TP4 - Word Embeddings

Atul Sinha
e-mail: Atul.Sinha@unige.ch

April 19, 2021

1 Goal

1.1 Word2Vec Model

In 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 word 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 ($m = 1$), we then have the dataset:
`([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox), ...`
of (context, target) pairs. In the SkipGram model, we attempt to predict the context words 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
`D : (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 V from our training documents, e.g. 10000 unique words ($|V| = 10000$). 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 $P(context|target)$. Let us denote the input by x (target word), the output by c (context word), and h_x and h_c as the representation (embedding) of the target and context word. The representation is computed by:

$$h_x = Wx$$

$$h_c = W'c$$

where W and W' are matrices of dimension (Embedding size $\times$ Vocabulary size) and the learnable parameters of the model. The embedding size is a hyperparameter. The probability model is then given by:

$$P(c|x) = \frac{\exp(h_c \cdot h_x)}{\sum_{i=1}^{|V|} \exp(h_i \cdot h_x)}$$

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

$$\begin{aligned}
L &= \lambda \sum_x \log(P(c-m, \ldots, c-1, c+1, \ldots c+m|x)) \\
&= \lambda \sum_x \log(\prod_{j=0, j\neq m}^{2m} P(c-m+j|x)) \\
&= \lambda \sum_x \sum_{j=0, j\neq m}^{2m} \log P(c-m+j|x)) \\
&= \lambda \sum_{(c,x)\in \mathtt{D}} \log P(c|x)
\end{aligned}$$

The training is done using stochastic gradient descent (SGD) or mini-batch SGD. After training, the embeddings can be extracted from the W and W' matrices as $e_x = Wx$ or $e_x = (W + W')x/2$

1.2 Negative Sampling

Evaluating the denominator in the expression of $P(c|x)$ for each training step is very costly as we need to evaluate it for each word in the vocabulary. This will update all rows of E' at every step by simultaneously making the embedding for x closer to c and farther from all other terms $c' \neq c, c' \in V$ together. This is a characteristic of the softmax function. Negative sampling is an approximation to the above optimisation problem (or any multi-class classification problem) where at each step, we only update one row of the classifier E' (corresponding to c). The probability model is modified as :

$$P(c|x) = \sigma(h_c \cdot h_x) = \frac{1}{1 + e^{-h_c \cdot h_x}}$$

where $\sigma(.)$ is the sigmoid function. The goal would be to predict high probabilities (=1) for $(c, t) \in \mathtt{D}$. This essentially brings the embedding for x closer to c but does not push it away from the other words in the vocabulary. Thus we need to add negative samples to enforce that. For each target word, we sample a fixed number n of negative examples uniformly from the vocabulary (excluding the target word itself). Let's consider the previous example : "The quick brown fox jumps over the lazy dog." Using a window size of 1, we got the dataset:
`([the, brown], quick), ([quick, fox], brown), ([brown, jumped], fox), ...`
of (context, target) pairs. The negative examples for `quick` after sampling from the whole vocabulary could be `[jumps, lazy]`. Further we can attach labels 1 for positive examples and 0 for negative examples, i.e. we want to predict high probability values for positive examples and low probability values for negative examples and get the following dataset : `D : ((quick, the),1), ((quick, brown), 1), ((quick, jumps), 0), ((quick, lazy), 0), ((brown, quick), 1),`

```
((brown, fox), 1) ...
```

We can rewrite the probability model to also include the labels as :

$$P(y = 1|(c, x)) = \sigma(h_c \cdot h_x) = \frac{1}{1 + e^{-h_i \cdot h_x}}$$

The likelihood can then be written as :

$$\begin{aligned} L &= \lambda \sum_{((x,c),y)\in D} \log P(y|(c, x)) \\ &= \lambda \sum_{((x,c),y)\in D} y \log P(y|(c, x)) + (1 - y) \log(1 - P(y|(c, x))) \end{aligned}$$

which is the familiar cross entropy loss function used for binary classification.

2 Tasks

2.1 SkipGram Model

- 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`
- Generate a dataset of word pairs by means of a window size of 2 (two words-to-the-left of the target, two words-to-the-right of the target).
- Train a neural network on this dataset using the SkipGram model. You can use Pytorch or Tensorflow.
- 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 (TP3 #2.2.3) and summarize them in a table.

2.2 Negative Sampling

- Train another neural network on the same dataset with negative sampling.
- Plot the training curve and compare it with the above model.
- Find 5 most likely nearby words of 10 extremely frequent words, as before.

Submission deadline

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 4 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 TP4 before May 2, 2021, 23:59PM. Later submission will not be accepted.