TRANSLATION WITH A SEQUENCE TO SEQUENCE NETWORK AND ATTENTION

- Translate from French to English

- Sequence to sequence network

  : two RNN work together to transform one sequence to another.

  - Encoder: condences an input sequence into a vector
  - Decoder : unfolds the vector into a new sequence

Requirements

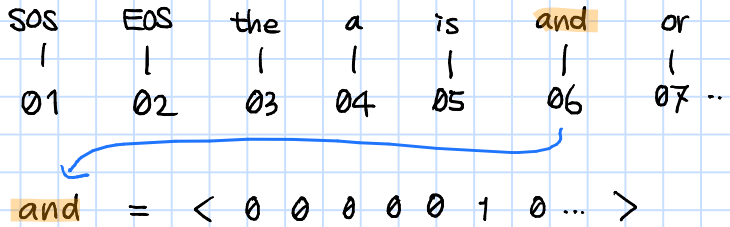
```
from __future__ import unicode_literals, print_function, division
from io import open
import unicodedata
import string
import re
import random

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

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

Loading data files

- English to French pairs
- unique index per word & one-hot vector



- class Lang: helper class
  - word → index (word2index)
  - index → word (index2word)
  - count of each word (word2count)

```
SOS_token = 0
EOS_token = 1

class Lang:
    def __init__(self, name):
        self.name = name
        self.word2index = {}
        self.word2count = {}
        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
```

To simplify:

- turn Unicode characters to ASCII
- make everything lowercase
- trim most punctuation

```
def unicodeToAscii(s):
    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
```

To read the data file :

- split the file into lines
  ↓
- split lines into pairs

To translate from other language → English : reverse flag

```
def readLangs(lang1, lang2, reverse=False):
    print("Reading lines...")

    # Read the file and split into lines
    lines = open('data/%s-%s.txt' % (lang1, lang2), encoding='utf-8').\
        read().strip().split('\n')

    # split every line into pairs and normalize
    pairs = [[normalizeString(s) for s in l.split('\t')] 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
```

To train quickly :

- trim the dataset to only relatively short & simple sentences.
- maximum length : 10 words (includes ending punctuation)
- filtering to sentences that translate to the form "I am" or "He is" etc.

```
MAX_LENGTH = 10

eng_prefixes = (
    "i am ", " i am", "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_LENGH and \
        p[1].startswith(eng_prefixes)

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

Full process for preparing the data is:

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