Word Embedding

L.J. Brown

September 28, 2017

We outline our first attempt at a method to map all unique words found in a corpus to vectors in a continuous vector space. The hope is that some relationships between words found in the corpus will be preserved through this mapping and will manifest as characteristics of the vector space. The method involves building a co-occurrence matrix from the corpus that represents how frequently word pairs occur together. We then search for word vectors with the soft constraint that given a word vector pair, their inner product will yield a value close to the two values in the co-occurrence matrix associated with those two words. We do this using Stochastic Gradient Descent, which draws heavily on the implementations by ”Word2vec” and ”GloVe”, and then with experimental methods using Eigen Decomposition and Singular Value Decomposition to compare results.

We first map all n unique words found in the corpus, $W = \{w_1, \dots, w_n\}$, to integers, $I = \{1, \dots, n\}$, using a function f.

$$n \equiv \text{Number of unique words in corpus, } W \equiv \text{Set of unique words in corpus}$$

$$W = \{w_1, \dots, w_n\}$$

$$I = \{1, \dots, n\}$$

$$f : W \leftrightarrow I$$

Next we build an $n \times n$ cooccurrence matrix, $\boldsymbol{A}_{n\times n}$, whose rows and columns correspond to the set of unique words, W, and whose elements are the output of a cooccurrence function, $\rho(w_i, w_j)$, which takes words as inputs. For each element, $a_{r_i c_j}$, the words used as inputs for ρ are the words whose integer mappings correspond to the elements row, r_i, and column, c_j, ($f(r_i) = w_i$ and $f(c_j) = w_j$).

$$\boldsymbol{A}_{n\times n} = \begin{array}{c} \\ r_1 \\ r_2 \\ \vdots \\ r_4 \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & \dots & c_n \end{array} \\ \begin{pmatrix} \rho(f(r_1), f(c_1)) & \rho(f(r_1), f(c_2)) & \dots & \rho(f(r_1), f(c_n)) \\ \rho(f(r_2), f(c_1)) & \rho(f(r_2), f(c_2)) & \dots & \rho(f(r_2), f(c_n)) \\ \vdots & \vdots & \ddots & \vdots \\ \rho(f(r_n), f(c_1)) & \rho(f(r_n), f(c_2)) & \dots & \rho(f(r_n), f(c_n)) \end{pmatrix} \end{array}$$

$$\text{where } a_{ij} = \rho(w_i, w_j)$$

Next we define the cooccurrence function, $\rho(w_i, w_j)$, by first introducing some new variables. For each of the m sentences in the corpus we define s_i as the sequence of words in that sentence. Where S is a sequence containing the m sequences, $s_i, \ldots, s_m$, corresponding to the m sentences of the corpus.

$$S = (s_1, \ldots, s_m)$$

For example if the i^{th} sentence in the corpus is: "God made mud.", then the corresponding sequence, s_i, in S would be

$$s_i = (\tilde{w_1}, \tilde{w_2}, \tilde{w_3})$$

$$\text{where } \tilde{w_1}, \tilde{w_2}, \tilde{w_3} \,\epsilon\, W$$

Then we define a distance function, $d(s_i, \{\tilde{w_j}, \tilde{w_k}\})$, whose parameters are a sequence of words, s_i, in S (corresponding to a sentence of the corpus) and a set or an unordered pair of words that are members of the sequence s_i. The distance function, $d(s_i, \{\tilde{w_j}, \tilde{w_k}\})$, returns a value one more than the number of words between the word pair, $\{\tilde{w_j}, \tilde{w_k}\}$, in sentence corresponding to the sequence, s_i.

$$d(s_i, \{\tilde{w_j}, \tilde{w_k}\}) =\mid j - k \mid$$

Finally we define the cooccurrence function, $\rho(\tilde{w_i}, \tilde{w_j})$, as

$$\rho(w_i, w_j) = \sum_{s \epsilon S} \sum_{p \epsilon \{[s]^2\}} \begin{cases} \log\left(\frac{1}{d(s,p)}\right), \text{ if } w_i, w_j \,\epsilon\, p \, \text{and if } w_i \neq w_j & , otherwise \\ 0, \text{ otherwise} \end{cases}$$

Where $\{[s_i]^2\}$is the set of all unordered pairs of words in the sequence s_i.

Stated simply, $\rho(w_i, w_j)$ finds all times that the words w_i and w_j appear together in sentences of the corpus, and sums the $\log\left(\frac{1}{\text{their distance apart}}\right)$. Note that the value of the cooccurrence function, $\rho(w_i, w_j)$, when the two inputs are the same word is zero. This corresponds to the diagonal entries of the cooccurrence matrix $\boldsymbol{A}$. The matrix $\boldsymbol{A}$ will also be symmetric about its diagonal due to the property of $\rho(w_i, w_j)$ that,

$$\rho(w_i, w_j) = \rho(w_j, w_i)$$

therefore,

$$a_{ij} = a_{ji}, \text{ in the matrix } \boldsymbol{A}.$$

Next we search for a set of n word vectors, $\vec{w_i}, \ldots, \vec{w}_n$, of dimensionality v that best satisfy the condition that the inner product between any two word vectors is a value close to the two elements in matrix $\boldsymbol{A}$ corresponding to the two words intersection. We exclude the diagonal elements from meeting this condition which correspond to the intersection of one word with itself. We will represent the desired n word vectors as the column vectors of a matrix $\boldsymbol{W}_{v \times n}$.

$$\boldsymbol{W}_{v\times n} = \left(\left[\vec{w}_1\right], \ldots, \left[\vec{w}_n\right]\right)$$

In order to formally define the properties of the word vectors we wish to find, we write a set of $\frac{n^2-n}{2}$ soft constraints, $C = \{c_{12}, c_{13}, \cdots, c_{\binom{n}{2}}\}$, as

$$c_{ij} \equiv \vec{w}_i^T \cdot \vec{w}_j :\Leftrightarrow a_{ij}, \text{ where } i \neq j.$$

where $\dfrac{n^2-n}{2}$ is the number of non-diagonal elements in the matrix $\boldsymbol{A}$.

However, order does not matter for the constraints when taking the inner product of two word vectors as $\boldsymbol{A}$ is symmetric about its diagonal and

$$\vec{w}_i^T \cdot \vec{w}_j = \vec{w}_j^T \cdot \vec{w}_i$$

$$a_{ij} = a_{ji}$$

We define an error function, ε, between two word vectors, $\vec{w}_i$ and $\vec{w}_j$, as

$$\varepsilon(i,j) = \vec{w}_i^T \cdot \vec{w}_j - a_{ij}$$

where a_{ij} is an element in the Matrix $\boldsymbol{A}$

The method we choose to find the values of the matrix $\boldsymbol{W}$ is **Stochastic Gradient Descent**. We define an objective function, J, to minimize as

$$J = \frac{1}{2\binom{n}{2}} \sum_{r=1}^{n} \sum_{c=r+1}^{n} \begin{cases} \varepsilon(r,c)^2, \, if \, r \neq c \\ 0, \, otherwise \end{cases}$$

When J is at a minimum then the column vectors of $\boldsymbol{W}$ will best meet our set of soft constraints, C.

We next define an individual objective function for each word vector, $J_{\vec{w}_i}$. During the optimization process, and during each iteration, we randomly select a column vector of $\boldsymbol{W}$, $\vec{w}_i$, to update by computing the gradient of the chosen individual word vector, $\frac{\partial J_{\vec{w}_i}}{\partial \vec{w}_i}$. We define the objective function for an individual word vector, $\vec{w}_i$, as

$$J_{\vec{w}_i} = \frac{1}{2} \sum_{j=1}^{n} \begin{cases} \varepsilon(i,j)^2, \, if \, i \neq j \\ 0, \, otherwise \end{cases}$$

And compute the gradient of that specific objective function, $\frac{\partial J_{\vec{w}_i}}{\partial \vec{w}_i}$, as

$$\frac{\partial J_{\vec{w}_i}}{\partial \vec{w}_i} = \sum_{j=1}^{n} \begin{cases} \vec{w}_i \odot \left(\vec{w}_i^T \cdot \vec{w}_j - y_{ij}\right), \, if \, i \neq j \\ 0, \, otherwise \end{cases}$$

The optimization process in pseudo code is outlined bellow:

$n \equiv$ Number of unique words in corpus
$m \equiv$ Number of training iterations
$\eta \equiv$ Learning rate

```
For $i = 1, 2, \ldots, m$, do:
    $x :=$ random number range$[1, n]$
    $\vec{w}_x := \vec{w}_x - \eta \frac{\partial J_{\vec{w}_x}}{\partial \vec{w}_x}$
```

The conditions we set for the set of word vectors can be stated in another way, our goal is to decompose the square, real, symmetric, cooccurrence matrix $\boldsymbol{A}_{n\times n}$ into a word vector matrix, $\boldsymbol{W}$, multiplied by its transpose. This would satisfies the conditions that the inner product between any two word vectors is equal to the two elements in matrix $\boldsymbol{A}$ corresponding to the two words intersection.

$$\boldsymbol{A} = \boldsymbol{W}\boldsymbol{W}^T$$

Real symmetric matrices can be decomposed into the form

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$$

Where $\boldsymbol{Q}$ is an orthogonal matrix and $\boldsymbol{\Lambda}$ is a diagonal matrix whose entries are the eigenvalues of $\boldsymbol{A}$. If the eigenvalues of $\boldsymbol{A}$ are all positive then we can write as

$$\boldsymbol{W} = \boldsymbol{Q}\sqrt{\boldsymbol{\Lambda}}$$

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T = \boldsymbol{Q}\sqrt{\boldsymbol{\Lambda}}\sqrt{\boldsymbol{\Lambda}}^T\boldsymbol{Q}^T = \boldsymbol{Q}\sqrt{\boldsymbol{\Lambda}}\left(\boldsymbol{Q}\sqrt{\boldsymbol{\Lambda}}\right)^T = \boldsymbol{W}\boldsymbol{W}^T$$

$$\boldsymbol{\Lambda} = \boldsymbol{\Lambda}^T \text{ and } \sqrt{\boldsymbol{\Lambda}} = \sqrt{\boldsymbol{\Lambda}}^T$$

However, it is not guaranteed that the eigenvalues of $\boldsymbol{A}$ will all be positive. But we can force this to be the case by making use of the free diagonals of the cooccurrence matrix $\boldsymbol{A}$ and making it diagonally dominant.

$$\text{A matrix is diagonally dominant if } |\boldsymbol{a}_{ii}| \geq \sum_{j\neq i} |\boldsymbol{a}_{ij}| \text{ for all } i.$$

If we add the condition that $\boldsymbol{a}_{ii} > 0$ for all i, then this matrix will also be positive definite and we can use singular value decomposition to compute $\tilde{\boldsymbol{A}} = \boldsymbol{W}\boldsymbol{W}^T$. If $\tilde{\boldsymbol{A}}$ is a symmetric positive definite matrix then by the spectral theorem we know that $\boldsymbol{\Sigma} = \boldsymbol{\Lambda}$ and $\boldsymbol{U} = \boldsymbol{V} = \boldsymbol{Q}$.

$$\tilde{\boldsymbol{A}} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

$$\tilde{\boldsymbol{A}}^T\tilde{\boldsymbol{A}} = \tilde{\boldsymbol{A}}\tilde{\boldsymbol{A}}^T, \text{ therefore } \boldsymbol{U} = \boldsymbol{V}$$

$$\text{And we can write } \tilde{\boldsymbol{A}} \text{ as } \boldsymbol{W}\boldsymbol{W}^T \text{ where } \boldsymbol{W} = \boldsymbol{V}\sqrt{\boldsymbol{\Lambda}}$$