```
%matplotlib inline
```

Generating Names with a Character-Level RNN

Author: `Sean Robertson <https://github.com/spro/practical-pytorch>`_

In the :doc: `last tutorial </intermediate/char_rnn_classification_tutorial>` we used a
origin. This time we'll turn around and generate names from languages.

::

```
> python sample.py Russian RUS
Rovakov
Uantov
Shavakov

> python sample.py German GER
Gerren
Ereng
Rosher

> python sample.py Spanish SPA
Salla
Parer
Allan

> python sample.py Chinese CHI
Chan
Hang
Iun
```

We are still hand-crafting a small RNN with a few linear layers. The big difference is instead of pre
letters of a name, we input a category and output one letter at a time. Recurrently predicting chara
done with words or other higher order constructs) is often referred to as a "language model".

Recommended Reading:

I assume you have at least installed PyTorch, know Python, and understand Tensors:

- http://pytorch.org/ For installation instructions

- :doc: `/beginner/former_torchies_tutorial` if you are former Lua Torch user

It would also be useful to know about RNNs and how they work:

- `The Unreasonable Effectiveness of Recurrent Neural Networks <http://karpath
  effectiveness/>`__ shows a bunch of real life examples

- `Understanding LSTM Networks <http://colah.github.io/posts/2015-08-Understan` but also informative about RNNs in general

I also suggest the previous tutorial, :doc: `/intermediate/char_rnn_classification_tutorial`

Preparing the Data

.. Note:: Download the data from `here <https://download.pytorch.org/tutorial/data.zip>`

See the last tutorial for more detail of this process. In short, there are a bunch of plain text files `da` line. We split lines into an array, convert Unicode to ASCII, and end up with a dictionary `{language`

```
from __future__ import unicode_literals, print_function, division
from io import open
import glob
import os
import unicodedata
import string

all_letters = "АаӘәБбВвГгҒғДдЕеЁёЖжЗзИиЙйКкҚқЛлМмНнҢңОоӨөПпРрСсТтУуҮүҰұФфХхҺһЦцЧчШ
n_letters = len(all_letters) + 1 # Plus EOS marker
def findFiles(path): return glob.glob(path)

# Turn a Unicode string to plain ASCII, thanks to http://stackoverflow.com/a/51823
def unicodeToAscii(s):
    temp = (c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
        and c in all_letters)

    return ''.join(temp)

# 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]

# Build the category_lines dictionary, a list of lines per category
category_lines = {}
all_categories = []
for filename in findFiles('*.txt'):
    category = os.path.splitext(os.path.basename(filename))[0]
    all_categories.append(category)
    lines = readLines(filename)
    category_lines[category] = lines
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
if n_categories == 0:
    raise RuntimeError('Data not found. Make sure that you downloaded data '
        'from https://download.pytorch.org/tutorial/data.zip and extract it to '
        'the current directory.')

print('# categories:', n_categories, all_categories)
```

```
print('# categories:', n_categories, all_categories)
print(unicodeToAscii("O'Néàl"))
```

```
# categories: 1 ['Kazakh']
'
```

Creating the Network

This network extends `the last tutorial's RNN <#Creating-the-Network>`__ with an extra a
concatenated along with the others. The category tensor is a one-hot vector just like the letter inpu

We will interpret the output as the probability of the next letter. When sampling, the most likely out

I added a second linear layer `o2o` (after combining hidden and output) to give it more muscle to w
`randomly zeros parts of its input <https://arxiv.org/abs/1207.0580>`__ with a given
fuzz inputs to prevent overfitting. Here we're using it towards the end of the network to purposely a
variety.

.. figure:: https://i.imgur.com/jzVrf7f.png :alt:

```
import torch
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(n_categories + input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(n_categories + input_size + hidden_size, output_size)
        self.o2o = nn.Linear(hidden_size + output_size, output_size)
        self.dropout = nn.Dropout(0.1)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, category, input, hidden):
        input_combined = torch.cat((category, input, hidden), 1)
        hidden = self.i2h(input_combined)
        output = self.i2o(input_combined)
        output_combined = torch.cat((hidden, output), 1)
        output = self.o2o(output_combined)
        output = self.dropout(output)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

Training

Preparing for Training

First of all, helper functions to get random pairs of (category, line):

```
import random

# Random item from a list
def randomChoice(l):
    return l[random.randint(0, len(l) - 1)]

# Get a random category and random line from that category
def randomTrainingPair():
    category = randomChoice(all_categories)
    line = randomChoice(category_lines[category])
    return category, line
```

For each timestep (that is, for each letter in a training word) the inputs of the network will be `(cat` and the outputs will be `(next letter, next hidden state)`. So for each training set, we'll neec of output/target letters.

Since we are predicting the next letter from the current letter for each timestep, the letter pairs are e.g. for `"ABCD<EOS>"` we would create ("A", "B"), ("B", "C"), ("C", "D"), ("D", "EOS").

.. figure:: https://i.imgur.com/JH58tXY.png :alt:

The category tensor is a `one-hot tensor <https://en.wikipedia.org/wiki/One-hot>`__ of s feed it to the network at every timestep - this is a design choice, it could have been included as pai strategy.

```
# One-hot vector for category
def categoryTensor(category):
    li = all_categories.index(category)
    tensor = torch.zeros(1, n_categories)
    tensor[0][li] = 1
    return tensor

# One-hot matrix of first to last letters (not including EOS) for input
def inputTensor(line):
    tensor = torch.zeros(len(line), 1, n_letters)
    for li in range(len(line)):
        letter = line[li]
        tensor[li][0][all_letters.find(letter)] = 1
    return tensor

# LongTensor of second letter to end (EOS) for target
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
    letter_indexes.append(n_letters - 1) # EOS
    return torch.LongTensor(letter_indexes)
```

For convenience during training we'll make a `randomTrainingExample` function that fetches a rar the required (category, input, target) tensors.

```
# Make category, input, and target tensors from a random category, line pair
def randomTrainingExample():
    category, line = randomTrainingPair()
    category_tensor = categoryTensor(category)
    input_line_tensor = inputTensor(line)
    target_line_tensor = targetTensor(line)
    return category_tensor, input_line_tensor, target_line_tensor
```

Training the Network

In contrast to classification, where only the last output is used, we are making a prediction at every step.

The magic of autograd allows you to simply sum these losses at each step and call backward at tl

```
criterion = nn.NLLLoss()

learning_rate = 0.0005

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

    rnn.zero_grad()

    loss = 0

    for i in range(input_line_tensor.size(0)):
        output, hidden = rnn(category_tensor, input_line_tensor[i], hidden)
        l = criterion(output, target_line_tensor[i])
        loss += l
        print(output)
        print(l)

    loss.backward()

    for p in rnn.parameters():
        p.data.add_(-learning_rate, p.grad.data)

    return output, loss.item() / input_line_tensor.size(0)
```

To keep track of how long training takes I am adding a `timeSince(timestamp)` function which re

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
def timeSince(since):
    now = time.time()
    s = now - since
    m = math.floor(s / 60)
    s -= m * 60
```

```
    return '%dm %ds' % (m, s)
```

Training is business as usual - call train a bunch of times and wait a few minutes, printing the curr
examples, and keeping store of an average loss per `plot_every` examples in `all_losses` for plo

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

n_iters = 100000
print_every = 10000
plot_every = 10000
all_losses = []
total_loss = 0 # Reset every plot_every iters

start = time.time()

for iter in 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,

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

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
          -5.5431, -5.8018, -6.1904, -5.2851, -5.3925, -5.3923, -5.3925, -6.322
          -5.8056, -5.8571, -5.5418, -5.2383, -6.1800, -5.8073, -5.7606, -5.854
          -5.7635, -3.6099, -5.3925, -4.4253, -5.3925, -5.8688, -5.6504, -5.702
          -5.6846, -5.8535, -5.4406, -5.9855, -5.3925, -6.1552, -5.8842, -5.794
          -5.8726, -5.8918, -0.7560]], grad_fn=<LogSoftmaxBackward>)
tensor(0.7560, grad_fn=<NllLossBackward>)
tensor([[-4.6281, -3.1661, -4.5636, -4.5844, -4.6652, -4.1046, -4.7222, -4.563
          -4.7283, -4.5304, -4.5736, -4.5997, -4.4677, -4.5140, -4.7091, -4.197
          -4.6893, -4.6063, -4.7057, -4.6141, -4.8666, -4.4197, -4.8130, -3.839
          -4.5842, -4.5636, -4.6039, -4.4740, -4.6201, -4.5779, -4.6931, -4.113
          -4.6673, -4.5636, -4.8101, -4.0087, -4.5620, -4.6248, -4.5513, -4.541
          -4.6883, -4.6651, -4.7704, -4.5636, -4.7231, -4.0332, -4.7686, -4.372
          -4.5950, -4.3242, -4.5866, -4.5190, -4.7331, -4.4052, -4.7081, -4.563
          -4.5912, -4.7118, -4.6747, -4.5230, -4.8114, -4.7121, -4.5636, -4.753
          -4.7707, -4.6137, -4.6334, -4.5636, -4.8301, -4.6747, -4.6576, -4.652
          -4.6668, -4.1540, -4.6818, -4.3322, -4.5636, -4.5886, -4.6839, -4.642
          -4.6440, -4.6542, -4.5822, -4.7202, -4.5862, -4.6898, -4.6652, -4.783
          -4.7237, -4.6464, -3.4559]], grad_fn=<LogSoftmaxBackward>)
tensor(4.5779, grad_fn=<NllLossBackward>)
tensor([[-4.8036, -2.5995, -4.8663, -4.7017, -4.9600, -3.8596, -5.1162, -4.812
          -4.9464, -4.6034, -4.7928, -4.6027, -4.6433, -4.4391, -4.8361, -4.010
          -4.9837, -4.8543, -4.8934, -4.7180, -5.0950, -4.4323, -5.0203, -3.398
          -4.7099, -4.8359, -4.7891, -4.4967, -4.8260, -4.7321, -4.7906, -3.864
          -4.8225, -4.1655, -4.6686, -3.8087, -4.8111, -4.9639, -4.8028, -4.640
          -4.9090, -4.9786, -4.9342, -4.7973, -4.9732, -3.7585, -4.9923, -4.278
          -4.7285, -4.2376, -4.8036, -4.5357, -5.0045, -4.4355, -4.9406, -4.646
          -4.7342, -4.8847, -4.9221, -4.5853, -5.1255, -4.7020, -4.9303, -5.002
          -4.8709, -4.8229, -4.7900, -4.6351, -5.0372, -4.6686, -4.8542, -4.845
          -4.8448, -3.9108, -4.7786, -4.2527, -4.8845, -4.7680, -4.8161, -4.851
          -4.7978, -4.8710, -4.7279, -4.9468, -4.7258, -4.9727, -4.8642, -4.934
          -4.8962, -4.8339, -2.6719]], grad_fn=<LogSoftmaxBackward>)
tensor(3.9108, grad_fn=<NllLossBackward>)
tensor([[-4.9127, -2.3155, -4.8125, -4.8125, -4.8125, -3.7963, -5.3461, -5.022
          -5.2324, -4.6598, -5.0590, -4.7561, -4.7541, -4.8125, -5.0551, -3.877
          -5.1159, -5.0019, -5.1468, -4.9049, -5.3723, -4.4818, -5.3038, -3.154
          -4.8554, -5.0639, -5.0031, -4.5124, -4.9444, -4.8389, -5.0152, -3.762
          -5.0379, -4.1920, -5.3467, -3.6750, -4.9946, -5.2189, -4.8878, -4.771
          -5.0810, -5.2501, -5.1258, -5.0082, -4.8125, -3.5990, -5.2129, -4.337
          -4.8125, -4.2307, -5.0161, -4.5730, -5.2793, -4.6102, -5.1618, -4.745
          -4.8812, -5.0726, -5.2597, -4.7139, -5.4160, -4.8630, -5.1400, -5.309
          -5.0899, -5.0323, -4.9765, -4.7334, -5.2641, -5.0539, -5.0154, -4.812
          -4.8125, -3.7885, -4.9272, -4.2174, -5.1444, -5.0449, -4.9865, -5.033
          -4.9480, -5.0333, -4.8653, -5.1488, -4.9068, -5.2141, -4.8125, -5.153
          -5.1304, -5.0813, -2.1305]], grad_fn=<LogSoftmaxBackward>)
tensor(3.7622, grad_fn=<NllLossBackward>)
tensor([[-5.2048, -2.0109, -5.4023, -5.0214, -5.5579, -3.6975, -5.7278, -5.225
          -5.0214, -4.8110, -5.3689, -4.9532, -4.9587, -4.6675, -5.3326, -3.850
          -5.4855, -5.0214, -5.4147, -5.0739, -5.7310, -4.6707, -5.5278, -2.938
          -5.1445, -5.2715, -5.3154, -5.0214, -5.0214, -5.0528, -5.3061, -3.668
          -5.2870, -4.2624, -5.6596, -3.6289, -5.0214, -5.5658, -5.1979, -4.898
          -5.0214, -5.6281, -5.3936, -5.1903, -5.4592, -3.5693, -5.4987, -4.442
          [illegible]
          [illegible]
          [illegible]
          -5.2529, -3.6906, -5.0785, -4.3822, -5.4537, -5.0214, -5.0214, -5.282
          -5.2310, -5.3582, -5.0787, -5.4433, -5.1375, -5.5487, -5.3176, -5.340
          -5.3919, -5.3137, -1.6922]], grad_fn=<LogSoftmaxBackward>)
tensor(3.6975, grad_fn=<NllLossBackward>)
tensor([[-5.4322, -1.7068, -5.5763, -5.1933, -5.1737, -3.7051, -5.9711, -5.424
          -5.1737, -4.8766, -5.6108, -5.1059, -5.1462, -4.7360, -5.5996, -3.823
          -5.7461, -5.5273, -5.6507, -5.2877, -5.9736, -4.8051, -5.8174, -5.173
```

```
            -5.2616, -5.4149, -5.5103, -4.7888, -5.1737, -5.1737, -5.4589, -3.644
            -5.4495, -4.3394, -5.9462, -3.6592, -5.5608, -5.1737, -5.3363, -5.103
            -5.5138, -5.8734, -5.6232, -5.3959, -5.6504, -3.5061, -5.7170, -4.428
            -5.1737, -4.2717, -5.5905, -4.8190, -5.7279, -4.9088, -5.5684, -5.057
            -5.2807, -5.5523, -5.8299, -5.0485, -6.1470, -5.1675, -5.5781, -5.173
            -5.5778, -5.5462, -5.3691, -5.0194, -5.8800, -5.5305, -5.4252, -5.600
            -5.1737, -5.1737, -5.2210, -4.3678, -5.1737, -5.1737, -5.3745, -5.426
            -5.4146, -5.4974, -5.2499, -5.6752, -5.3073, -5.8537, -5.6023, -5.546
            -5.6064, -5.1737, -1.3380]], grad_fn=<LogSoftmaxBackward>)
tensor(1.7068, grad_fn=<NllLossBackward>)
tensor([[-5.8159, -1.7668, -5.8920, -5.5091, -6.1845, -3.8038, -6.3889, -5.732
            -6.1735, -5.4580, -6.0233, -5.2917, -5.4159, -5.0088, -5.9435, -3.922
            -6.1135, -5.8721, -6.0133, -5.4580, -6.2499, -5.0790, -6.1373, -2.714
            -5.6242, -5.6949, -5.8073, -4.9674, -5.6436, -5.5394, -5.8096, -5.458
            -5.7905, -4.5053, -6.3183, -3.6226, -5.8220, -6.2562, -5.6930, -5.381
            -5.9182, -6.2297, -5.9868, -5.7260, -5.9814, -3.5537, -6.0480, -4.570
            -5.6273, -4.3943, -5.4580, -5.1228, -6.1106, -5.1499, -5.9168, -5.458
            -5.5935, -5.8060, -6.1716, -5.3198, -6.5399, -5.4117, -5.9041, -6.271
            -5.4580, -5.8512, -5.6485, -5.3159, -6.1514, -5.7976, -5.8385, -5.880
            -5.8348, -3.7840, -5.5412, -4.5646, -6.0421, -5.4580, -5.4580, -5.458
            -5.7622, -5.8427, -5.5949, -6.0298, -5.6422, -6.1590, -5.9943, -5.891
            -5.4580, -5.4580, -1.0954]], grad_fn=<LogSoftmaxBackward>)
tensor(2.7144, grad_fn=<NllLossBackward>)
tensor([[-5.4866, -5.4866, -5.9390, -5.5280, -5.4866, -3.6966, -6.4681, -5.785
            -6.2924, -5.1079, -6.0901, -5.3141, -5.3945, -5.4866, -6.0299, -3.831
            -6.2231, -5.9366, -6.0782, -5.5787, -6.3429, -5.0871, -6.1993, -2.542
            -5.6508, -5.7185, -5.8921, -4.9289, -5.7291, -5.6386, -5.8615, -3.575
            -5.4866, -4.4946, -6.3789, -3.5621, -5.8864, -5.4866, -5.7364, -5.463
            -5.9147, -6.3992, -6.0328, -5.7358, -6.0753, -3.4412, -6.1670, -4.638
            -5.6257, -4.3561, -6.0322, -5.1202, -6.2213, -5.2075, -5.9407, -5.355
            -5.6216, -5.8873, -6.2743, -5.3847, -6.6662, -5.4866, -5.9461, -6.426
            -5.8829, -5.9654, -5.6706, -5.3265, -6.2722, -5.8759, -5.8714, -5.991
            -5.8518, -5.4866, -5.5204, -4.5039, -6.1131, -5.4866, -5.7490, -5.809
            -5.7741, -5.9728, -5.4866, -6.0982, -5.6478, -6.2861, -5.9778, -5.899
            -5.9491, -6.0122, -0.7219]], grad_fn=<LogSoftmaxBackward>)
tensor(0.7219, grad_fn=<NllLossBackward>)
tensor([[-4.6578, -3.1849, -4.6756, -4.5625, -4.6101, -4.1244, -4.7320, -4.683
            -4.7603, -4.5772, -4.5791, -4.5545, -4.4845, -4.5182, -4.6574, -4.159
            -4.7048, -4.5758, -4.7174, -4.6287, -4.8444, -4.3435, -4.8037, -3.785
            -4.5924, -4.6925, -4.5872, -4.5208, -4.6163, -4.5074, -4.6972, -4.163
            -4.6620, -4.2311, -4.8444, -4.0763, -4.5302, -4.6417, -4.5592, -4.506
            -4.7359, -4.6570, -4.7601, -4.6323, -4.7558, -4.0210, -4.7860, -4.562
            -4.6390, -4.3297, -4.5987, -4.4877, -4.7275, -4.5625, -4.7925, -4.598
            -4.5594, -4.6660, -4.6635, -4.5150, -4.8017, -4.6750, -4.6771, -4.730
            -4.7166, -4.6195, -4.6624, -4.5375, -4.5625, -4.6284, -4.6545, -4.651
            -4.5625, -4.2116, -4.6780, -4.2897, -4.6693, -4.5837, -4.7097, -4.562
            -4.7010, -4.6727, -4.6103, -4.7230, -4.5913, -4.7009, -4.6561, -4.745
            -4.5625, -4.5769, -3.4666]], grad_fn=<LogSoftmaxBackward>)
tensor(3.1849, grad_fn=<NllLossBackward>)
tensor([[-4.6870, -2.6919, -4.8570, -4.7475, -4.6870, -3.8852, -5.1027, -4.795
            -4.6870, -4.5866, -4.8596, -4.6531, -4.6474, -4.5103, -4.8668, -4.035
            -4.9557, -4.8088, -4.6870, -4.7226, -4.6870, -4.4571, -5.0747, -3.321
                                                                               62
                                                                               35
                                                                               11
            -4.7290, -4.2092, -4.8294, -4.4855, -5.0249, -4.4672, -4.9490, -4.656
            -4.7693, -4.6870, -4.9471, -4.6026, -5.1760, -4.6910, -4.9676, -5.001
            -5.0334, -4.8188, -4.8170, -4.6246, -5.1008, -4.8053, -4.8518, -4.880
            -4.8662, -3.9036, -4.7809, -4.2498, -4.9827, -4.8034, -4.8536, -4.827
            -4.8239, -4.8698, -4.7598, -4.9373, -4.7525, -4.9338, -4.9008, -4.948
            -4.8364, -4.8834, -2.6629]], grad_fn=<LogSoftmaxBackward>)
tensor(4.5103, grad_fn=<NllLossBackward>)
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
tensor([[-4.8836, -2.0129, -5.0180, -4.7290, -5.1581, -3.5849, -5.3202, -4.883
         -5.1482, -4.5889, -4.9586, -4.6341, -4.6358, -4.4621, -4.9761, -3.731
         -5.0772, -4.9485, -5.0894, -4.8284, -5.3823, -4.4130, -5.2296, -2.905
         -4.8281, -4.9829, -4.9513, -4.4619, -4.9333, -4.7529, -4.9412, -3.606
         -4.9692, -4.1214, -5.3536, -3.5420, -4.9045, -5.1355, -4.8328, -4.668
         -5.0172, -5.1595, -5.0650, -4.9073, -5.1474, -3.4767, -5.0851, -4.238
         -4.8542, -4.0812, -5.0089, -4.7290, -5.1843, -4.4910, -5.0158, -4.642
         -4.7803, -5.0100, -5.1712, -4.6309, -4.7290, -4.7682, -5.0800, -5.192
         -5.0430, -4.9757, -4.8982, -4.6287, -5.2197, -4.9070, -4.9575, -5.031
         -4.9140, -3.6436, -4.8103, -4.1459, -5.0741, -4.9099, -4.9257, -4.983
         -4.7290, -5.0405, -4.7377, -5.1058, -4.8199, -5.2108, -5.0462, -5.025
         -5.0182, -5.0094, -4.7290]], grad_fn=<LogSoftmaxBackward>)
tensor(2.9057, grad_fn=<NllLossBackward>)
tensor([[-5.2766, -1.9535, -5.3673, -5.0628, -5.6081, -3.7204, -5.7834, -5.287
         -5.6249, -5.0465, -5.4321, -4.9493, -4.9919, -4.6780, -5.4114, -3.860
         -5.5248, -5.3308, -5.4412, -5.0922, -5.7761, -4.7483, -5.6174, -2.846
         -5.0465, -5.2351, -5.3260, -4.6893, -5.2411, -5.1378, -5.3496, -3.611
         -5.3071, -4.2888, -5.6943, -3.6268, -5.3413, -5.6296, -5.2126, -5.040
         -5.3504, -5.7074, -5.4623, -5.2520, -5.4923, -3.5233, -5.5716, -5.046
         -5.1565, -4.2393, -5.0465, -4.7581, -5.6160, -5.0465, -5.3929, -4.946
         -5.1775, -5.4094, -5.6032, -4.9946, -5.8754, -5.0669, -5.4256, -5.711
         -5.3971, -5.3811, -5.2092, -5.0465, -5.6764, -5.3283, -5.3403, -5.370
         -5.3305, -3.7115, -5.0607, -4.3067, -5.5098, -5.3197, -5.2579, -5.289
         -5.2445, -5.4492, -5.0374, -5.4960, -5.1427, -5.5939, -5.3704, -5.359
         -5.3771, -5.4145, -1.6015]], grad_fn=<LogSoftmaxBackward>)
tensor(5.0465, grad_fn=<NllLossBackward>)
tensor([[-5.1836, -1.5141, -5.3665, -4.9942, -5.5604, -3.4219, -5.7562, -5.204
         -5.5631, -4.6024, -5.3902, -4.9288, -4.8241, -4.4925, -5.3711, -3.568
         -5.5405, -5.3286, -5.4219, -5.1373, -5.6580, -4.5134, -5.5585, -2.473
         -5.0156, -5.1766, -5.2237, -4.5072, -5.0565, -5.0366, -5.2058, -4.928
         -5.2158, -4.0611, -5.7295, -3.2876, -4.9288, -5.6845, -5.1267, -4.850
         -5.2948, -5.6560, -5.3220, -5.2287, -5.4768, -3.2545, -5.4847, -4.138
         -5.0649, -3.9692, -5.3992, -4.6764, -5.5088, -4.6741, -5.3239, -4.855
         -4.9288, -5.3014, -5.6263, -4.8200, -5.8815, -4.9288, -5.3506, -5.675
         -5.3038, -5.3171, -5.1262, -4.8372, -5.6020, -5.2670, -5.3366, -5.307
         -5.2472, -3.3398, -5.0480, -4.1450, -5.4363, -5.3414, -5.1153, -5.197
         -5.1536, -5.2622, -5.0597, -5.4630, -5.0353, -4.9288, -5.3338, -5.335
         -5.3744, -5.3350, -4.9288]], grad_fn=<LogSoftmaxBackward>)
tensor(1.5141, grad_fn=<NllLossBackward>)
tensor([[-5.8247, -1.7164, -5.9263, -5.5072, -6.2233, -3.7393, -5.4784, -5.763
         -6.1783, -5.1079, -6.0476, -5.3301, -5.4184, -4.9850, -5.9568, -3.929
         -6.1613, -5.9222, -6.0472, -5.5722, -6.2897, -5.1123, -6.1818, -5.478
         -5.6665, -5.7357, -5.8509, -4.9853, -5.6765, -5.5355, -5.8020, -3.711
         -5.8075, -4.5466, -6.3625, -3.6100, -5.4784, -6.2925, -5.7214, -5.421
         -5.9385, -6.2932, -6.0171, -5.7232, -5.4784, -3.5310, -6.0612, -4.568
         -5.6535, -4.3972, -5.9556, -5.1595, -6.1613, -5.1667, -5.9406, -5.381
         -5.6148, -5.8455, -5.4784, -5.3353, -6.5603, -5.4352, -5.9447, -6.327
         -5.9648, -5.4784, -5.4784, -5.4784, -6.2042, -5.8548, -5.4784, -5.478
         -5.8608, -3.7512, -5.5687, -4.5868, -6.0633, -5.9559, -5.7374, -5.762
         -5.7686, -5.8622, -5.4784, -6.0473, -5.6814, -6.1787, -6.0118, -5.913
         -5.9356, -5.9756, -1.0032]], grad_fn=<LogSoftmaxBackward>)
tensor(1.0032, grad_fn=<NllLossBackward>)
```



```
                                                                          33
                                                                          59
                                                                          89
         -4.5455, -4.6708, -4.5653, -4.4350, -4.5818, -4.5357, -4.6543, -4.524
         -4.6284, -4.2255, -4.7717, -3.9703, -4.5242, -4.5860, -4.5242, -4.524
         -4.6497, -4.6268, -4.7319, -4.5936, -4.6846, -3.9946, -4.7299, -4.333
         -4.5560, -4.2853, -4.5480, -4.4799, -4.6947, -4.3662, -4.6691, -4.524
         -4.5523, -4.6733, -4.6360, -4.4841, -4.7730, -4.6728, -4.7274, -4.714
         -4.7319, -4.5750, -4.5947, -4.4850, -4.7917, -4.6358, -4.6189, -4.613
         -4.6279, -4.1110, -4.6431, -4.2934, -4.5242, -4.5496, -4.6454, -4.603
```

```
        -4.6052, -4.6156, -4.5432, -4.6817, -4.5474, -4.6513, -4.6263, -4.744
        -4.6852, -4.6076, -3.4132]], grad_fn=<LogSoftmaxBackward>)
tensor(3.7898, grad_fn=<NllLossBackward>)
tensor([[-4.7814, -2.6207, -4.6625, -4.6993, -4.9451, -3.8759, -5.0719, -4.784
        -4.9912, -4.6082, -4.8391, -4.6282, -4.6098, -4.4290, -4.8407, -4.030
        -4.9327, -4.7844, -4.9058, -4.6824, -5.1389, -4.4325, -5.0176, -3.319
        -4.7045, -4.8095, -4.8086, -4.4708, -4.8424, -4.6625, -4.8801, -3.805
        -4.8180, -4.1806, -5.0595, -4.6625, -4.7834, -4.9302, -4.6625, -4.687
        -4.8298, -4.9952, -4.9410, -4.7461, -4.9512, -3.6997, -5.0039, -4.359
        -4.6854, -4.2265, -4.8143, -4.4622, -5.0132, -4.4626, -4.8682, -4.612
        -4.7461, -4.8614, -4.9489, -4.6197, -5.1459, -4.6965, -4.9284, -5.016
        -4.9098, -4.8346, -4.7563, -4.6382, -5.0962, -4.6625, -4.8257, -4.867
        -4.8117, -3.8749, -4.6960, -4.1838, -4.9314, -4.7609, -4.8283, -4.807
        -4.7894, -4.9374, -4.6418, -4.9108, -4.6992, -4.9414, -4.7980, -4.889
        -4.8268, -4.8553, -2.6319]], grad_fn=<LogSoftmaxBackward>)
tensor(4.6282, grad_fn=<NllLossBackward>)
tensor([[-4.9640, -2.1818, -5.0858, -4.8265, -5.2232, -3.6948, -5.3498, -4.971
        -5.2194, -4.6623, -5.0747, -4.7436, -4.7740, -4.5447, -5.0742, -3.877
        -5.1671, -5.0298, -4.8037, -4.8885, -4.8037, -4.4595, -5.2976, -4.803
        -4.8679, -4.8037, -5.0128, -4.5066, -5.0019, -4.8359, -5.0474, -3.715
        -5.0598, -4.1545, -5.3559, -3.6108, -5.0206, -5.2873, -4.8995, -4.728
        -5.0456, -5.2490, -5.1029, -4.9703, -5.1919, -3.5797, -5.1899, -4.277
        -4.8955, -4.1783, -4.8037, -4.5395, -5.2714, -4.5734, -5.0614, -4.714
        -4.8955, -5.0773, -5.2354, -4.7169, -5.4762, -4.8325, -5.1218, -5.319
        -5.1417, -5.0336, -4.9613, -4.8037, -5.2772, -4.9969, -5.0299, -5.058
        -5.0304, -3.7425, -4.9126, -4.2038, -5.1431, -5.0093, -4.9309, -5.020
        -4.9447, -5.0060, -4.8858, -5.1523, -4.9177, -5.2021, -5.1091, -5.136
        -5.0798, -4.8037, -2.0560]], grad_fn=<LogSoftmaxBackward>)
tensor(3.7425, grad_fn=<NllLossBackward>)
tensor([[-5.2056, -2.0120, -5.3663, -5.0067, -5.5071, -3.7291, -5.6851, -5.275
        -5.5264, -4.7472, -5.3931, -4.9338, -4.9477, -4.6570, -5.3480, -3.817
        -5.4215, -5.2868, -5.4337, -5.1209, -5.6788, -4.6536, -5.5926, -2.882
        -5.1036, -5.2749, -5.2840, -4.6231, -5.1456, -5.0440, -5.2644, -5.011
        -5.2615, -4.2421, -5.6971, -3.6220, -5.2642, -5.0114, -5.1109, -4.972
        -5.3480, -5.6193, -5.3636, -5.2500, -5.4171, -3.5194, -5.5002, -4.377
        -5.1776, -4.2542, -5.3093, -4.7542, -5.5717, -4.8035, -5.4059, -4.924
        -5.0892, -5.3335, -5.5853, -4.8948, -5.7825, -5.0114, -5.3857, -5.660
        -5.3287, -5.3174, -5.1915, -4.9124, -5.5573, -5.3100, -5.3014, -5.011
        -5.2221, -3.7304, -5.1184, -4.3120, -5.4452, -5.3679, -5.0114, -5.265
        -5.1880, -5.3264, -5.0114, -5.4082, -5.1438, -5.5526, -5.3856, -5.011
        -5.4088, -5.3770, -1.6224]], grad_fn=<LogSoftmaxBackward>)
tensor(4.6536, grad_fn=<NllLossBackward>)
tensor([[-5.3614, -5.0955, -5.5444, -5.1264, -5.0955, -3.5323, -5.9347, -5.347
        -5.6863, -5.0955, -5.5198, -4.9729, -5.0136, -4.7252, -5.4969, -3.714
        -5.6605, -5.4317, -5.5873, -5.2748, -5.9006, -5.0955, -5.7356, -2.605
        -5.2526, -5.3859, -5.4591, -4.6960, -5.0955, -5.2137, -5.4114, -3.467
        -5.4091, -4.2533, -5.8353, -3.4793, -5.4195, -5.0955, -5.2662, -4.973
        -5.4900, -5.8422, -5.5207, -5.0955, -5.5833, -3.4237, -5.6272, -4.385
        -5.0955, -4.1680, -5.5699, -4.7658, -5.7159, -4.8219, -5.4985, -5.045
        -5.1416, -5.4860, -5.7472, -5.0955, -6.0218, -5.1322, -5.0955, -5.882
        -5.4999, -5.4440, -5.3078, -5.0955, -5.7203, -5.0955, -5.4116, -5.498
        [illegible]                                                        85
        [illegible]                                                        95
```

This file was updated remotely or in another tab. To force a save, overwriting the last update, select Save from the File menu

```
tensor(1.1759, grad_fn=<NllLossBackward>)
tensor([[-4.5977, -4.5298, -4.6723, -4.5973, -4.5298, -4.0565, -4.7325, -4.600
        -4.6606, -4.5170, -4.5632, -4.5415, -4.4914, -4.4325, -4.6465, -4.125
        -4.6688, -4.5590, -4.5298, -4.5546, -4.8284, -4.4235, -4.8094, -3.831
        -4.5521, -4.6763, -4.5383, -4.4409, -4.6636, -4.5211, -4.6564, -4.066
        -4.5990, -4.2148, -4.7870, -3.9603, -4.5831, -4.5860, -4.5263, -4.490
        -4.6909, -4.6585, -4.7250, -4.6220, -4.7355, -3.9879, -4.7263, -4.344
```