test preprocessing code adopted from https://pytorch.org/tutorials/intermediate/seq2seq_translat (https://pytorch.org/tutorials/intermediate/seq2seq_transla

test process and train

In [148]:
```
import unicodedata
import string
import re
import random
import time
import math

import torch
import torch.nn as nn
from torch.autograd import Variable
from torch import optim
import torch.nn.functional as F
```

In [149]:
```
USE_CUDA = False
```

In [150]:
```
PAD_token = 0
SOS_token = 1
EOS_token = 2

class Lang:
    def __init__(self, name):
        self.name = name
        self.word2index = {}
        self.word2count = {}
        self.index2word = {PAD_token: "PAD", SOS_token: "SOS", EOS_token:
        self.n_words = 3 # Count SOS and EOS

    def index_words(self, sentence):
        for word in sentence.split(' '):
            self.index_word(word)

    def index_word(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
```

In [151]:
```
# Turn a Unicode string to plain ASCII, thanks to http://stackoverflow.co
def unicode_to_ascii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
    )

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

In [152]:
```
def read_langs(lang1, lang2, reverse=False):
    print("Reading lines...")

    pairs = []
    line1 = open('data/train.de').read().strip().split('\n')
    line2 = open('data/train.en').read().strip().split('\n')  #.splitline

    for i in range(len(line1)):

        pairs.append([normalize_string(line1[i]), normalize_string(line2[

    # 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
```

In [153]:
```
def prepare_data(lang1_name, lang2_name, reverse=False):
    input_lang, output_lang, pairs = read_langs(lang1_name, lang2_name, r
    print("Read %s sentence pairs" % len(pairs))



    print("Indexing words...")
    for pair in pairs:
        input_lang.index_words(pair[0])
        output_lang.index_words(pair[1])

    return input_lang, output_lang, pairs

input_lang, output_lang, pairs = prepare_data('ger', 'en')

# Print an example pair
print(random.choice(pairs))
```

```
Reading lines...
Read 196884 sentence pairs
Indexing words...
['und der grund warum ich das machen könnte ist weil säugetiere eine r
eihe dieser schwefelwasserstoff ereignisse durchgemacht haben und unse
re körper sich angepasst haben .', 'and the reason i could do that is
because we mammals have gone through a series of these hydrogen sulfid
e events , and our bodies have adapted .']
```

In [154]:
```
print(pairs[13])
```

```
['das meiste ist unerforscht , und doch gibt es schönheiten wie diese
, die uns fesseln und uns vertrauter mit ihm machen .', "it 's mostly
unexplored , and yet there are beautiful sights like this that captiva
te us and make us become familiar with it ."]
```

In [155]:
```
# Return a list of indexes, one for each word in the sentence
def indexes_from_sentence(lang, sentence):
    return [lang.word2index[word] for word in sentence.split(' ')]

def variable_from_sentence(lang, sentence):
    indexes = indexes_from_sentence(lang, sentence)
    indexes.append(EOS_token)
    var = Variable(torch.LongTensor(indexes).view(-1, 1))
    if USE_CUDA: var = var.cuda()
    return var

def variables_from_pair(pair):
    input_variable = variable_from_sentence(input_lang, pair[0])
    target_variable = variable_from_sentence(output_lang, pair[1])
    return (input_variable, target_variable)
```

In [156]:
```
def find_max_len(pair):
    result = 0
    target = ""
    for sents in pair:
        for item in sents:
            if len(item) > result:
                result = len(item)
                target = item
#     print("longest: ", target)
    return result
```

In [157]:
```
def paddingSOS(vector, max_len):
    vector = [SOS_token]+vector
    while len(vector)< max_len:
        vector.append(PAD_token)
    return vector
```

In [158]:
```
def paddingEOS(vector, max_len):
    vector = vector + [EOS_token]
    while len(vector)< max_len:
        vector.append(PAD_token)
    return vector
```

In [159]:
```

print('input_lang 0: ', input_lang.index2word[0])
print('input_lang 1: ', input_lang.index2word[1])
print('input_lang 2: ', input_lang.index2word[2])
```

```
input_lang 0:  PAD
input_lang 1:  SOS
input_lang 2:  EOS
```

In [ ]:

In [160]:
```
max_len = find_max_len(pairs)+2
print('max_len: ', max_len)
```

```
max_len:  3014
```

In [161]:
```
print('input: ', input_lang.name)
print('output: ', output_lang.name)
```

```
input:  ger
output:  en
```

data store pair sentences converted to indexes

In [164]:
```
def pair_to_indexes(pairs, max_len):
    result = []
    for pair in pairs:
        # add start token for english
        sent2 = paddingSOS(indexes_from_sentence(output_lang, pair[1]), m
        # add end token for german
        sent1 = paddingEOS(indexes_from_sentence(input_lang, pair[0]), ma
        result.append([sent1, sent2])

    return result
```

In [165]:
```
data = pair_to_indexes(pairs, max_len)
```

In [172]:
```
print(pairs[55])
print(data[55])
```

```
['sie sehen sie sich da unten in dieser welt an .', "they 're actuall
y looking at them down in that world ."]
[[43, 83, 43, 91, 348, 320, 20, 280, 142, 117, 9, 2, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
```

In [ ]: