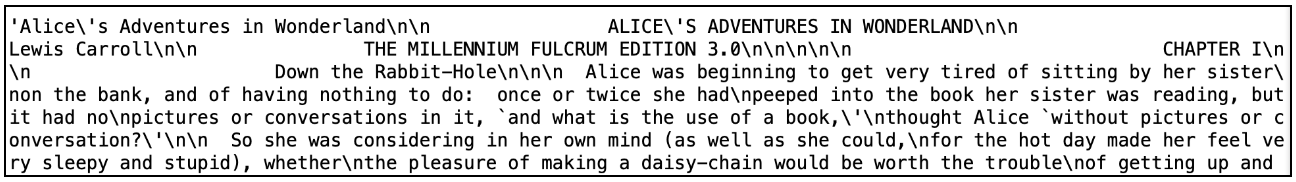
Aim

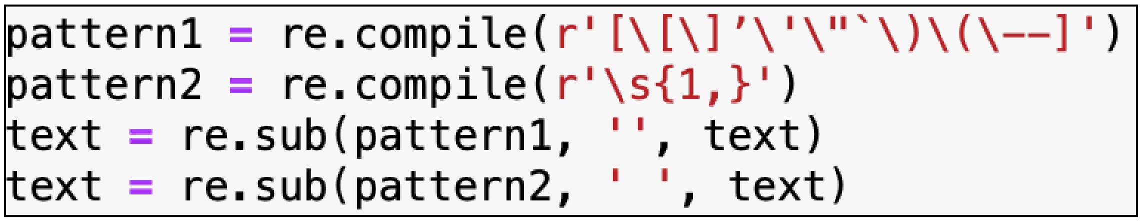
Using the provided code, tokenize your chosen text into words and train the recurrent network model to predict the next word in the text. The main emphasis in the work should be on methods of working with text and ways of encoding a word at the input to the model.

Data prettifying and analysis

As a text, a fragment from ‘Alice In Wonderland’ was chosen. In its raw view, it looks dirty, with tons of unnecessary symbols, so Python regular expressions library ‘re’ can solve this problem.

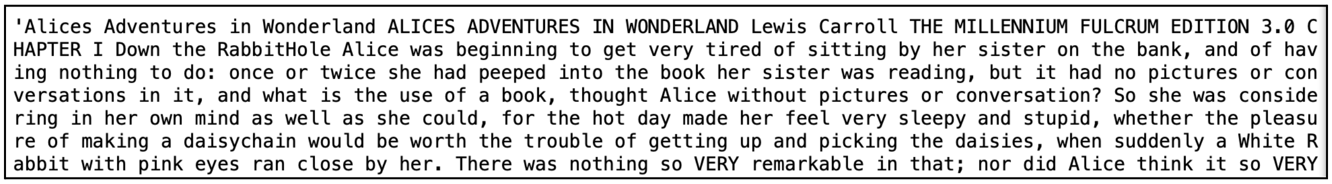


```
'Alice\'s Adventures in Wonderland\n\n                 ALICE\'S ADVENTURES IN WONDERLAND\n\n
Lewis Carroll\n\n                THE MILLENNIUM FULCRUM EDITION 3.0\n\n\n\n\n                              CHAPTER I\n
\n                          Down the Rabbit-Hole\n\n\n  Alice was beginning to get very tired of sitting by her sister\
non the bank, and of having nothing to do:  once or twice she had\npeeped into the book her sister was reading, but
it had no\npictures or conversations in it, `and what is the use of a book,\'\nthought Alice `without pictures or c
onversation?\'\n\n  So she was considering in her own mind (as well as she could,\nfor the hot day made her feel ve
ry sleepy and stupid), whether\nthe pleasure of making a daisy-chain would be worth the trouble\nof getting up and
```



```
pattern1 = re.compile(r'[\[\]’\'\"`\)\(\--]')
pattern2 = re.compile(r'\s{1,}')
text = re.sub(pattern1, '', text)
text = re.sub(pattern2, ' ', text)
```

Here we have two patterns, the first one removes all the punctuation marks and quotes, the second gets rid of unimportant symbols like ‘\n’, ‘ ’, ‘\u’ symbols for our NLP process.

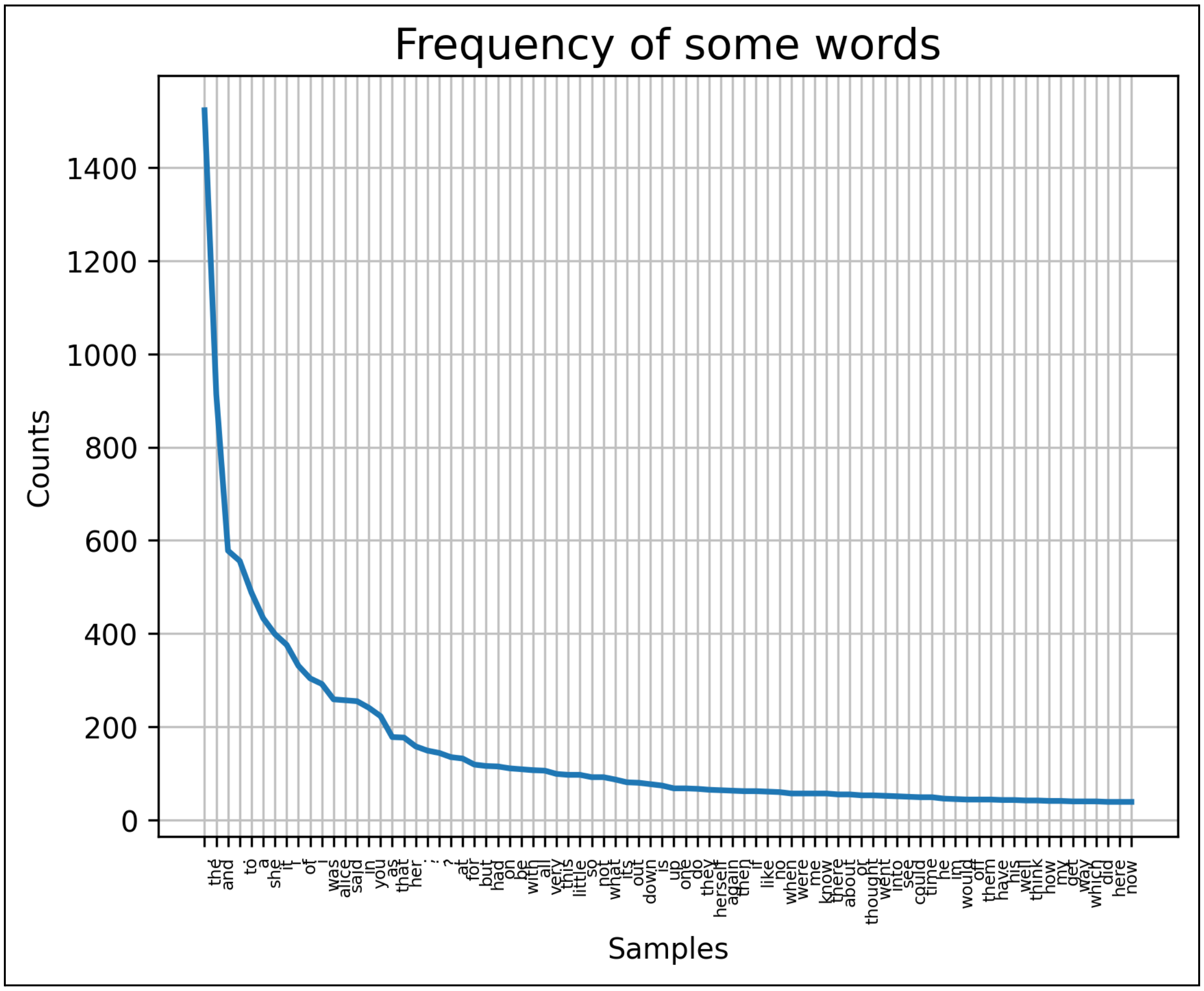


```
'Alices Adventures in Wonderland ALICES ADVENTURES IN WONDERLAND Lewis Carroll THE MILLENNIUM FULCRUM EDITION 3.0 C
HAPTER I Down the RabbitHole Alice was beginning to get very tired of sitting by her sister on the bank, and of hav
ing nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or con
versations in it, and what is the use of a book, thought Alice without pictures or conversation? So she was conside
ring in her own mind as well as she could, for the hot day made her feel very sleepy and stupid, whether the pleasu
re of making a daisychain would be worth the trouble of getting up and picking the daisies, when suddenly a White R
abbit with pink eyes ran close by her. There was nothing so VERY remarkable in that; nor did Alice think it so VERY
```

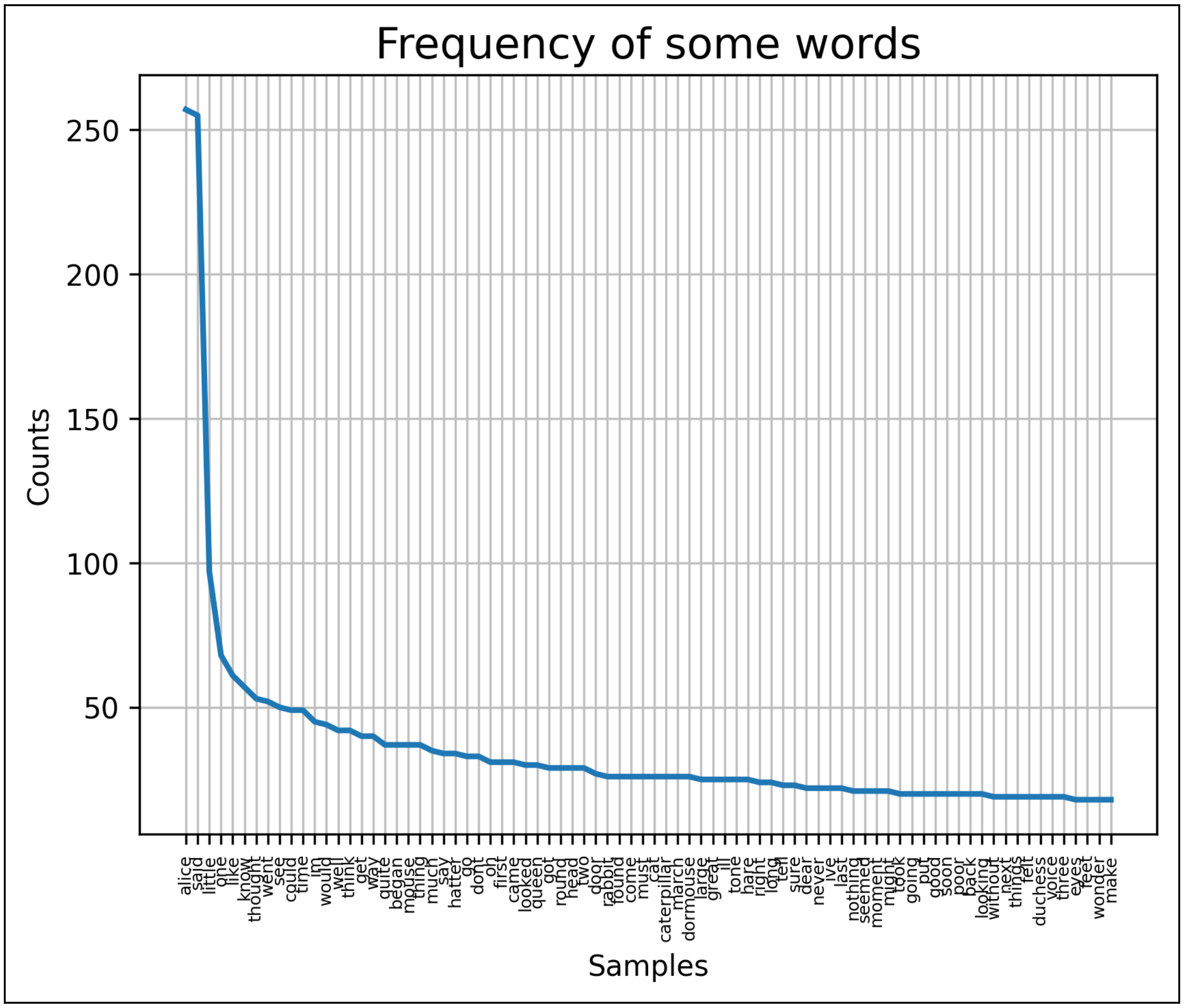
Having our NLTK library for processing and analyzing text, it’s worth it to see the frequency distribution of the words present. For this particular case, first 20000 words (symbols) from the text are taken, which form a quite representable fragment. To make a job more effective, a word tokenization algorithm is used from the NLTK library. It connects an unique numerical value to each text object (word or punctuation mark), so the processing is done with an array of arrays (id, object) instead of plain string.

```
word_tokens = nltk.word_tokenize(text.lower())[0:20000]
```

The first plot represents a frequency of most 50 popular text objects.

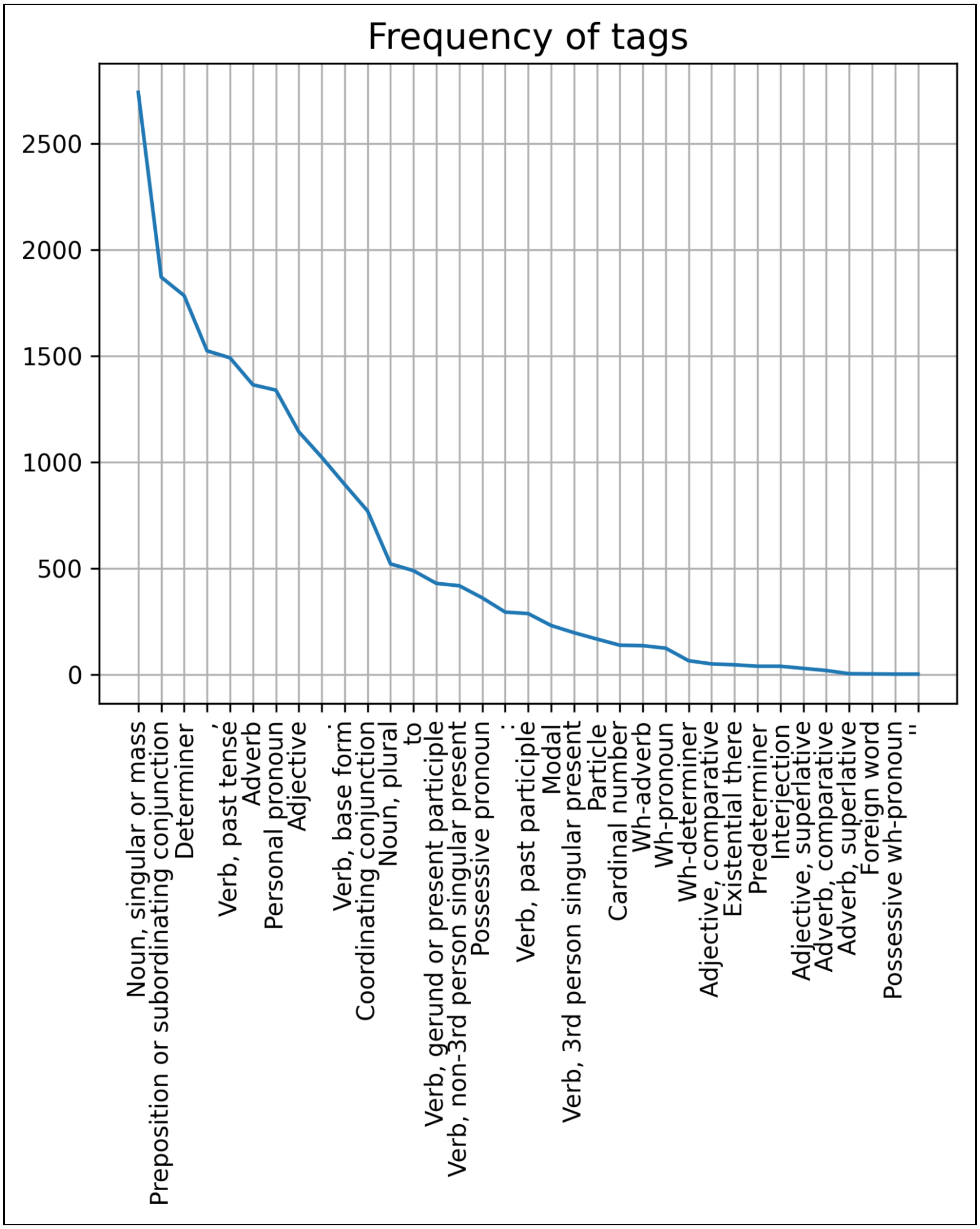


It’s clearly noticeable that the most popular one is the punctuation mark ‘,’, which occurred more than 1500 times. Other samples to the right, not surprisingly, mostly are pronouns, prepositions, conjunctions and articles. The next step is to remove some unimportant elements from text like those, which are the most frequent on our chart.



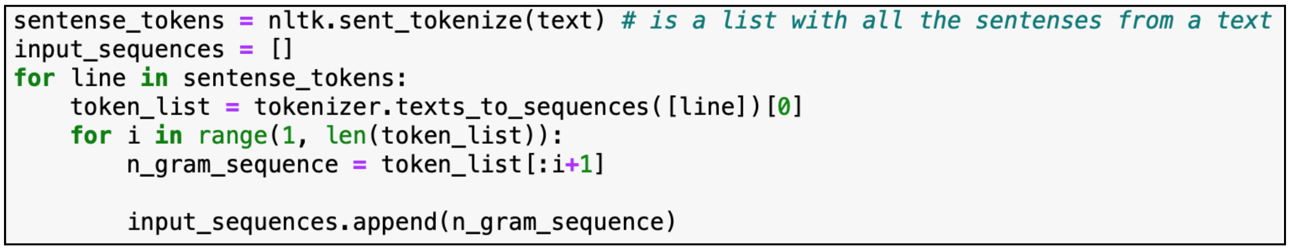
It was expected that in a fairytale about Alice, the most popular word will be ‘alice’. Other informative tokens are ‘said’, ‘little’, ‘one’, ‘like’, ‘know’, which values are extremely high - 257, 255, 97,68, unlike other ~99%, which is under 50.

The next graph represents a numeric occurrence value of different types of words. Right here, the view is a bit different. Three most popular word types are noun, conjunction and determiner.

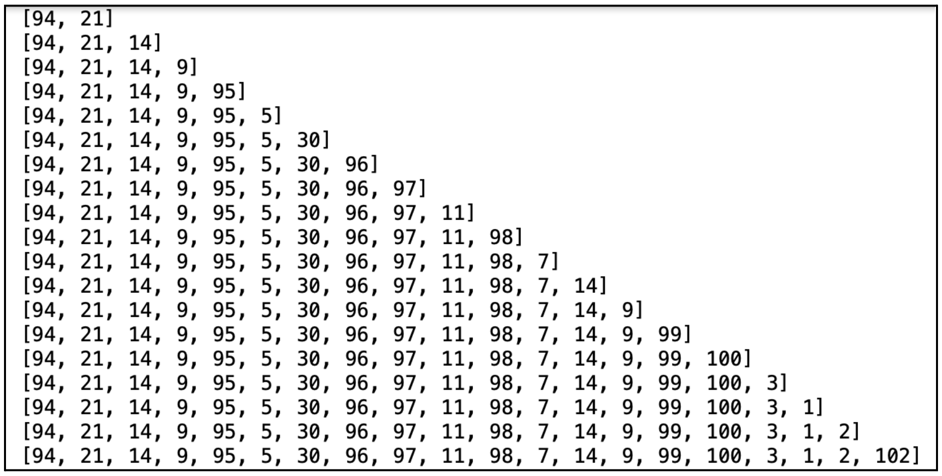

Frequency of tags
2500
2000
1500
1000
500
0
Noun, singular or mass
Preposition or subordinating conjunction
Determiner
,
Verb, past tense
Adverb
Personal pronoun
Adjective
.
Verb, base form
Coordinating conjunction
Noun, plural
to
Verb, gerund or present participle
Verb, non-3rd person singular present
Possessive pronoun
:
Verb, past participle
Modal
Verb, 3rd person singular present
Particle
Cardinal number
Wh-adverb
Wh-pronoun
Wh-determiner
Adjective, comparative
Existential there
Predeterminer
Interjection
Adjective, superlative
Adverb, comparative
Adverb, superlative
Foreign word
Possessive wh-pronoun
''

Data preparing

As our aim is to build a ‘next word’ predictor, firstly we need to determine how the data will be given to a model, and how to represent it. Here, again, some of the NLTK mechanisms are needed. The approach is in representing words as some unique values, and afterwards sentences will be nothing but the sequence of those values. For example if we have such a ‘vocabulary’ data structure as {child:343, loves:4, cat:64, my:”516}, the sentence ‘my child loves my cat’ can be determined as the array [516, 343, 4, 516, 64]. As an option, an array of tokenized words (a sentence) without the last word could be our train data and the last word is an expected output. Not one full sentence participates, but its ‘n_gram’ sequences - all the possible sequences from the sentence, where their length is between 2 and a length of the sentence - starting from a group of the first two words, ending by a full sentence.



```
sentense_tokens = nltk.sent_tokenize(text) # is a list with all the sentenses from a text
input_sequences = []
for line in sentense_tokens:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]

        input_sequences.append(n_gram_sequence)
```



```
[94, 21]
[94, 21, 14]
[94, 21, 14, 9]
[94, 21, 14, 9, 95]
[94, 21, 14, 9, 95, 5]
[94, 21, 14, 9, 95, 5, 30]
[94, 21, 14, 9, 95, 5, 30, 96]
[94, 21, 14, 9, 95, 5, 30, 96, 97]
[94, 21, 14, 9, 95, 5, 30, 96, 97, 11]
[94, 21, 14, 9, 95, 5, 30, 96, 97, 11, 98]
[94, 21, 14, 9, 95, 5, 30, 96, 97, 11, 98, 7]
[94, 21, 14, 9, 95, 5, 30, 96, 97, 11, 98, 7, 14]
[94, 21, 14, 9, 95, 5, 30, 96, 97, 11, 98, 7, 14, 9]
[94, 21, 14, 9, 95, 5, 30, 96, 97, 11, 98, 7, 14, 9, 99]
[94, 21, 14, 9, 95, 5, 30, 96, 97, 11, 98, 7, 14, 9, 99, 100]
[94, 21, 14, 9, 95, 5, 30, 96, 97, 11, 98, 7, 14, 9, 99, 100, 3]
[94, 21, 14, 9, 95, 5, 30, 96, 97, 11, 98, 7, 14, 9, 99, 100, 3, 1]
[94, 21, 14, 9, 95, 5, 30, 96, 97, 11, 98, 7, 14, 9, 99, 100, 3, 1, 2]
[94, 21, 14, 9, 95, 5, 30, 96, 97, 11, 98, 7, 14, 9, 99, 100, 3, 1, 2, 102]
```

However the input layer shape of our model can’t be varied, so pad sequences can resolve that. We measure the length of the longest sentence and then lengthen every one to that number, inserting zeros instead of ‘empty values’.

```
# pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))
```

```
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36
2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37
3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38
4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39
5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40
6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18
7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41
8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1
9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43
2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2
3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44
4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9
5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45
6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45,3
7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45,3,46
8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45,3,46,19
9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45,3,46,19,4
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45,3,46,19,4,20
1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45,3,46,19,4,20,47
2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45,3,46,19,4,20,47,1
3,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45,3,46,19,4,20,47,1,48
4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45,3,46,19,4,20,47,1,48,10
5,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45,3,46,19,4,20,47,1,48,10,3
6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45,3,46,19,4,20,47,1,48,10,3,49
7,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45,3,46,19,4,20,47,1,48,10,3,49,21
8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45,3,46,19,4,20,47,1,48,10,3,49,21,2
9,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,15,16,5,17,15,16,5,17,33,34,1,35,36,37,38,39,40,18,41,1,42,11,43,2,44,9,45,3,46,19,4,20,47,1,48,10,3,49,21,2,5
```

And finally out training data and labels are separated the way discussed before.

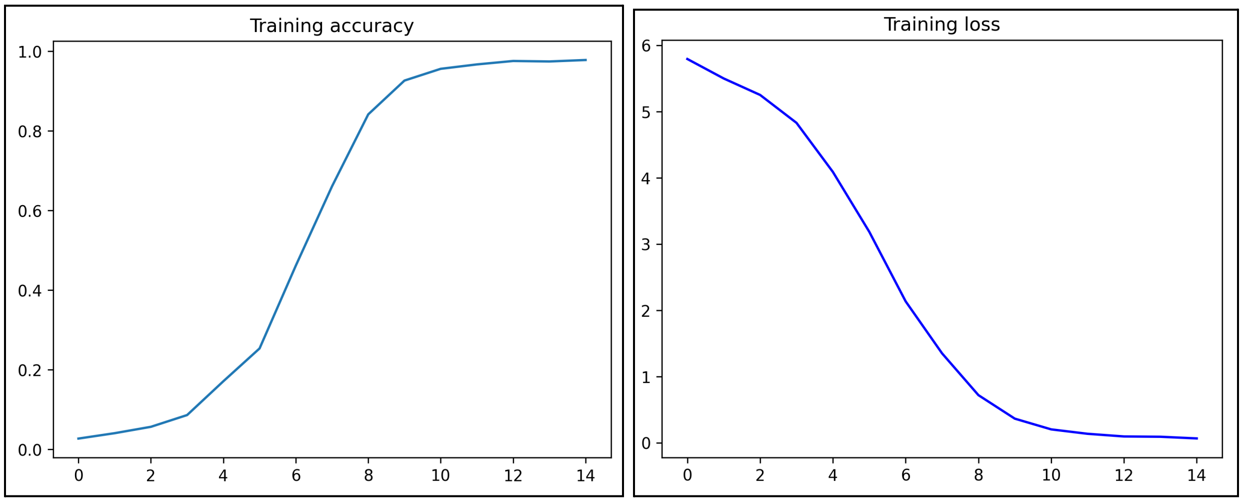
```
# create predictors and label
xs, labels = input_sequences[:,:-1],input_sequences[:,-1]

ys = to_categorical(labels, num_classes=total_words)
```

Model

As a model it is customary to use a recurrent one. We create a sequential model with the first layer as the word embedding layer. And then applying bidirectional LSTM, where parameter return_sequence is marked as True so that the word generation keeps in consideration, previous and even the words coming ahead in the sequence. The output layer has softmax so as to get the probability of the word to be predicted next. ‘Adam’ was chosen as the optimizer, learning rate is 0.01 this time.

```
model = Sequential()
model.add(Embedding(total_words, 100, input_length=max_sequence_len-1))
model.add(Bidirectional(LSTM(150, return_sequences = True)))
model.add(Dense(total_words, activation='softmax'))
adam = Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])
history = model.fit(xs, ys, epochs=15, verbose=1)
```

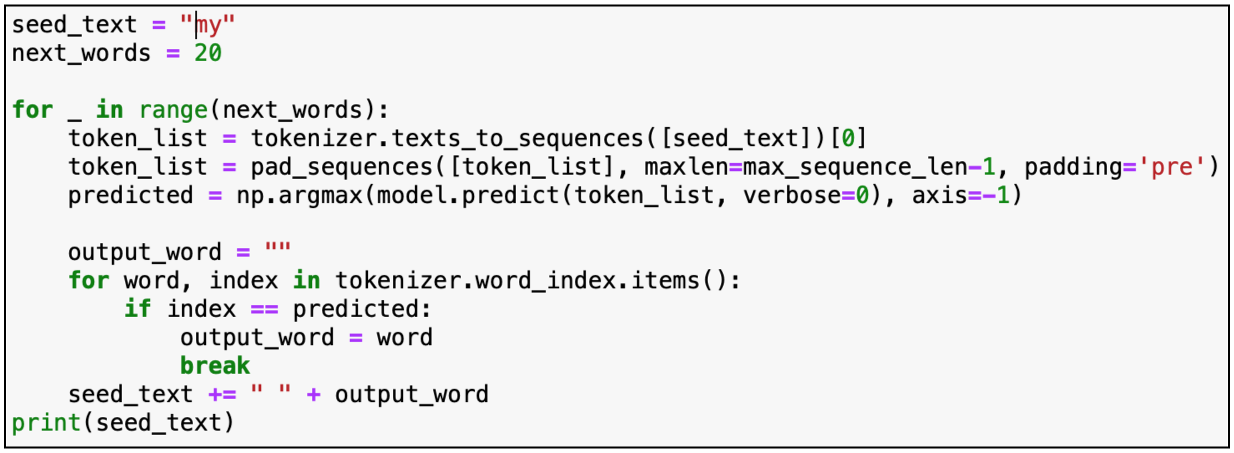
15 was chosen as the number of epochs for training. Following pictures show the result.



On epoch number 8 our model performed with an accuracy of ~0.8, later - more, which is a quite good result. A final step is to test our model and ask to generate some text.

Evaluation

Here, we tokenize our input (seed) text to a sequence of numbers, padding it. We give this array to a ‘model.predict()’ function which outputs a list of probabilities of all the words being next, where we chose one with the highest probability. We take a numeric value and find a word with the help of a ‘word_index’ dictionary, where unique words and their unique values are stored. We add a new word to our existing seed text.



```
seed_text = "my"
next_words = 20

for _ in range(next_words):
    token_list = tokenizer.texts_to_sequences([seed_text])[0]
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
    predicted = np.argmax(model.predict(token_list, verbose=0), axis=-1)

    output_word = ""
    for word, index in tokenizer.word_index.items():
        if index == predicted:
            output_word = word
            break
    seed_text += " " + output_word
print(seed_text)
```

“alice had been to the seaside once in her life and had come to the general conclusion that wherever you go to on the english coast you find a number of bathing machines in the sea some children digging in the sand with wooden spades then a row of lodging houses and behind them a station and hot buttered toast she very soon finished it off the cake to see what coming to herself rather commotion in the pool as it went to get very tired of sitting by her sister on the bank and of having nothing to do once”

“I must be getting somewhere near the center of the earth but the rabbit no longer to be seen she found herself in a long low hall which lit up by a row of lamps hanging from the roof of the hall in fact she now more than nine feet high and she at once took up the little golden key and hurried off to the garden door she sadly down the middle wondering how she ever to get out again with a little eyes but it too slippery and when she had tired herself out with trying the poor little”

- is the text which was generated from seed words ‘alice’ and ‘I’. It’s not perfect, we can see a similarity to the initial one, sometimes it looks weird but it’s not hard to get a sense of it.

It’s worth lemmatizing.

“she said aloud a or hippopotamus but then she remembered how small she now and she soon made out that it only a mouse that had slipped in like herself in a dreamy sort of do cat eat bat i wonder what i should be like then a great hurry to change the subject of conversation and all dripping wet cross and uncomfortable she felt certain it must be really offended of the and she ran across the field after it and fortunately just in time to see it pop down a large rabbithole under the hedge of his tail and”

- It's more uncomfortable to read, but it’s easier for a machine to learn and make word embeddings. While we had 972 words after tokenizing, applying lemmatizing, we got 863 - some of the words became exactly similar after reduction to the initial form.

To sum up, we preprocessed our data, cleaned it. We have used a recurrent neural network model to build a word predictor. We have encoded our sentences, as a sequence of numbers, each word having its own. The architecture of NN is trivial but quite good results can be seen while evaluating. We haven’t got a ‘new’ text, but some pieces of the initial one were reproduced.