TP3 - Zipf's Law and Word2Vec Model

Pablo Strasser
e-mail: Pablo.Strasser@unige.ch
Exercice based on Michal Muszynski.

March 25, 2019

1 Goal

1.1 Zipf's Law

In the first part of this exercise we study the statistics of word occurrences in texts. An observation made by George Kingsley Zipf (1902-1950) states that, in general:

- A small number of words occur very frequently
- Many words occur rarely

More formally, the above Zipf's law states that the *probability* of encountering the r-th most common word is inversely proportional to its rank r, that is,

$$P(r) \approx 0.1/r. \tag{1}$$

For example, according to the above equation, the most frequently occurred word w_1 with $r = 1$ has a probability of $P(1) \approx 0.1$, meaning that roughly one of every ten words is w_1. The second most frequent word has a probability $P(2) \approx 0.05$, meaning that one of every twenty word is w_2, and so on.

Terminologies

Frequency (f) number of occurrences of a word in a text

Corpus a collection of documents of a particular kind

Rank of a word (r) a word's ordinal number in a list sorted by decreasing frequency (f)

Vocabulary set of all unique words in a corpus

1.2 Word2Vec Model

In the second part of this exercise we become familiar with the concept of Word2Vec models. Let's start with an idea that we train a simple neural network with one hidden layer to perform a certain task. We train the neural network in the following way. We select a specific word in the middle of a sentence (the input word), then we look at the words nearby and pick one randomly. The trained neural network is able to tell us the probability for every work in our vocabulary of being the nearby word that we chose. The output probability is related to how likely each

vocabulary word is nearby our input word. To do so, we train the neural network by feeding it word pairs found in our training documents. The below example shows some of the training samples that we would take from our dataset. Let's consider the sentence "The quick brown fox jumps over the lazy dog."as an example of the dataset. We first form a dataset of words and the contexts in which they appear. We could define context in any way that makes sense, for example, words-to-the-left of the target, words-to-the-right of the target. Using a window size of 1, we then have the dataset:
`([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox), ...`
of (context, target) pairs. Let's recall that we attempt to predict each context word from its target word, so the task becomes to predict 'the' and 'brown' from 'quick', 'quick' and 'fox' from 'brown', etc. Therefore our dataset becomes
`(quick, the), (quick, brown), (brown, quick), (brown, fox), ...`
of (input, output) pairs.
However, we can not feed the neural network words as a text string, so we need find a numerical representation of words. To do this, we select a vocabulary of words from our training documents, e.g. 10000 unique words. Then, we represent each input word as a one-hot vector (one-hot encoding). Each vector would have 10000 elements (one for every word in our vocabulary) in which "1" is placed in the position corresponding to the given word and and "0" in all of the other positions. The output of the network is a single vector also with 10,000 components for every word in our vocabulary. We train our network to compute $P(context|input)$. Let denote by x the input and by c the target (context). Let denote by h_x and h_c the representation of the target and input. The representation is computed by:

$$h_x = Wx$$
$$h_c = Wc$$

Where the dimension of W is (Embedding size,Vocabulary size). The probability model is then given by:

$$P(c|x) = \frac{\exp(h_c \cdot h_x)}{\sum_i \exp(h_i \cdot h_x)} \tag{2}$$

Our goal is then to maximize the average likelihood on the whole corpus. The likelihood is defined as:

$$\log(P(c|x)) \tag{3}$$

Once done, the vector embedding can be read directly on the w matrix.

2 Tasks

2.1 Verify the Zipf's Law

1. Choose one or more French books as your corpus from the project Guttenberg web site `http://www.gutenberg.org/`, for example
   - "20000 Lieues Sous Les Mers" `http://www.gutenberg.org/ebooks/5097`
   - "Les Trois Mousquetaires" `http://www.gutenberg.org/ebooks/13951`
   - "L'homme Qui Rit" `http://www.gutenberg.org/ebooks/5423`
   - "L'Odyssée" `http://www.gutenberg.org/ebooks/14286`

- "Histoire de la Révolution franaise" `http://www.gutenberg.org/ebooks/9945`

2. Verify the Zipf's Law. For this you need to:

   - Identify all unique words in your corpus. One way to do this is to tokenize your corpus by splitting based on white space characters. If a token match a predefined regular expression, then memorize it as a valid word. This is for filtering non-word tokens like "****", "—", etc.
   - Count the frequencies of all words in your corpus and arranges them in a list according to their rank. You may program in `Python`, `Matlab`, `Perl`, etc.
   - Transform the frequencies into probabilities by normalizing each frequency using the total number of words in your corpus. On the same diagram, plot the obtained probabilities against their ranks, the theoretical relationship according to the formula of Zipf's law and a linear regression (least-squares fit) line. Please report the values of R-squared, p-value and attach a residual plot for the linear regression. Justify whether linear regression models can be used to explore the dependence between words' probabilities and their ranks. Comment on how the approximation fits the theoretical formula.

3. From the data you obtained, find 10 examples of extremely frequent, very rare, and averagely frequent words to fill out the following table

	Very Frequent Words	Averagely Frequent Words	Very Rare Words
1			
2			
⋮			
10			

   Intuitively, which of the above three word categories could be more useful in information retrieval? Which of these categories is likely to have large `tf-idf` values? Why?

2.2 Word2Vec Model

- Encode the selected unique words in your corpus using one-hot encoding.
- Generate a dataset of word pairs by means of a window size of 1 (words-to-the-left of the target, words-to-the-right of the target).
- Train a neural network on this dataset. To train the neural network you can use Pytorch or Tensorflow. An introduction to Tensorflow and Pytorch will be given at the exercice session.
- Plot the training curve (likelihood/epoch) and give the running time per epoch.
- If possible give the running time per epoch on GPU and compare with respect to a CPU.
- Find 5 most likely nearby words of 10 extremely frequent words in your corpus (task 2.1) and summarize them in a table.

3 Assessment

The assessment is based on your report. It should include all experimental results, your answers to all questions, and your analysis and comments of the experimental results. It should be around 5 pages for this assignment. Please try to detail the report by giving examples and conclusions. Please archive your report and codes in "PrenomNomTP3.zip", and upload to `http://moodle.unige.ch` under TP3 before Monday, April 8, 2019.