Translation with a sequence to sequence

FullName : Bouabd Chaimaa

In This notebook we will carry out translate sentences from French to English using sequence to sequence model

Basic NumPy and Pandas Initialization

```
import numpy as np
import pandas as pd
import os
```

Mounting Google Drive in Colab

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Unzipping Dataset from Google Drive

```
!unzip -q /content/drive/MyDrive/archive.zip
```

Loading a Subset of the 'en-fr.csv' Dataset (First 8 Million Rows)

```
# full dataset not taking load
df = pd.read_csv("/content/en-fr.csv", nrows=8000000)
```

Importing and Setting Up PyTorch for NLP Task

```
# import packages
from __future__ import unicode_literals, print_function, division
from io import open
import unicodedata
import string
import re
import random

import torch
import torch.nn as nn
from torch import optim
import torch.nn.functional as F

# device type "cuda" or "cpu"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

Language Processing and Text Normalization Functions

```
SOS_token = 0
EOS_token = 1

class Lang:
    def __init__(self, name):
        self.name = name
        self.word2index = {} # word → index (word2index)
        self.word2count = {} # index → word (index2word)
        self.index2word = {0: "SOS", 1: "EOS"}
        self.n_words = 2 # count SOS and EOS

    def addSentence(self, sentence):
        for word in sentence.split(' '):
            self.addWord(word)

    def addWord(self, word):
        if word not in self.word2index:
            self.word2index[word] = self.n_words
            self.word2count[word] = 1
            self.index2word[self.n_words] = word
            self.n_words += 1
        else:
            self.word2count[word] += 1

def unicodeToAscii(s):
    '''
    For each character, there are two normal forms:
    normal form C and normal form D.
    Normal form D (NFD) is also known as canonical decomposition, and
translates each character into its decomposed form.
    Normal form C (NFC) first applies a canonical decomposition, then
composes pre-combined characters again.
    '''
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
    )

# Lowercase, trim, and remove non-letter characters
def normalizeString(s):
    s = unicodeToAscii(s.lower().strip())
    s = re.sub(r"([.!?])", r" \1", s)
    s = re.sub(r"[^a-zA-Z.!?]+", r" ", s)
    return s
```

Setting Maximum Length and Hidden Size for Sequence Translation Model

```
MAX_LENGTH = 30 # max 10 words including ending punctuation
hidden_size = 256
```

Reading Language Pairs and Filtering Sentences

```
def readLang(lang1, lang2, reverse=False):
    print("Reading lines...")
    # split into two lines
    lines = df.values.tolist()
    # select everyline into pairs and normalize
    pairs = [
        [normalizeString(str(s)) for s in l] for l in lines
    ]
    # Reverse pairs, make Lang instances
    if reverse:
        pairs = [list(reversed(p)) for p in pairs]
        input_lang = Lang(lang2)
        output_lang = Lang(lang1)
    else:
        input_lang = Lang(lang1)
        output_lang = Lang(lang2)
    return input_lang, output_lang, pairs

# filtering to sentences that translate to the form “I am” or “He is”
etc.
eng_prefixes = (
    "i am ", "i m ",
    "he is", "he s ",
    "she is", "she s ",
    "you are", "you re ",
    "we are", "we re ",
    "they are", "they re "
)

def filterPair(p):
    return len(p[0].split(' ')) < MAX_LENGTH and len(p[1].split(' '))
< MAX_LENGTH and p[1].startswith(eng_prefixes)

def filterPairs(pairs):
    return [pair for pair in pairs if filterPair(pair)]
```

The full process for preparing the data is:

- Read text file and split into lines, split lines into pairs
- Normalize text, filter by length and content
- Make word lists from sentences in pairs

Preparing Data for Sequence-to-Sequence Model

```python
def prepareData(lang1, lang2, reverse=False):
    input_lang, output_lang, pairs = readLang(lang1, lang2, reverse)
    print(f"Read sentence pairs: {len(pairs)}")
    pairs = filterPairs(pairs)
    print(f"Trimmed to sentence pairs: {len(pairs)}")
    print(f"COUNTING WORDS...")
    for pair in pairs:
        input_lang.addSentence(pair[0])
        output_lang.addSentence(pair[1])
    print("Counted Words...")
    print(input_lang.name, input_lang.n_words)
    print(output_lang.name, output_lang.n_words)
    return input_lang, output_lang, pairs

input_lang, output_lang, pairs = prepareData(df.columns.tolist()[0],
df.columns.tolist()[1], True)
print(random.choice(pairs)) # random choice of pairs
```

```
Reading lines...
Read sentence pairs: 8000000
Trimmed to sentence pairs: 16663
COUNTING WORDS...
Counted Words...
fr 18781
en 14262
['elle vit actuellement a brantford ontario et travaille dans
plusieurs disciplines artistiques dont la broderie perlee la peinture
la photographie et le film .', 'she is currently based in brantford
ontario and works in a variety of media including beadwork painting
photography and film .']
```

Seq2Seq Model

A Sequence to Sequence network, or seq2seq network, or Encoder Decoder network, is a model consisting of two RNNs called the encoder and decoder. The encoder reads an input sequence and outputs a single vector, and the decoder reads that vector to produce an output sequence.

The Encoder

The encoder of a seq2seq network is a RNN that outputs some value for every word from the input sentence. For every input word the encoder outputs a vector and a hidden state, and uses the hidden state for the next input word.

The Decoder

In the simplest seq2seq decoder we use only last output of the encoder. This last output is sometimes called the context vector as it encodes context from the entire sequence. This context vector is used as the initial hidden state of the decoder.

At every step of decoding, the decoder is given an input token and hidden state. The initial input token is the start-of-string SOS token, and the first hidden state is the context vector (the encoder's last hidden state).

Encoder and Decoder Models for Sequence-to-Sequence Translation

```
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)

    def forward(self, input, hidden):
        embedding = self.embedding(input).view(1, 1, -1)
        output = embedding
        output, hidden = self.gru(output, hidden)
        return output, hidden

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


class DecoderRNN(nn.Module):
    """docstring for DecoderRNN"""
    def __init__(self, hidden_size, output_size):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.gru = nn.GRU(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, hidden = self.gru(output, hidden)
        output = self.softmax(self.out(output[0]))
        return output, hidden

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

Functions to Convert Sentences to Tensors

```
def indexesFromSentence(lang, sentence):
    return [lang.word2index[word] for word in sentence.split(' ')]

def tensorFromSentence(lang, sentence):
    indexes = indexesFromSentence(lang, sentence)
    indexes.append(EOS_token)
    return torch.tensor(indexes, dtype=torch.long,
device=device).view(-1, 1)

def tensorFromPair(pair):
    input_tensor = tensorFromSentence(input_lang, pair[0])
    target_tensor = tensorFromSentence(output_lang, pair[1])
    return (input_tensor, target_tensor)
```

Functions for Preparing Training Data

```
# Training
# Preparing Training Data
# To train, for each pair we will need an input tensor (indexes of the
words in the input sentence) and target tensor (indexes of the words
in the target sentence).
# While creating these vectors we will append the EOS token to both
sequences.

def indexesFromSentence(lang, sentence):
    return [lang.word2index[word] for word in sentence.split(' ')]

def tensorFromSentence(lang, sentence):
    indexes = indexesFromSentence(lang, sentence)
    indexes.append(EOS_token)
    return torch.tensor(indexes, dtype=torch.long,
device=device).view(-1, 1)

def tensorFromPair(pair):
    input_tensor = tensorFromSentence(input_lang, pair[0])
    target_tensor = tensorFromSentence(output_lang, pair[1])
    return (input_tensor, target_tensor)
```

Training Function for Sequence-to-Sequence Model

```
teacher_forcing_ratio = 0.5

def train(input_tensor, target_tensor, encoder, decoder,
encoder_optimizer, decoder_optimizer, criterion,
max_length=MAX_LENGTH):
    encoder_hidden = encoder.initHidden()
    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = input_tensor.size(0)
    target_length = target_tensor.size(0)

    encoder_outputs = torch.zeros(max_length, encoder.hidden_size,
device=device)

    loss = 0
    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden
        )
        encoder_outputs[ei] = encoder_output[0, 0]

    decoder_input = torch.tensor([[SOS_token]], device=device)
    decoder_hidden = encoder_hidden
    use_teacher_forcing = True if random.random() <
teacher_forcing_ratio else False

    if use_teacher_forcing:
        # Teacher forcing: Feed the target as the next input
        for di in range(target_length):
            decoder_output, decoder_hidden = decoder(
                decoder_input, decoder_hidden
            )
            loss += criterion(decoder_output, target_tensor[di])
            decoder_input = target_tensor[di] # Teacher forcing
    else:
        # Without teacher forcing: use its own predictions as the next
input
        for di in range(target_length):
            decoder_output, decoder_hidden = decoder(
                decoder_input, decoder_hidden
            )
            topv, topi = decoder_output.topk(1)
            decoder_input = topi.squeeze().detach() # detach from
history as input
            loss += criterion(decoder_output, target_tensor[di])
            if decoder_input.item() == EOS_token:
                break
    loss.backward()
    encoder_optimizer.step()
    decoder_optimizer.step()
    return loss.item() / target_length
```

Utility Functions for Time Formatting

```
import time
import math

def asMinutes(s):
    m = math.floor(s / 60)
    s -= m * 60
    return '%dm %ds' % (m, s)

def timeSince(since, percent):
    now = time.time()
    s = now - since
    es = s / (percent)
    rs = es - s
    return '%s (- %s)' % (asMinutes(s), asMinutes(s))
```

The whole training process looks like this:

- Start a timer

- Initialize optimizers and criterion
- Create set of training pairs
- Start empty losses array for plotting

Function for Plotting Training Loss

```
def showplot(points):
    plt.figure()
    fig, ax = plt.subplots()
    loc = ticker.MultipleLocator(base=0.2)
    ax.yaxis.set_major_locator(loc)
    plt.plot(points)
    plt.show()
```

Function for Training the Sequence-to-Sequence Model Iteratively

```
import matplotlib.pyplot as plt
plt.switch_backend('agg')
import matplotlib.ticker as ticker

def trainIters(encoder, decoder, n_iters, print_every=1000,
plot_every=100, learning_rate=0.01):
    start = time.time()
    plot_losses = []
    print_loss_total = 0 # Reset every print_every
    plot_loss_total = 0 # Reset every plot_every

    encoder_optimizer = optim.SGD(encoder.parameters(),
lr=learning_rate)
    decoder_optimizer = optim.SGD(decoder.parameters(),
lr=learning_rate)

    training_pairs = [tensorFromPair(random.choice(pairs)) for i in
range(n_iters)]
    criterion = nn.NLLLoss()

    for iter in range(1, n_iters + 1):
        training_pair = training_pairs[iter - 1]
        input_tensor = training_pair[0]
        target_tensor = training_pair[1]

        loss = train(
            input_tensor, target_tensor,
            encoder, decoder,
            encoder_optimizer, decoder_optimizer,
            criterion
        )

        print_loss_total += loss
        plot_loss_total += loss
        if iter % print_every == 0:
            print_loss_avg = print_loss_total / print_every
            print_loss_total = 0
            print('%s (%d %d%%) %.4f' % (timeSince(start, iter /
n_iters),
                                         iter, iter / n_iters * 100,
print_loss_avg))

        if iter % plot_every == 0:
            plot_loss_avg = plot_loss_total / plot_every
            plot_losses.append(plot_loss_avg)
            plot_loss_total = 0
    showplot(plot_losses)
```

Evaluation is mostly the same as training, but there are no targets so we simply feed the decoder's predictions back to itself for each step. Every time it predicts a word we add it to the output string, and if it predicts the EOS token we stop there.

Training a Sequence-to-Sequence Model for Language Translation

```
def evaluate(encoder, decoder, sentence, max_length=MAX_LENGTH):
    with torch.no_grad():
        input_tensor = tensorFromSentence(input_lang, sentence)
        input_length = input_tensor.size()[0]
        encoder_hidden = encoder.initHidden()

        encoder_outputs = torch.zeros(max_length, encoder.hidden_size,
device=device)

        for ei in range(input_length):
            encoder_output, encoder_hidden = encoder(input_tensor[ei],
                                                     encoder_hidden)
            encoder_outputs[ei] += encoder_output[0, 0]

        decoder_input = torch.tensor([[SOS_token]], device=device)  #
SOS

        decoder_hidden = encoder_hidden

        decoded_words = []

        for di in range(max_length):
            decoder_output, decoder_hidden = decoder(
                decoder_input, decoder_hidden)
            topv, topi = decoder_output.data.topk(1)
            if topi.item() == EOS_token:
                decoded_words.append('<EOS>')
                break
            else:

decoded_words.append(output_lang.index2word[topi.item()])

            decoder_input = topi.squeeze().detach()

        return decoded_words

encoder1 = EncoderRNN(input_lang.n_words, hidden_size).to(device)
decoder1 = DecoderRNN(hidden_size, output_lang.n_words).to(device)

trainIters(encoder1, decoder1, 75000, print_every=5000)
```

def evaluateRandomly(encoder, decoder, n=10): for i in range(n): pair = random.choice(pairs) print('>', pair[0]) print('=', pair[1]) output_words = evaluate(encoder, decoder, pair[0]) output_sentence = ' '.join(output_words) print('<', output_sentence) print('') evaluateRandomly(encoder1, decoder1) def evaluateRandomly(encoder, decoder, n=10): for i in range(n): pair = random.choice(pairs) print('>', pair[0]) print('=', pair[1]) output_words = evaluate(encoder, decoder, pair[0]) output_sentence = ' '.join(output_words) print('<', output_sentence) print('') evaluateRandomly(encoder1, decoder1) def evaluateRandomly(encoder, decoder, n=10): for i in range(n): pair = random.choice(pairs) print('>', pair[0]) print('=', pair[1]) output_words = evaluate(encoder, decoder, pair[0]) output_sentence = ' '.join(output_words) print('<', output_sentence) print('') evaluateRandomly(encoder1, decoder1) def evaluateRandomly(encoder, decoder, n=10): for i in range(n): pair = random.choice(pairs) print('>', pair[0]) print('=', pair[1]) output_words = evaluate(encoder, decoder, pair[0]) output_sentence = ' '.join(output_words) print('<', output_sentence) print('') evaluateRandomly(encoder1, decoder1) def evaluateRandomly(encoder, decoder, n=10): for i in range(n): pair = random.choice(pairs) print('>', pair[0]) print('=', pair[1]) output_words = evaluate(encoder, decoder, pair[0]) output_sentence = ' '.join(output_words) print('<', output_sentence) print('') evaluateRandomly(encoder1, decoder1)

Evaluating Sequence-to-Sequence Model on Random Examples

```
def evaluateRandomly(encoder, decoder, n=10):
    for i in range(n):
        pair = random.choice(pairs)
        print('>', pair[0])
        print('=', pair[1])
        output_words = evaluate(encoder, decoder, pair[0])
        output_sentence = ' '.join(output_words)
        print('<', output_sentence)
        print('')
evaluateRandomly(encoder1, decoder1)
```

```
> en outre il jouit d une vaste reputation en tant que communicateur
sur les questions complexes de politique publique .
= he is also widely regarded as a gifted communicator on complex
public policy questions .
< he is also a to a of and and as as and as . and . . <EOS>

> ils sont apres tout des proprietaires de droits et sont interesses a
encourager le respect des regles .
= they are rights owners after all and are interested in encouraging
respect for the rules .
< they are designed to the and and and and are rights . <EOS>

> je suis heureuse de vous annoncer que le ministere du patrimoine
canadien appuiera la conference inaugurale du reseau des villes
creatives .
= i am pleased to tell you that canadian heritage will support the
founding conference of the creative cities network .
< i am pleased to present the canadian you the to the the the the .
<EOS>

> tout le monde aime travailler avec lui et il arrive a interesser
tout le monde .
= he s somebody that everyone likes to work with and is able to get
everyone interested .
< he is always he to he and to to and and and he to to the and to
<EOS>

> il est desormais secretaire d etat amerique latine et afrique
francophonie .
= he is now secretary of state latin america and africa francophonie .
< he is now the secretary of the american and of the the and . <EOS>

> je constate une baisse dans leur estime de soi .
= i m noticing a lowering in their self esteem .
< i m still from a i i am from i <EOS>

> j ai le plaisir d annoncer que le canada va lancer un nouveau
programme de main d uvre pour les ameriques qui soutiendra la colombie
.
= i am pleased to announce that canada will launch a new labour
program for the americas that will include support for colombia .
< i m pleased to announce that that canada will continue to be a
program for a new program . <EOS>

> l evaluation est faite par la commission de la fonction publique qui
peut deleguer ce pouvoir aux organisations .
= they are assessed by public service commission which can delegate
this power to organizations .
< they are available by the commission of the commission commission by
the commission . <EOS>
```

```
> de tels organismes vous font connaitre differentes strategies
financieres a adopter pour eviter d eventuelles difficultes .
= they are there to help you with financial strategies to avoid
trouble in the future .
< you are looking to make the to and and and and <EOS>

> ils font aussi un effort pour reduire la tension dans d autres
secteurs du systeme de sante tels que dans les salles d urgence .
= they are also making efforts to relieve pressures found elsewhere in
the health system such as emergency rooms .
< they are also to to of health health such as as as as as . <EOS>
```

Translating Text Using Trained Sequence-to-Sequence Model

```
# Test
def translateText(input_text):
    output_words = evaluate(
        encoder1, decoder1, normalizeString(input_text)
    )
    output_sentence = ' '.join(output_words)
    return f"Output Sentence: {output_sentence}"

translateText(input_text=input("Type sentence: "))
```

```
Type sentence:  ils font aussi un effort pour reduire la tension dans
d autres secteurs

'Output Sentence: they are also a in a of the in the . <EOS>'
```