lab11

April 8, 2021

1 LSTM Lab

To be done on your own: find another dataset of sentence pairs in a different domain and see if you can preprocess the data and train a chatbot model on it using the same code we developed today. Report your results.

[1]:
```
import torch
from torch.jit import script, trace
import torch.nn as nn
from torch import optim
import torch.nn.functional as F
import csv
import random
import re
import os
import unicodedata
import codecs
from io import open
import itertools
import math
import pickle
from torch.autograd import Variable
from tqdm.contrib.discord import tqdm, trange

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

[2]:
```
# Reserved word tokens

PAD_token = 0  # Used for padding short sentences
SOS_token = 1  # Start-of-sentence token
EOS_token = 2  # End-of-sentence token

class Voc:

    def __init__(self):
        self.trimmed = False
```

```
        self.word2index = {}
        self.word2count = {}
        self.index2word = {PAD_token: "PAD", SOS_token: "SOS", EOS_token: "EOS"}
        self.num_words = 3  # Count SOS, EOS, PAD

    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.num_words
            self.word2count[word] = 1
            self.index2word[self.num_words] = word
            self.num_words += 1
        else:
            self.word2count[word] += 1

    # Remove words below a certain count threshold

    def trim(self, min_count):
        if self.trimmed:
            return
        self.trimmed = True

        keep_words = []

        for k, v in self.word2count.items():
            if v >= min_count:
                keep_words.append(k)

        print('keep_words {} / {} = {:.4f}'.format(
            len(keep_words), len(self.word2index), len(keep_words) / len(self.
↪word2index)
        ))

        # Reinitialize dictionaries

        self.word2index = {}
        self.word2count = {}
        self.index2word = {PAD_token: "PAD", SOS_token: "SOS", EOS_token: "EOS"}
        self.num_words = 3 # Count default tokens

        for word in keep_words:
            self.addWord(word)
```

[3]:
```
MAX_LENGTH = 10  # Maximum sentence length to consider

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)
    s = re.sub(r"\s+", r" ", s).strip()
    return s

# Read query/response pairs and return a Voc object

def readVocs(datafile):
    print("Reading lines...")
    # Read the file and split into lines
    lines = open(datafile, 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]
    voc = Voc()
    return voc, pairs

# Boolean function returning True iff both sentences in a pair 'p' are under␣
 ↪the MAX_LENGTH threshold

def filterPair(p):
    # Input sequences need to preserve the last word for EOS token
    return len(p[0].split(' ')) < MAX_LENGTH and len(p[1].split(' ')) <␣
 ↪MAX_LENGTH

# Filter pairs using the filterPair predicate

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

# Using the functions defined above, return a populated voc object and pairs␣
 ↪list

def loadPrepareData(datafile):
    print("Start preparing training data ...")
```

```
    voc, pairs = readVocs(datafile)
    print("Read {!s} sentence pairs".format(len(pairs)))
    pairs = filterPairs(pairs)
    print("Trimmed to {!s} sentence pairs".format(len(pairs)))
    print("Counting words...")
    for pair in pairs:
        voc.addSentence(pair[0])
        voc.addSentence(pair[1])
    print("Counted words:", voc.num_words)
    return voc, pairs
```

[4]:
```
# wget https://github.com/dsai-asia/RTML/raw/main/Labs/11-LSTMs/chatDataset.txt

# Load/Assemble Voc and pairs

datafile = './data/formatted_movie_lines.txt'
voc, pairs = loadPrepareData(datafile)

# Print some pairs to validate

print("\npairs:")
for pair in pairs[:10]:
    print(pair)
```

```
Start preparing training data …
Reading lines…
Read 221282 sentence pairs
Trimmed to 64271 sentence pairs
Counting words…
Counted words: 18008

pairs:
['there .', 'where ?']
['you have my word . as a gentleman', 'you re sweet .']
['hi .', 'looks like things worked out tonight huh ?']
['you know chastity ?', 'i believe we share an art instructor']
['have fun tonight ?', 'tons']
['well no . . .', 'then that s all you had to say .']
['then that s all you had to say .', 'but']
['but', 'you always been this selfish ?']
['do you listen to this crap ?', 'what crap ?']
['what good stuff ?', 'the real you .']
```

[5]:
```
MIN_COUNT = 3    # Minimum word count threshold for trimming

def trimRareWords(voc, pairs, MIN_COUNT):
    # Trim words used under the MIN_COUNT from the voc
```

```
    voc.trim(MIN_COUNT)
    # Filter out pairs with trimmed words
    keep_pairs = []
    for pair in pairs:
        input_sentence = pair[0]
        output_sentence = pair[1]
        keep_input = True
        keep_output = True
        # Check input sentence
        for word in input_sentence.split(' '):
            if word not in voc.word2index:
                keep_input = False
                break
        # Check output sentence
        for word in output_sentence.split(' '):
            if word not in voc.word2index:
                keep_output = False
                break

        # Only keep pairs that do not contain trimmed word(s) in their input or␣
↪output sentence
        if keep_input and keep_output:
            keep_pairs.append(pair)

    print("Trimmed from {} pairs to {}, {:.4f} of total".format(len(pairs),␣
↪len(keep_pairs), len(keep_pairs) / len(pairs)))
    return keep_pairs

# Trim vocabulary and pairs

pairs = trimRareWords(voc, pairs, MIN_COUNT)
```

```
keep_words 7823 / 18005 = 0.4345
Trimmed from 64271 pairs to 53165, 0.8272 of total
```

[6]:
```
print(pairs)
```

```
['yeah .', 'what have you got ?'], ['so one of them got to him .', 'maybe .'],
['maybe .', 'you know who ?'], ['you know who ?', 'maybe .'], ['who else knows
?', 'just us .'], ['who else knows ?', 'just us .'], ['it s all over paul .',
'move ! or you die right here !'], ['they kid a lot .', 'i would not be too sure
.'], ['where are you taking us now ?', 'home .'], ['home .', 'you couldn t wait
until morning ?'], ['you said we would be safe in philadelphia .', 'i was wrong
.'], ['no no doctor . . .', 'but why ?'], ['does anybody know i m here ?', 'only
the elders .'], ['only the elders .', 'how long ?'], ['how long ?', 'what ?'],
['what ?', 'how long have i been here ?'], ['how long have i been here ?', 'two
days .'], ['tell him his tea stinks .', 'you tell him . when you re able .'],
```

['something wrong with buttons ?', 'buttons are hochmut .'], ['buttons are hochmut .', 'hochmut ?'], ['the . . . bullets ?', 'oh . the bullets .'], ['when will you be going ?', 'not long . . . a few days .'], ['i know .', 'i was being foolish ?'], ['you re so sure of that ?', 'aren t you ? after today ?'], ['that s your way not mine .', 'that s god s way !'], ['anybody know about this ?', 'i didn t even know about it .'], ['it s me .', 'johnny ! where the hell have you been ?'], ['where i m at is maybe .', 'say again ?'], ['say again ?', 'make that .'], ['you know where he is .', 'wrong .'], ['wrong .', 'you d lie to protect him .'], ['you d lie to protect him .', 'probably .'], ['you re the first one he ll contact .', 'he s got my number .'], ['did you find him ?', 'not yet .'], ['everything okay ?', 'yes thank you very much .'], ['yes thank you very much .', 'john said you re amish .'], ['john said you re amish .', 'yes .'], ['yes .', 'oh .'], ['good morning .', 'you didn t have to . . .'], ['who was that man ?', 'his name is john book .'], ['is the english dead ?', 'no . . .'], ['no . . .', 'looks dead . . .'], ['that has no place in this house .', 'i know .'], ['i am not a child .', 'you are acting like one !'], ['you are acting like one !', 'i will be the judge of that .'], ['no rachel . . .', 'i have to help him !'], ['she say where he is ?', 'i don t think she knows .'], ['what about carter ?', 'tight . but i m working on him .'], ['tight . but i m working on him .', 'lean on him .'], ['he s not in this building .', 'all right where is he ?'], ['he ll live .', 'you might have killed him !'], ['get back in there .', 'my son is out there !'], ['i don t want to stay here .', 'they are english . they don t understand .'], ['aunt em !', 'fifty seven fifty eight'], ['dorothy ! dorothy ! we re busy !', 'oh all right .'], ['yes .', 'oh i hope we got them in time .'], ['oh i hope we got them in time .', 'yes .'], ['to see ?', 'if she . . .'], ['if she . . .', 'if she ?'], ['. . . oz has spoken !', 'who are you ?'], ['oh', 'goodbye folks !'], ['oh !', 'who did it ? now wait a minute .'], ['run toto run !', 'catch him you fool !'], ['toto too .', 'oh now ?'], ['oh now ?', 'whenever you wish .'], ['are you ready now ?', 'yes . say goodbye toto .'], ['aw come come come', 'no they won t honestly .'], ['no they won t honestly .', 'oh'], ['that s our farm !', 'oh yes .'], ['yes . . .that s aunt em .', 'her her name is emily .'], ['uh huh .', 'what s she doing now ?'], ['oh no no !', 'that s all the crystal s gone dark .'], ['that s all the crystal s gone dark .', 'oh you . . . .'], ['oh', 'but you couldn t have been could you ?'], ['now you re seeing reason .', 'no'], ['did you say something ?', 'oil can . . . .'], ['oh', '. . .oh did that hurt ?'], ['oh', 'all hollow . oh'], ['oh i ll get it !', 'oh ! oh !'], ['we might .', 'oh'], ['and bears !', 'what sort of an animal is that ?'], ['no . why only oh', 'oh ! oh tin man ! oh !'], ['oh ! something bit me too !', 'now come on you re acting silly'], ['dorothy !', 'i knew you would !'], ['oh my !', 'lions and tigers and bears !'], ['lions and tigers and bears !', 'oh my !'], ['oh my !', 'lions and tigers and bears !'], ['lions and tigers and bears !', 'oh my !'], ['oh my !', 'lions and tigers and bears !'], ['lions and tigers and bears !', 'oh my !'], ['oh my !', 'lions and tigers and bears !'], ['lions and tigers and bears !', 'oh my !'], ['a home', 'the nerve .'], ['yes let s run !', 'yes .'], ['ho ho ho ho', 'hah !'], ['i d sooner wait outside .', 'but why ? why ?'], ['but why ? why ?', 'because i m still scared !'], ['because i m still scared !', 'oh come on .'], ['oh come on .', 'ohh !'], ['oh oh come on .', 'huh ? what d he say ?'], ['does it work ?', 'no but it s wonderful for threatening

with .'], ['no but it s wonderful for threatening with .', 'oh'], ['they re coming back !', 'ohhh !'], ['. . .don t talk .', 'it s pleasant down that way too .'], ['. . .yes .', 'ohhhh'], ['ohhhh', 'ohh !'], ['oh ! oh ! ohhh !', 'did i scare you ?'], ['they would ?', 'um hmm .'], ['um hmm .', 'where s kansas ?'], ['oh i ll try ! really i will .', 'to oz ?'], ['to oz ?', 'to oz !'], ['what ?', 'yes . oh look'], ['oil can what ?', 'oil can ? oh oh here it is !'], ['he said his mouth .', 'here here'], ['here here', 'the other side . . . .'], ['the other side . . . .', 'yes there .'], ['oh oh', 'oh are you are you all right ?'], ['oh come on now everybody', 'did you just hear what i just heard ?'], ['no no no no !', 'ohh ! ohh ! ohh !'], ['yeah .', 'ah'], ['ah', 'dorothy next !'], ['who at ?', 'i don t know .'], ['what happened ?', 'somebody pulled my tail .'], ['somebody pulled my tail .', 'oh you did it yourself !'], ['oh you did it yourself !', 'i oh'], ['i oh', 'here come on .'], ['here come on .', 'what was that ?'], ['fine . he s got a plan', 'and you re going to lead us .'], ['and you re going to lead us .', 'yeah . me ?'], ['yeah . me ?', 'yes you .'], ['up !', 'now . . .'], ['where where do we go now ?', 'this way ! come on !'], ['i haven t slept in weeks .', 'why don t you try counting sheep ?'], ['it s a whatzis .', 'it s a whatzis ?'], ['it s a whatzis ?', 'whozat ?'], ['whozat ?', 'whozat ?'], ['no ? now wait a minute .', 'you don t neither'], ['where do we go now ?', 'yeah .'], ['. . . about dorothy .', 'dorothy ? well what has dorothy done ?'], ['you mean she bit you ?', 'no her dog !'], ['no her dog !', 'oh she bit her dog eh ?'], ['oh she bit her dog eh ?', 'no !'], ['no sir !', 'no sir !'], ['that s right .', 'we do .'], ['we do .', 'to oz ?'], ['to oz ?', 'to oz !'], ['animals that that eat straw ?', 'some but mostly lions and tigers and bears .'], ['then i m sure to get a brain', 'a heart'], ['come on come on', 'hurry hurry'], ['hey dorothy !', 'dorothy !'], ['oh oh poor dorothy !', 'don t cry you ll rust yourself again !'], ['let s .', 'yes .'], ['ha ha ha', 'ho ho ho'], ['oh did did you see that ?', 'oh look out .'], ['oh look out .', 'you know something ?'], ['now now don t fret .', 'oh dear dear .'], ['oh dear dear .', 'we ll get you together !'], ['. . .to dorothy !', 'oh'], ['oh', 'come on fellows !'], ['no you don t .', 'oh no !'], ['you put up a great fight lion .', 'yeah'], ['oh upstairs quickly !', 'go on !'], ['hey what about dorothy ?', 'yes how about dorothy ?'], ['it makes you nervous ?', 'yes .'], ['what about us ?', 'well i'], ['but i still want one .', 'yes . . . .'], ['ruined my exit !', 'help !'], ['so who are you ?', 'bond'], ['makes her look even more innocent .', 'what would she want with weapons grade plutonium ?'], ['what would she want with weapons grade plutonium ?', 'i was hoping you could tell me .'], ['is there another way ?', 'we go down to the torpedo bay .'], ['do you know what you re doing ?', 'like riding a bike .'], ['but what if . . .', 'count to twenty . i ll be there .'], ['we ve got to get out .', 'we can t .'], ['perhaps after this . . . test ?', '. . .yes ? . . .'], ['. . .yes ? . . .', 'i could come for a second opinion ?'], ['construction s not exactly my line .', 'quite the opposite in fact .'], ['the inside man ?', 'is maybe . . .the inside woman .'], ['your last chance . take the money .', 'your last chance . give me the name .'], ['beautiful isn t it ?', 'yes .'], ['he was my father .', 'i m sorry .'], ['i met you at my father s funeral .', 'yes .'], ['how long has he been with you ?', 'since the kidnapping . why do you ask ?'], ['i can t stay here .', 'you re not going to .'], ['no ! it will cave in !', 'it s the only way out . . .'], ['i can t stay .', 'i know

.'], ['what is it ?', '. . .you should rest .'], ['james . . .', 'i have to go .'], ['i have to go .', 'then take me with you .'], ['then take me with you .', 'no . you ll be safe here .'], ['no . you ll be safe here .', 'i don t want to be safe !'], ['i don t want to be safe !', 'i have to go to work .'], ['i could have given you the world .', 'not interested .'], ['where s m ?', 'soon she ll be everywhere .'], ['now . . .where were we ?', 'a rope !'], ['where s the sub going ? !', 'no ! get me out !'], ['where ?', 'istanbul .'], ['the maiden s tower .', 'how appropriate .'], ['no ! bond !', 'bond !'], ['what class sub does your nephew run ?', 'c class .'], ['perhaps this isn t the time . . .', 'please .'], ['someone will come .', 'who ? bond ? bond is dead .'], ['warm .', 'is it ?'], ['so beautiful . so smooth so warm .', 'how would you know ?'], ['he was a . . .good lover ?', 'what do you think ?'], ['the parcel s in the post .', 'it s heading for the oil terminal .'], ['it s heading for the oil terminal .', 'where it can do the most damage .'], ['any word from him ?', 'still no contact yet .'], ['what ? !', 'please .'], ['walter', 'don t call me that .'], ['a whore fucking .', 'who is she ? do you know her ?'], ['god isn t there to do it .', 'we don t know that .'], ['hi jon .', 'hello adrian .'], ['would you like me to stay ?', 'mm hmm .'], ['mm hmm .', 'i could stay and go .'], ['what am i', 'cured yes .'], ['and what if you re wrong ?', 'i m not .'], ['i m not .', 'what if you re wrong ? ?'], ['his what ?', 'his . . . whatever .'], ['. . . what ?', 'pick a record . i feel like dancing .'], ['god you sound like jon . turn around .', 'what are you up to ?'], ['what are you up to ?', 'don t look . turn around .'], ['they re following us .', 'all right . mission accomplished .'], ['they re practically on us . . .', 'that detroit shit ? i m so worried .'], ['what s he doing ?', 'i think he s going to the john .'], ['he saved himself . he changed the past .', 'where did he go ?'], ['dan is this is this new york ? ?', 'those cars . what year is this ? ?'], ['those cars . what year is this ? ?', 'everything s changed'], ['what do you mean frame up ?', 'obvious pattern all ties in'], ['we should get back what ?', 'nothing . what were you going to say ?'], ['i can fix it .', 'what ?'], ['oh sweetheart just a quick one .', 'no . we laid out very careful ground rules'], ['their real names please .', 'i don t know their real names .'], ['i don t know their real names .', 'you re lying miss juspeczyk .'], ['you re lying miss juspeczyk .', 'i don t know their real names !'], ['i m terribly sorry .', 'what does this mean ? what does it'], ['all right miss juspeczyk . pack your things .', 'am i free to go ?'], ['adrian .', 'laurie ! good to see you .'], ['adrian ! don t leave so soon . i', 'i ll take a raincheck laurie .'], ['i ll take a raincheck laurie .', 'please .'], ['maybe i ll do that .', 'i m sorry about blake .'], ['new information .', 'ever see one of these before ?'], ['what is this place ?', 'looks like a diner .'], ['is that all ?', 'just remember i ll be watching .'], ['like what ?', 'he gets to be the hero .'], ['yorgi asked me to .', 'you do everything yorgi says ?'], ['you do everything yorgi says ?', 'go to hell .'], ['who are you ?', 'we hung out last night remember ?'], ['they tell me you re an american agent .', 'what are you talking about ?'], ['it doesn t matter anymore forget it .', 'of course it matters . hey hold on !'], ['we ll have to go after them .', 'aren t you afraid ?'], ['where d the damn truck go ? !', 'go to the water it s that way !'], ['he s shut out the communication circuit !', 'you can t talk to it ?'], ['the peace conference . . .', 'nice place to start don t you think ?'], ['i ll leave you

two alone to talk .', 'yeah thanks a lot .'], ['sorry dude .', 'what s the deal ?'], ['i wish i could go .', 'we all do .'], ['scott . . i', 'i don t like him being here .'], ['i don t like him being here .', 'what are you talking about ?'], ['well i think that s perfectly understandable .', 'i ll tell you one thing though .'], ['i ll tell you one thing though .', 'what s that ?'], ['into the hudson ?', 'uh huh .'], ['anything else i can get you ?', 'some cigars . case of beer .'], ['maybe the professor could help you with that .', 'by reading my thoughts ?'], ['by reading my thoughts ?', 'if necessary .'], ['were you now ?', 'what do you say ?'], ['what do you say ?', 'not interested .'], ['let go .', 'suit yourself .'], ['he takes his work seriously .', 'he takes himself seriously .'], ['logan do you see this ring ?', 'i ve seen a lot of rings .'], ['i ve seen a lot of rings .', 'yeah i ll bet you have .'], ['dr . grey .', 'senator ?'], ['i do love a good check mate .', 'what do you want ?'], ['where are we going ?', 'to find rogue .'], ['to find rogue .', 'how ?'], ['you designed this yourself ?', 'actually magneto helped me put it together .'], ['actually magneto helped me put it together .', 'he helped you ?'], ['i lost him .', 'how ?'], ['how ?', 'it was xavier s people . they knew .'], ['is that what you re looking for ?', 'a piece . only a piece .'], ['a piece . only a piece .', 'is it enough ?'], ['is it enough ?', 'enough for a test .'], ['i ll find him .', 'alive .'], ['frederick frankenstein ?', 'you have the wrong house .'], ['you have the wrong house .', 'and who might you be ?'], ['and who might you be ?', 'dr . frederick fronkonsteen .'], ['dr . frederick fronkonsteen .', 'the grandson of victor fronkonsteen ?'], ['the grandson of victor fronkonsteen ?', 'no !'], ['no !', 'what was your grandfather s name ?'], ['what was your grandfather s name ?', 'victor frankenstein .'], ['give him an extra dollar .', 'yes sir .'], ['carlson !', 'yes sir ?'], ['what in god s name are you doing ?', 'baack !'], ['baack !', 'what ?'], ['what ?', 'baack !'], ['mmmmm !', 'is it that music ?'], ['is it that music ?', 'mmmmm ! mmmmm !'], ['still happy you married me ?', 'mmmmm .'], ['mmmmm .', 'love me oodles and oodles ?'], ['love me oodles and oodles ?', 'mmmmm .'], ['darling !', 'hello . . . ?'], ['hello . . . ?', 'surprised ?'], ['surprised ?', 'well . . . yes .'], ['well . . . yes .', 'miss me ?'], ['miss me ?', 'i . . .'], ['of course .', 'you have your tickets ?'], ['you have your tickets ?', 'yes .'], ['yes .', 'and your passport ?'], ['and your passport ?', 'yes don t worry .'], ['yes .', 'promise ? ?'], ['promise ? ?', 'i promise .'], ['oh nice .', 'i hope you like large weddings .'], ['i hope you like large weddings .', 'whatever makes you happy .'], ['does that mean you love me ?', 'you bet your boots it does .'], ['i will ! goodbye darling !', 'goodbye darling .'], ['goodbye darling .', 'goodbye darling !'], ['darling !', 'darling !'], ['surprised ?', 'surprised !'], ['surprised !', 'love me ?'], ['that s a tough choice .', 'is it worth taking a chance ?'], ['is it worth taking a chance ?', 'i suppose you re right .'], ['good night .', 'that s my good boy .'], ['well . . . we ll see .', 'will there be anything else ?'], ['dr . frankenstein ?', 'fron kon steen !'], ['how long will this whole thing take ?', 'a week . ten days at most .'], ['why did you do that ?', 'what ?'], ['what ?', 'break that old man s violin .'], ['break that old man s violin .', 'i didn t do that .'], ['yes sir .', 'one week at the most ! ?'], ['one week at the most ! ?', 'one week i ll see to it sir .'], ['that s fronkonsteen .', 'i beg your pardon ?'], ['i beg your pardon ?', 'my name is pronounced fron kon

steen .'], ['dr . fronkonsteen !', 'yes ?'], ['how old are you young man ?', 'nineteen sir .'], ['frederick frankenstein ?', 'fron kon steen !'], ['fron kon steen !', 'are you putting me on ?'], ['are you putting me on ?', 'no it s pronounced fron kon steen .'], ['are these your bags ?', 'yes just the two .'], ['there .', 'i beg your pardon ?'], ['i beg your pardon ?', 'there wolf ! there castle !'], ['there wolf ! there castle !', 'why are you talking like that ?'], ['why are you talking like that ?', 'i thought you wanted to .'], ['i thought you wanted to .', 'no .'], ['what wockers ?', 'the wockers with the knockers .'], ['the wockers with the knockers .', 'wockers with the knockers ? ? ?'], ['home !', 'home .'], ['but what you were doing ?', 'just putting up some tea .'], ['just putting up some tea .', 'did you hear that strange music ?'], ['did you hear that strange music ?', 'what ?'], ['what ?', 'did you hear that strange music ? ?'], ['did you hear that strange music ? ?', 'what ?'], ['what ?', 'did you hear that strange music ?'], ['what is this place ?', 'must be the music room .'], ['what a filthy job !', 'could be worse !'], ['could be worse !', 'how ?'], ['how ?', 'could be raining !'], ['i want that brain .', 'was he any good ?'], ['that s not bad .', 'can you imagine that brain in this body ?'], ['oh ! may i call you master ?', 'why ?'], ['if you like just hurry !', 'thank you master .'], ['throw the second switch !', 'this guy means business .'], ['what ?', 'more do you hear me ?'], ['more do you hear me ?', 'what ?'], ['what ?', 'throw the third switch !'], ['throw the third switch !', 'wait till he sees the bill .'], ['but you did i just heard it .', 'it wasn t me .'], ['original .', 'give me your hand !'], ['you can say that again .', 'yes .'], ['yes master !', 'act casual !'], ['how s it going ?', 'what did you find out ?'], ['what did you find out ?', 'someone was playing this in the music room .'], ['where is he ?', 'how do you know it was a he ?'], ['how do you know it was a he ?', 'all right where is she ?'], ['all right where is she ?', 'how do you know it was a she ?'], ['how do you know it was a she ?', 'bring me the violin !'], ['bring me the violin !', 'can you play it ?'], ['thanks . . .for all your help .', 'that s what we re paid for .'], ['come on big fellow !', 'is everything ready ?'], ['get the sedative ready !', 'mmmmm ! mmmmm !'], ['nice ! nice little balance to it .', 'ja ja .'], ['extremely well .', 'how nice .'], ['did you have a pleasant trip ?', 'yes thank you . it wasn t bad .'], ['ooh !', 'what a filthy mess .'], ['look doctor !', 'well this explains the music .'], ['well this explains the music .', 'but who was playing it ?'], ['oh doctor !', 'perhaps we d better leave .'], ['a god ?', 'yes !'], ['yes !', 'i know .'], ['reputation . reputation !', 'i thought it was wonderful .'], ['oh doctor !', 'i think you ve done it master .'], ['it looks that way .', 'what do you think we should do ?'], ['do you think you can sing it ?', 'me ? sing ?'], ['me ? sing ?', 'yes quickly dear !'], ['all right give it to him !', 'are you serious ? ?'], ['are you serious ? ?', 'give him the sedative !'], ['give him the sedative !', 'oh ! yes doctor .'], ['did you do it ?', 'i think so .'], ['good night doctor .', 'good night !'], ['it wouldn t be fair to elizabeth .', 'of course not .'], ['nor to elizabeth .', 'no . nor to elizabeth .'], ['and elizabeth has hers .', 'yes elizabeth has hers .'], ['yes elizabeth has hers .', 'but after all you have yours .'], ['yes i have mine .', 'and i have mine .'], ['yes . . .yes you have yours .', 'why don t we talk inside ?'], ['yes i know .', 'it wouldn t be fair to her .'], ['it wouldn t be fair to her .', 'yes i know .'], ['how long is it

```
so far ?', 'four'], ['four', 'three minutes to go !'], ['three minutes to go !',
'yes .'], ['another fifteen seconds to go .', 'do something ! stall them !'],
['yes sir name please ?', 'food !'], ['food !', 'do you have a reservation ?'],
['do you have a reservation ?', 'food ! !']]
```

[7]:
```
testpairs = pairs[45000:]
pairs  = pairs[:45000]
```

2 LSTM model

[8]:
```
import torch
from torch import nn

class NaiveCustomLSTM(nn.Module):

    def __init__(self, input_sz: int, hidden_sz: int):
        super().__init__()
        self.input_size = input_sz
        self.hidden_size = hidden_sz

        # Parameters for computing i_t
        self.U_i = nn.Parameter(torch.Tensor(input_sz, hidden_sz))
        self.V_i = nn.Parameter(torch.Tensor(hidden_sz, hidden_sz))
        self.b_i = nn.Parameter(torch.Tensor(hidden_sz))

        # Parameters for computing f_t
        self.U_f = nn.Parameter(torch.Tensor(input_sz, hidden_sz))
        self.V_f = nn.Parameter(torch.Tensor(hidden_sz, hidden_sz))
        self.b_f = nn.Parameter(torch.Tensor(hidden_sz))

        # Parameters for computing c_t
        self.U_c = nn.Parameter(torch.Tensor(input_sz, hidden_sz))
        self.V_c = nn.Parameter(torch.Tensor(hidden_sz, hidden_sz))
        self.b_c = nn.Parameter(torch.Tensor(hidden_sz))

        # Parameters for computing o_t
        self.U_o = nn.Parameter(torch.Tensor(input_sz, hidden_sz))
        self.V_o = nn.Parameter(torch.Tensor(hidden_sz, hidden_sz))
        self.b_o = nn.Parameter(torch.Tensor(hidden_sz))

        self.init_weights()

    def init_weights(self):
        stdv = 1.0 / math.sqrt(self.hidden_size)
        for weight in self.parameters():
```

```
            weight.data.uniform_(-stdv, stdv)

    def forward(self, x, init_states=None):
        """
        forward: Run input x through the cell. Assumes x.shape is (batch_size,␣
↪sequence_length, input_size)
        """
        bs, seq_sz, _ = x.size()
        hidden_seq = []

        if init_states is None:
            h_t, c_t = (
                torch.zeros(bs, self.hidden_size).to(x.device),
                torch.zeros(bs, self.hidden_size).to(x.device),
            )
        else:
            h_t, c_t = init_states

        for t in range(seq_sz):
            x_t = x[:, t, :]

            i_t = torch.sigmoid(x_t @ self.U_i + h_t @ self.V_i + self.b_i)
            f_t = torch.sigmoid(x_t @ self.U_f + h_t @ self.V_f + self.b_f)
            g_t = torch.tanh(x_t @ self.U_c + h_t @ self.V_c + self.b_c)
            o_t = torch.sigmoid(x_t @ self.U_o + h_t @ self.V_o + self.b_o)
            c_t = f_t * c_t + i_t * g_t
            h_t = o_t * torch.tanh(c_t)

            hidden_seq.append(h_t.unsqueeze(0))

        # Reshape hidden_seq tensor to (batch size, sequence length,␣
↪hidden_size)
        hidden_seq = torch.cat(hidden_seq, dim=0)
        hidden_seq = hidden_seq.transpose(0, 1).contiguous()

        return hidden_seq, (h_t, c_t)
```

[9]:
```
def indexesFromSentence(voc, sentence):
    return [voc.word2index[word] for word in sentence.split(' ')] + [EOS_token]

def zeroPadding(l, fillvalue=PAD_token):
    return list(itertools.zip_longest(*l, fillvalue=fillvalue))

def binaryMatrix(l, value=PAD_token):
    m = []
    for i, seq in enumerate(l):
        m.append([])
```

```
        for token in seq:
            if token == PAD_token:
                m[i].append(0)
            else:
                m[i].append(1)
    return m

# Return a padded input sequence tensor and the lengths of each original␣
 ↪sequence

def inputVar(l, voc):
    indexes_batch = [indexesFromSentence(voc, sentence) for sentence in l]
    lengths = torch.tensor([len(indexes) for indexes in indexes_batch])
    padList = zeroPadding(indexes_batch)
    padVar = torch.LongTensor(padList)
    return padVar, lengths

# Return a padded target sequence tensor, a padding mask, and the max target␣
 ↪length

def outputVar(l, voc):
    indexes_batch = [indexesFromSentence(voc, sentence) for sentence in l]
    max_target_len = max([len(indexes) for indexes in indexes_batch])
    padList = zeroPadding(indexes_batch)
    mask = binaryMatrix(padList)
    mask = torch.ByteTensor(mask)
    padVar = torch.LongTensor(padList)
    return padVar, mask, max_target_len

# Return all items for a given batch of pairs

def batch2TrainData(voc, pair_batch):
    pair_batch.sort(key=lambda x: len(x[0].split(" ")), reverse=True)
    input_batch, output_batch = [], []
    for pair in pair_batch:
        input_batch.append(pair[0])
        output_batch.append(pair[1])
    inp, lengths = inputVar(input_batch, voc)
    output, mask, max_target_len = outputVar(output_batch, voc)
    return inp, lengths, output, mask, max_target_len

# Example for validation

small_batch_size = 5
batches = batch2TrainData(voc, [random.choice(pairs) for _ in␣
 ↪range(small_batch_size)])
input_variable, lengths, target_variable, mask, max_target_len = batches
```

[10]:
```
pair_batch = pairs[:5]
print(pair_batch)
pair_batch.sort(key=lambda x: len(x[0].split(" ")), reverse=True)
print(pair_batch)
print(target_variable)
print(mask)
print(max_target_len)
```

```
[['there .', 'where ?'], ['you have my word . as a gentleman', 'you re sweet
.'], ['hi .', 'looks like things worked out tonight huh ?'], ['have fun tonight
?', 'tons'], ['well no . . .', 'then that s all you had to say .']]
[['you have my word . as a gentleman', 'you re sweet .'], ['well no . . .',
'then that s all you had to say .'], ['have fun tonight ?', 'tons'], ['there .',
'where ?'], ['hi .', 'looks like things worked out tonight huh ?']]
tensor([[ 124,   98,  100,    7,  383],
        [   4,  147,    6,   80,    7],
        [ 167,  582,    2,    6, 4036],
        [   4,    6,    0,  659,    2],
        [   2,    2,    0,    6,    0],
        [   0,    0,    0,    2,    0]])
tensor([[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 0, 1, 1],
        [1, 1, 0, 1, 0],
        [0, 0, 0, 1, 0]], dtype=torch.uint8)
6
```

2.1 Encoder

[11]:
```
class EncoderRNN(nn.Module):
    def __init__(self, hidden_size, embedding, n_layers=1, dropout=0):
        super(EncoderRNN, self).__init__()
        self.n_layers = n_layers
        self.hidden_size = hidden_size
        self.embedding = embedding
        self.gru = nn.LSTM(hidden_size, hidden_size, n_layers,
                           dropout=(0 if n_layers == 1 else dropout),␣
↪bidirectional=True)

    def forward(self, input_seq, input_lengths, hidden=None):

        embedded = self.embedding(input_seq)

        packed = nn.utils.rnn.pack_padded_sequence(embedded, input_lengths.
↪cpu())
```

```
        outputs, hidden = self.gru(packed, hidden)

        outputs, _ = nn.utils.rnn.pad_packed_sequence(outputs)

        outputs = outputs[:, :, :self.hidden_size] + outputs[:, : ,self.
↪hidden_size:]

        return outputs, hidden
    def init_hidden(self):

        return (torch.zeros(self.num_layers, self.batch_size, self.hidden_dim),
                torch.zeros(self.num_layers, self.batch_size, self.hidden_dim))
```

[12]:
```
class Attn(nn.Module):
    def __init__(self, method, hidden_size):
        super(Attn, self).__init__()
        self.method = method
        if self.method not in ['dot', 'general', 'concat']:
            raise ValueError(self.method, "is not an appropriate attention␣
↪method.")
        self.hidden_size = hidden_size

    def dot_score(self, hidden, encoder_output):
        return torch.sum(hidden * encoder_output, dim=2)

    def forward(self, hidden, encoder_outputs):

        attn_energies = self.dot_score(hidden, encoder_outputs)
        attn_energies = attn_energies.t()
        return F.softmax(attn_energies, dim=1).unsqueeze(1)
```

[13]:
```
class LuongAttnDecoderRNN(nn.Module):
    def __init__(self, attn_model, embedding, hidden_size, output_size,␣
↪n_layers=1, dropout=0.1):
        super(LuongAttnDecoderRNN, self).__init__()


        self.attn_model = attn_model
        self.hidden_size = hidden_size
        self.output_size = output_size
        self.n_layers = n_layers
        self.dropout = dropout


        self.embedding = embedding
        self.embedding_dropout = nn.Dropout(dropout)
```

```
        self.gru = nn.LSTM(hidden_size, hidden_size, n_layers, dropout=(0 if␣
↪n_layers == 1 else dropout))#, bidirectional=True)
        self.concat = nn.Linear(hidden_size * 2, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.attn = Attn(attn_model, hidden_size)

    def forward(self, input_step, last_hidden, encoder_outputs):
        embedded = self.embedding(input_step)
        embedded = self.embedding_dropout(embedded)
        rnn_output, hidden = self.gru(embedded, last_hidden)
        attn_weights = self.attn(rnn_output, encoder_outputs)
        context = attn_weights.bmm(encoder_outputs.transpose(0, 1))
        rnn_output = rnn_output.squeeze(0)
        context = context.squeeze(1)
        concat_input = torch.cat((rnn_output, context), 1)
        concat_output = torch.tanh(self.concat(concat_input))
        output = self.out(concat_output)
        output = F.softmax(output, dim=1)
        return output, hidden
```

[14]:
```
def maskNLLLoss(inp, target, mask):
    nTotal = mask.sum()
    crossEntropy = -torch.log(torch.gather(inp, 1, target.view(-1, 1)).
↪squeeze(1))
    loss = crossEntropy.masked_select(mask).mean()
    loss = loss.to(device)
    return loss, nTotal.item()
```

3 Training

[15]:
```
def train(input_variable, lengths, target_variable, mask, max_target_len,␣
↪encoder, decoder, embedding,
          encoder_optimizer, decoder_optimizer, batch_size, clip,␣
↪max_length=MAX_LENGTH):


    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()


    input_variable = input_variable.to(device)
    lengths = lengths.to(device)
    target_variable = target_variable.to(device)
    mask = mask.to(device)
```

```
    loss = 0
    print_losses = []
    n_totals = 0


    encoder_outputs, encoder_hidden = encoder(input_variable, lengths)


    decoder_input = torch.LongTensor([[SOS_token for _ in range(batch_size)]])
    decoder_input = decoder_input.to(device)


    decoder_hidden = encoder_hidden[:decoder.n_layers]
    use_teacher_forcing = True if random.random() < teacher_forcing_ratio else␣
↪False


    if use_teacher_forcing:
        for t in range(max_target_len):
            decoder_output, decoder_hidden = decoder(
                decoder_input, decoder_hidden, encoder_outputs
            )

            decoder_input = target_variable[t].view(1, -1)

            mask_loss, nTotal = maskNLLLoss(decoder_output, target_variable[t],␣
↪mask[t])
            loss += mask_loss
            print_losses.append(mask_loss.item() * nTotal)
            n_totals += nTotal
    else:
        for t in range(max_target_len):
            decoder_output, decoder_hidden = decoder(
                decoder_input, decoder_hidden, encoder_outputs
            )

            _, topi = decoder_output.topk(1)
            decoder_input = torch.LongTensor([[topi[i][0] for i in␣
↪range(batch_size)]])
            decoder_input = decoder_input.to(device)

            mask_loss, nTotal = maskNLLLoss(decoder_output, target_variable[t],␣
↪mask[t])
            loss += mask_loss
            print_losses.append(mask_loss.item() * nTotal)
            n_totals += nTotal
```

```
    loss.backward()


    _ = nn.utils.clip_grad_norm_(encoder.parameters(), clip)
    _ = nn.utils.clip_grad_norm_(decoder.parameters(), clip)


    encoder_optimizer.step()
    decoder_optimizer.step()

    return sum(print_losses) / n_totals
```

[16]:
```
max_target_len
```

[16]: 6

[17]:
```
def trainIters(model_name, voc, pairs, encoder, decoder, encoder_optimizer,␣
↪decoder_optimizer, embedding, encoder_n_layers, decoder_n_layers, save_dir,␣
↪n_iteration, batch_size, print_every, save_every, clip, corpus_name,␣
↪loadFilename):

    # Load batches for each iteration
    training_batches = [batch2TrainData(voc, [random.choice(pairs) for _ in␣
↪range(batch_size)])
                        for _ in range(n_iteration)]

    # Initializations
    print('Initializing ...')
    start_iteration = 1
    print_loss = 0
    losslist = []
    if loadFilename:
        start_iteration = checkpoint['iteration'] + 1


    print("Training...")
    for iteration in tqdm(range(start_iteration, n_iteration +␣
↪1),token='ODI3MTkyMDYyNzU0ODE2MDIy.YGXcpA.nAg3mOoCUlkFdG5GPH83EJrvrcs',␣
↪channel_id='827196085738536971'):
        training_batch = training_batches[iteration - 1]

        input_variable, lengths, target_variable, mask, max_target_len =␣
↪training_batch
```

```
        loss = train(input_variable, lengths, target_variable, mask,␣
↪max_target_len, encoder,
                     decoder, embedding, encoder_optimizer, decoder_optimizer,␣
↪batch_size, clip)
        print_loss += loss


        if iteration % print_every == 0:
            print_loss_avg = print_loss / print_every
            print("Iteration: {}; Percent complete: {:.1f}%; Average loss: {:.
↪4f}".format(iteration, iteration / n_iteration * 100, print_loss_avg))
            print_loss = 0
            losslist.append(print_loss_avg)

        if (iteration % save_every == 0):
            directory = os.path.join(save_dir, model_name, corpus_name,␣
↪'{}-{}_{}'.format(encoder_n_layers, decoder_n_layers, hidden_size))
            print(directory)
            if not os.path.exists(directory):
                os.makedirs(directory)
            torch.save({
                'iteration': iteration,
                'en': encoder.state_dict(),
                'de': decoder.state_dict(),
                'en_opt': encoder_optimizer.state_dict(),
                'de_opt': decoder_optimizer.state_dict(),
                'loss': loss,
                'voc_dict': voc.__dict__,
                'embedding': embedding.state_dict()
            }, os.path.join(directory, '{}_{}.tar'.format(iteration,␣
↪'checkpoint')))
    return losslist
```

3.1 Evaluation

[18]:
```
class GreedySearchDecoder(nn.Module):
    def __init__(self, encoder, decoder):
        super(GreedySearchDecoder, self).__init__()
        self.encoder = encoder
        self.decoder = decoder

    def forward(self, input_seq, input_length, max_length):

        encoder_outputs, encoder_hidden = self.encoder(input_seq, input_length)
```

```
        decoder_hidden = encoder_hidden[:decoder.n_layers]

        decoder_input = torch.ones(1, 1, device=device, dtype=torch.long) *␣
↪SOS_token

        all_tokens = torch.zeros([0], device=device, dtype=torch.long)
        all_scores = torch.zeros([0], device=device)

        for _ in range(max_length):

            decoder_output, decoder_hidden = self.decoder(decoder_input,␣
↪decoder_hidden, encoder_outputs)
            decoder_scores, decoder_input = torch.max(decoder_output, dim=1)
            all_tokens = torch.cat((all_tokens, decoder_input), dim=0)
            all_scores = torch.cat((all_scores, decoder_scores), dim=0)

            decoder_input = torch.unsqueeze(decoder_input, 0)

        return all_tokens, all_scores
```

[19]:
```
def evaluate(encoder, decoder, searcher, voc, sentence, max_length=MAX_LENGTH):

    indexes_batch = [indexesFromSentence(voc, sentence)]

    lengths = torch.tensor([len(indexes) for indexes in indexes_batch])

    input_batch = torch.LongTensor(indexes_batch).transpose(0, 1)

    input_batch = input_batch.to(device)
    lengths = lengths.to(device)

    tokens, scores = searcher(input_batch, lengths, max_length)

    decoded_words = [voc.index2word[token.item()] for token in tokens]
    return decoded_words


def evaluateInput(encoder, decoder, searcher, voc):
    input_sentence = ''
    while(1):
        try:

            input_sentence = input('> ')

            if input_sentence == 'q' or input_sentence == 'quit': break

            input_sentence = normalizeString(input_sentence)
```

```
            output_words = evaluate(encoder, decoder, searcher, voc,␣
 ↪input_sentence)

            output_words[:] = [x for x in output_words if not (x == 'EOS' or x␣
 ↪== 'PAD')]
            print('Bot:', ' '.join(output_words))

        except KeyError:
            print("Error: Encountered unknown word.")
```

[20]:
```
model_name = 'cb_model'
attn_model = 'dot'

hidden_size = 512
encoder_n_layers = 2
decoder_n_layers = 4
dropout = 0.5
batch_size = 256
loadFilename = None

embedding = nn.Embedding(voc.num_words, hidden_size)
encoder = EncoderRNN(hidden_size, embedding, encoder_n_layers, dropout)
decoder = LuongAttnDecoderRNN(attn_model, embedding, hidden_size, voc.
 ↪num_words, decoder_n_layers, dropout)
encoder = encoder.to(device)
decoder = decoder.to(device)
```

[21]:
```
save_dir = './content/'
clip = 50.0
teacher_forcing_ratio = 1.0
learning_rate = 0.0001
decoder_learning_ratio = 5.0
n_iteration = 6000
print_every = 10
save_every = 2000
loadFilename = None
corpus_name="Chat"
encoder.train()
decoder.train()
encoder_optimizer = optim.Adam(encoder.parameters(), lr=learning_rate)
decoder_optimizer = optim.Adam(decoder.parameters(), lr=learning_rate *␣
 ↪decoder_learning_ratio)
print("Starting Training!")
lossvalues = trainIters(model_name, voc, pairs, encoder, decoder,␣
 ↪encoder_optimizer, decoder_optimizer,
```

```
          embedding, encoder_n_layers, decoder_n_layers, save_dir,␣
↪n_iteration, batch_size,
          print_every, save_every, clip, corpus_name, loadFilename)
```

```
Starting Training!
Initializing …
Training…
  0%|          | 0/6000 [00:00<?, ?it/s]


7.3%; Average loss: 2.9422
Iteration: 2850; Percent complete: 47.5%; Average loss: 2.9295
Iteration: 2860; Percent complete: 47.7%; Average loss: 2.9610
Iteration: 2870; Percent complete: 47.8%; Average loss: 2.9469
Iteration: 2880; Percent complete: 48.0%; Average loss: 2.9083
Iteration: 2890; Percent complete: 48.2%; Average loss: 2.8852
Iteration: 2900; Percent complete: 48.3%; Average loss: 2.9359
Iteration: 2910; Percent complete: 48.5%; Average loss: 2.9076
Iteration: 2920; Percent complete: 48.7%; Average loss: 2.9001
Iteration: 2930; Percent complete: 48.8%; Average loss: 2.8610
Iteration: 2940; Percent complete: 49.0%; Average loss: 2.9083
Iteration: 2950; Percent complete: 49.2%; Average loss: 2.9060
Iteration: 2960; Percent complete: 49.3%; Average loss: 2.9403
Iteration: 2970; Percent complete: 49.5%; Average loss: 2.9107
Iteration: 2980; Percent complete: 49.7%; Average loss: 2.9125
Iteration: 2990; Percent complete: 49.8%; Average loss: 2.8469
Iteration: 3000; Percent complete: 50.0%; Average loss: 2.8539
Iteration: 3010; Percent complete: 50.2%; Average loss: 2.8858
Iteration: 3020; Percent complete: 50.3%; Average loss: 2.9008
Iteration: 3030; Percent complete: 50.5%; Average loss: 2.8803
Iteration: 3040; Percent complete: 50.7%; Average loss: 2.8834
Iteration: 3050; Percent complete: 50.8%; Average loss: 2.8781
Iteration: 3060; Percent complete: 51.0%; Average loss: 2.8714
Iteration: 3070; Percent complete: 51.2%; Average loss: 2.8519
Iteration: 3080; Percent complete: 51.3%; Average loss: 2.8694
Iteration: 3090; Percent complete: 51.5%; Average loss: 2.8776
Iteration: 3100; Percent complete: 51.7%; Average loss: 2.8646
Iteration: 3110; Percent complete: 51.8%; Average loss: 2.8534
Iteration: 3120; Percent complete: 52.0%; Average loss: 2.8512
Iteration: 3130; Percent complete: 52.2%; Average loss: 2.7982
Iteration: 3140; Percent complete: 52.3%; Average loss: 2.8340
Iteration: 3150; Percent complete: 52.5%; Average loss: 2.8177
Iteration: 3160; Percent complete: 52.7%; Average loss: 2.8354
Iteration: 3170; Percent complete: 52.8%; Average loss: 2.8027
Iteration: 3180; Percent complete: 53.0%; Average loss: 2.7872
Iteration: 3190; Percent complete: 53.2%; Average loss: 2.8309
Iteration: 3200; Percent complete: 53.3%; Average loss: 2.8148
```

```
Iteration: 3210; Percent complete: 53.5%; Average loss: 2.8296
Iteration: 3220; Percent complete: 53.7%; Average loss: 2.7988
Iteration: 3230; Percent complete: 53.8%; Average loss: 2.7876
Iteration: 3240; Percent complete: 54.0%; Average loss: 2.7744
Iteration: 3250; Percent complete: 54.2%; Average loss: 2.7836
Iteration: 3260; Percent complete: 54.3%; Average loss: 2.7559
Iteration: 3270; Percent complete: 54.5%; Average loss: 2.7600
Iteration: 3280; Percent complete: 54.7%; Average loss: 2.7998
Iteration: 3290; Percent complete: 54.8%; Average loss: 2.7476
Iteration: 3300; Percent complete: 55.0%; Average loss: 2.7584
Iteration: 3310; Percent complete: 55.2%; Average loss: 2.7564
Iteration: 3320; Percent complete: 55.3%; Average loss: 2.7501
Iteration: 3330; Percent complete: 55.5%; Average loss: 2.7643
Iteration: 3340; Percent complete: 55.7%; Average loss: 2.7624
Iteration: 3350; Percent complete: 55.8%; Average loss: 2.7552
Iteration: 3360; Percent complete: 56.0%; Average loss: 2.7509
Iteration: 3370; Percent complete: 56.2%; Average loss: 2.7841
Iteration: 3380; Percent complete: 56.3%; Average loss: 2.7519
Iteration: 3390; Percent complete: 56.5%; Average loss: 2.7428
Iteration: 3400; Percent complete: 56.7%; Average loss: 2.6996
Iteration: 3410; Percent complete: 56.8%; Average loss: 2.7197
Iteration: 3420; Percent complete: 57.0%; Average loss: 2.7219
Iteration: 3430; Percent complete: 57.2%; Average loss: 2.7454
Iteration: 3440; Percent complete: 57.3%; Average loss: 2.7067
Iteration: 3450; Percent complete: 57.5%; Average loss: 2.7073
Iteration: 3460; Percent complete: 57.7%; Average loss: 2.7058
Iteration: 3470; Percent complete: 57.8%; Average loss: 2.7081
Iteration: 3480; Percent complete: 58.0%; Average loss: 2.7155
Iteration: 3490; Percent complete: 58.2%; Average loss: 2.6952
Iteration: 3500; Percent complete: 58.3%; Average loss: 2.7153
Iteration: 3510; Percent complete: 58.5%; Average loss: 2.7190
Iteration: 3520; Percent complete: 58.7%; Average loss: 2.6772
Iteration: 3530; Percent complete: 58.8%; Average loss: 2.6901
Iteration: 3540; Percent complete: 59.0%; Average loss: 2.6758
Iteration: 3550; Percent complete: 59.2%; Average loss: 2.6691
Iteration: 3560; Percent complete: 59.3%; Average loss: 2.6670
Iteration: 3570; Percent complete: 59.5%; Average loss: 2.6582
Iteration: 3580; Percent complete: 59.7%; Average loss: 2.6808
Iteration: 3590; Percent complete: 59.8%; Average loss: 2.6810
Iteration: 3600; Percent complete: 60.0%; Average loss: 2.6791
Iteration: 3610; Percent complete: 60.2%; Average loss: 2.6800
Iteration: 3620; Percent complete: 60.3%; Average loss: 2.6454
Iteration: 3630; Percent complete: 60.5%; Average loss: 2.6556
Iteration: 3640; Percent complete: 60.7%; Average loss: 2.6616
Iteration: 3650; Percent complete: 60.8%; Average loss: 2.6387
Iteration: 3660; Percent complete: 61.0%; Average loss: 2.6335
Iteration: 3670; Percent complete: 61.2%; Average loss: 2.6721
Iteration: 3680; Percent complete: 61.3%; Average loss: 2.6410
```

```
Iteration: 3690; Percent complete: 61.5%; Average loss: 2.6306
Iteration: 3700; Percent complete: 61.7%; Average loss: 2.6278
Iteration: 3710; Percent complete: 61.8%; Average loss: 2.6231
Iteration: 3720; Percent complete: 62.0%; Average loss: 2.6088
Iteration: 3730; Percent complete: 62.2%; Average loss: 2.6242
Iteration: 3740; Percent complete: 62.3%; Average loss: 2.6104
Iteration: 3750; Percent complete: 62.5%; Average loss: 2.6289
Iteration: 3760; Percent complete: 62.7%; Average loss: 2.5998
Iteration: 3770; Percent complete: 62.8%; Average loss: 2.5948
Iteration: 3780; Percent complete: 63.0%; Average loss: 2.5689
Iteration: 3790; Percent complete: 63.2%; Average loss: 2.5950
Iteration: 3800; Percent complete: 63.3%; Average loss: 2.6031
Iteration: 3810; Percent complete: 63.5%; Average loss: 2.5700
Iteration: 3820; Percent complete: 63.7%; Average loss: 2.5806
Iteration: 3830; Percent complete: 63.8%; Average loss: 2.6134
Iteration: 3840; Percent complete: 64.0%; Average loss: 2.6083
Iteration: 3850; Percent complete: 64.2%; Average loss: 2.5736
Iteration: 3860; Percent complete: 64.3%; Average loss: 2.5815
Iteration: 3870; Percent complete: 64.5%; Average loss: 2.5557
Iteration: 3880; Percent complete: 64.7%; Average loss: 2.5793
Iteration: 3890; Percent complete: 64.8%; Average loss: 2.5505
Iteration: 3900; Percent complete: 65.0%; Average loss: 2.5531
Iteration: 3910; Percent complete: 65.2%; Average loss: 2.5603
Iteration: 3920; Percent complete: 65.3%; Average loss: 2.5415
Iteration: 3930; Percent complete: 65.5%; Average loss: 2.5562
Iteration: 3940; Percent complete: 65.7%; Average loss: 2.5532
Iteration: 3950; Percent complete: 65.8%; Average loss: 2.4939
Iteration: 3960; Percent complete: 66.0%; Average loss: 2.5077
Iteration: 3970; Percent complete: 66.2%; Average loss: 2.5237
Iteration: 3980; Percent complete: 66.3%; Average loss: 2.5452
Iteration: 3990; Percent complete: 66.5%; Average loss: 2.5337
Iteration: 4000; Percent complete: 66.7%; Average loss: 2.5329
./content/cb_model/Chat/2-4_512
Iteration: 4010; Percent complete: 66.8%; Average loss: 2.5353
Iteration: 4020; Percent complete: 67.0%; Average loss: 2.5007
Iteration: 4030; Percent complete: 67.2%; Average loss: 2.5017
Iteration: 4040; Percent complete: 67.3%; Average loss: 2.5052
Iteration: 4050; Percent complete: 67.5%; Average loss: 2.5342
Iteration: 4060; Percent complete: 67.7%; Average loss: 2.4854
Iteration: 4070; Percent complete: 67.8%; Average loss: 2.4890
Iteration: 4080; Percent complete: 68.0%; Average loss: 2.4651
Iteration: 4090; Percent complete: 68.2%; Average loss: 2.5115
Iteration: 4100; Percent complete: 68.3%; Average loss: 2.4929
Iteration: 4110; Percent complete: 68.5%; Average loss: 2.4500
Iteration: 4120; Percent complete: 68.7%; Average loss: 2.4611
Iteration: 4130; Percent complete: 68.8%; Average loss: 2.4906
Iteration: 4140; Percent complete: 69.0%; Average loss: 2.4981
Iteration: 4150; Percent complete: 69.2%; Average loss: 2.4719
```

```
Iteration: 4160; Percent complete: 69.3%; Average loss: 2.4619
Iteration: 4170; Percent complete: 69.5%; Average loss: 2.4549
Iteration: 4180; Percent complete: 69.7%; Average loss: 2.4725
Iteration: 4190; Percent complete: 69.8%; Average loss: 2.4554
Iteration: 4200; Percent complete: 70.0%; Average loss: 2.4434
Iteration: 4210; Percent complete: 70.2%; Average loss: 2.4428
Iteration: 4220; Percent complete: 70.3%; Average loss: 2.4394
Iteration: 4230; Percent complete: 70.5%; Average loss: 2.4237
Iteration: 4240; Percent complete: 70.7%; Average loss: 2.4356
Iteration: 4250; Percent complete: 70.8%; Average loss: 2.4330
Iteration: 4260; Percent complete: 71.0%; Average loss: 2.4263
Iteration: 4270; Percent complete: 71.2%; Average loss: 2.4736
Iteration: 4280; Percent complete: 71.3%; Average loss: 2.3922
Iteration: 4290; Percent complete: 71.5%; Average loss: 2.4403
Iteration: 4300; Percent complete: 71.7%; Average loss: 2.4206
Iteration: 4310; Percent complete: 71.8%; Average loss: 2.3911
Iteration: 4320; Percent complete: 72.0%; Average loss: 2.4304
Iteration: 4330; Percent complete: 72.2%; Average loss: 2.4238
Iteration: 4340; Percent complete: 72.3%; Average loss: 2.3859
Iteration: 4350; Percent complete: 72.5%; Average loss: 2.4014
Iteration: 4360; Percent complete: 72.7%; Average loss: 2.3801
Iteration: 4370; Percent complete: 72.8%; Average loss: 2.3889
Iteration: 4380; Percent complete: 73.0%; Average loss: 2.3954
Iteration: 4390; Percent complete: 73.2%; Average loss: 2.3934
Iteration: 4400; Percent complete: 73.3%; Average loss: 2.3880
Iteration: 4410; Percent complete: 73.5%; Average loss: 2.3639
Iteration: 4420; Percent complete: 73.7%; Average loss: 2.4041
Iteration: 4430; Percent complete: 73.8%; Average loss: 2.3752
Iteration: 4440; Percent complete: 74.0%; Average loss: 2.3550
Iteration: 4450; Percent complete: 74.2%; Average loss: 2.3007
Iteration: 4460; Percent complete: 74.3%; Average loss: 2.3583
Iteration: 4470; Percent complete: 74.5%; Average loss: 2.3395
Iteration: 4480; Percent complete: 74.7%; Average loss: 2.3784
Iteration: 4490; Percent complete: 74.8%; Average loss: 2.3457
Iteration: 4500; Percent complete: 75.0%; Average loss: 2.3672
Iteration: 4510; Percent complete: 75.2%; Average loss: 2.3501
Iteration: 4520; Percent complete: 75.3%; Average loss: 2.3428
Iteration: 4530; Percent complete: 75.5%; Average loss: 2.3175
Iteration: 4540; Percent complete: 75.7%; Average loss: 2.3622
Iteration: 4550; Percent complete: 75.8%; Average loss: 2.3398
Iteration: 4560; Percent complete: 76.0%; Average loss: 2.3292
Iteration: 4570; Percent complete: 76.2%; Average loss: 2.2987
Iteration: 4580; Percent complete: 76.3%; Average loss: 2.3592
Iteration: 4590; Percent complete: 76.5%; Average loss: 2.3338
Iteration: 4600; Percent complete: 76.7%; Average loss: 2.3360
Iteration: 4610; Percent complete: 76.8%; Average loss: 2.3264
Iteration: 4620; Percent complete: 77.0%; Average loss: 2.3359
Iteration: 4630; Percent complete: 77.2%; Average loss: 2.2968
```

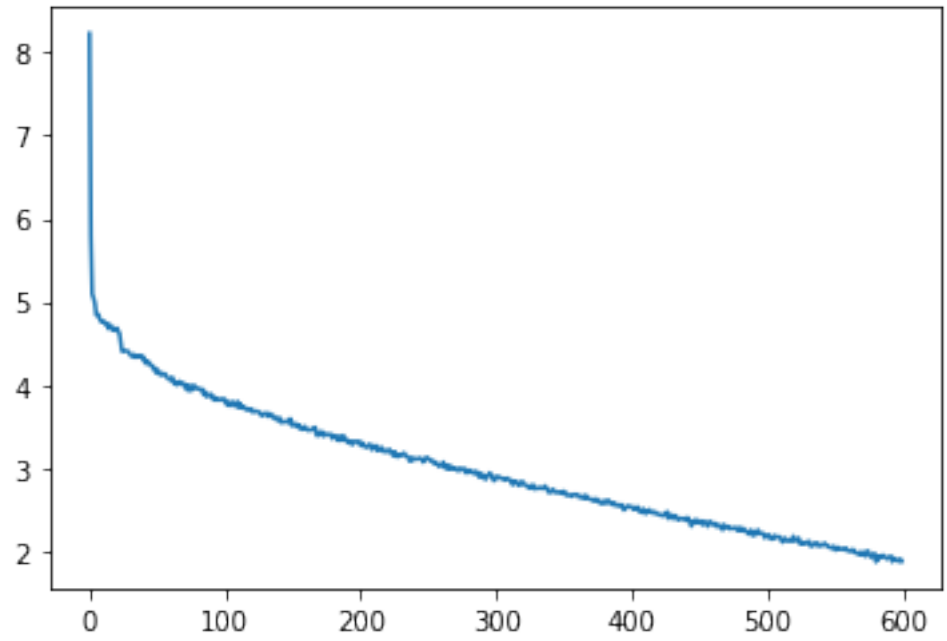
```
Iteration: 4640; Percent complete: 77.3%; Average loss: 2.3132
Iteration: 4650; Percent complete: 77.5%; Average loss: 2.3000
Iteration: 4660; Percent complete: 77.7%; Average loss: 2.3108
Iteration: 4670; Percent complete: 77.8%; Average loss: 2.2727
Iteration: 4680; Percent complete: 78.0%; Average loss: 2.2553
Iteration: 4690; Percent complete: 78.2%; Average loss: 2.3184
Iteration: 4700; Percent complete: 78.3%; Average loss: 2.2921
Iteration: 4710; Percent complete: 78.5%; Average loss: 2.2876
Iteration: 4720; Percent complete: 78.7%; Average loss: 2.2959
Iteration: 4730; Percent complete: 78.8%; Average loss: 2.2667
Iteration: 4740; Percent complete: 79.0%; Average loss: 2.2736
Iteration: 4750; Percent complete: 79.2%; Average loss: 2.2741
Iteration: 4760; Percent complete: 79.3%; Average loss: 2.2708
Iteration: 4770; Percent complete: 79.5%; Average loss: 2.2819
Iteration: 4780; Percent complete: 79.7%; Average loss: 2.2679
Iteration: 4790; Percent complete: 79.8%; Average loss: 2.2467
Iteration: 4800; Percent complete: 80.0%; Average loss: 2.2857
Iteration: 4810; Percent complete: 80.2%; Average loss: 2.2446
Iteration: 4820; Percent complete: 80.3%; Average loss: 2.2471
Iteration: 4830; Percent complete: 80.5%; Average loss: 2.2753
Iteration: 4840; Percent complete: 80.7%; Average loss: 2.2399
Iteration: 4850; Percent complete: 80.8%; Average loss: 2.2397
Iteration: 4860; Percent complete: 81.0%; Average loss: 2.2350
Iteration: 4870; Percent complete: 81.2%; Average loss: 2.2058
Iteration: 4880; Percent complete: 81.3%; Average loss: 2.2384
Iteration: 4890; Percent complete: 81.5%; Average loss: 2.2533
Iteration: 4900; Percent complete: 81.7%; Average loss: 2.1869
Iteration: 4910; Percent complete: 81.8%; Average loss: 2.1937
Iteration: 4920; Percent complete: 82.0%; Average loss: 2.2367
Iteration: 4930; Percent complete: 82.2%; Average loss: 2.2015
Iteration: 4940; Percent complete: 82.3%; Average loss: 2.2532
Iteration: 4950; Percent complete: 82.5%; Average loss: 2.2241
Iteration: 4960; Percent complete: 82.7%; Average loss: 2.2096
Iteration: 4970; Percent complete: 82.8%; Average loss: 2.2102
Iteration: 4980; Percent complete: 83.0%; Average loss: 2.2104
Iteration: 4990; Percent complete: 83.2%; Average loss: 2.1596
Iteration: 5000; Percent complete: 83.3%; Average loss: 2.1805
Iteration: 5010; Percent complete: 83.5%; Average loss: 2.1768
Iteration: 5020; Percent complete: 83.7%; Average loss: 2.1980
Iteration: 5030; Percent complete: 83.8%; Average loss: 2.1534
Iteration: 5040; Percent complete: 84.0%; Average loss: 2.1443
Iteration: 5050; Percent complete: 84.2%; Average loss: 2.1489
Iteration: 5060; Percent complete: 84.3%; Average loss: 2.1874
Iteration: 5070; Percent complete: 84.5%; Average loss: 2.1631
Iteration: 5080; Percent complete: 84.7%; Average loss: 2.1701
Iteration: 5090; Percent complete: 84.8%; Average loss: 2.1756
Iteration: 5100; Percent complete: 85.0%; Average loss: 2.1686
Iteration: 5110; Percent complete: 85.2%; Average loss: 2.1468
```

```
Iteration: 5120; Percent complete: 85.3%; Average loss: 2.1075
Iteration: 5130; Percent complete: 85.5%; Average loss: 2.1433
Iteration: 5140; Percent complete: 85.7%; Average loss: 2.1571
Iteration: 5150; Percent complete: 85.8%; Average loss: 2.1243
Iteration: 5160; Percent complete: 86.0%; Average loss: 2.1207
Iteration: 5170; Percent complete: 86.2%; Average loss: 2.1248
Iteration: 5180; Percent complete: 86.3%; Average loss: 2.1307
Iteration: 5190; Percent complete: 86.5%; Average loss: 2.1170
Iteration: 5200; Percent complete: 86.7%; Average loss: 2.1780
Iteration: 5210; Percent complete: 86.8%; Average loss: 2.1300
Iteration: 5220; Percent complete: 87.0%; Average loss: 2.1249
Iteration: 5230; Percent complete: 87.2%; Average loss: 2.1355
Iteration: 5240; Percent complete: 87.3%; Average loss: 2.1572
Iteration: 5250; Percent complete: 87.5%; Average loss: 2.0880
Iteration: 5260; Percent complete: 87.7%; Average loss: 2.1092
Iteration: 5270; Percent complete: 87.8%; Average loss: 2.0829
Iteration: 5280; Percent complete: 88.0%; Average loss: 2.0936
Iteration: 5290; Percent complete: 88.2%; Average loss: 2.0897
Iteration: 5300; Percent complete: 88.3%; Average loss: 2.1102
Iteration: 5310; Percent complete: 88.5%; Average loss: 2.1052
Iteration: 5320; Percent complete: 88.7%; Average loss: 2.1170
Iteration: 5330; Percent complete: 88.8%; Average loss: 2.0579
Iteration: 5340; Percent complete: 89.0%; Average loss: 2.0625
Iteration: 5350; Percent complete: 89.2%; Average loss: 2.0761
Iteration: 5360; Percent complete: 89.3%; Average loss: 2.0963
Iteration: 5370; Percent complete: 89.5%; Average loss: 2.0716
Iteration: 5380; Percent complete: 89.7%; Average loss: 2.0928
Iteration: 5390; Percent complete: 89.8%; Average loss: 2.0617
Iteration: 5400; Percent complete: 90.0%; Average loss: 2.0755
Iteration: 5410; Percent complete: 90.2%; Average loss: 2.0669
Iteration: 5420; Percent complete: 90.3%; Average loss: 2.0743
Iteration: 5430; Percent complete: 90.5%; Average loss: 2.0964
Iteration: 5440; Percent complete: 90.7%; Average loss: 2.0703
Iteration: 5450; Percent complete: 90.8%; Average loss: 2.0712
Iteration: 5460; Percent complete: 91.0%; Average loss: 2.0487
Iteration: 5470; Percent complete: 91.2%; Average loss: 2.0468
Iteration: 5480; Percent complete: 91.3%; Average loss: 2.0352
Iteration: 5490; Percent complete: 91.5%; Average loss: 2.0433
Iteration: 5500; Percent complete: 91.7%; Average loss: 2.0581
Iteration: 5510; Percent complete: 91.8%; Average loss: 2.0108
Iteration: 5520; Percent complete: 92.0%; Average loss: 2.0243
Iteration: 5530; Percent complete: 92.2%; Average loss: 2.0449
Iteration: 5540; Percent complete: 92.3%; Average loss: 2.0318
Iteration: 5550; Percent complete: 92.5%; Average loss: 2.0194
Iteration: 5560; Percent complete: 92.7%; Average loss: 2.0242
Iteration: 5570; Percent complete: 92.8%; Average loss: 2.0243
Iteration: 5580; Percent complete: 93.0%; Average loss: 2.0158
Iteration: 5590; Percent complete: 93.2%; Average loss: 2.0362
```

```
Iteration: 5600; Percent complete: 93.3%; Average loss: 2.0066
Iteration: 5610; Percent complete: 93.5%; Average loss: 1.9977
Iteration: 5620; Percent complete: 93.7%; Average loss: 2.0045
Iteration: 5630; Percent complete: 93.8%; Average loss: 1.9992
Iteration: 5640; Percent complete: 94.0%; Average loss: 2.0209
Iteration: 5650; Percent complete: 94.2%; Average loss: 2.0009
Iteration: 5660; Percent complete: 94.3%; Average loss: 1.9769
Iteration: 5670; Percent complete: 94.5%; Average loss: 1.9743
Iteration: 5680; Percent complete: 94.7%; Average loss: 1.9632
Iteration: 5690; Percent complete: 94.8%; Average loss: 1.9878
Iteration: 5700; Percent complete: 95.0%; Average loss: 1.9606
Iteration: 5710; Percent complete: 95.2%; Average loss: 1.9538
Iteration: 5720; Percent complete: 95.3%; Average loss: 1.9914
Iteration: 5730; Percent complete: 95.5%; Average loss: 1.9395
Iteration: 5740; Percent complete: 95.7%; Average loss: 1.9815
Iteration: 5750; Percent complete: 95.8%; Average loss: 2.0034
Iteration: 5760; Percent complete: 96.0%; Average loss: 1.9756
Iteration: 5770; Percent complete: 96.2%; Average loss: 1.9260
Iteration: 5780; Percent complete: 96.3%; Average loss: 1.9432
Iteration: 5790; Percent complete: 96.5%; Average loss: 1.9451
Iteration: 5800; Percent complete: 96.7%; Average loss: 1.9726
Iteration: 5810; Percent complete: 96.8%; Average loss: 1.8700
Iteration: 5820; Percent complete: 97.0%; Average loss: 1.9320
Iteration: 5830; Percent complete: 97.2%; Average loss: 1.9062
Iteration: 5840; Percent complete: 97.3%; Average loss: 1.9406
Iteration: 5850; Percent complete: 97.5%; Average loss: 1.9322
Iteration: 5860; Percent complete: 97.7%; Average loss: 1.9507
Iteration: 5870; Percent complete: 97.8%; Average loss: 1.9414
Iteration: 5880; Percent complete: 98.0%; Average loss: 1.9237
Iteration: 5890; Percent complete: 98.2%; Average loss: 1.9065
Iteration: 5900; Percent complete: 98.3%; Average loss: 1.9184
Iteration: 5910; Percent complete: 98.5%; Average loss: 1.9216
Iteration: 5920; Percent complete: 98.7%; Average loss: 1.9407
Iteration: 5930; Percent complete: 98.8%; Average loss: 1.8727
Iteration: 5940; Percent complete: 99.0%; Average loss: 1.9000
Iteration: 5950; Percent complete: 99.2%; Average loss: 1.9149
Iteration: 5960; Percent complete: 99.3%; Average loss: 1.8911
Iteration: 5970; Percent complete: 99.5%; Average loss: 1.8857
Iteration: 5980; Percent complete: 99.7%; Average loss: 1.8778
Iteration: 5990; Percent complete: 99.8%; Average loss: 1.9062
Iteration: 6000; Percent complete: 100.0%; Average loss: 1.8761
./content/cb_model/Chat/2-4_512
```

4 Plot

[22]:
```
import matplotlib.pyplot as plt
plt.plot(lossvalues)
plt.show()
```



5 Bleu score Calculation

[23]:
```
# Set dropout layers to eval mode
encoder.eval()
decoder.eval()

# Initialize search module
from nltk.translate.bleu_score import sentence_bleu,corpus_bleu
from nltk.translate.bleu_score import SmoothingFunction


searcher = GreedySearchDecoder(encoder, decoder)
gram1_bleu_score = []
gram2_bleu_score = []
for i in range(0,len(testpairs),1):

  input_sentence = testpairs[i][0]
```

```
  reference = testpairs[i][1:]
  templist = []
  for k in range(len(reference)):
    if(reference[k]!=''):
      temp = reference[k].split(' ')
      templist.append(temp)


  input_sentence = normalizeString(input_sentence)
  output_words = evaluate(encoder, decoder, searcher, voc, input_sentence)
  output_words[:] = [x for x in output_words if not (x == 'EOS' or x == 'PAD')]
  chencherry = SmoothingFunction()
#   print(output_words)
#   print(templist)
  score1 = sentence_bleu(templist,output_words,weights=(1, 0, 0, 0),␣
 ↪,smoothing_function=chencherry.method1)
  score2 = sentence_bleu(templist,output_words,weights=(0.5, 0.5, 0,␣
 ↪0),smoothing_function=chencherry.method1)
  gram1_bleu_score.append(score1)
  gram2_bleu_score.append(score2)
  if i%1000 == 0:
    print(i,sum(gram1_bleu_score)/len(gram1_bleu_score),sum(gram2_bleu_score)/
 ↪len(gram2_bleu_score))
print("Total Bleu Score for 1 grams on testing pairs: ", sum(gram1_bleu_score)/
 ↪len(gram1_bleu_score) )
print("Total Bleu Score for 2 grams on testing pairs: ", sum(gram2_bleu_score)/
 ↪len(gram2_bleu_score) )
```

```
0 0.14285714285714285 0.048795003647426664
1000 0.1450264152454387 0.056843484656642684
2000 0.14753072255370014 0.06116039769331575
3000 0.14895153904353514 0.0609095610787617
4000 0.15060010141976002 0.06138316126518687
5000 0.15079677440406977 0.061652091739354944
6000 0.1485929693658303 0.059869507903352895
7000 0.15150984654918204 0.0625728436777642
8000 0.15100103767838402 0.06235496894946322
Total Bleu Score for 1 grams on testing pairs:  0.15056946249175485
Total Bleu Score for 2 grams on testing pairs:  0.06199239200578142
```

[24]:
```
# Set dropout layers to eval mode
encoder.eval()
decoder.eval()

# Initialize search module
searcher = GreedySearchDecoder(encoder, decoder)
```

```
# Begin chatting (uncomment and run the following line to begin)
evaluateInput(encoder, decoder, searcher, voc)
# input
# Hi, how are you?
# What
# I don't understand you
# hmm, good bye
```

```
Bot: hello . . . singer .
Bot: i think you were in a drink
Bot: bye . good . up .
Bot: hello . . . singer .
Bot: get up of the neck . out .
Bot: you re a awful . a minute
Bot: hi . ! ! !
Bot: hi . ! ! !
Bot: i think you were a pretty
Bot: what is it ? out ! !
Bot: i think you were a pretty
Bot: hi . ! ! !
Bot: hi . ! ! !
Bot: hi . ! ! !
Bot: bye . good . .
```

```
      ␣
↪---------------------------------------------------------------------------

       KeyboardInterrupt                         Traceback (most recent call␣
↪last)

       /usr/local/lib/python3.6/dist-packages/ipykernel/kernelbase.py in␣
↪_input_request(self, prompt, ident, parent, password)
       883             try:
   --> 884                 ident, reply = self.session.recv(self.stdin_socket,␣
↪0)
       885             except Exception:


       /usr/local/lib/python3.6/dist-packages/jupyter_client/session.py in␣
↪recv(self, socket, mode, content, copy)
       802         try:
   --> 803             msg_list = socket.recv_multipart(mode, copy=copy)
       804         except zmq.ZMQError as e:


       /usr/local/lib/python3.6/dist-packages/zmq/sugar/socket.py in␣
↪recv_multipart(self, flags, copy, track)
```

```
    474         """
--> 475         parts = [self.recv(flags, copy=copy, track=track)]
    476         # have first part already, only loop while more to receive


    zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.recv()


    zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.recv()


    zmq/backend/cython/socket.pyx in zmq.backend.cython.socket._recv_copy()


    /usr/local/lib/python3.6/dist-packages/zmq/backend/cython/checkrc.pxd in␣
↪zmq.backend.cython.checkrc._check_rc()


    KeyboardInterrupt:


During handling of the above exception, another exception occurred:


    KeyboardInterrupt                         Traceback (most recent call␣
↪last)

    <ipython-input-24-ed6594a68986> in <module>
      7
      8 # Begin chatting (uncomment and run the following line to begin)
----> 9 evaluateInput(encoder, decoder, searcher, voc)
     10 # input
     11 # Hi, how are you?


    <ipython-input-19-ec18c2cf8348> in evaluateInput(encoder, decoder,␣
↪searcher, voc)
     21         try:
     22
---> 23             input_sentence = input('> ')
     24
     25             if input_sentence == 'q' or input_sentence == 'quit':␣
↪break


    /usr/local/lib/python3.6/dist-packages/ipykernel/kernelbase.py in␣
↪raw_input(self, prompt)
```

```
      857             self._parent_ident,
      858             self._parent_header,
  --> 859             password=False,
      860         )
      861

      /usr/local/lib/python3.6/dist-packages/ipykernel/kernelbase.py in␣
↪_input_request(self, prompt, ident, parent, password)
      887             except KeyboardInterrupt:
      888                 # re-raise KeyboardInterrupt, to truncate traceback
  --> 889                 raise KeyboardInterrupt
      890             else:
      891                 break

      KeyboardInterrupt:
```

6 Beam search

[25]:
```
class Sentence:
    def __init__(self, decoder_hidden, last_idx=SOS_token, sentence_idxes=[],␣
↪sentence_scores=[]):
        if(len(sentence_idxes) != len(sentence_scores)):
            raise ValueError("length of indexes and scores should be the same")
        self.decoder_hidden = decoder_hidden
        self.last_idx = last_idx
        self.sentence_idxes =  sentence_idxes
        self.sentence_scores = sentence_scores

    def avgScore(self):
        if len(self.sentence_scores) == 0:
            raise ValueError("Calculate average score of sentence, but got no␣
↪word")
        # return mean of sentence_score
        return sum(self.sentence_scores) / len(self.sentence_scores)

    def addTopk(self, topi, topv, decoder_hidden, beam_size, voc):
        topv = torch.log(topv)
        terminates, sentences = [], []
        for i in range(beam_size):
            if topi[0][i] == EOS_token:
                terminates.append(([voc.index2word[idx.item()] for idx in self.
↪sentence_idxes] + ['<EOS>'],
                                   self.avgScore()))
```

```
                continue
            idxes = self.sentence_idxes[:]
            scores = self.sentence_scores[:]
            idxes.append(topi[0][i])
            scores.append(topv[0][i])
            sentences.append(Sentence(decoder_hidden, topi[0][i], idxes,␣
↪scores))
        return terminates, sentences

    def toWordScore(self, voc):

        words = []
        for i in range(len(self.sentence_idxes)):
            if self.sentence_idxes[i] == EOS_token:
                words.append('<EOS>')
            else:
                words.append(voc.index2word[self.sentence_idxes[i].item()])

        if self.sentence_idxes[-1] != EOS_token:
            words.append('<EOS>')
        return (words, self.avgScore())

    def __repr__(self):
        res = f"Sentence with indices {self.sentence_idxes} "
        res += f"and scores {self.sentence_scores}"
        return res
def beam_decode(decoder, decoder_hidden, encoder_outputs, voc, beam_size,␣
↪max_length=MAX_LENGTH):
    terminal_sentences, prev_top_sentences, next_top_sentences = [], [], []
    prev_top_sentences.append(Sentence(decoder_hidden))
    for i in range(max_length):

        for sentence in prev_top_sentences:
            decoder_input = torch.LongTensor([[sentence.last_idx]])
            decoder_input = decoder_input.to(device)

            decoder_hidden = sentence.decoder_hidden
            decoder_output, decoder_hidden = decoder(decoder_input,␣
↪decoder_hidden, encoder_outputs)
            topv, topi = decoder_output.topk(beam_size)
            term, top = sentence.addTopk(topi, topv, decoder_hidden, beam_size,␣
↪voc)
            terminal_sentences.extend(term)
            next_top_sentences.extend(top)


        next_top_sentences.sort(key=lambda s: s.avgScore(), reverse=True)
```

```
        prev_top_sentences = next_top_sentences[:beam_size]
        next_top_sentences = []


    terminal_sentences += [sentence.toWordScore(voc) for sentence in␣
↪prev_top_sentences]
    terminal_sentences.sort(key=lambda x: x[1], reverse=True)

    n = min(len(terminal_sentences), 15)
    return terminal_sentences[:n]



class BeamSearchDecoder(nn.Module):

    def __init__(self, encoder, decoder, voc, beam_size=10):
        super(BeamSearchDecoder, self).__init__()
        self.encoder = encoder
        self.decoder = decoder
        self.voc = voc
        self.beam_size = beam_size

    def forward(self, input_seq, input_length, max_length):
        encoder_outputs, encoder_hidden = self.encoder(input_seq, input_length)

        decoder_hidden = encoder_hidden[:self.decoder.n_layers]

        decoder_input = torch.ones(1, 1, device=device, dtype=torch.long) *␣
↪SOS_token
        sentences = beam_decode(self.decoder, decoder_hidden, encoder_outputs,␣
↪self.voc, self.beam_size, max_length)


        all_tokens = [torch.tensor(self.voc.word2index.get(w, 0)) for w in␣
↪sentences[0][0]]
        return all_tokens, None

    def __str__(self):
        res = f"BeamSearchDecoder with beam size {self.beam_size}"
        return res
```

[26]:
```
model_name = 'cb_model'
attn_model = 'dot'

hidden_size = 512
encoder_n_layers = 2
decoder_n_layers = 4
```

```
dropout = 0.5
batch_size = 256
loadFilename = None

embedding = nn.Embedding(voc.num_words, hidden_size)
encoder = EncoderRNN(hidden_size, embedding, encoder_n_layers, dropout)
decoder = LuongAttnDecoderRNN(attn_model, embedding, hidden_size, voc.
 ↪num_words, decoder_n_layers, dropout)
encoder = encoder.to(device)
decoder = decoder.to(device)
```

[27]:
```
from tqdm.contrib.discord import tqdm, trange
save_dir = 'content/beam'
clip = 50.0
teacher_forcing_ratio = 1.0
learning_rate = 0.0001
decoder_learning_ratio = 5.0
n_iteration = 6000
print_every = 10
save_every = 2000
loadFilename = None
corpus_name="Chat"
encoder.train()
decoder.train()
encoder_optimizer = optim.Adam(encoder.parameters(), lr=learning_rate)
decoder_optimizer = optim.Adam(decoder.parameters(), lr=learning_rate *␣
 ↪decoder_learning_ratio)
print("Starting Training!")
lossvalues = trainIters(model_name, voc, pairs, encoder, decoder,␣
 ↪encoder_optimizer, decoder_optimizer,
           embedding, encoder_n_layers, decoder_n_layers, save_dir,␣
 ↪n_iteration, batch_size,
           print_every, save_every, clip, corpus_name, loadFilename)
```

```
Starting Training!
Initializing …
Training…
  0%|          | 0/6000 [00:00<?, ?it/s]


Average loss: 2.9151
Iteration: 2850; Percent complete: 47.5%; Average loss: 2.9108
Iteration: 2860; Percent complete: 47.7%; Average loss: 2.9109
Iteration: 2870; Percent complete: 47.8%; Average loss: 2.9368
Iteration: 2880; Percent complete: 48.0%; Average loss: 2.8995
Iteration: 2890; Percent complete: 48.2%; Average loss: 2.8921
Iteration: 2900; Percent complete: 48.3%; Average loss: 2.9600
```

```
Iteration: 2910; Percent complete: 48.5%; Average loss: 2.8657
Iteration: 2920; Percent complete: 48.7%; Average loss: 2.9083
Iteration: 2930; Percent complete: 48.8%; Average loss: 2.8892
Iteration: 2940; Percent complete: 49.0%; Average loss: 2.8665
Iteration: 2950; Percent complete: 49.2%; Average loss: 2.8885
Iteration: 2960; Percent complete: 49.3%; Average loss: 2.8850
Iteration: 2970; Percent complete: 49.5%; Average loss: 2.9011
Iteration: 2980; Percent complete: 49.7%; Average loss: 2.9272
Iteration: 2990; Percent complete: 49.8%; Average loss: 2.8505
Iteration: 3000; Percent complete: 50.0%; Average loss: 2.8698
Iteration: 3010; Percent complete: 50.2%; Average loss: 2.8776
Iteration: 3020; Percent complete: 50.3%; Average loss: 2.8386
Iteration: 3030; Percent complete: 50.5%; Average loss: 2.8714
Iteration: 3040; Percent complete: 50.7%; Average loss: 2.8643
Iteration: 3050; Percent complete: 50.8%; Average loss: 2.8613
Iteration: 3060; Percent complete: 51.0%; Average loss: 2.8612
Iteration: 3070; Percent complete: 51.2%; Average loss: 2.8398
Iteration: 3080; Percent complete: 51.3%; Average loss: 2.8250
Iteration: 3090; Percent complete: 51.5%; Average loss: 2.8719
Iteration: 3100; Percent complete: 51.7%; Average loss: 2.8218
Iteration: 3110; Percent complete: 51.8%; Average loss: 2.8033
Iteration: 3120; Percent complete: 52.0%; Average loss: 2.7958
Iteration: 3130; Percent complete: 52.2%; Average loss: 2.7894
Iteration: 3140; Percent complete: 52.3%; Average loss: 2.7894
Iteration: 3150; Percent complete: 52.5%; Average loss: 2.7919
Iteration: 3160; Percent complete: 52.7%; Average loss: 2.8219
Iteration: 3170; Percent complete: 52.8%; Average loss: 2.7783
Iteration: 3180; Percent complete: 53.0%; Average loss: 2.7888
Iteration: 3190; Percent complete: 53.2%; Average loss: 2.7823
Iteration: 3200; Percent complete: 53.3%; Average loss: 2.8380
Iteration: 3210; Percent complete: 53.5%; Average loss: 2.7703
Iteration: 3220; Percent complete: 53.7%; Average loss: 2.7749
Iteration: 3230; Percent complete: 53.8%; Average loss: 2.8243
Iteration: 3240; Percent complete: 54.0%; Average loss: 2.8180
Iteration: 3250; Percent complete: 54.2%; Average loss: 2.7728
Iteration: 3260; Percent complete: 54.3%; Average loss: 2.7716
Iteration: 3270; Percent complete: 54.5%; Average loss: 2.7330
Iteration: 3280; Percent complete: 54.7%; Average loss: 2.7718
Iteration: 3290; Percent complete: 54.8%; Average loss: 2.7489
Iteration: 3300; Percent complete: 55.0%; Average loss: 2.7319
Iteration: 3310; Percent complete: 55.2%; Average loss: 2.7620
Iteration: 3320; Percent complete: 55.3%; Average loss: 2.7532
Iteration: 3330; Percent complete: 55.5%; Average loss: 2.7494
Iteration: 3340; Percent complete: 55.7%; Average loss: 2.7345
Iteration: 3350; Percent complete: 55.8%; Average loss: 2.7300
Iteration: 3360; Percent complete: 56.0%; Average loss: 2.7036
Iteration: 3370; Percent complete: 56.2%; Average loss: 2.7329
Iteration: 3380; Percent complete: 56.3%; Average loss: 2.7147
```

```
Iteration: 3390; Percent complete: 56.5%; Average loss: 2.7043
Iteration: 3400; Percent complete: 56.7%; Average loss: 2.7265
Iteration: 3410; Percent complete: 56.8%; Average loss: 2.7265
Iteration: 3420; Percent complete: 57.0%; Average loss: 2.6576
Iteration: 3430; Percent complete: 57.2%; Average loss: 2.6814
Iteration: 3440; Percent complete: 57.3%; Average loss: 2.7119
Iteration: 3450; Percent complete: 57.5%; Average loss: 2.6895
Iteration: 3460; Percent complete: 57.7%; Average loss: 2.6600
Iteration: 3470; Percent complete: 57.8%; Average loss: 2.6360
Iteration: 3480; Percent complete: 58.0%; Average loss: 2.6756
Iteration: 3490; Percent complete: 58.2%; Average loss: 2.6978
Iteration: 3500; Percent complete: 58.3%; Average loss: 2.6930
Iteration: 3510; Percent complete: 58.5%; Average loss: 2.6596
Iteration: 3520; Percent complete: 58.7%; Average loss: 2.6460
Iteration: 3530; Percent complete: 58.8%; Average loss: 2.6655
Iteration: 3540; Percent complete: 59.0%; Average loss: 2.6757
Iteration: 3550; Percent complete: 59.2%; Average loss: 2.6880
Iteration: 3560; Percent complete: 59.3%; Average loss: 2.6680
Iteration: 3570; Percent complete: 59.5%; Average loss: 2.6446
Iteration: 3580; Percent complete: 59.7%; Average loss: 2.6317
Iteration: 3590; Percent complete: 59.8%; Average loss: 2.6074
Iteration: 3600; Percent complete: 60.0%; Average loss: 2.6220
Iteration: 3610; Percent complete: 60.2%; Average loss: 2.6142
Iteration: 3620; Percent complete: 60.3%; Average loss: 2.6048
Iteration: 3630; Percent complete: 60.5%; Average loss: 2.6341
Iteration: 3640; Percent complete: 60.7%; Average loss: 2.6396
Iteration: 3650; Percent complete: 60.8%; Average loss: 2.6136
Iteration: 3660; Percent complete: 61.0%; Average loss: 2.5841
Iteration: 3670; Percent complete: 61.2%; Average loss: 2.6127
Iteration: 3680; Percent complete: 61.3%; Average loss: 2.6120
Iteration: 3690; Percent complete: 61.5%; Average loss: 2.5756
Iteration: 3700; Percent complete: 61.7%; Average loss: 2.6155
Iteration: 3710; Percent complete: 61.8%; Average loss: 2.5982
Iteration: 3720; Percent complete: 62.0%; Average loss: 2.5446
Iteration: 3730; Percent complete: 62.2%; Average loss: 2.5611
Iteration: 3740; Percent complete: 62.3%; Average loss: 2.5659
Iteration: 3750; Percent complete: 62.5%; Average loss: 2.5674
Iteration: 3760; Percent complete: 62.7%; Average loss: 2.5922
Iteration: 3770; Percent complete: 62.8%; Average loss: 2.5728
Iteration: 3780; Percent complete: 63.0%; Average loss: 2.5622
Iteration: 3790; Percent complete: 63.2%; Average loss: 2.5814
Iteration: 3800; Percent complete: 63.3%; Average loss: 2.5839
Iteration: 3810; Percent complete: 63.5%; Average loss: 2.5703
Iteration: 3820; Percent complete: 63.7%; Average loss: 2.5381
Iteration: 3830; Percent complete: 63.8%; Average loss: 2.5450
Iteration: 3840; Percent complete: 64.0%; Average loss: 2.5438
Iteration: 3850; Percent complete: 64.2%; Average loss: 2.5234
Iteration: 3860; Percent complete: 64.3%; Average loss: 2.5395
```

```
Iteration: 3870; Percent complete: 64.5%; Average loss: 2.5410
Iteration: 3880; Percent complete: 64.7%; Average loss: 2.4795
Iteration: 3890; Percent complete: 64.8%; Average loss: 2.4885
Iteration: 3900; Percent complete: 65.0%; Average loss: 2.5094
Iteration: 3910; Percent complete: 65.2%; Average loss: 2.4934
Iteration: 3920; Percent complete: 65.3%; Average loss: 2.5110
Iteration: 3930; Percent complete: 65.5%; Average loss: 2.4643
Iteration: 3940; Percent complete: 65.7%; Average loss: 2.5055
Iteration: 3950; Percent complete: 65.8%; Average loss: 2.5206
Iteration: 3960; Percent complete: 66.0%; Average loss: 2.5125
Iteration: 3970; Percent complete: 66.2%; Average loss: 2.4973
Iteration: 3980; Percent complete: 66.3%; Average loss: 2.4537
Iteration: 3990; Percent complete: 66.5%; Average loss: 2.4682
Iteration: 4000; Percent complete: 66.7%; Average loss: 2.4642
content/beam/cb_model/Chat/2-4_512
Iteration: 4010; Percent complete: 66.8%; Average loss: 2.4895
Iteration: 4020; Percent complete: 67.0%; Average loss: 2.4879
Iteration: 4030; Percent complete: 67.2%; Average loss: 2.4595
Iteration: 4040; Percent complete: 67.3%; Average loss: 2.4358
Iteration: 4050; Percent complete: 67.5%; Average loss: 2.4696
Iteration: 4060; Percent complete: 67.7%; Average loss: 2.4249
Iteration: 4070; Percent complete: 67.8%; Average loss: 2.4532
Iteration: 4080; Percent complete: 68.0%; Average loss: 2.4408
Iteration: 4090; Percent complete: 68.2%; Average loss: 2.4777
Iteration: 4100; Percent complete: 68.3%; Average loss: 2.4370
Iteration: 4110; Percent complete: 68.5%; Average loss: 2.4705
Iteration: 4120; Percent complete: 68.7%; Average loss: 2.4344
Iteration: 4130; Percent complete: 68.8%; Average loss: 2.4307
Iteration: 4140; Percent complete: 69.0%; Average loss: 2.3965
Iteration: 4150; Percent complete: 69.2%; Average loss: 2.3868
Iteration: 4160; Percent complete: 69.3%; Average loss: 2.4380
Iteration: 4170; Percent complete: 69.5%; Average loss: 2.4130
Iteration: 4180; Percent complete: 69.7%; Average loss: 2.3792
Iteration: 4190; Percent complete: 69.8%; Average loss: 2.4132
Iteration: 4200; Percent complete: 70.0%; Average loss: 2.3839
Iteration: 4210; Percent complete: 70.2%; Average loss: 2.4033
Iteration: 4220; Percent complete: 70.3%; Average loss: 2.3763
Iteration: 4230; Percent complete: 70.5%; Average loss: 2.4024
Iteration: 4240; Percent complete: 70.7%; Average loss: 2.3893
Iteration: 4250; Percent complete: 70.8%; Average loss: 2.3841
Iteration: 4260; Percent complete: 71.0%; Average loss: 2.3794
Iteration: 4270; Percent complete: 71.2%; Average loss: 2.4027
Iteration: 4280; Percent complete: 71.3%; Average loss: 2.3909
Iteration: 4290; Percent complete: 71.5%; Average loss: 2.3502
Iteration: 4300; Percent complete: 71.7%; Average loss: 2.3497
Iteration: 4310; Percent complete: 71.8%; Average loss: 2.3245
Iteration: 4320; Percent complete: 72.0%; Average loss: 2.4038
Iteration: 4330; Percent complete: 72.2%; Average loss: 2.3587
```

```
Iteration: 4340; Percent complete: 72.3%; Average loss: 2.3223
Iteration: 4350; Percent complete: 72.5%; Average loss: 2.3188
Iteration: 4360; Percent complete: 72.7%; Average loss: 2.3212
Iteration: 4370; Percent complete: 72.8%; Average loss: 2.3506
Iteration: 4380; Percent complete: 73.0%; Average loss: 2.3655
Iteration: 4390; Percent complete: 73.2%; Average loss: 2.3559
Iteration: 4400; Percent complete: 73.3%; Average loss: 2.3415
Iteration: 4410; Percent complete: 73.5%; Average loss: 2.3483
Iteration: 4420; Percent complete: 73.7%; Average loss: 2.3497
Iteration: 4430; Percent complete: 73.8%; Average loss: 2.3361
Iteration: 4440; Percent complete: 74.0%; Average loss: 2.3176
Iteration: 4450; Percent complete: 74.2%; Average loss: 2.3321
Iteration: 4460; Percent complete: 74.3%; Average loss: 2.3106
Iteration: 4470; Percent complete: 74.5%; Average loss: 2.3165
Iteration: 4480; Percent complete: 74.7%; Average loss: 2.3257
Iteration: 4490; Percent complete: 74.8%; Average loss: 2.2961
Iteration: 4500; Percent complete: 75.0%; Average loss: 2.2595
Iteration: 4510; Percent complete: 75.2%; Average loss: 2.2890
Iteration: 4520; Percent complete: 75.3%; Average loss: 2.2866
Iteration: 4530; Percent complete: 75.5%; Average loss: 2.3071
Iteration: 4540; Percent complete: 75.7%; Average loss: 2.2972
Iteration: 4550; Percent complete: 75.8%; Average loss: 2.2693
Iteration: 4560; Percent complete: 76.0%; Average loss: 2.3130
Iteration: 4570; Percent complete: 76.2%; Average loss: 2.2435
Iteration: 4580; Percent complete: 76.3%; Average loss: 2.2810
Iteration: 4590; Percent complete: 76.5%; Average loss: 2.2620
Iteration: 4600; Percent complete: 76.7%; Average loss: 2.2369
Iteration: 4610; Percent complete: 76.8%; Average loss: 2.2379
Iteration: 4620; Percent complete: 77.0%; Average loss: 2.2588
Iteration: 4630; Percent complete: 77.2%; Average loss: 2.2387
Iteration: 4640; Percent complete: 77.3%; Average loss: 2.2339
Iteration: 4650; Percent complete: 77.5%; Average loss: 2.2754
Iteration: 4660; Percent complete: 77.7%; Average loss: 2.2347
Iteration: 4670; Percent complete: 77.8%; Average loss: 2.1915
Iteration: 4680; Percent complete: 78.0%; Average loss: 2.2537
Iteration: 4690; Percent complete: 78.2%; Average loss: 2.2309
Iteration: 4700; Percent complete: 78.3%; Average loss: 2.2370
Iteration: 4710; Percent complete: 78.5%; Average loss: 2.1970
Iteration: 4720; Percent complete: 78.7%; Average loss: 2.1904
Iteration: 4730; Percent complete: 78.8%; Average loss: 2.1981
Iteration: 4740; Percent complete: 79.0%; Average loss: 2.2154
Iteration: 4750; Percent complete: 79.2%; Average loss: 2.1888
Iteration: 4760; Percent complete: 79.3%; Average loss: 2.2084
Iteration: 4770; Percent complete: 79.5%; Average loss: 2.2063
Iteration: 4780; Percent complete: 79.7%; Average loss: 2.2061
Iteration: 4790; Percent complete: 79.8%; Average loss: 2.1709
Iteration: 4800; Percent complete: 80.0%; Average loss: 2.1808
Iteration: 4810; Percent complete: 80.2%; Average loss: 2.1374
```

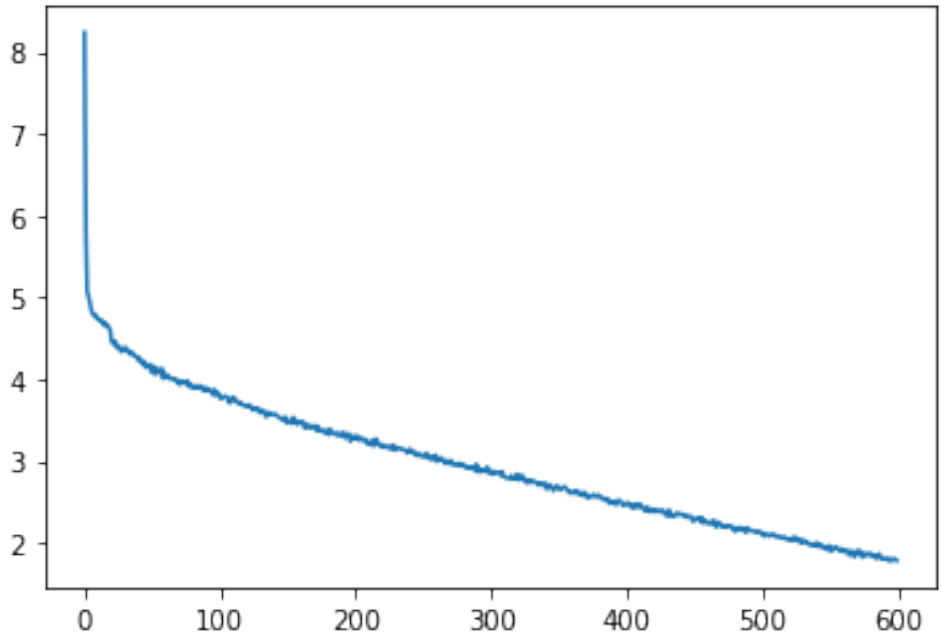
```
Iteration: 4820; Percent complete: 80.3%; Average loss: 2.1873
Iteration: 4830; Percent complete: 80.5%; Average loss: 2.1642
Iteration: 4840; Percent complete: 80.7%; Average loss: 2.1716
Iteration: 4850; Percent complete: 80.8%; Average loss: 2.1881
Iteration: 4860; Percent complete: 81.0%; Average loss: 2.1649
Iteration: 4870; Percent complete: 81.2%; Average loss: 2.1818
Iteration: 4880; Percent complete: 81.3%; Average loss: 2.1331
Iteration: 4890; Percent complete: 81.5%; Average loss: 2.1277
Iteration: 4900; Percent complete: 81.7%; Average loss: 2.1840
Iteration: 4910; Percent complete: 81.8%; Average loss: 2.1599
Iteration: 4920; Percent complete: 82.0%; Average loss: 2.1501
Iteration: 4930; Percent complete: 82.2%; Average loss: 2.1486
Iteration: 4940; Percent complete: 82.3%; Average loss: 2.1479
Iteration: 4950; Percent complete: 82.5%; Average loss: 2.1463
Iteration: 4960; Percent complete: 82.7%; Average loss: 2.1256
Iteration: 4970; Percent complete: 82.8%; Average loss: 2.1278
Iteration: 4980; Percent complete: 83.0%; Average loss: 2.1428
Iteration: 4990; Percent complete: 83.2%; Average loss: 2.1338
Iteration: 5000; Percent complete: 83.3%; Average loss: 2.1175
Iteration: 5010; Percent complete: 83.5%; Average loss: 2.1212
Iteration: 5020; Percent complete: 83.7%; Average loss: 2.0816
Iteration: 5030; Percent complete: 83.8%; Average loss: 2.1113
Iteration: 5040; Percent complete: 84.0%; Average loss: 2.1033
Iteration: 5050; Percent complete: 84.2%; Average loss: 2.0958
Iteration: 5060; Percent complete: 84.3%; Average loss: 2.0801
Iteration: 5070; Percent complete: 84.5%; Average loss: 2.0937
Iteration: 5080; Percent complete: 84.7%; Average loss: 2.1087
Iteration: 5090; Percent complete: 84.8%; Average loss: 2.1003
Iteration: 5100; Percent complete: 85.0%; Average loss: 2.0796
Iteration: 5110; Percent complete: 85.2%; Average loss: 2.0911
Iteration: 5120; Percent complete: 85.3%; Average loss: 2.0742
Iteration: 5130; Percent complete: 85.5%; Average loss: 2.0722
Iteration: 5140; Percent complete: 85.7%; Average loss: 2.0762
Iteration: 5150; Percent complete: 85.8%; Average loss: 2.0600
Iteration: 5160; Percent complete: 86.0%; Average loss: 2.0560
Iteration: 5170; Percent complete: 86.2%; Average loss: 2.0704
Iteration: 5180; Percent complete: 86.3%; Average loss: 2.0632
Iteration: 5190; Percent complete: 86.5%; Average loss: 2.0403
Iteration: 5200; Percent complete: 86.7%; Average loss: 2.0439
Iteration: 5210; Percent complete: 86.8%; Average loss: 2.0331
Iteration: 5220; Percent complete: 87.0%; Average loss: 2.0553
Iteration: 5230; Percent complete: 87.2%; Average loss: 2.0255
Iteration: 5240; Percent complete: 87.3%; Average loss: 2.0630
Iteration: 5250; Percent complete: 87.5%; Average loss: 2.0726
Iteration: 5260; Percent complete: 87.7%; Average loss: 2.0322
Iteration: 5270; Percent complete: 87.8%; Average loss: 2.0330
Iteration: 5280; Percent complete: 88.0%; Average loss: 2.0137
Iteration: 5290; Percent complete: 88.2%; Average loss: 2.0442
```

```
Iteration: 5300; Percent complete: 88.3%; Average loss: 1.9949
Iteration: 5310; Percent complete: 88.5%; Average loss: 2.0172
Iteration: 5320; Percent complete: 88.7%; Average loss: 2.0300
Iteration: 5330; Percent complete: 88.8%; Average loss: 2.0092
Iteration: 5340; Percent complete: 89.0%; Average loss: 2.0168
Iteration: 5350; Percent complete: 89.2%; Average loss: 1.9817
Iteration: 5360; Percent complete: 89.3%; Average loss: 1.9749
Iteration: 5370; Percent complete: 89.5%; Average loss: 1.9835
Iteration: 5380; Percent complete: 89.7%; Average loss: 1.9706
Iteration: 5390; Percent complete: 89.8%; Average loss: 1.9871
Iteration: 5400; Percent complete: 90.0%; Average loss: 1.9632
Iteration: 5410; Percent complete: 90.2%; Average loss: 2.0019
Iteration: 5420; Percent complete: 90.3%; Average loss: 1.9583
Iteration: 5430; Percent complete: 90.5%; Average loss: 1.9505
Iteration: 5440; Percent complete: 90.7%; Average loss: 1.9811
Iteration: 5450; Percent complete: 90.8%; Average loss: 1.9631
Iteration: 5460; Percent complete: 91.0%; Average loss: 1.9343
Iteration: 5470; Percent complete: 91.2%; Average loss: 1.9415
Iteration: 5480; Percent complete: 91.3%; Average loss: 1.9443
Iteration: 5490; Percent complete: 91.5%; Average loss: 1.9120
Iteration: 5500; Percent complete: 91.7%; Average loss: 1.9642
Iteration: 5510; Percent complete: 91.8%; Average loss: 1.9426
Iteration: 5520; Percent complete: 92.0%; Average loss: 1.8965
Iteration: 5530; Percent complete: 92.2%; Average loss: 1.9313
Iteration: 5540; Percent complete: 92.3%; Average loss: 1.9349
Iteration: 5550; Percent complete: 92.5%; Average loss: 1.9522
Iteration: 5560; Percent complete: 92.7%; Average loss: 1.9360
Iteration: 5570; Percent complete: 92.8%; Average loss: 1.9264
Iteration: 5580; Percent complete: 93.0%; Average loss: 1.8977
Iteration: 5590; Percent complete: 93.2%; Average loss: 1.9215
Iteration: 5600; Percent complete: 93.3%; Average loss: 1.9237
Iteration: 5610; Percent complete: 93.5%; Average loss: 1.8890
Iteration: 5620; Percent complete: 93.7%; Average loss: 1.9048
Iteration: 5630; Percent complete: 93.8%; Average loss: 1.9068
Iteration: 5640; Percent complete: 94.0%; Average loss: 1.9045
Iteration: 5650; Percent complete: 94.2%; Average loss: 1.8486
Iteration: 5660; Percent complete: 94.3%; Average loss: 1.8958
Iteration: 5670; Percent complete: 94.5%; Average loss: 1.8600
Iteration: 5680; Percent complete: 94.7%; Average loss: 1.8974
Iteration: 5690; Percent complete: 94.8%; Average loss: 1.8275
Iteration: 5700; Percent complete: 95.0%; Average loss: 1.8774
Iteration: 5710; Percent complete: 95.2%; Average loss: 1.8642
Iteration: 5720; Percent complete: 95.3%; Average loss: 1.9147
Iteration: 5730; Percent complete: 95.5%; Average loss: 1.8571
Iteration: 5740; Percent complete: 95.7%; Average loss: 1.8788
Iteration: 5750; Percent complete: 95.8%; Average loss: 1.8228
Iteration: 5760; Percent complete: 96.0%; Average loss: 1.8714
Iteration: 5770; Percent complete: 96.2%; Average loss: 1.8544
```

```
Iteration: 5780; Percent complete: 96.3%; Average loss: 1.8524
Iteration: 5790; Percent complete: 96.5%; Average loss: 1.8717
Iteration: 5800; Percent complete: 96.7%; Average loss: 1.8500
Iteration: 5810; Percent complete: 96.8%; Average loss: 1.8666
Iteration: 5820; Percent complete: 97.0%; Average loss: 1.8433
Iteration: 5830; Percent complete: 97.2%; Average loss: 1.8255
Iteration: 5840; Percent complete: 97.3%; Average loss: 1.8458
Iteration: 5850; Percent complete: 97.5%; Average loss: 1.8075
Iteration: 5860; Percent complete: 97.7%; Average loss: 1.8282
Iteration: 5870; Percent complete: 97.8%; Average loss: 1.8654
Iteration: 5880; Percent complete: 98.0%; Average loss: 1.8225
Iteration: 5890; Percent complete: 98.2%; Average loss: 1.7983
Iteration: 5900; Percent complete: 98.3%; Average loss: 1.8047
Iteration: 5910; Percent complete: 98.5%; Average loss: 1.8012
Iteration: 5920; Percent complete: 98.7%; Average loss: 1.8122
Iteration: 5930; Percent complete: 98.8%; Average loss: 1.8057
Iteration: 5940; Percent complete: 99.0%; Average loss: 1.7791
Iteration: 5950; Percent complete: 99.2%; Average loss: 1.8054
Iteration: 5960; Percent complete: 99.3%; Average loss: 1.7852
Iteration: 5970; Percent complete: 99.5%; Average loss: 1.7875
Iteration: 5980; Percent complete: 99.7%; Average loss: 1.8060
Iteration: 5990; Percent complete: 99.8%; Average loss: 1.8011
Iteration: 6000; Percent complete: 100.0%; Average loss: 1.7820
content/beam/cb_model/Chat/2-4_512
```

[28]:
```python
import matplotlib.pyplot as plt

plt.plot(lossvalues)
plt.show()
```



7 Bleu score Calculation for Beam search

[29]:
```
# Set dropout layers to eval mode
encoder.eval()
decoder.eval()

# Initialize search module
from nltk.translate.bleu_score import sentence_bleu,corpus_bleu
from nltk.translate.bleu_score import SmoothingFunction

##############################################################################
# Difference between greedy search and beam search is here

# greedy search
# searcher = GreedySearchDecoder(encoder, decoder)

# beam search
searcher = BeamSearchDecoder(encoder, decoder, voc, 10)
##############################################################################
gram1_bleu_score = []
gram2_bleu_score = []
for i in range(0,len(testpairs),1):

  input_sentence = testpairs[i][0]
```

```
  reference = testpairs[i][1:]
  templist = []
  for k in range(len(reference)):
    if(reference[k]!=''):
      temp = reference[k].split(' ')
      templist.append(temp)


  input_sentence = normalizeString(input_sentence)
  output_words = evaluate(encoder, decoder, searcher, voc, input_sentence)
  output_words[:] = [x for x in output_words if not (x == 'EOS' or x == 'PAD')]
  chencherry = SmoothingFunction()
  score1 = sentence_bleu(templist,output_words,weights=(1, 0, 0, 0)␣
 ↪,smoothing_function=chencherry.method1)
  score2 = sentence_bleu(templist,output_words,weights=(0.5, 0.5, 0,␣
 ↪0),smoothing_function=chencherry.method1)
  gram1_bleu_score.append(score1)
  gram2_bleu_score.append(score2)
  if i%1000 == 0:
    print(i,sum(gram1_bleu_score)/len(gram1_bleu_score),sum(gram2_bleu_score)/
 ↪len(gram2_bleu_score))
print("Total Bleu Score for 1 grams on testing pairs: ", sum(gram1_bleu_score)/
 ↪len(gram1_bleu_score))
print("Total Bleu Score for 2 grams on testing pairs: ", sum(gram2_bleu_score)/
 ↪len(gram2_bleu_score))
```

```
0 0.13406400920712788 0.04739878501170794
1000 0.15674150177584226 0.07069958145080933
2000 0.14601108728119894 0.06506785101638089
3000 0.14666470225263117 0.06565489601654281
4000 0.14810825443924747 0.06559820531880635
5000 0.14754701043529136 0.06574311412989854
6000 0.14687792357817048 0.06503098534501695
7000 0.15310102450351762 0.07107164419517663
8000 0.15160559356415987 0.07011479924099687
Total Bleu Score for 1 grams on testing pairs:  0.15082862204169686
Total Bleu Score for 2 grams on testing pairs:  0.06961184299487612
```

[30]:
```
encoder.eval()
decoder.eval()
searcher = BeamSearchDecoder(encoder, decoder, voc, 10)
evaluateInput(encoder, decoder, searcher, voc)
```

```
Bot: it s me .
Bot: just fine i m fine .
Bot: of course .
```

```
Bot: thank you .
Bot: i love you .
Bot: i don t think so .
Bot: i don t want to talk about it .
Bot: i m sorry .
Bot: i don t want to know
Bot: hello what are you doing here ?
Bot: what ?
Bot: yeah i think so .
Bot: i want to go back to


      ␣
↪---------------------------------------------------------------------------

       KeyboardInterrupt                         Traceback (most recent call␣
↪last)

       /usr/local/lib/python3.6/dist-packages/ipykernel/kernelbase.py in␣
↪_input_request(self, prompt, ident, parent, password)
       883             try:
   --> 884                 ident, reply = self.session.recv(self.stdin_socket,␣
↪0)
       885             except Exception:


       /usr/local/lib/python3.6/dist-packages/jupyter_client/session.py in␣
↪recv(self, socket, mode, content, copy)
       802         try:
   --> 803             msg_list = socket.recv_multipart(mode, copy=copy)
       804         except zmq.ZMQError as e:


       /usr/local/lib/python3.6/dist-packages/zmq/sugar/socket.py in␣
↪recv_multipart(self, flags, copy, track)
       474         """
   --> 475         parts = [self.recv(flags, copy=copy, track=track)]
       476         # have first part already, only loop while more to receive


       zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.recv()


       zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.recv()


       zmq/backend/cython/socket.pyx in zmq.backend.cython.socket._recv_copy()
```

```
       /usr/local/lib/python3.6/dist-packages/zmq/backend/cython/checkrc.pxd in␣
↪zmq.backend.cython.checkrc._check_rc()


       KeyboardInterrupt:


   During handling of the above exception, another exception occurred:


       KeyboardInterrupt                         Traceback (most recent call␣
↪last)

       <ipython-input-30-72fb9220f3db> in <module>
         2 decoder.eval()
         3 searcher = BeamSearchDecoder(encoder, decoder, voc, 10)
   ----> 4 evaluateInput(encoder, decoder, searcher, voc)


       <ipython-input-19-ec18c2cf8348> in evaluateInput(encoder, decoder,␣
↪searcher, voc)
        21         try:
        22
   ---> 23             input_sentence = input('> ')
        24
        25             if input_sentence == 'q' or input_sentence == 'quit':␣
↪break


       /usr/local/lib/python3.6/dist-packages/ipykernel/kernelbase.py in␣
↪raw_input(self, prompt)
       857             self._parent_ident,
       858             self._parent_header,
   --> 859             password=False,
       860         )
       861


       /usr/local/lib/python3.6/dist-packages/ipykernel/kernelbase.py in␣
↪_input_request(self, prompt, ident, parent, password)
       887             except KeyboardInterrupt:
       888                 # re-raise KeyboardInterrupt, to truncate traceback
   --> 889                 raise KeyboardInterrupt
       890             else:
       891                 break
```

```
        KeyboardInterrupt:
```

[ ]:

[ ]:

[ ]: