```
!apt-get install graphviz
!pip install torchview
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
graphviz is already the newest version (2.42.2-6).
0 upgraded, 0 newly installed, 0 to remove and 38 not upgraded.
Requirement already satisfied: torchview in
/usr/local/lib/python3.10/dist-packages (0.2.6)
```

```
import glob
import unicodedata
import string
import requests
import random
import sys
import torch
from torch.autograd import Variable
from torch import nn
from torchview import draw_graph
import random
import time
import math
from torch.autograd import Variable
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
```

Data Prep

Download and Tensorify

```
all_letters = string.ascii_letters + " .,;'-"
n_letters = len(all_letters)
print(n_letters)
def findFiles(path): return glob.glob(path)
```

```
# Turn a Unicode string to plain ASCII, thanks to
        http://stackoverflow.com/a/518232/2809427
def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
        and c in all_letters
    )


# 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]


# Find letter index from all_letters, e.g. "a" = 0
def letterToIndex(letter):
    return all_letters.find(letter)


# Turn a line into a <line_length x 1 x n_letters>,
# or 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 # one-hot-encoded
        representation
    return tensor


# Define the GitHub repository URL and branch
github_url =
        'https://api.github.com/repos/DrUzair/NLP/contents/data/names'
branch = 'f8e0c40481b1c1e32440b1da39c8bdfc9f070ffa'


# Initialize dictionaries to store data
all_categories = []
category_lines = {}


# Make a request to the GitHub API to get the list of files in the
        directory
response = requests.get(f'{github_url}?ref={branch}')


if response.status_code == 200:
    file_data = response.json()
```

```
    for file_info in file_data:
        if file_info['type'] == 'file' and
        file_info['name'].endswith('.txt'):
            file_url = file_info['download_url']
            category = file_info['name'].split('.')[0]

            # Add the category to the list
            all_categories.append(category)

            # Make a request to download the file
            file_response = requests.get(file_url)

            if file_response.status_code == 200:
                # Read and store the file content
                lines = file_response.text.split('\n')
                category_lines[category] = lines
            else:
                print(f"Failed to download file: {file_info['name']}")
else:
    print(f"Failed to retrieve file list from GitHub:
        {response.status_code}")

n_categories = len(all_categories)
```

```
58
```

```
print(all_categories)
print(len(all_categories))
```

```
['Arabic', 'Chinese', 'Czech', 'Dutch', 'English', 'French', 'German',
'Greek', 'Irish', 'Italian', 'Japanese', 'Korean', 'Polish',
'Portuguese', 'Russian', 'Scottish', 'Spanish', 'Vietnamese']
18
```

Estimate base-line error

We have 18 classes. The cross-entropy of a random classifier with 18 classes can be calculated using the formula:

$H(p,q) = -\sum_x p(x) \log q(x)$

where $p(x)$ is the true probability distribution and $q(x)$ is the predicted probability distribution.

- In the case of a random classifier, the predicted probability distribution is uniform, i.e., $q(x) = 1/18$ for all x.
- In the context of classification, the true probability distribution is typically represented as a one-hot vector, where the probability of the true class is 1 and the probabilities of all other classes are 0. In this case, the true probability distribution would be a one-hot vector with 1 in one of the 18 positions and 0 in all other positions. In this case, $p(x) = 1$ for the true class and $p(x) = 0$ for all other classes.

Substituting these values, the summation in the formula reduces to:

$H(p,q)=-\sum_xp(x)\logq(x)=-\logq(x)=-log\frac{1}{18}=\log18≈2.89$

```
category_lines['Arabic']
```

```
['Khoury',
 'Nahas',
 'Daher',
 'Gerges',
 'Nazari',
 'Maalouf',
 'Gerges',
 'Naifeh',
 'Guirguis',
 'Baba',
 'Sabbagh',
 'Attia',
 'Tahan',
 'Haddad',
 'Aswad',
 'Najjar',
 'Dagher',
 'Maloof',
 'Isa',
 'Asghar',
 'Nader',
 'Gaber',
 'Abboud',
```

```
            'Maalouf',
            'Zogby',
            'Srour',
            'Bahar',
            'Mustafa',
            'Hanania',
            'Daher',
            'Tuma',
            'Nahas',
            'Saliba',
            'Shamoon',
            'Handal',
            'Baba',
            'Amari',
            'Bahar',
            'Atiyeh',
            'Said',
            'Khouri',
            'Tahan',
            'Baba',
            'Mustafa',
            'Guirguis',
            'Sleiman',
            'Seif',
            'Dagher',
            'Bahar',
            'Gaber',
            'Harb',
            'Seif',
            'Asker',
            'Nader',
            'Antar',
            'Awad',
            'Srour',
            'Shadid',
            'Hajjar',
            'Hanania',
            'Kalb',
            'Shadid',
            'Bazzi',
```

```
            'Mustafa',
            'Masih',
            'Ghanem',
            'Haddad',
            'Isa',
            'Antoun',
            'Sarraf',
            'Sleiman',
            'Dagher',
            'Najjar',
            'Malouf',
            'Nahas',
            'Naser',
            'Saliba',
            'Shamon',
            'Malouf',
            'Kalb',
            'Daher',
            'Maalouf',
            'Wasem',
            'Kanaan',
            'Naifeh',
            'Boutros',
            'Moghadam',
            'Masih',
            'Sleiman',
            'Aswad',
            'Cham',
            'Assaf',
            'Quraishi',
            'Shalhoub',
            'Sabbag',
            'Mifsud',
            'Gaber',
            'Shammas',
            'Tannous',
            'Sleiman',
            'Bazzi',
            'Quraishi',
            'Rahal',
```

```
        'Cham',
        'Ghanem',
        'Ghanem',
        'Naser',
        'Baba',
        'Shamon',
        'Almasi',
        'Basara',
        'Quraishi',
        'Bata',
        'Wasem',
        'Shamoun',
        'Deeb',
        'Touma',
        'Asfour',
        'Deeb',
        'Hadad',
        'Naifeh',
        'Touma',
        'Bazzi',
        'Shamoun',
        'Nahas',
        'Haddad',
        'Arian',
        'Kouri',
        'Deeb',
        'Toma',
        'Halabi',
        'Nazari',
        'Saliba',
        'Fakhoury',
        'Hadad',
        'Baba',
        'Mansour',
        'Sayegh',
        'Antar',
        'Deeb',
        'Morcos',
        'Shalhoub',
        'Sarraf',
```

```
        'Amari',
        'Wasem',
        'Ganim',
        'Tuma',
        'Fakhoury',
        'Hadad',
        'Hakimi',
        'Nader',
        'Said',
        'Ganim',
        'Daher',
        'Ganem',
        'Tuma',
        'Boutros',
        'Aswad',
        'Sarkis',
        'Daher',
        'Toma',
        'Boutros',
        'Kanaan',
        'Antar',
        'Gerges',
        'Kouri',
        'Maroun',
        'Wasem',
        'Dagher',
        'Naifeh',
        'Bishara',
        'Ba',
        'Cham',
        'Kalb',
        'Bazzi',
        'Bitar',
        'Hadad',
        'Moghadam',
        'Sleiman',
        'Shamoun',
        'Antar',
        'Atiyeh',
        'Koury',
```

```
        'Nahas',
        'Kouri',
        'Maroun',
        'Nassar',
        'Sayegh',
        'Haik',
        'Ghanem',
        'Sayegh',
        'Salib',
        'Cham',
        'Bata',
        'Touma',
        'Antoun',
        'Antar',
        'Bata',
        'Botros',
        'Shammas',
        'Ganim',
        'Sleiman',
        'Seif',
        'Moghadam',
        'Ba',
        'Tannous',
        'Bazzi',
        'Seif',
        'Salib',
        'Hadad',
        'Quraishi',
        'Halabi',
        'Essa',
        'Bahar',
        'Kattan',
        'Boutros',
        'Nahas',
        'Sabbagh',
        'Kanaan',
        'Sayegh',
        'Said',
        'Botros',
        'Najjar',
```

```
        'Toma',
        'Bata',
        'Atiyeh',
        'Halabi',
        'Tannous',
        'Kouri',
        'Shamoon',
        'Kassis',
        'Haddad',
        'Tuma',
        'Mansour',
        'Antar',
        'Kassis',
        'Kalb',
        'Basara',
        'Rahal',
        'Mansour',
        'Handal',
        'Morcos',
        'Fakhoury',
        'Hadad',
        'Morcos',
        'Kouri',
        'Quraishi',
        'Almasi',
        'Awad',
        'Naifeh',
        'Koury',
        'Asker',
        'Maroun',
        'Fakhoury',
        'Sabbag',
        'Sarraf',
        'Shamon',
        'Assaf',
        'Boutros',
        'Malouf',
        'Nassar',
        'Qureshi',
        'Ghanem',
```

```
        'Srour',
        'Almasi',
        'Qureshi',
        'Ghannam',
        'Mustafa',
        'Najjar',
        'Kassab',
        'Shadid',
        'Shamoon',
        'Morcos',
        'Atiyeh',
        'Isa',
        'Ba',
        'Baz',
        'Asker',
        'Seif',
        'Asghar',
        'Hajjar',
        'Deeb',
        'Essa',
        'Qureshi',
        'Abboud',
        'Ganem',
        'Haddad',
        'Koury',
        'Nassar',
        'Abadi',
        'Toma',
        'Tannous',
        'Harb',
        'Issa',
        'Khouri',
        'Mifsud',
        'Kalb',
        'Gaber',
        'Ganim',
        'Boulos',
        'Samaha',
        'Haddad',
        'Sabbag',
```

```
        'Wasem',
        'Dagher',
        'Rahal',
        'Atiyeh',
        'Antar',
        'Asghar',
        'Mansour',
        'Awad',
        'Boulos',
        'Sarraf',
        'Deeb',
        'Abadi',
        'Nazari',
        'Daher',
        'Gerges',
        'Shamoon',
        'Gaber',
        'Amari',
        'Sarraf',
        'Nazari',
        'Saliba',
        'Naifeh',
        'Nazari',
        'Hakimi',
        'Shamon',
        'Abboud',
        'Quraishi',
        'Tahan',
        'Safar',
        'Hajjar',
        'Srour',
        'Gaber',
        'Shalhoub',
        'Attia',
        'Safar',
        'Said',
        'Ganem',
        'Nader',
        'Asghar',
        'Mustafa',
```

```
            'Said',
            'Antar',
            'Botros',
            'Nader',
            'Ghannam',
            'Asfour',
            'Tahan',
            'Mansour',
            'Attia',
            'Touma',
            'Najjar',
            'Kassis',
            'Abboud',
            'Bishara',
            'Bazzi',
            'Shalhoub',
            'Shalhoub',
            'Safar',
            'Khoury',
            'Nazari',
            'Sabbag',
            'Sleiman',
            'Atiyeh',
            'Kouri',
            'Bitar',
            'Zogby',
            'Ghanem',
            'Assaf',
            'Abadi',
            'Arian',
            'Shalhoub',
            'Khoury',
            'Morcos',
            'Shamon',
            'Wasem',
            'Abadi',
            'Antoun',
            'Baz',
            'Naser',
            'Assaf',
```

```
        'Saliba',
        'Nader',
        'Mikhail',
        'Naser',
        'Daher',
        'Morcos',
        'Awad',
        'Nahas',
        'Sarkis',
        'Malouf',
        'Mustafa',
        'Fakhoury',
        'Ghannam',
        'Shadid',
        'Gaber',
        'Koury',
        'Atiyeh',
        'Shamon',
        'Boutros',
        'Sarraf',
        'Arian',
        'Fakhoury',
        'Abadi',
        'Kassab',
        'Nahas',
        'Quraishi',
        'Mansour',
        'Samaha',
        'Wasem',
        'Seif',
        'Fakhoury',
        'Saliba',
        'Cham',
        'Bahar',
        'Shamoun',
        'Essa',
        'Shamon',
        'Asfour',
        'Bitar',
        'Cham',
```

```
        'Tahan',
        'Tannous',
        'Daher',
        'Khoury',
        'Shamon',
        'Bahar',
        'Quraishi',
        'Ghannam',
        'Kassab',
        'Zogby',
        'Basara',
        'Shammas',
        'Arian',
        'Sayegh',
        'Naifeh',
        'Mifsud',
        'Sleiman',
        'Arian',
        'Kassis',
        'Shamoun',
        'Kassis',
        'Harb',
        'Mustafa',
        'Boulos',
        'Asghar',
        'Shamon',
        'Kanaan',
        'Atiyeh',
        'Kassab',
        'Tahan',
        'Bazzi',
        'Kassis',
        'Qureshi',
        'Basara',
        'Shalhoub',
        'Sayegh',
        'Haik',
        'Attia',
        'Maroun',
        'Kassis',
```

```
        'Sarkis',
        'Harb',
        'Assaf',
        'Kattan',
        'Antar',
        'Sleiman',
        'Touma',
        'Sarraf',
        'Bazzi',
        'Boulos',
        'Baz',
        'Issa',
        'Shamon',
        'Shadid',
        'Deeb',
        'Sabbag',
        'Wasem',
        'Awad',
        'Mansour',
        'Saliba',
        'Fakhoury',
        'Arian',
        'Bishara',
        'Dagher',
        'Bishara',
        'Koury',
        'Fakhoury',
        'Naser',
        'Nader',
        'Antar',
        'Gerges',
        'Handal',
        'Hanania',
        'Shadid',
        'Gerges',
        'Kassis',
        'Essa',
        'Assaf',
        'Shadid',
        'Seif',
```

```
        'Shalhoub',
        'Shamoun',
        'Hajjar',
        'Baba',
        'Sayegh',
        'Mustafa',
        'Sabbagh',
        'Isa',
        'Najjar',
        'Tannous',
        'Hanania',
        'Ganem',
        'Gerges',
        'Fakhoury',
        'Mifsud',
        'Nahas',
        'Bishara',
        'Bishara',
        'Abadi',
        'Sarkis',
        'Masih',
        'Isa',
        'Attia',
        'Kalb',
        'Essa',
        'Boulos',
        'Basara',
        'Halabi',
        'Halabi',
        'Dagher',
        'Attia',
        'Kassis',
        'Tuma',
        'Gerges',
        'Ghannam',
        'Toma',
        'Baz',
        'Asghar',
        'Zogby',
        'Aswad',
```

```
        'Hadad',
        'Dagher',
        'Naser',
        'Shadid',
        'Atiyeh',
        'Zogby',
        'Abboud',
        'Tannous',
        'Khouri',
        'Atiyeh',
        'Ganem',
        'Maalouf',
        'Isa',
        'Maroun',
        'Issa',
        'Khouri',
        'Harb',
        'Nader',
        'Awad',
        'Nahas',
        'Said',
        'Baba',
        'Totah',
        'Ganim',
        'Handal',
        'Mansour',
        'Basara',
        'Malouf',
        'Said',
        'Botros',
        'Samaha',
        'Safar',
        'Tahan',
        'Botros',
        'Shamoun',
        'Handal',
        'Sarraf',
        'Malouf',
        'Bishara',
        'Aswad',
```

```
        'Khouri',
        'Baz',
        'Asker',
        'Toma',
        'Koury',
        'Gerges',
        'Bishara',
        'Boulos',
        'Najjar',
        'Aswad',
        'Shamon',
        'Kouri',
        'Srour',
        'Assaf',
        'Tannous',
        'Attia',
        'Mustafa',
        'Kattan',
        'Asghar',
        'Amari',
        'Shadid',
        'Said',
        'Bazzi',
        'Masih',
        'Antar',
        'Fakhoury',
        'Shadid',
        'Masih',
        'Handal',
        'Sarraf',
        'Kassis',
        'Salib',
        'Hajjar',
        'Totah',
        'Koury',
        'Totah',
        'Mustafa',
        'Sabbagh',
        'Moghadam',
        'Toma',
```

```
        'Srour',
        'Almasi',
        'Totah',
        'Maroun',
        'Kattan',
        'Naifeh',
        'Sarkis',
        'Mikhail',
        'Nazari',
        'Boutros',
        'Guirguis',
        'Gaber',
        'Kassis',
        'Masih',
        'Hanania',
        'Maloof',
        'Quraishi',
        'Cham',
        'Hadad',
        'Tahan',
        'Bitar',
        'Arian',
        'Gaber',
        'Baz',
        'Mansour',
        'Kalb',
        'Sarkis',
        'Attia',
        'Antar',
        'Asfour',
        'Said',
        'Essa',
        'Koury',
        'Hadad',
        'Tuma',
        'Moghadam',
        'Sabbagh',
        'Amari',
        'Dagher',
        'Srour',
```

```
        'Antoun',
        'Sleiman',
        'Maroun',
        'Tuma',
        'Nahas',
        'Hanania',
        'Sayegh',
        'Amari',
        'Sabbagh',
        'Said',
        'Cham',
        'Asker',
        'Nassar',
        'Bitar',
        'Said',
        'Dagher',
        'Safar',
        'Khouri',
        'Totah',
        'Khoury',
        'Salib',
        'Basara',
        'Abboud',
        'Baz',
        'Isa',
        'Cham',
        'Amari',
        'Mifsud',
        'Hadad',
        'Rahal',
        'Khoury',
        'Bazzi',
        'Basara',
        'Totah',
        'Ghannam',
        'Koury',
        'Malouf',
        'Zogby',
        'Zogby',
        'Boutros',
```

```
        'Nassar',
        'Handal',
        'Hajjar',
        'Maloof',
        'Abadi',
        'Maroun',
        'Mifsud',
        'Kalb',
        'Amari',
        'Hakimi',
        'Boutros',
        'Masih',
        'Kattan',
        'Haddad',
        'Arian',
        'Nazari',
        'Assaf',
        'Attia',
        'Wasem',
        'Gerges',
        'Asker',
        'Tahan',
        'Fakhoury',
        'Shadid',
        'Sarraf',
        'Attia',
        'Naifeh',
        'Aswad',
        'Deeb',
        'Tannous',
        'Totah',
        'Cham',
        'Baba',
        'Najjar',
        'Hajjar',
        'Shamoon',
        'Handal',
        'Awad',
        'Guirguis',
        'Awad',
```

```
        'Ganem',
        'Naifeh',
        'Khoury',
        'Hajjar',
        'Moghadam',
        'Mikhail',
        'Ghannam',
        'Guirguis',
        'Tannous',
        'Kanaan',
        'Handal',
        'Khoury',
        'Kalb',
        'Qureshi',
        'Najjar',
        'Atiyeh',
        'Gerges',
        'Nassar',
        'Tahan',
        'Hadad',
        'Fakhoury',
        'Salib',
        'Wasem',
        'Bitar',
        'Fakhoury',
        'Attia',
        'Awad',
        'Totah',
        'Deeb',
        'Touma',
        'Botros',
        'Nazari',
        'Nahas',
        'Kouri',
        'Ghannam',
        'Assaf',
        'Asfour',
        'Sarraf',
        'Naifeh',
        'Toma',
```

```
            'Asghar',
            'Abboud',
            'Issa',
            'Sabbag',
            'Sabbagh',
            'Isa',
            'Koury',
            'Kattan',
            'Shamoon',
            'Rahal',
            'Kalb',
            'Naser',
            'Masih',
            'Sayegh',
            'Dagher',
            'Asker',
            'Maroun',
            'Dagher',
            'Sleiman',
            'Botros',
            'Sleiman',
            'Harb',
            'Tahan',
            'Tuma',
            'Said',
            'Hadad',
            'Samaha',
            'Harb',
            'Cham',
            'Atiyeh',
            'Haik',
            'Malouf',
            'Bazzi',
            'Harb',
            'Malouf',
            'Ghanem',
            'Cham',
            'Asghar',
            'Samaha',
            'Khouri',
```

```
        'Nassar',
        'Rahal',
        'Baz',
        'Kalb',
        'Rahal',
        'Gerges',
        'Cham',
        'Sayegh',
        'Shadid',
        'Morcos',
        'Shamoon',
        'Hakimi',
        'Shamoon',
        'Qureshi',
        'Ganim',
        'Shadid',
        'Khoury',
        'Boutros',
        'Hanania',
        'Antoun',
        'Naifeh',
        'Deeb',
        'Samaha',
        'Awad',
        'Asghar',
        'Awad',
        'Saliba',
        'Shamoun',
        'Mikhail',
        'Hakimi',
        'Mikhail',
        'Cham',
        'Halabi',
        'Sarkis',
        'Kattan',
        'Nazari',
        'Safar',
        'Morcos',
        'Khoury',
        'Essa',
```

```
        'Nassar',
        'Haik',
        'Shadid',
        'Fakhoury',
        'Najjar',
        'Arian',
        'Botros',
        'Daher',
        'Saliba',
        'Saliba',
        'Kattan',
        'Hajjar',
        'Nader',
        'Daher',
        'Nassar',
        'Maroun',
        'Harb',
        'Nassar',
        'Antar',
        'Shammas',
        'Toma',
        'Antar',
        'Koury',
        'Nader',
        'Botros',
        'Bahar',
        'Najjar',
        'Maloof',
        'Salib',
        'Malouf',
        'Mansour',
        'Bazzi',
        'Atiyeh',
        'Kanaan',
        'Bishara',
        'Hakimi',
        'Saliba',
        'Tuma',
        'Mifsud',
        'Hakimi',
```

```
        'Assaf',
        'Nassar',
        'Sarkis',
        'Bitar',
        'Isa',
        'Halabi',
        'Shamon',
        'Qureshi',
        'Bishara',
        'Maalouf',
        'Srour',
        'Boulos',
        'Safar',
        'Shamoun',
        'Ganim',
        'Abadi',
        'Koury',
        'Shadid',
        'Zogby',
        'Boutros',
        'Shadid',
        'Hakimi',
        'Bazzi',
        'Isa',
        'Totah',
        'Salib',
        'Shamoon',
        'Gaber',
        'Antar',
        'Antar',
        'Najjar',
        'Fakhoury',
        'Malouf',
        'Salib',
        'Rahal',
        'Boulos',
        'Attia',
        'Said',
        'Kassis',
        'Bahar',
```

```
        'Bazzi',
        'Srour',
        'Antar',
        'Nahas',
        'Kassis',
        'Samaha',
        'Quraishi',
        'Asghar',
        'Asker',
        'Antar',
        'Totah',
        'Haddad',
        'Maloof',
        'Kouri',
        'Basara',
        'Bata',
        'Antar',
        'Shammas',
        'Arian',
        'Gerges',
        'Seif',
        'Almasi',
        'Tuma',
        'Shamoon',
        'Khoury',
        'Hakimi',
        'Abboud',
        'Baz',
        'Seif',
        'Issa',
        'Nazari',
        'Harb',
        'Shammas',
        'Amari',
        'Totah',
        'Malouf',
        'Sarkis',
        'Naser',
        'Zogby',
        'Handal',
```

```
 'Naifeh',
 'Cham',
 'Hadad',
 'Gerges',
 'Kalb',
 'Shalhoub',
 'Saliba',
 'Tannous',
 'Tahan',
 'Tannous',
 'Kassis',
 'Shadid',
 'Sabbag',
 'Tahan',
 'Abboud',
 'Nahas',
 'Shamoun',
 ...]
```

randomTrainingExample

```
def randomTrainingExample():
    category = all_categories[random.randint(0, len(all_categories) -
        1)]
    category_names = category_lines[category]
    name = category_names[random.randint(0, len(category_names) - 1)]
    category_tensor =
        Variable(torch.LongTensor([all_categories.index(category)]))
    name_tensor = Variable(lineToTensor(name))
    return category, name, category_tensor, name_tensor


category, name, category_tensor, name_tensor = randomTrainingExample()
print('category =', category, '/ name =', name, 'name_tensor shape',
        name_tensor.shape)
```

```
category = Italian / name = Affini name_tensor shape torch.Size([6, 1,
58])
```

Define Model

This class defines a simple Multilayer Perceptron (MLP) neural network using PyTorch's `nn.Module` module. An MLP is a type of feedforward neural network, consisting of multiple layers of nodes (neurons), each connected to the next layer. The class is defined in Python and utilizes the PyTorch library.

Let's break down the key components of this class:

1. **Constructor (`__init__`) method:**
   - It takes three parameters: `input_dim`, `hidden_dim`, and `output_dim`, representing the dimensions of the input, hidden, and output layers, respectively.
   - Initializes three linear layers (`nn.Linear`): `fc1`, `fc2`, and `softmax`.
     - `fc1`: First fully connected (linear) layer mapping input_dim to hidden_dim.
     - `fc2`: Second fully connected (linear) layer mapping hidden_dim to output_dim.
     - `softmax`: Log-softmax activation function applied along dimension `1` (usually used for classification problems).
2. **Forward method (`forward`):**
   - It defines the forward pass of the network, describing how input data moves through the layers.
   - Takes an input tensor `x_in` of shape (batch_size, input_dim).
   - Applies the first linear layer (`fc1`) followed by a Rectified Linear Unit (ReLU) activation function (`F.relu`).
   - Applies the second linear layer (`fc2`) followed by another ReLU activation.
   - Applies the log-softmax activation (`softmax`) on the output.
   - Returns the final output tensor.

In summary, this MLP consists of an input layer (`fc1`), a hidden layer with a ReLU activation, and an output layer (`fc2`) with a log-softmax activation. The goal is to map input data (name) to a probability distribution over different classes (name language).

- Note that for binary classification, the output_dim would typically be set to `1`, and a sigmoid activation might be used instead of softmax.

```
from torch import nn, sigmoid
import torch.nn.functional as F
```

```
class MultilayerPerceptron(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        """
        Args:
            input_dim (int): the size of the input vectors
            hidden_dim (int): the output size of the first Linear
        layer
            output_dim (int): the output size of the output Linear
        layer
        """
        super(MultilayerPerceptron, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        self.fc3 = nn.Linear(hidden_dim, output_dim)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, x_in):
        """The forward pass of the MLP

        Args:
            x_in (torch.Tensor): an input data tensor
                x_in.shape should be (batch, input_dim)
        Returns:
            the resulting tensor. tensor.
        """
        hidden = F.relu(self.fc1(x_in))
        output = F.relu(self.fc2(hidden))
        output = F.relu(self.fc3(output))
        output = self.softmax(output)
        return output
```

Training

categoryFromOutput

The function `categoryFromOutput(output)` is a utility function that takes the output tensor from the neural network and returns the predicted category.

- **Input:**
  - `output`: The output tensor from the neural network. It is assumed to be a PyTorch tensor.

- **Processing:**
  - `output.data`: Accesses the underlying tensor of the `output` variable.
  - `topk(1)`: Returns the indices of the top `k` elements along a specified dimension. In this case, it finds the index of the maximum element along dimension 1 (which typically corresponds to the classes in a classification problem).
  - `top_n, top_i`: Unpacks the result of `topk(1)`. `top_n` is the maximum value, and `top_i` is the index of the maximum value.
- **Output:**
  - `category_i`: Extracts the actual index of the predicted category from `top_i`.
  - `all_categories[category_i]`: Uses the index to look up the corresponding category label in the `all_categories` list.
  - Returns a tuple containing the predicted category label and its index.

`all_categories` is a list containing the category labels, and this function provides a convenient way to obtain the predicted category and its index from the network's output.

```
def categoryFromOutput(output):
    top_n, top_i = output.data.topk(1)  # Tensor out of Variable with
        .data
    category_i = top_i[0][0]
    return all_categories[category_i], category_i
```

Initialize model

```
n_hidden = 1000
mlp = MultilayerPerceptron(input_dim=n_letters, hidden_dim=n_hidden,
        output_dim=n_categories)
#model_graph = draw_graph(mlp, input_size=(1, 58), device='meta')
#model_graph.visual_graph

learning_rate = 0.001 # If you set this too high, it might explode. If
        too low, it might not learn
optimizer = torch.optim.SGD(mlp.parameters(), lr=learning_rate)
criterion = nn.NLLLoss()



def train(category_tensor, name_tensor):
    # step 1. zero the gradients
    optimizer.zero_grad()
```

```
    # step 2. compute the output
    output = mlp(name_tensor)

    # step 3. The loss is computed using
    loss = criterion(output, category_tensor)

    # step 4. The gradients of the loss are computed using
    loss.backward()

    # step 5. The weights of the MLP are updated using
    optimizer.step()

    return output, loss.item()
```

training loop

1. The for loop iterates over a range of values from 1 to n_epochs + 1. In each iteration of the loop, a random training example is selected using the randomTrainingExample() function. This function returns the category and name of a random training example, as well as the corresponding category tensor and line tensor.

2. The name_tensor is a tensor representation of the name, and tensor_1d is a 1-dimensional tensor obtained by summing the line_tensor along the first dimension.

3. The train() function is called with the category_tensor and tensor_1d as inputs. This function returns the output of the model and the loss.

4. The current_loss variable is updated with the loss from the current iteration.

- If the current iteration is a multiple of print_every, the epoch number, loss, name, and guess are printed to the console. The guess is the predicted category for the current name, and correct is a string indicating whether the prediction was correct or not.
- If the current iteration is a multiple of plot_every, the current loss average is added to the list of losses, and current_loss is reset to 0.

```
# Keep track of losses for plotting
current_loss = 0
all_losses = []
```

```
n_epochs = 1000000
print_every = 5000
plot_every = 1000
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 epoch in range(1, n_epochs + 1):
    category, name, category_tensor, name_tensor =
        randomTrainingExample()
    name_tensor = name_tensor.sum(0)
    output, loss = train(category_tensor, name_tensor)
    current_loss += loss
    # Print epoch number, loss, name and guess
    if epoch % print_every == 0:
        guess, guess_i = categoryFromOutput(output)
        correct = '✓' if guess == category else '✗ (%s)' % category
        print('%d %d%% (%s) %.4f %s / %s %s' % (
        epoch, epoch / n_epochs * 100, timeSince(start), loss, name,
        guess, correct))

    # Add current loss avg to list of losses
    if epoch % plot_every == 0:
        all_losses.append(current_loss / plot_every)
        current_loss = 0

torch.save(mlp, 'name-classifier.pt')

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

```
5000 0% (0m 14s) 2.8122 Sinagra / Japanese ✗ (Italian)
10000 1% (0m 25s) 3.0219 Charoshnikov / Japanese ✗ (Russian)
15000 1% (0m 36s) 3.0825 Loifman / Japanese ✗ (Russian)
20000 2% (0m 47s) 1.3737 Schoettmer / German ✓
```

25000 2% (0m 58s) 2.7306 Bartosz / Portuguese ✗ (Polish)
30000 3% (1m 9s) 3.1332 Jiu / Vietnamese ✗ (Chinese)
35000 3% (1m 21s) 0.2639 Jaskolski / Polish ✓
40000 4% (1m 32s) 3.1728 Sen / German ✗ (Japanese)
45000 4% (1m 43s) 1.4931 Vo / Vietnamese ✓
50000 5% (1m 54s) 2.5944 Yount / Vietnamese ✗ (French)
55000 5% (2m 5s) 2.6465 Gaspari / Portuguese ✗ (Italian)
60000 6% (2m 15s) 0.8050 O'Hara / Irish ✓
65000 6% (2m 26s) 0.9416 Mclaughlin / Irish ✗ (Scottish)
70000 7% (2m 37s) 3.3074 Reijnders / Scottish ✗ (Dutch)
75000 7% (2m 48s) 1.4904 Kieu / Vietnamese ✓
80000 8% (2m 59s) 3.9048 Mertens / German ✗ (Dutch)
85000 8% (3m 10s) 3.1327 Kringos / Scottish ✗ (Greek)
90000 9% (3m 21s) 3.3454 Fenyo / Irish ✗ (Czech)
95000 9% (3m 32s) 3.1031 San / Arabic ✗ (Korean)
100000 10% (3m 43s) 2.3514 Winograd / Scottish ✗ (Polish)
105000 10% (3m 54s) 0.8233 Mozdzierz / Polish ✓
110000 11% (4m 5s) 0.4000 Altoviti / Italian ✓
115000 11% (4m 18s) 1.3433 Bock / Czech ✓
120000 12% (4m 30s) 1.0034 Ito / Japanese ✓
125000 12% (4m 42s) 0.0533 Takizawa / Japanese ✓
130000 13% (4m 54s) 3.2843 Siskin / Polish ✗ (German)
135000 13% (5m 6s) 3.0812 Ojeda / Japanese ✗ (Spanish)
140000 14% (5m 18s) 3.1473 Kondo / English ✗ (Japanese)
145000 14% (5m 31s) 1.7880 Knopf / Czech ✗ (German)
150000 15% (5m 43s) 3.0971 Rheem / German ✗ (Korean)
155000 15% (5m 55s) 1.8804 Mckenzie / Polish ✗ (Scottish)
160000 16% (6m 7s) 3.0732 Kuiper / Czech ✗ (Dutch)
165000 16% (6m 19s) 1.5308 Sarkis / Arabic ✓
170000 17% (6m 31s) 0.1731 Soares / Portuguese ✓
175000 17% (6m 44s) 1.7499 Hussey / English ✓
180000 18% (6m 56s) 3.2053 Kokoris / Polish ✗ (Greek)
185000 18% (7m 8s) 3.0369 Roles / Spanish ✗ (English)
190000 19% (7m 21s) 0.6745 Wirner / German ✓
195000 19% (7m 33s) 0.2113 Sokolof / Polish ✓
200000 20% (7m 45s) 0.4640 Watson / Scottish ✓
205000 20% (7m 58s) 0.1672 Favreau / French ✓
210000 21% (8m 10s) 0.5752 Leclerc / French ✓
215000 21% (8m 22s) 3.2289 Mendelsohn / Scottish ✗ (German)
220000 22% (8m 34s) 0.0565 Núñez / Spanish ✓

225000 22% (8m 46s) 1.0628 Kaluza / Czech ✗ (Polish)
230000 23% (8m 58s) 0.1900 Czajka / Polish ✓
235000 23% (9m 10s) 0.7740 Brioschi / Italian ✓
240000 24% (9m 22s) 2.9833 Pastore / Portuguese ✗ (Italian)
245000 24% (9m 35s) 0.6933 Crawley / English ✓
250000 25% (9m 47s) 3.2448 / Vietnamese ✗ (Portuguese)
255000 25% (9m 59s) 0.3425 Ferreiro / Portuguese ✓
260000 26% (10m 10s) 0.3050 Watt / Scottish ✓
265000 26% (10m 22s) 2.9139 Jeon / French ✗ (Korean)
270000 27% (10m 34s) 0.7048 Munro / Scottish ✓
275000 27% (10m 47s) 0.0132 O'Shea / Irish ✓
280000 28% (10m 59s) 3.0117 Cheung / Vietnamese ✗ (Chinese)
285000 28% (11m 11s) 2.7428 Oatridge / Italian ✗ (English)
290000 28% (11m 24s) 2.8904 Dzhunusov / Korean ✗ (Russian)
295000 29% (11m 36s) 0.1994 Guirguis / Arabic ✓
300000 30% (11m 49s) 2.8904 Jong / Korean ✓
305000 30% (12m 1s) 1.0523 Kenyon / English ✓
310000 31% (12m 13s) 0.1275 Kassis / Arabic ✓
315000 31% (12m 26s) 1.9631 Elliot / Spanish ✗ (English)
320000 32% (12m 38s) 1.0338 Bleier / French ✗ (German)
325000 32% (12m 50s) 0.1286 Iturburua / Spanish ✓
330000 33% (13m 3s) 1.2286 Pollard / Spanish ✗ (English)
335000 33% (13m 15s) 3.6030 Glukhman / German ✗ (Russian)
340000 34% (13m 28s) 0.2503 Watson / Scottish ✓
345000 34% (13m 40s) 2.8904 Song / Korean ✗ (Chinese)
350000 35% (13m 53s) 1.4459 Pace / Czech ✗ (Italian)
355000 35% (14m 5s) 3.3029 Voakes / Czech ✗ (English)
360000 36% (14m 18s) 0.2228 Diep / Vietnamese ✓
365000 36% (14m 31s) 3.1281 Morcos / Italian ✗ (Arabic)
370000 37% (14m 43s) 0.1131 Crawford / Scottish ✓
375000 37% (14m 57s) 2.9989 Garber / German ✗ (English)
380000 38% (15m 10s) 0.5530 Metjeka / Czech ✓
385000 38% (15m 24s) 0.4622 Sinagra / Italian ✓
390000 39% (15m 36s) 1.3268 Mateu / French ✗ (Spanish)
395000 39% (15m 49s) 0.0477 Momotani / Japanese ✓
400000 40% (16m 2s) 0.0029 O'Brien / Irish ✓
405000 40% (16m 14s) 2.8904 Fotopoulos / Korean ✗ (Greek)
410000 41% (16m 27s) 2.9085 Huan / German ✗ (Chinese)
415000 41% (16m 39s) 4.0210 Geracimos / Italian ✗ (Greek)
420000 42% (16m 51s) 0.3860 Ballalatak / Czech ✓

425000 42% (17m 4s) 0.1394 Touma / Arabic ✓
430000 43% (17m 17s) 2.4438 Morello / Spanish ✗ (Italian)
435000 43% (17m 29s) 0.0058 Majewski / Polish ✓
440000 44% (17m 41s) 0.0416 Serafim / Portuguese ✓
445000 44% (17m 54s) 2.9522 Pae / French ✗ (Korean)
450000 45% (18m 7s) 0.0580 Serafim / Portuguese ✓
455000 45% (18m 19s) 0.2173 Kouri / Arabic ✓
460000 46% (18m 32s) 0.2269 Santos / Portuguese ✓
465000 46% (18m 45s) 0.0756 Kattan / Arabic ✓
470000 47% (18m 58s) 3.0571 Penders / Czech ✗ (Dutch)
475000 47% (19m 10s) 2.2803 Docherty / Scottish ✗ (English)
480000 48% (19m 23s) 2.8932 Paschalis / English ✗ (Greek)
485000 48% (19m 36s) 2.8904 Chun / Korean ✓
490000 49% (19m 48s) 0.1269 O'Driscoll / Irish ✓
495000 49% (20m 1s) 0.3045 Lawerenz / German ✓
500000 50% (20m 14s) 1.1720 Ustohal / Czech ✓
505000 50% (20m 26s) 1.0521 Gerald / English ✗ (Irish)
510000 51% (20m 39s) 0.2537 Ruiz / Spanish ✓
515000 51% (20m 52s) 2.5482 Rossi / Spanish ✗ (Italian)
520000 52% (21m 5s) 1.3870 Farnum / German ✗ (English)
525000 52% (21m 16s) 1.8548 Mas / Spanish ✓
530000 53% (21m 27s) 1.1091 Soma / Japanese ✓
535000 53% (21m 38s) 0.0237 Matos / Portuguese ✓
540000 54% (21m 49s) 0.3661 Cober / Czech ✓
545000 54% (22m 0s) 0.0759 Jones / Scottish ✓
550000 55% (22m 11s) 0.0695 Stawski / Polish ✓
555000 55% (22m 22s) 0.2914 Chishu / Japanese ✓
560000 56% (22m 33s) 0.0371 Johnston / Scottish ✓
565000 56% (22m 44s) 0.0088 Lefévre / French ✓
570000 56% (22m 55s) 2.9094 Drivakis / Czech ✗ (Greek)
575000 57% (23m 6s) 0.0449 Piazza / Italian ✓
580000 57% (23m 16s) 0.0636 Shamon / Arabic ✓
585000 58% (23m 27s) 1.5553 Rao / Italian ✓
590000 59% (23m 38s) 0.2797 Lawrence / English ✓
595000 59% (23m 50s) 0.1230 Garcia / Portuguese ✓
600000 60% (24m 2s) 1.3615 Santana / Portuguese ✗ (Spanish)
605000 60% (24m 15s) 0.0376 Moghadam / Arabic ✓
610000 61% (24m 27s) 1.2214 Praseta / Czech ✓
615000 61% (24m 40s) 3.0473 Iskyul / English ✗ (Russian)
620000 62% (24m 52s) 2.8904 She / Korean ✗ (Chinese)

625000 62% (25m 5s) 0.2289 Doyle / Irish ✓
630000 63% (25m 18s) 2.8904 Oh / Korean ✓
635000 63% (25m 30s) 0.0001 Kawatake / Japanese ✓
640000 64% (25m 43s) 2.5233 Lucas / Italian ✗ (English)
645000 64% (25m 55s) 2.8904 Zhui / Korean ✗ (Chinese)
650000 65% (26m 8s) 2.8904 Tugarov / Korean ✗ (Russian)
655000 65% (26m 20s) 2.9625 Cha / Czech ✗ (Korean)
660000 66% (26m 33s) 2.8904 Zha / Korean ✗ (Chinese)
665000 66% (26m 46s) 3.3747 Spiker / Czech ✗ (Dutch)
670000 67% (26m 58s) 0.0483 Oleastro / Spanish ✓
675000 67% (27m 10s) 0.1038 Bissette / French ✓
680000 68% (27m 23s) 2.8904 Xiong / Korean ✗ (Chinese)
685000 68% (27m 35s) 0.1841 Albrecht / German ✓
690000 69% (27m 47s) 0.0551 Kocian / Czech ✓
695000 69% (28m 0s) 0.0538 Black / Scottish ✓
700000 70% (28m 13s) 2.8904 Snaijer / Korean ✗ (Dutch)
705000 70% (28m 25s) 2.8904 Senyakovich / Korean ✗ (Russian)
710000 71% (28m 38s) 2.8904 Romeijnsen / Korean ✗ (Dutch)
715000 71% (28m 50s) 2.8904 Jong / Korean ✓
720000 72% (29m 4s) 3.2548 Han / Vietnamese ✗ (Korean)
725000 72% (29m 17s) 0.2844 Shinko / Japanese ✓
730000 73% (29m 29s) 0.0535 Böhme / German ✓
735000 73% (29m 42s) 3.1267 Rao / Spanish ✗ (Chinese)
740000 74% (29m 54s) 1.6186 Stuart / German ✗ (English)
745000 74% (30m 8s) 0.1003 Cassidy / Irish ✓
750000 75% (30m 20s) 2.9801 Ryoo / Irish ✗ (Korean)
755000 75% (30m 33s) 0.1316 Page / French ✓
760000 76% (30m 46s) 0.1753 Soares / Portuguese ✓
765000 76% (30m 57s) 2.8904 Bagdasarov / Korean ✗ (Russian)
770000 77% (31m 8s) 0.0084 Sniegowski / Polish ✓
775000 77% (31m 19s) 0.1397 Wykruta / Czech ✓
780000 78% (31m 30s) 2.8904 Jon / Korean ✓
785000 78% (31m 41s) 0.4054 an / Vietnamese ✓
790000 79% (31m 52s) 0.0596 Whyte / Scottish ✓
795000 79% (32m 3s) 0.0323 Coelho / Portuguese ✓
800000 80% (32m 14s) 0.0231 Ly / Vietnamese ✓
805000 80% (32m 25s) 0.6775 Kneib / German ✓
810000 81% (32m 36s) 0.0646 Davidson / Scottish ✓
815000 81% (32m 47s) 0.0162 Di pietro / Italian ✓
820000 82% (32m 57s) 0.0367 Fionn / Irish ✓

825000 82% (33m 8s) 2.8904 Langbroek / Korean ✗ (Dutch)
830000 83% (33m 19s) 0.2859 Armando / Spanish ✓
835000 83% (33m 30s) 3.0405 Belesis / French ✗ (Greek)
840000 84% (33m 41s) 0.1788 Scarpa / Italian ✓
845000 84% (33m 52s) 0.0648 Rusnak / Polish ✓
850000 85% (34m 3s) 3.3764 Vandale / Spanish ✗ (Dutch)
855000 85% (34m 14s) 0.0843 Archambault / French ✓
860000 86% (34m 25s) 0.0923 Gordon / Scottish ✓
865000 86% (34m 35s) 3.1032 Mak / Polish ✗ (Chinese)
870000 87% (34m 46s) 0.1656 Poulin / French ✓
875000 87% (34m 57s) 2.8904 Hahulin / Korean ✗ (Russian)
880000 88% (35m 8s) 1.0608 Gaspar / Portuguese ✗ (Spanish)
885000 88% (35m 19s) 2.9735 Fei / German ✗ (Chinese)
890000 89% (35m 30s) 0.0659 Sierzant / Polish ✓
895000 89% (35m 40s) 3.9750 Bahmutoff / German ✗ (Russian)
900000 90% (35m 51s) 0.1358 Kara / Czech ✓
905000 90% (36m 2s) 0.2153 Hochberg / German ✓
910000 91% (36m 13s) 2.8904 Fung / Korean ✗ (Chinese)
915000 91% (36m 24s) 0.2007 Koch / German ✓
920000 92% (36m 34s) 2.8904 Andruhov / Korean ✗ (Russian)
925000 92% (36m 45s) 2.4805 Barber / German ✗ (English)
930000 93% (36m 56s) 2.8904 Fotopoulos / Korean ✗ (Greek)
935000 93% (37m 7s) 0.0266 Hashimoto / Japanese ✓
940000 94% (37m 18s) 2.8996 Bai / Vietnamese ✗ (Chinese)
945000 94% (37m 29s) 0.0001 Bicchieri / Italian ✓
950000 95% (37m 40s) 3.2576 Schoonraad / Italian ✗ (Dutch)
955000 95% (37m 51s) 0.6576 Martin / French ✓
960000 96% (38m 1s) 0.0035 Vo / Vietnamese ✓
965000 96% (38m 13s) 0.0000 Macghabhann / Irish ✓
970000 97% (38m 23s) 1.0619 Franco / Spanish ✗ (Portuguese)
975000 97% (38m 34s) 0.9300 Rey / French ✓
980000 98% (38m 45s) 0.2613 Kyubei / Japanese ✓
985000 98% (38m 56s) 2.8904 Lai / Korean ✗ (Chinese)
990000 99% (39m 7s) 0.0116 Palmeiro / Portuguese ✓
995000 99% (39m 18s) 0.0295 Bonnet / French ✓
1000000 100% (39m 29s) 2.8904 So / Korean ✓
[2.8886846449375154, 2.8829496495723723, 2.8736342966556547,
2.8675319027900694, 2.855524185180664, 2.845370704650879,
2.840475560903549, 2.823385104894638, 2.8021208670139313,
2.7984105306863785, 2.7744062572717665, 2.762119504213333,

2.7601056272983553, 2.738252814412117, 2.7152938364744186,
2.7068535701036454, 2.68692944419384, 2.6739510185718536,
2.6526829508543015, 2.630745654165745, 2.6307583589553833,
2.589990207493305, 2.602242308139801, 2.548279051691294,
2.5848550623357296, 2.5564189337193968, 2.490727790027857,
2.5446399863362315, 2.4858048963844777, 2.5038624077737333,
2.5030919774770735, 2.4722040488421917, 2.4670026052445175,
2.479907234877348, 2.4774372902810575, 2.4204777524098753,
2.410773107632995, 2.4065671894103287, 2.3885983275324105,
2.3956566247306763, 2.378784779779613, 2.3722410248368977,
2.362623919073492, 2.389413634592667, 2.384823826547712,
2.365850778080523, 2.2812263146592304, 2.326048707768321,
2.3396152583742515, 2.264638162717223, 2.327535546980798,
2.2992420558184383, 2.3054225936718287, 2.2728520451672374,
2.287997187688947, 2.261465719839558, 2.241857475991361,
2.2882492115832864, 2.3266468914151193, 2.2657587321179453,
2.213020103370771, 2.2727119780711362, 2.1951727435870563,
2.2365803208518775, 2.261059893934056, 2.2034793226923792,
2.1859296008339153, 2.2628050150424244, 2.2243117830082775,
2.232941036120057, 2.2554129783567043, 2.1775570890312084,
2.2208964891228824, 2.213054741090571, 2.136896571743302,
2.157083583464846, 2.135433335337206, 2.1149136072949912,
2.1851918688481673, 2.1824451375436036, 2.1367681471006943,
2.1102869959836825, 2.1421987096429804, 2.134086894020438,
2.1952939639636315, 2.117095196744427, 2.129410374856554,
2.088285431671422, 2.0759161913990973, 2.1361062205820343,
2.117436904428527, 2.060828638728708, 2.154660062291703,
2.1031837686151267, 2.1060057714905125, 2.124592445190996,
2.094455037923064, 2.087424096420873, 2.08154610180459,
2.013839926118366, 2.1128670558430023, 2.095421027955861,
2.0627994657106754, 2.104142153324443, 2.1068375967449975,
2.0701574039194237, 2.0393403440490365, 2.048767996286973,
1.9952171721081249, 2.0426238059462047, 2.0860028774733657,
2.057364037933985, 2.076657881744206, 2.0890304927000254,
2.0438181824006607, 2.0152833372442984, 2.055920824032015,
2.1142899753982203, 2.0636297378975432, 2.0232862961180507,
2.03593892154959, 2.0570790489483626, 1.9779650084646418,
2.0527905301062854, 2.012762551810127, 1.9505836436637618,
1.9428068772116676, 1.974258339041582, 1.9209855702373206,
1.9787346530216745, 2.0180451575091576, 1.9923986596670002,

2.0428012646577556, 1.9362165449006716, 1.952587641977705,
1.9441956671053195, 1.9913732678974048, 1.9940667704332153,
2.004821539420285, 2.0118669256120336, 1.960673931458252,
1.9535598408843506, 1.9112625966308405, 1.9338051704694517,
1.8886778238645057, 1.8911638278625906, 1.9164857267859625,
1.9084790431093424, 1.908760763750084, 1.8978838901357957,
1.874042279885034, 1.8957030200947775, 1.9145519442209116,
1.854791109306996, 1.9023663829496362, 1.87688775073085,
1.9302192604914308, 1.960060084341443, 1.9476097579937195,
1.8989259852376998, 1.9242395064842859, 1.8668535216809832,
1.8970900061675346, 1.9279883327458447, 1.9001175168168556,
1.8945139638332185, 1.9023714544033283, 1.8474354987659025,
1.902765650673944, 1.8797213269800996, 1.8991928627168526,
1.9047865140722715, 1.9205641540790674, 1.8772556880873454,
1.8903906938817436, 1.8699054990265285, 1.827333597540748,
1.8917576184487552, 1.7870218255920336, 1.7977997786143096,
1.8483142951314804, 1.7621608733620087, 1.8536379920873978,
1.8282629608062562, 1.7916774509729003, 1.843333829684445,
1.8225229925579043, 1.7741709905277967, 1.8013410423618625,
1.83690457965112, 1.789631754233691, 1.803473297544464,
1.8456434584177623, 1.8118969370962585, 1.7658279407173687,
1.7784183663591975, 1.8173726592854365, 1.792839579974192,
1.8284133087369847, 1.7862782830903599, 1.8251039608614519,
1.7967610934388067, 1.819263399585703, 1.836336660024943,
1.8280188148371235, 1.7443474285006815, 1.7820079524995527,
1.819205686394591, 1.6913413045656926, 1.736344261950002,
1.767732164171226, 1.8111449855882675, 1.689556700852685,
1.7755973507312302, 1.696945653153467, 1.7173197490817402,
1.742061279106142, 1.7536106252987675, 1.7525005365777178,
1.686735540317255, 1.6396260502612567, 1.7289533484699933,
1.7188250314824691, 1.7095866028896998, 1.7331944133560901,
1.782304936993256, 1.712361060066207, 1.7721604361186183,
1.7126674247643787, 1.7452899185246207, 1.6985595673497083,
1.6653170302500948, 1.6800811470348562, 1.6540835724868492,
1.7654702854174247, 1.703269861697685, 1.8042334598891465,
1.6838918762746615, 1.696367799874366, 1.7079424638217606,
1.6727592814609233, 1.800132491646421, 1.69780456969043,
1.6528662632909545, 1.6240001016583119, 1.6614519561645575,
1.697098920558201, 1.714395977121021, 1.6913673364585848,
1.6623415481630073, 1.6433101551794025, 1.6755409998428368,

1.6681418724365795, 1.6706112150577237, 1.5978265548316257,
1.6268381372390486, 1.650134242248654, 1.6672266806429397,
1.630938341262401, 1.6485579730895115, 1.6021494367786508,
1.6259233248321068, 1.6111371224810356, 1.5877607403018483,
1.604791539522359, 1.641482344932243, 1.638067888631238,
1.6367043636027812, 1.6822372136487989, 1.6020575229753267,
1.6116041076249967, 1.6140642049623128, 1.6167925254395232,
1.617104691905799, 1.6054247998371576, 1.610649964943601,
1.5377346900920748, 1.596983379137673, 1.5690377189142455,
1.5649273808093276, 1.6140390890409908, 1.5808791331014058,
1.6696949706241866, 1.5961599060645968, 1.5976246319139318,
1.5901499590166603, 1.5914393281204684, 1.5655985698412624,
1.6719874407686002, 1.5083704318413875, 1.5566632252738928,
1.578814954968402, 1.5836332176435126, 1.5554772258820757,
1.5388997649314116, 1.6431859110252771, 1.5117015641137113,
1.5572835418897448, 1.5307859002685873, 1.5143387124954024,
1.5570339704888574, 1.5557318467289807, 1.4602786660195062,
1.5193813065933792, 1.549204606936316, 1.4870978130879813,
1.5963174029714537, 1.55847965639842, 1.5561593751877958,
1.4958337749806088, 1.5248302869032633, 1.5611551991190062,
1.5917038419700693, 1.5567193102433157, 1.5041239104376565,
1.521136073886868, 1.5631750292082776, 1.5601088882348122,
1.4399155845246279, 1.535355657051885, 1.5930806236079953,
1.5497496919931946, 1.5679600445528994, 1.5245235180298915,
1.4556700982210642, 1.5181404901269853, 1.5180370357270803,
1.4592033945003349, 1.5547772135646518, 1.577846230894851,
1.5059034068625479, 1.5259096893421664, 1.5356705466900857,
1.5192784244142568, 1.4048031587688783, 1.5001476870427142,
1.4517560443991406, 1.5851703190847584, 1.4869324559772503,
1.4754151684225913, 1.5211028802215834, 1.5211957286879405,
1.5678640102342396, 1.530988456512705, 1.4738205795254544,
1.4750135238975344, 1.5418514345299046, 1.4526078233184962,
1.489967091589613, 1.444929454310186, 1.4626931097964189,
1.550827113658539, 1.4960965585586914, 1.417389953935035,
1.5753096591688773, 1.4405912526678923, 1.4288590079676797,
1.4891901165685175, 1.4303559345236936, 1.4357238653052773,
1.5304983665593246, 1.4859934560155699, 1.5451061531378436,
1.4603192382222552, 1.5570095369953925, 1.41191693510826,
1.4483941903556987, 1.4587029757443233, 1.492509736498838,
1.4775148904110282, 1.3952520053945627, 1.4099725794716214,

1.4335149986605975, 1.44178578328, 1.426162060165272,
1.434097174075438, 1.4244869197184407, 1.4725482143920672,
1.4986305593310663, 1.4878548857626739, 1.3693468988852693,
1.4596582153206692, 1.4098017654474615, 1.4284091912449348,
1.4593890576655577, 1.4401891931883293, 1.4943953686910099,
1.400084144125005, 1.380510258072787, 1.4756120874992849,
1.389535075134303, 1.4513256328681272, 1.3759089301663625,
1.4466691685277038, 1.4752187997376822, 1.439189151808463,
1.4651664120600645, 1.4287547704799057, 1.4255640291570453,
1.4643974665254937, 1.3498069385963027, 1.3656484882932856,
1.3695015038552738, 1.4334936388394772, 1.429753199715391,
1.3662248314339027, 1.3904304881756233, 1.4136121929786405,
1.4273960711341673, 1.4157342939301147, 1.4162553133391558,
1.3511017576151645, 1.3592840134623, 1.4523102293845036,
1.399281440549541, 1.3707734288609628, 1.372678211522105,
1.3763281099661508, 1.3564178279121597, 1.4416278539820855,
1.3944040500420631, 1.3488552670295202, 1.3755263148774393,
1.424529600444521, 1.3534888055680276, 1.3792112493069943,
1.4127609900472144, 1.4007084545325925, 1.35792744399601,
1.3788086106899682, 1.4004157155564316, 1.3753171995785851,
1.3645454869662699, 1.4134379217936075, 1.3690872142078825,
1.4120582414753067, 1.3966770857280317, 1.4048415580975369,
1.3641281347142467, 1.327903502170864, 1.381242313239636,
1.3493856010571585, 1.416358665564986, 1.3831197456745388,
1.3631839203399638, 1.349657707768383, 1.297718027410905,
1.3777016197546854, 1.3716830626158334, 1.3984075186982263,
1.4211060530149697, 1.3949180715408538, 1.3385085412155369,
1.32793440785644, 1.3462352307777583, 1.287082344482653,
1.3822264961278123, 1.4162821679899062, 1.3262274066082664,
1.3937132241653016, 1.401502763431119, 1.3436990360391865,
1.3260878075358924, 1.3090388118610154, 1.3545458798989989,
1.3932846850179703, 1.3787996855616584, 1.4010612149642039,
1.4143871944683188, 1.3539611770581905, 1.3813321203675113,
1.3330845892109429, 1.3430102806000286, 1.3454412635672979,
1.3431926499126274, 1.3999018077215883, 1.3233919384063737,
1.3934394259396068, 1.3722094873567057, 1.3463343458409927,
1.3202962870637567, 1.2676011537539744, 1.3574172676157437,
1.3347877586349568, 1.3364879385647337, 1.339310186394965,
1.3538518521324259, 1.3724626159916078, 1.2830082006705978,
1.3172010023435545, 1.3367650569232032, 1.3377741163207015,

1.369201940253799, 1.2435966450925189, 1.3215826052214097,
1.2900538668495565, 1.3362859029242682, 1.2681146891211375,
1.2960919611380233, 1.3929753032694607, 1.3138322023390356,
1.3104295414211047, 1.3049206257163424, 1.2652336592140654,
1.3008027874143837, 1.2849874573849565, 1.305720741794983,
1.24586447961928, 1.3216200013889212, 1.3403119544929396,
1.290848363370897, 1.3183686774448093, 1.3302034703152894,
1.243009652289254, 1.404038285271523, 1.345688407055728,
1.2258096491772486, 1.2964456061893435, 1.3270928349447535,
1.2350110534382275, 1.3095459893907806, 1.3188227755133932,
1.3055051357017664, 1.4056494946644216, 1.2835537095258478,
1.3290533107242073, 1.2705398018679448, 1.2764035177052901,
1.2351075590082492, 1.3225885240223105, 1.2880809920575667,
1.2395951510246377, 1.3358501068014366, 1.276605191092036,
1.2519522008012063, 1.2688793872591264, 1.3119998600711684,
1.344454498246187, 1.3223803272304802, 1.3525371111593554,
1.2245346204374492, 1.3521337766569959, 1.2783834877873559,
1.263973551287003, 1.2087964966501876, 1.357018701430853,
1.3238635281456628, 1.27080584239144, 1.3617815598317633,
1.183435478287174, 1.296890007429871, 1.2391037860736978,
1.2911558835370598, 1.332970227393107, 1.3545963729489008,
1.2949679524406483, 1.262822942003485, 1.3691202152037507,
1.328852615292973, 1.2825872001350185, 1.3155878568632478,
1.321085980059328, 1.2506970333821263, 1.2638255122977153,
1.2957222084821405, 1.2513846994895212, 1.2836648770325119,
1.2718711512545915, 1.2117109604845209, 1.264834843752833,
1.2866293406317164, 1.262135402784098, 1.2644810986757602,
1.2810042404554314, 1.2791093346423854, 1.2489051918309393,
1.2099656991552543, 1.2833734327844728, 1.275541155534951,
1.2490759398886777, 1.2828093953723327, 1.3185430878405768,
1.2662146732276913, 1.2303861683811665, 1.258305468607859,
1.286080268720883, 1.2278570020983164, 1.2567605218371027,
1.206223130668367, 1.212821848438887, 1.2838516686852963,
1.2886941518341761, 1.2936356076077666, 1.2801931695735385,
1.23804501067021, 1.2535417770838504, 1.2566549688373352,
1.2346883159835096, 1.204268854517059, 1.3196028362782526,
1.2212042295976826, 1.2319539414856473, 1.2064789458369396,
1.2880288955938106, 1.2303121256380218, 1.2922472500182736,
1.2752984232494629, 1.2722585566090956, 1.1580332555519,
1.2496214586960397, 1.2958726789334851, 1.2996565557981585,

1.3015189234337303, 1.2648082419524453, 1.175476193064713,
1.21062547273023, 1.274017458211109, 1.294093540790318,
1.3243022865376934, 1.2294607097044064, 1.2184512910444227,
1.2003506630545957, 1.2372565610523858, 1.319711252232648,
1.224894461736514, 1.2413229768551672, 1.2782538857167474,
1.2770013176570194, 1.271436141164926, 1.254854719930476,
1.265549992234823, 1.2186882532132015, 1.210114923425261,
1.3288069651094092, 1.2280791939750106, 1.223980315905886,
1.2691149135608344, 1.24718883082104, 1.179904621110192,
1.2484693280620869, 1.2523801860383919, 1.235250006806337,
1.2622207920714672, 1.242190103361515, 1.1269021288970926,
1.223477253649046, 1.1464490555374978, 1.2739907490550577,
1.1743053660950635, 1.2482075145605267, 1.2041215111073194,
1.2174669019805477, 1.1723018560417209, 1.158591863973184,
1.1738223224451583, 1.1684319893657498, 1.2427821126770793,
1.2281413935087258, 1.2150202764590814, 1.2767824921947573,
1.257261792153523, 1.2809513713584275, 1.2788142527125923,
1.2147782743094557, 1.2039399318015955, 1.232300559525024,
1.2383994895006258, 1.1783353743844314, 1.2525686795431084,
1.1923569656120576, 1.2600180776117964, 1.2551327512776425,
1.2041037807908797, 1.2086904989683753, 1.1988046084225161,
1.2247912478255836, 1.189082460372972, 1.1952138364338143,
1.2376139301212283, 1.1728241921130993, 1.264266686372761,
1.1681475820445608, 1.1817130612558826, 1.1641700647447064,
1.242594394622683, 1.2189146671901372, 1.2314428362370344,
1.1596942841433335, 1.2035065797857751, 1.1620587144545227,
1.1685982577874439, 1.187932123661291, 1.1645141995866752,
1.1329063970812623, 1.1368287285187608, 1.1770224365837048,
1.2087120927929809, 1.2311592494461707, 1.187974530408885,
1.1724067378163328, 1.1911090839786176, 1.2345375981242064,
1.1999707845433567, 1.2642848918472374, 1.2225363343882971,
1.1757819890738237, 1.226770054942816, 1.1883376884698202,
1.1894324442352462, 1.2142125400602217, 1.1295832771138812,
1.1815848448123845, 1.1413445305224459, 1.224446483272681,
1.1813656758652296, 1.203038646552328, 1.1712214011776942,
1.2238877250690408, 1.1132578758635636, 1.212373780474809,
1.211343330472037, 1.2484539247925495, 1.1061364674389045,
1.217774417739156, 1.163532662155508, 1.2081878837407176,
1.2068326437255343, 1.207018533599612, 1.2093712749039713,
1.1515206657405697, 1.1870697213200225, 1.1955093830609502,

1.1621317633263248, 1.2341095242744313, 1.1045334585773137,
1.2674078954797885, 1.1912813444833903, 1.0883588345832949,
1.0931600891905435, 1.2435383184991158, 1.136373139498077,
1.2004113497638464, 1.1920536968742583, 1.1761525884425683,
1.1514958320288844, 1.1489095451784679, 1.1434316682566497,
1.1478593586002694, 1.2348935636946075, 1.14994058864267,
1.2112560604690243, 1.2396130439186135, 1.1452126526001791,
1.201859940869289, 1.1969405712686805, 1.133005622987918,
1.2064011863765687, 1.1806881095859731, 1.16032226742701,
1.1228959979552096, 1.1877553806299097, 1.1425621543471676,
1.2113953572464633, 1.1098270838957542, 1.1490544183572502,
1.0842014018611454, 1.189219214296127, 1.2308921279386105,
1.097237430432384, 1.159253782008107, 1.0920990763180003,
1.2070277774916003, 1.13365080873706, 1.152196877260467,
1.118115969859552, 1.1110632441486863, 1.1975758596415045,
1.1367690736450133, 1.283328689338785, 1.2240723441471966,
1.1490308229079125, 1.1394264823553562, 1.2064815307197205,
1.091354915982367, 1.1548644872332856, 1.202373421549884,
1.1560661824746057, 1.1256456897242102, 1.1830698992789532,
1.1594166695596135, 1.1457926813593053, 1.1562768612927543,
1.1335560458662657, 1.1550373110681427, 1.1883138482173408,
1.0765216170073855, 1.140550220510512, 1.1744465197029252,
1.1623989264162793, 1.1593502157502602, 1.193811260037482,
1.1617817117470404, 1.1451759833154087, 1.2231159270899912,
1.2249494515989419, 1.1388999769308112, 1.2296319528246702,
1.1829651034721662, 1.1381424703769785, 1.2309936042359633,
1.1904621473205412, 1.1264753556024862, 1.1214941560303868,
1.1298946916371755, 1.091548304133807, 1.1490404873093039,
1.2180786931707275, 1.2027255522437617, 1.1315683646273647,
1.1485607435828562, 1.1789176022902508, 1.1732408439503124,
1.1648670526325733, 1.1328761055726673, 1.1252474705971927,
1.1089509069796555, 1.1909286656700737, 1.1747383646131921,
1.1877359677878643, 1.1797915943109725, 1.1661993493461924,
1.2136521555632318, 1.176892441943202, 1.145533843110678,
1.0892791997950375, 1.129922679043926, 1.1899314689966098,
1.1853433532415534, 1.1359752912158816, 1.1605404784584346,
1.087247708855883, 1.1299648347711309, 1.1199830473967918,
1.0957356320368936, 1.1580090506143952, 1.1744992117354065,
1.1716653837046644, 1.152811085619409, 1.1045287474285097,
1.1797336559684095, 1.2515676708398338, 1.0862732682599467,

1.1946182119581616, 1.102978099553856, 1.1642293878807741,
1.1175329824073024, 1.175559512411721, 1.068668494873267,
1.1173914597371253, 1.1884333333798838, 1.2163534389875454,
1.1878537686248938, 1.1365436707211736, 1.283106051572175,
1.2129769222604403, 1.159818677451489, 1.18119707474735,
1.1999900676034072, 1.129026899830611, 1.177374570644735,
1.2198428710200628, 1.0925365752675926, 1.2411763469900825,
1.1364957177815218, 1.1169322982483192, 1.0978599280609629,
1.1896392853980124, 1.208133157526534, 1.0435104122905154,
1.0505423443355948, 1.1159014233128286, 1.1913026480901496,
1.1522929181826875, 1.1672254597437957, 1.2190171104177587,
1.0974410542041015, 1.0951011547788103, 1.1936315921605751,
1.1486113592955172, 1.1271254133366588, 1.1610462775910433,
1.1738063537892471, 1.1346139728579565, 1.2471594745535304,
1.0916047358947867, 1.1164278092949944, 1.1119731856171557,
1.2139269989532508, 1.2018609706189722, 1.1719310817387258,
1.1910456158317764, 1.0803400846352764, 1.1477145598898821,
1.125217364046659, 1.244103095776905, 1.1581878627624766,
1.159521861709502, 1.1198334557758514, 1.1345303399534188,
1.0506265693134513, 1.1144930942710363, 1.080403611999669,
1.1306023272652268, 1.1134026894523765, 1.1786354505711034,
1.0338374443924252, 1.214342131554817, 1.132260681879282,
1.1118271254136365, 1.0851901260156909, 1.1356179206500079,
1.1466262229603672, 1.1738874722774808, 1.2000161589652794,
1.1253682552794322, 1.1751256537611938, 1.2317549361460347,
1.0688188495809627, 1.0753162803633323, 1.1598633458298342,
1.1323609685550033, 1.1082886127321478, 1.0620311655201438,
1.2123638009013884, 1.175218913273584, 1.0962332841890317,
1.0434608497121631, 1.120349283580408, 1.1675008411015706,
1.065141628686389, 1.156005642038032, 1.11208411209543,
1.2122637776582645, 1.1542306955623967, 1.1808788746132626,
1.1374921493535635, 1.1120267449996508, 1.0753026824034313,
1.1685665835167685, 1.2383820327104516, 1.1767342337539166,
1.165022979701034, 1.1377796773026563, 1.133123556805296,
1.1924725312014626, 1.1250565019883847, 1.162170983839949,
1.121017157513021, 1.0791034432578144, 1.0692765836558964,
1.1281247585283662, 1.149873439389522, 1.1163144776332028,
1.0999295401257574, 1.066255163969591, 1.1249440958598547,
1.155489514200539, 1.1029398777854553, 1.140477356970794,
1.0787329924736018, 1.1032495885067013, 1.090730198636797,

```
1.1238969078233283, 1.1578368978389877, 1.135024776899848,
1.2002457295462514, 1.0956440375322214, 1.1573130736889274,
1.2028120768560084, 1.1224531519856236, 1.1557890295862367,
1.089394919760649, 1.1002036919983156, 1.1516174681274085,
1.164710084581884, 1.0415508973385643, 1.1256786298539574,
1.1649285605433488, 1.124394179621328, 1.085242131868291,
1.1687695957096038, 1.09605753576028, 1.0594472472539858,
1.1060514940053299, 1.0758128899952266, 1.0902018838591139,
1.1735729661810954, 1.1519226119610948, 1.0985913579873063,
1.1108800909488605]

[<matplotlib.lines.Line2D at 0x7c3a9101a080>]
```

Evaluate

```
import torch
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

#mlp = torch.load('./name-classifier.pt')

# Keep track of correct guesses in a confusion matrix
confusion = torch.zeros(n_categories, n_categories)
n_confusion = 10000 # This is the number of examples that will be used
        to evaluate the performance of the model.

# Go through a bunch of examples and record which are correctly
        guessed
for i in range(n_confusion):
    category, name, category_tensor, name_tensor =
        randomTrainingExample()
    name_tensor = name_tensor.sum(0)
    if len(name) > 0:
      #print(name_tensor.shape)
      output = mlp(name_tensor)
      guess, guess_i = categoryFromOutput(output)
      #print(category, name, guess)
      category_i = all_categories.index(category)
      confusion[category_i][guess_i] += 1
    else:
```

```
        continue
        # print(category, name)

# 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()
```

```
<ipython-input-12-03ad0d7359f7>:37: UserWarning: FixedFormatter should
only be used together with FixedLocator
  ax.set_xticklabels([''] + all_categories, rotation=90)
<ipython-input-12-03ad0d7359f7>:38: UserWarning: FixedFormatter should
only be used together with FixedLocator
  ax.set_yticklabels([''] + all_categories)
```

Predict

```
mlp = torch.load('name-classifier.pt')

def predict(line, n_predictions=3):
    output = mlp(Variable(lineToTensor(line).sum(0)))
```

```
    # Get top N categories
    topv, topi = output.data.topk(n_predictions, 1, True)
    predictions = []

    for i in range(n_predictions):
        value = topv[0][i]
        category_index = topi[0][i]
        print('(%.2f) %s' % (value, all_categories[category_index]))
        predictions.append([value, all_categories[category_index]])

    return predictions


if __name__ == '__main__':
    predict('koh')
```

```
(-2.89) Korean
(-2.89) Portuguese
(-2.89) Polish
```