Word Embedding

L.J. Brown

September 29, 2017

1 Introduction

This paper outlines a few methods used in this repository to map all unique words found in a corpus to vectors in a continuous vector space. The idea, and 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. Successful implimentations and strides in this area include Latent Semantic Analysis, "Word2vec", and "GloVe".

2 General Strategy

The essential steps taken by each implementation within this repository are as follows:

1. **Construct a Cooccurrence Matrix**
   Build a square "Cooccurrence Matrix", $\boldsymbol{A}$, where each element roughly represents some function, f, of the frequency or probability that two words, w_i and w_j, occur together in the corpus (the diagonal elements being associated with the same word twice).

$$\boldsymbol{A}_{n\times n} = \begin{array}{c} \\ w_1 \\ w_2 \\ \vdots \\ w_n \end{array} \begin{array}{c} \begin{array}{cccc} w_1 & w_2 & \dots & w_n \end{array} \\ \begin{pmatrix} f(w_1,w_1) & f(w_1,w_2) & \dots & f(w_1,w_n) \\ f(w_2,w_1) & f(w_2,w_2) & \dots & f(w_2,w_n) \\ \vdots & \vdots & \ddots & \vdots \\ f(w_n,w_1) & f(w_n,w_2) & \dots & f(w_n,w_n) \end{pmatrix} \end{array}$$

2. **Decompose the Cooccurrence Matrix into unique Word Vectors**
   Search for word vectors with the soft constraint that given any 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. This constraint can be written as

$$\vec{w}_i \cdot \vec{w}_j^T \approx f(w_i, w_j)$$

If the desired word vectors, $(\vec{w}_1, \ldots, \vec{w}_n)$, are written as the columns of some "Word Vector Matrix", $\boldsymbol{W}$, then this constraint can be written as

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

Several methods to perform this decomposition step are explored within this repository.

3 Cooccurrence Matrix Decomposition Methods

The methods implimented in this repository to decompose the Cooccurrence Matrix, $\boldsymbol{A}$, into $\boldsymbol{W}\boldsymbol{W}^T$ are

1. **Stochastic Gradient Descent**
   An iterative optimization method used to minimize an objective function. Stochastic Gradient Descent (SGD) is often used over Batch Gradient Decent (BGD) for large noisy datasets. Our implementation of SGD Draws heavily on the implementations by "Word2vec" and "GloVe".

2. **Eigen Decomposition of a forced Symmetric Positive Matrix**
   First the Cooccurrence Matrix, $\boldsymbol{A}$, is updated to have positive Eigen values. This is achieved by modifying the unused diagonal entries to ensure diagonal dominance. The new real symmetric matrix, $\tilde{\boldsymbol{A}}$, can be written as $\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^T$, and because of the positive Eigen values, $\boldsymbol{W} = \boldsymbol{Q}\sqrt{\boldsymbol{\Lambda}}$ will satisfy the constraint.

3. **Singular Value Decomposition of a forced Symmetric Positive Definite Matrix**
   Similarly to the EVD method, the Cooccurrence Matrix, $\boldsymbol{A}$, is updated, but now to be positive definite. This is again achieved by modifying the unused diagonal entries to ensure diagonal dominance with the additional constraint that all diagonal entries are greater than zero. The new positive definite matrix, $\tilde{\boldsymbol{A}}$, can be written as $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, and because it is symmetric and positive definite, $\boldsymbol{U} = \boldsymbol{V}$, and $\boldsymbol{W} = \boldsymbol{V}\sqrt{\boldsymbol{\Lambda}}$ will satisfy the constraint.

4 Details of Cooccurrence Matrix

This section details how this repository defines it's $n \times n$ 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, $f(w_i, w_j)$, which takes words as inputs.

For clarity it is useful to define a function, $m(w_i) = i$, that converts unique words to unique integers in the range 1 to n and and its inverse, $m^{-1}(i) = w_i$, which does the opposite.

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

$$W = \{w_1, \ldots, w_n\},\ I = \{1, \ldots, n\}$$

$$m : W \to I,\ m^{-1} : W \leftarrow I$$

$$f(w_i, w_j) \equiv \text{Cooccurrence function defined bellow}$$

$$\boldsymbol{A}_{n\times n} = \begin{array}{c} \\ r_1 \\ r_2 \\ \vdots \\ r_n \end{array} \begin{array}{c} \begin{array}{cccc} c_1 & c_2 & \ldots & c_n \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix} \end{array}$$

$$\text{where } a_{ij} = f(m^{-1}(r_i), m^{-1}(c_j)) = f(w_i, w_j)$$

For each element, a_{ij}, the words used as inputs for f are the words whose integer mappings correspond to the elements row, r_i, and column, c_j.

In order to define the cooccurrence function, f, it is useful to introduce some new variables. For each of the m sentences in the corpus we define s_i as the sequence of words in that particular sentence. Where S is a **sequence** containing the m **sequences**, $(s_1, \ldots, s_m)$, 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

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

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

Also used in the construction of the concurrence matrix is 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. For example using the i^{th} sentence in the corpus again and a random pair from that sequence $\{\tilde{w_2}, \tilde{w_3}\} = \{\text{"}made\text{"}, \text{"}mud\text{"}\}$

$$d(s_i, \{\tilde{w_2}, \tilde{w_3}\}) = d(\{\text{"}God\text{"}, \text{"}made\text{"}, \text{"}mud\text{"}\}, \{\text{"made"}, \text{"mud"}\})$$

$$\text{It is important to note that although } \tilde{w_2}, \tilde{w_3}\, \epsilon\, W$$

$$\text{it is not necessarily true that } \tilde{w_2} = w_2 \text{ or } \tilde{w_3} = w_3.$$

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.

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

One more useful definition due to obscure notation, $\{[s_i]^2\}$ is the set of all unordered pairs of words in the sequence s_i. Using the example above for s_i

$$\{[s_i]^2\} = \{\{\text{"God"}, \text{"made"}\}, \{\text{"God"}, \text{"mud"}\}, \{\text{"made"}, \text{"mud"}\}\}$$

Finally the cooccurrence function, $f(w_i, w_j)$, is defined as

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

Stated simply, $f(w_i, w_j)$ 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, $f(w_i, w_j)$, 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 due to the property of $f(w_i, w_j)$ that,

$$f(w_i, w_j) = f(w_j, w_i)$$

therefore,

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

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

5 Details of Word Vectors and Word Vector Matrix

Defining Constraints The soft constraints imposed states that for any given word vector pair, $\vec{w}_i$ and $\vec{w}_j$, excluding duplicate pairs, $\vec{w}_i$ and $\vec{w}_i$, their inner product will yield a value close to the two values in the co-occurrence matrix associated with those two words. This constraint can be written as

$$\vec{w}_i \cdot \vec{w}_j^T :\Leftrightarrow f(w_i, w_j) = a_{ij}$$

Since $\boldsymbol{A}$ is symmetric order does not matter when taking the inner product of two word vectors

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

Therefore,

$$\vec{w}_i \cdot \vec{w}_j^T = \vec{w}_i^T \cdot \vec{w}_j \; :\Leftrightarrow \; a_{ij} = a_{ji}$$

If the desired word vectors, $(\vec{w}_1, \ldots, \vec{w}_n)$, are written as the columns of some "Word Vector Matrix", $\boldsymbol{W}$, then this constraint can be written as

$$\boldsymbol{W}\boldsymbol{W}^T :\Leftrightarrow \boldsymbol{A}$$

where $\boldsymbol{W}$ is written as,

$$\boldsymbol{W}_{n\times v} = \begin{pmatrix} \vec{w}_1 \\ \vdots \\ \vec{w}_n \end{pmatrix}$$

and where v is the dimensionality of the word vectors (not yet specified).

6 Decomposition using Stochastic Gradient Descent

An iterative optimization method used to minimize an objective function. Stochastic Gradient Descent (SGD) is often used over Batch Gradient Decent (BGD) for large noisy datasets. Our implementation of SGD Draws heavily on the implementations by "Word2vec" and "GloVe".

The definition of an error function, ε, between two word vectors, $\vec{w}_i$ and $\vec{w}_j$, based on the soft constraints chosen is

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

The chosen objective function, J, to minimize during the optimization process is

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

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

When J is at a minimum then the $\boldsymbol{W}$ will best meet our soft constraint.

Next is the definition 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

$$\text{For } i = 1, 2, \ldots, m, \text{ do:}$$

$$x := \text{random number range}[1, n]$$

$$\vec{w}_x := \vec{w}_x - \eta \frac{\partial J_{\vec{w}_x}}{\partial \vec{w}_x}$$

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 $\boldsymbol{U} = \boldsymbol{V} = \boldsymbol{Q}$.

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

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

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