append(category)
            lines.append(line)
            lines_length.append(len(line))
            # If our random word_i has the greatest lenght, save that number
            if(len(line) > max_length): max_length = len(line)
        # padding function
        line_tensor = batched_lines(lines,max_length)
        # just pack all the tags in to tensor form
        category_tensor = batched_categories(categories)
        return categories, lines, category_tensor, line_tensor
```

batched_lines function will pad all the word with all zero array after each word until the size of that word is equal to the biggest size in the list. Then pack those words into tensor.

batched_categories function is transform tags into tensor.

[ ]:
```
# https://www.marktechpost.com/2020/04/12/
 ↪implementing-batching-for-seq2seq-models-in-pytorch/
def batched_lines(names, max_word_size):
    rep = torch.zeros(max_word_size, len(names), n_letters)
    for name_index, name in enumerate(names):
        for letter_index, letter in enumerate(name):
            pos = all_letters.find(letter)
            rep[letter_index][name_index][pos] = 1
    return rep

def batched_categories(langs):
    rep = torch.zeros([len(langs)], dtype=torch.long)
    for index, lang in enumerate(langs):
        rep[index] = all_categories.index(lang)
    return rep
```

Lastly, the initeHidden must be able to generate hidden of (batch_size, hidden_size).

[ ]:
```
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.h2o = nn.Linear(hidden_size, output_size)
        # A bit more efficient than normal Softmax
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        # print(input.shape, hidden.shape)
        combined = torch.cat((input, hidden), 1)
        a = self.i2h(combined)
        hidden = torch.tanh(a)
        o = self.h2o(hidden)
        y_hat = self.softmax(o)
        # hidden = self.i2h(combined)
        # output = self.i2o(combined)
        # output = self.softmax(output)
        return y_hat, hidden

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

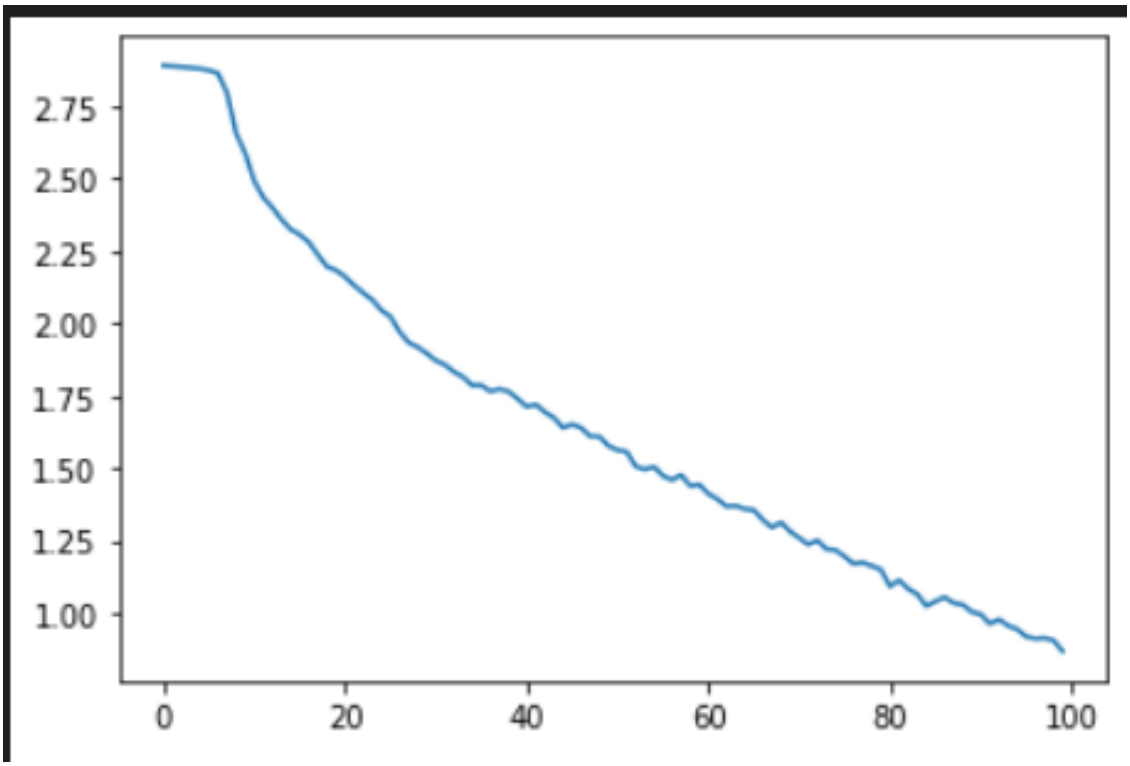
The result.

When train at batch_size = 1, the model trains slower. I think it is due to inefficient of moving data in and out of the GPU.

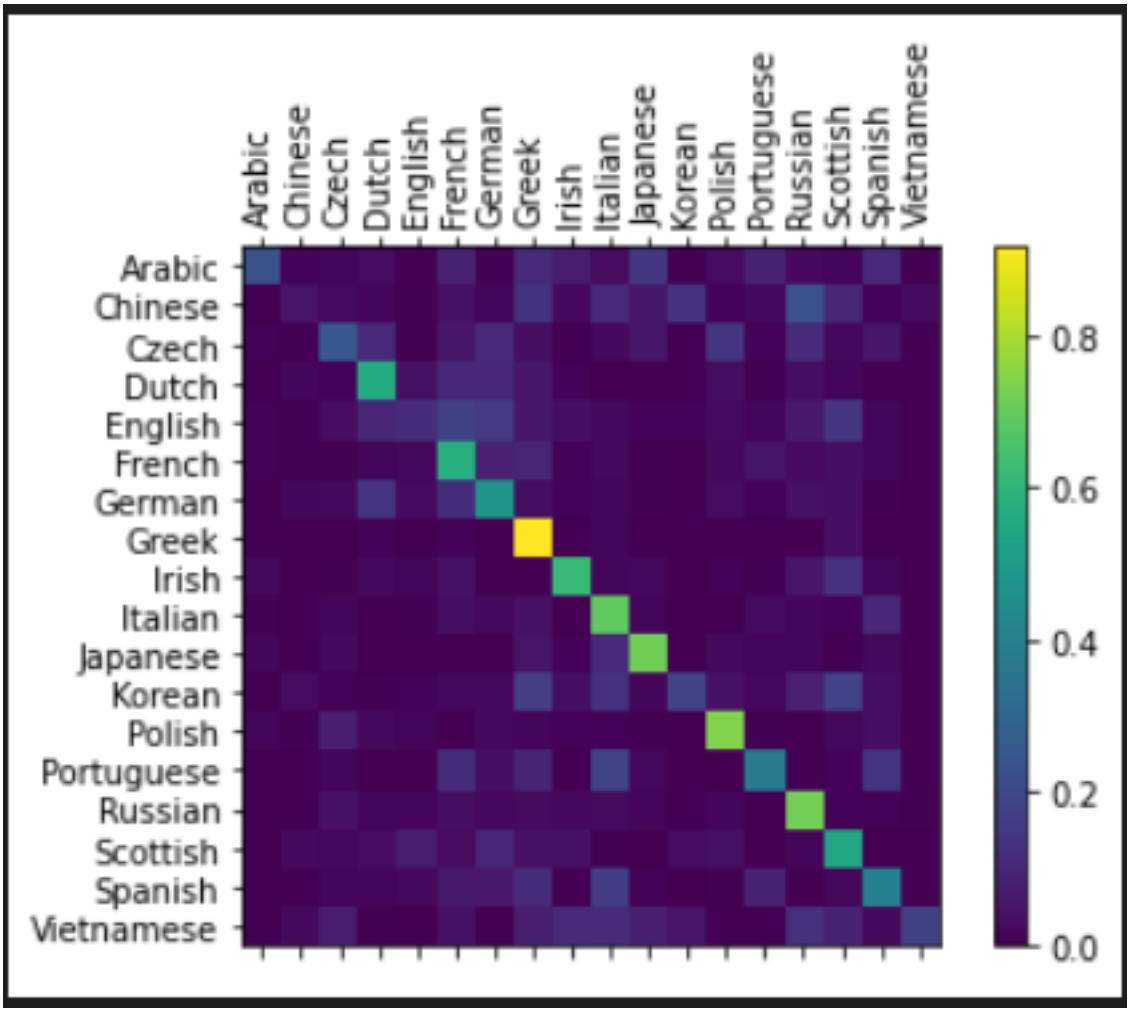
When train at batch_size > 1. - number of iter = 100000/batch_size: with this condition, the model sees the same number of words. It trains faster but perform noticibly worst. - number of iter = 100000: with this condition, the model sees more words (order of batch_size). It trains slower but perform close to the previous non-btach model

The graph is for iter = 100000 and batch size of 10

The loss



The confusion matrix



1.3 3. Do a bit of research on similar problems such as named entity recognition, find a dataset, train a model, and report your results.

I use the dataset from here becuase I want to perform NER.

https://towardsdatascience.com/named-entity-recognition-ner-using-keras-bidirectional-lstm-28cd3f301f54

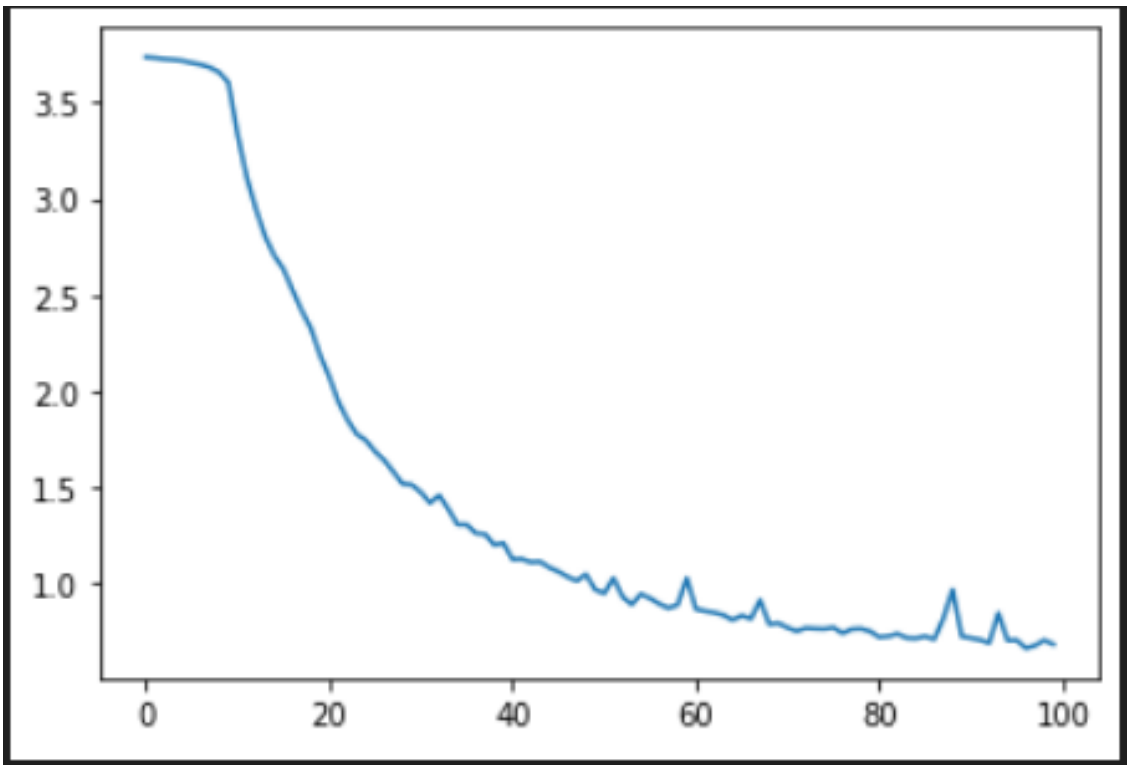
Later on, I found out that the source train the model differently from what we have done so far. Therefore, I change my mind to predict POS based on word.

I change the preparation section to prepare the dataset. This way, we predict a POS from a sequence of characters. (same idea of predict category based on a sequence)

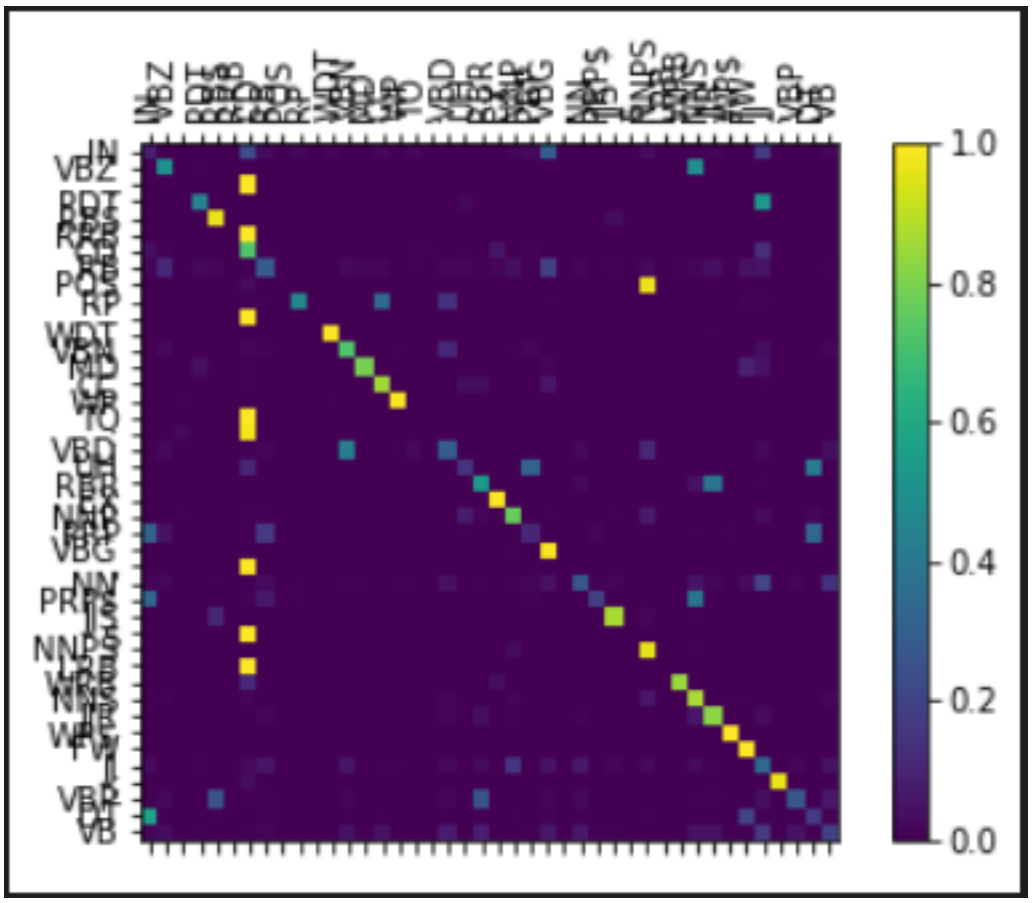
[ ]:
```
# Prepare Data
all_letters = string.ascii_letters + " .,;'"

category_lines = {}
all_categories = []
```

```
n_letters = len(all_letters)

for pos in list(set(data['POS'].to_list())):
    category_lines[pos] = []
    all_categories.append(pos)

for word,pos in zip(data['Word'].to_list(),data['POS'].to_list()):
    category_lines[pos].append(word)
```

The loss



the confusion matrix


VBZ
IN
VBZ
WDT
TO
VBD
VBG
NN
VBP
VB
1.0
0.8
0.6
0.4
0.2
0.0