Kissipo Learning for Deep Learning

Topic 25: Name prediction project (20min)

Hsueh-Ting Chu

KLDL-W9-T25

Course Schedule

- W1 - Course Introduction
- W2 - DL Programming Basics(1)
- W3 - DL Programming Basics(2)
- W4 - DL with TensorFlow
- W5 - Midterm
- W6 - DL with PyTorch
- W7 - AOI hands-on project
- W8 - RSD hands-on project
- W9 - NLP hands-on project
- W10 - Final exam

DL: Deep Learning
AOI: Automated Optical Inspection
RSD: Road Sign Detection
NLP: Natural Language Processing

Topics

- Topic 01: Introduction to Deep Learning (20min)
- Topic 02: Kissipo Learning for Deep Learning (20min)
- Topic 03: Python quick tutorial (20min)
- Topic 04: Numpy quick tutorial (15min)
- Topic 05: Pandas quick tutorial (15min)
- Topic 06: Scikit-learn quick tutorial (15min)
- Topic 07: OpenCV quick tutorial (15min)
- Topic 08: Image Processing basics (20min)
- Topic 09: Machine Learning basics (20min)
- Topic 10: Deep Learning basics (20min)
- Topic 11: TensorFlow overview (20min)
- Topic 12: CNN with TensorFlow (20min)
- Topic 13: RNN with TensorFlow (20min)
- Topic 14: PyTorch overview (20min)
- Topic 15: CNN with PyTorch (20min)
- Topic 16: RNN with Pytorch (20min)
- Topic 17: Introduction to AOI (20min)
- Topic 18: AOI simple Pipeline (A) (20min)
- Topic 19: AOI simple Pipeline (B) (20min)
- Topic 20: Introduction to Object detection (20min)
- Topic 21: YoloV5 Quick Tutorial (20min)
- Topic 22: Using YoloV5 for RSD (20min)
- Topic 23: Introduction to NLP (20min)
- Topic 24: Introduction to Word Embedding (20min)
- Topic 25: Name prediction project (20min)

Week 9 Topics

- Topic 23: Introduction to NLP (20min)
- Topic 24: Introduction to Word Embedding (20min)
- Topic 25: Name prediction project (20min)

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

NLP FROM SCRATCH: CLASSIFYING NAMES WITH A CHARACTER-LEVEL RNN

Author: Sean Robertson

We will be building and training a basic character-level RNN to classify words. This tutorial, along with the following two, show how to do preprocess data for NLP modeling "from scratch", in particular not using many of the convenience functions of *torchtext*, so you can see how preprocessing for NLP modeling works at a low level.

A character-level RNN reads words as a series of characters - outputting a prediction and "hidden state" at each step, feeding its previous hidden state into each next step. We take the final prediction to be the output, i.e. which class the word belongs to.

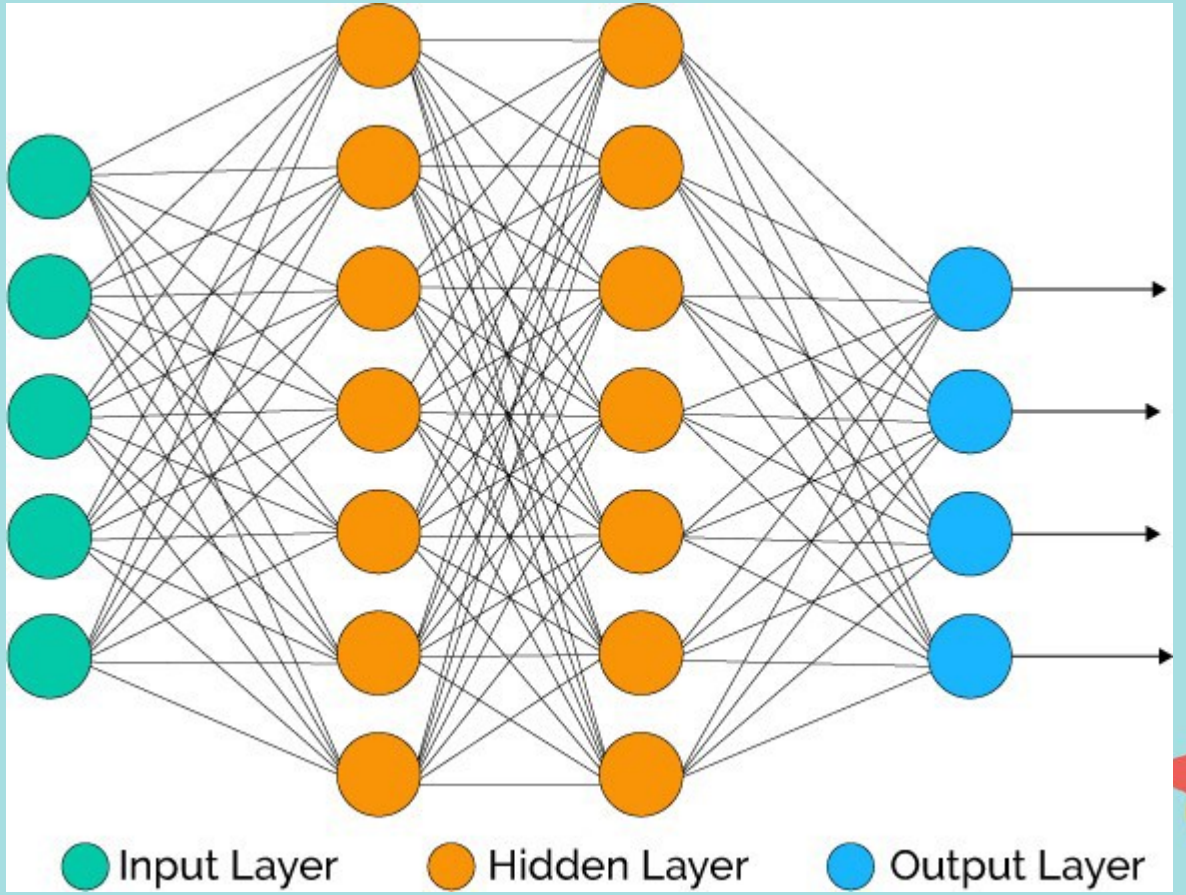
Specifically, we'll train on a few thousand surnames from 18 languages of origin, and predict which language a name is from based on the spelling:

```
$ python predict.py Hinton
(-0.47) Scottish
(-1.52) English
(-3.57) Irish

$ python predict.py Schmidhuber
(-0.19) German
(-2.48) Czech
(-2.68) Dutch
```

https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial.html

NLP with Deep Learning Models

Input:
Monica is a dog



Preparing the Data

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

def findFiles(path): return glob.glob(path)

print(findFiles('data/names/*.txt'))

import unicodedata
import string

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

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

print(unicodeToAscii('Ślusàrski'))
```

```
# Build the category_lines dictionary, a list of names per language
category_lines = {}
all_categories = []

# 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]

for filename in findFiles('data/names/*.txt'):
    category = os.path.splitext(os.path.basename(filename))[0]
    all_categories.append(category)
    lines = readLines(filename)
    category_lines[category] = lines

n_categories = len(all_categories)
```

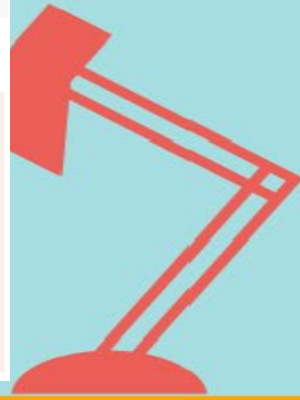
Turning Names into Tensors

- We use a “one-hot vector” of size <1 x n_letters> To represent a single letter.
- A one-hot vector is filled with 0s except for a 1 at index of the current letter, e.g. "b" = <0 1 0 0 0 …>.

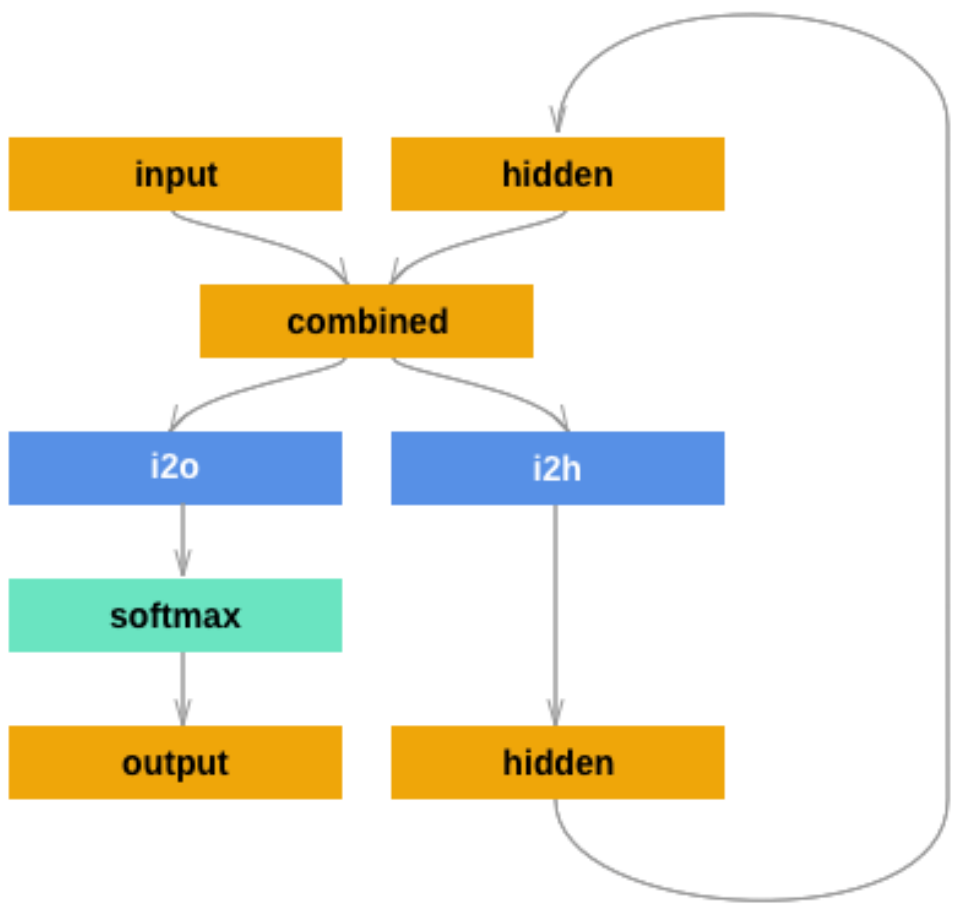
```
print(letterToTensor('J'))

print(lineToTensor('Jones').size())
```

Out:

```
tensor([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0.]])
torch.Size([5, 1, 57])
```



Creating the Network




```
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.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

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

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

Preparing for Training

```
def categoryFromOutput(output):
    top_n, top_i = output.topk(1)
    category_i = top_i[0].item()
    return all_categories[category_i], category_i

print(categoryFromOutput(output))
```

Out:

```
('Czech', 0)
```

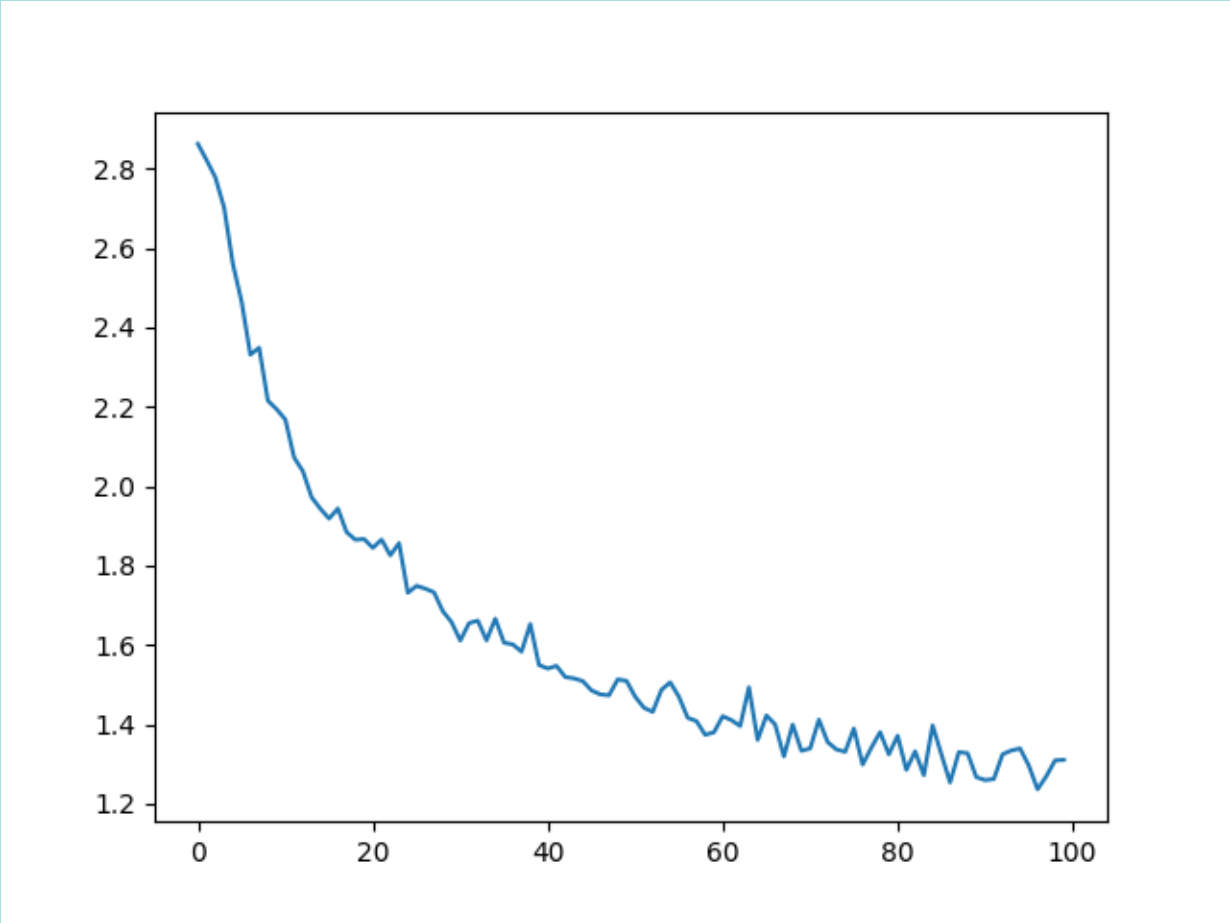
Training the Network

Each loop of training will:

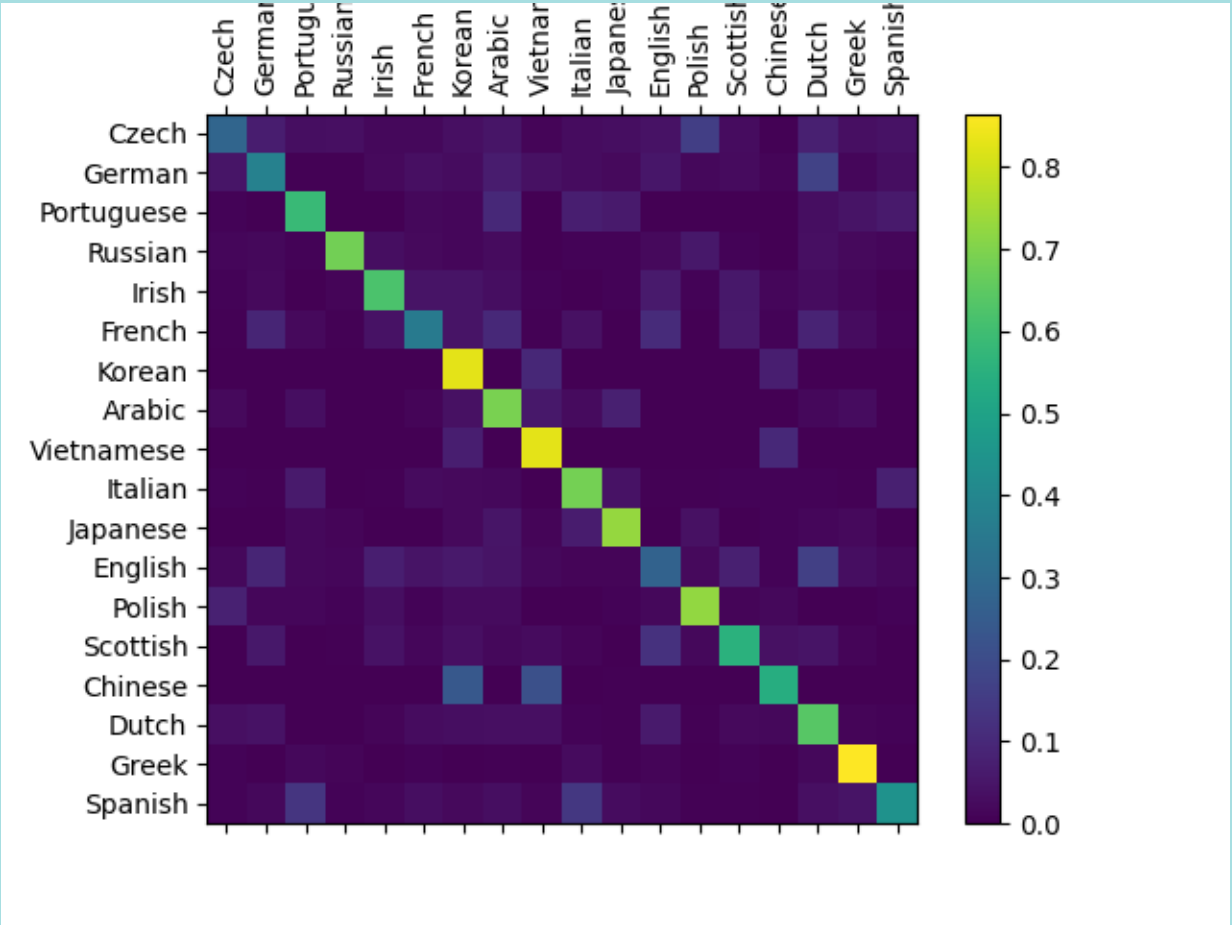
- Create input and target tensors
- Create a zeroed initial hidden state
- Read each letter in and Keep hidden state for next letter
- Compare final output to target
- Back-propagate
- Return the output and loss

```
learning_rate = 0.005 # If you set this too high, it might explo
learn

def train(category_tensor, line_tensor):
    hidden = rnn.initHidden()

    rnn.zero_grad()

    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by l
    for p in rnn.parameters():
        p.data.add_(p.grad.data, alpha=-learning_rate)

    return output, loss.item()
```

Plotting the Results



Evaluating the Results



Running on User Input

```
def predict(input_line, n_predictions=3):
    print('\n> %s' % input_line)
    with torch.no_grad():
        output = evaluate(lineToTensor(input_line))

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

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

predict('Dovesky')
predict('Jackson')
predict('Satoshi')
```

```
> Dovesky
(-0.48) Russian
(-1.32) Czech
(-2.92) English

> Jackson
(-0.39) Scottish
(-1.92) English
(-2.65) Russian

> Satoshi
(-0.76) Italian
(-1.50) Japanese
(-2.41) Arabic
```

Thanks!

Q&A