Deep Neural Networks for Learning Graph Representation

Julina Maharjan, jmaharja@kent.edu

Kent State University – KSU

April 2, 2021

Content

Content

Objective

Main Idea

model that generates low dimensional vector representation for each vertex by capturing the graph structural information

2 steps:

1. uses the Random Surfing model to capture the graph structure
   - instead of the sampling-based method for generating linear sequences
2. performs PMI matrix factorization as a solution for an objective function of SGNS
   - instead of SVD, uses the Stacked denoising autoencoders for extracting complex non-linear features

Approach

Many methods existed to learn effective vector representation for linear sequence in NLP
e.g. Random walk

Task Casting

Can we leverage this idea to learn vertex representations for graph structure?

- The graph structure can be converted into a large collection of linear structures
- And Skip-Gram model can be utilized to learn low-dimensional representations for vertices from such linear structures

Random-Surfing Model

Why?

- yields a probabilistic co-occurrence for weighted graphs
- very similar to the one obtained by sampling linear sequence from graphs (w/o sampling)

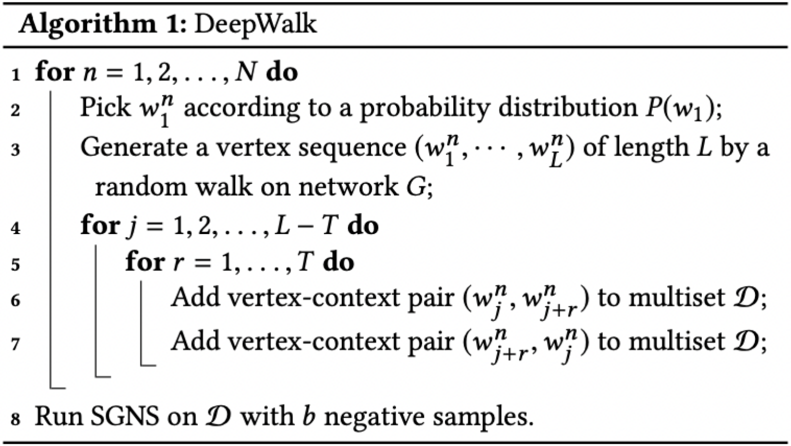
SGNS: Skip Gram Negative Sampling

why?

- the objective function of SGNS has a intrinsic relation with factorizing PPMI matrix of the words and contexts
- SVD only reduces to linear dimension reduction
- Levy and Goldberg showed that PPMI matrix itself is an explicit representation matrix of the graph
- SVD didn't outperform the representations from the PPMI matrix
- thus, the stacked autoencoders are used to learn low level features with high level abstraction

Content

Deep Walk



Algorithm 1: DeepWalk

```
for $n = 1, 2, \ldots, N$ do
    Pick $w_1^n$ according to a probability distribution $P(w_1)$;
    Generate a vertex sequence $(w_1^n, \cdots, w_L^n)$ of length $L$ by a
      random walk on network $G$;
    for $j = 1, 2, \ldots, L - T$ do
        for $r = 1, \ldots, T$ do
            Add vertex-context pair $(w_j^n, w_{j+r}^n)$ to multiset $\mathcal{D}$;
            Add vertex-context pair $(w_{j+r}^n, w_j^n)$ to multiset $\mathcal{D}$;
Run SGNS on $\mathcal{D}$ with $b$ negative samples.
```

Random Walk

Let $G = (V, E)$ be an undirected graph. Consider the random process that starts from some vertex $v \in V(G)$, and repeatedly moves to a neighbor of the current vertex chosen uniformly at random.
For $t \geq 0$, and for $u \in V(G)$, let $p_t(u)$ denote the probability that you are at vertex u at time $t . p_t \in \mathbb{R}^n$. Clearly, $\sum_{u \in V(G)} p_t(u) = 1$

$$p_{t+1}(v) = \sum_{(u,v) \in E(G)} p_t(u) \cdot \frac{1}{d(u)}$$

We can write this using matrix notation $W = W_G$:

$$[W_G]_{i,j} = \begin{cases} \frac{1}{d(j)} & if (i,j) \in E(G) \\ 0 & \text{otherwise} \end{cases}$$
$$W_G = AD^{-1}$$

Random Walk

Limitation:

- only transform unweighted graph structural information into linear sequence
- constraints on hyper parameters walk length, η and total walk, γ
- based on sampled linear sequences

Solution:

- Random Surfing: model that generates POM from a weighted graph which avoids the expensive sampling process

Skip Gram Negative Sampling Objective Function

Consider a word-context pair (w, c), Let $\Pr(D = 1 \mid w, c)$ be the probability that (w, c) came from the data, the distribution is modeled as:

$$\Pr(D = 1 \mid w, c) = \sigma(\vec{w}, \vec{c})$$

where $\vec{w}$ and $\vec{c}$ (each a d -dimensional vector) are the model parameters to be learned.

The negative sampling objective tries to maximize $\Pr(D = 1 \mid w, c)$ for observed (w, c) pairs while maximizing $\Pr(D = 0 \mid w, c)$ for randomly sampled "negative" examples. SGNS's objective for a single (w, c) observation is then:

$$\log \delta(\vec{w} \cdot \vec{c}) + k \cdot \mathbb{E}_{c_N \sim P_D} \left[\log \delta \left(-\vec{w} \cdot \overrightarrow{c_N} \right) \right]$$

where k is the number of "negative" samples and c_N is the sampled context, drawn according to the empirical unigram distribution $P_D(c) = \frac{\#(c)}{|D|}$

Skip Gram Negative Sampling

Setting and Notation: The skip-gram model assumes a corpus of words $w \in V_W$ and their contexts $c \in V_C$, where V_W and V_C are the word and context vocabularies.
The words come from an unannotated textual corpus of words $w_1, w_2, \ldots, w_n$ (typically n is in the billions) and the contexts for word w_i are the words surrounding it in an L-sized window
$w_{i-L}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+L}$
We denote the collection of observed words and context pairs as D.
We use $\#(w, c)$ to denote the number of times the pair (w, c) appears in D.
Similarly, $\#(w) = \sum_{c' \in V_C} \#(w, c')$ and $\#(w) = \sum_{w' \in V_w} \#(w', c)$ are the number of times w and c occurred in D, respectively.

SGNS Objective Function

$$\ell = \sum_{w \in V_W} \sum_{c \in V_C} \#(w, c) \left(\log \sigma(\vec{w} \cdot \vec{c}) + k \cdot \mathbb{E}_{c_N \sim P_D} \left[\log \sigma \left(-\vec{w} \cdot \vec{c}_N \right) \right] \right)$$

Simplifying above equation leads to following: (Paper: Neural Word Embedding as Implicit Matrix Factorization - Levy and Goldberg 20014)

$$\vec{w} \cdot \vec{c} = \log \left(\frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)} \cdot \frac{1}{k} \right) = \log \left(\frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)} \right) - \log k$$

Interestingly, the expression $\log \left(\frac{\#(w,c) \cdot |D|}{\#(w) \cdot \#(c)} \right)$ is the well-known pointwise mutual information (PMI) of (w, c), which we discuss in depth below. Finally, we can describe the matrix M that SGNS is factorizing:

$$M_{ij}^{\mathrm{SGNS}} = W_i \cdot C_j = \vec{w}_i \cdot \vec{c}_j = PMI\left(w_i, c_j\right) - \log k$$

PPMI

PPMI is a measure of association used in information theory and statistics.

$$\operatorname{pmi}(x;y) \equiv \log \frac{p(x,y)}{p(x)p(y)} = \log \frac{p(x \mid y)}{p(x)} = \log \frac{p(y \mid x)}{p(y)}$$

Auto Encoder

Auto Encoder performs two actions

- – an encoding step, followed by decoding step. In the encoding step, a function $f_{\theta_1}(\cdot)$ applied to the vector in the input space and send it to new feature space. An activation function is typically involved in this process to model the non-linearities between the two vector spaces – the space of input vectors and th space of latent vector representations.
- At the decoding step a reconstruction function $g_{\theta_2}(\cdot)$ is used to reconstruct th original input vectors back from the latent representation space.

Auto Encoder Objective Function

Let us assume that $f_{\theta_1}(x) = \sigma\left(W_1 x + b_1\right)$ and $g_{\theta_2}(y) = \sigma\left(W_2 y + b_2\right),$ where $\sigma(\cdot)$ is the activation function, $\theta_1 = \{W_1, b_1\}$ are the weights (parameters) involved in the encoder, and $\theta_2 = \{W_2, b_2\}$ are the weights (parameters) involved in the decoder. Here W_1 and W_2 are linear maps (matrices) transforming the vectors from the input space, and b_1 and b_2 are the bias vectors. Our goal is to minimize the following reconstruction loss function by finding θ_1 and θ_2 :

$$\sum L\left(x^{(i)}, g_{\theta_2}\left(f_{\theta_1}\left(x^{(i)}\right)\right)\right)$$

where L is sample-wise loss, and $x^(i)$ is the i-th instance.

Stacked Auto Encoder

- Multi-layer deep neural network consisting of multiple layers of such autoencoders.
- use the layer-wise training approach to extract essential regularities capturing different levels of abstractions from the data layer by layer, with higher layers conveying higher-level abstractions from the data.

Stacked denoising Auto Encoder

- partially corrupts the input data before taking the training step
- each input sample x is corrupted randomly by assigning some of the entries in the vector to 0 with a certain probability.
- **Loss function:**
  Similar to standard autoencoders, we again reconstruct data from the latent representations.

$$\min_{\theta_1,\theta_2} \sum_{i=1}^{n} L\left(x^{(i)}, g_{\theta_2}\left(f_{\theta_1}\left(\tilde{x}^{(i)}\right)\right)\right)$$

where $\tilde{x}^{(i)}$ is corrupted input data of $x^{(i)}$, and L is the standard squared loss.

Content

DNGR Model

3 main steps

- Random surfing model to capture graph structural information and generate a probabilistic co-occurrence matrix
- Calculation of PPMI matrix based on our probabilistic co-occurrence matrix
- A Stacked Denoising autoencoder to learn low-dimensional vertex representations

Main Components

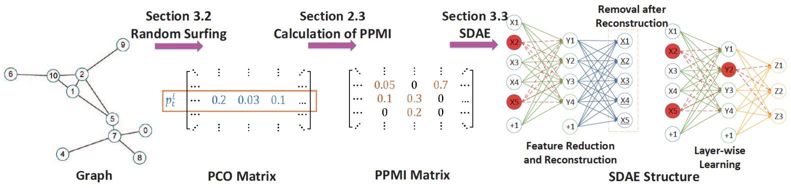


Figure: Main Components: Random Surfing, Calculation of PPMI matrix and feature reduction by SDAE

Random Surfing and Context Weighting

Why Random Surfing ?

- Sampling sequences have finite length, thus makes difficult to capture the accurate contextual info for vertices that appears at the edges of the sequence
- Also, it is very difficult to find hyper parameters ; walk length η and total walks γ

Random Surfing Model

Let current vertex be i-th vertex and A be a transition matrix.
Then, p_k is a row vector, whose j-th entry is the probability of reaching the j-th vertex after k steps

$$p_k = \alpha \cdot p_{k-1} A + (1 - \alpha) p_0$$

Stacked Denoising Autoencoder

- Matrix factorization of PPMI matrix to low dimensional word/vertex embedding
- SVD: Limitation in Linear Factorization
- Need non-linear factorization

Stacked Denoising Auto Encoders (SDAE) - to learn high-level abstractions at each layer - denoising to corrupt the input data corrupt each input sample x randomly by assigning some of the entries in the vector o 0 with a certain probability

Stacked Denoising Autoencoder

```
Input: PPMI matrix $X$, Number of SDAE layers $\Gamma$
1. Initialize SDAE
Set number of nodes $n_j$ in layer $j, X^{(1)} = X$
2. Greedy layer-wised training
For $j = 2$ to $\Gamma$
  2.1 Construct one hidden layer SDAE with input of $X^{(j)}$
  2.2 Learn hidden layer representation $h^{(j)}$
  2.3 $X^{(j)} = h^{(j)} \left(X^{(j)} \in \mathbb{R}^{n \times n_j}\right)$
Output: Matrix of vertices representations $R$
```

Stacked Denoising Autoencoder

- all the X_i's are input data
- Y_i's are learned representation in first layer
- Z_i's are learned representation in second layer

The objection function for SDAE:

$$\min_{\theta_1,\theta_2} \sum_{i=1}^{n} L\left(x^{(i)}, g_{\theta_2}\left(f_{\theta_1}\left(\tilde{x}^{(i)}\right)\right)\right)$$

where $x^{(i)}$ is corrupted input data of $x^{(i)}$, and L is the standard squared loss.

Content

Visualization of Wine

- Comparison on the performance of DNGR with SVD, DeepWalk and SGNS on Wine
- Vertex representation were fed to t-SNE; mapped to 2D.

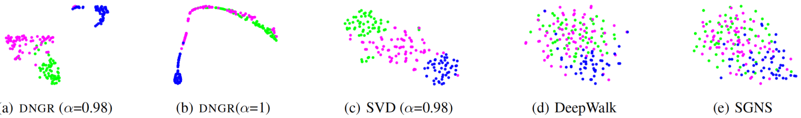


Figure: Visualization Results of wine

- DNGR ($\alpha = 0.98$) showed better clustering among all.

Content

Conclusion

- A Deep Graph Representation Model that encodes vertex representation in a low dimension
- Uses Random surfing and SDAE as the main components
- Experiments showed it outperformed many baseline algorithms

References I

Wei Lu Shaosheng Cao and Qiongkai Xu.
Deep neural networks for learning graph representations.
Xidian University, Singapore University of Technology and Design and Australian National University Nation UCT Australia -CRL.

Omer Levy and Yoav Goldberg.
Neuralword embedding as implicit matrix factorization.
Bar-Ilan University.

Bullinara and Levy.
Positive pointwise mutual information.
2007.

Mikolov et al.
Skip gram negative sampling.
2013.

References II

Al-Rfou Perozzi and Skiena.
Deepwalk.
2014.

Gutmann Michael and Hyv¨arinen Aapo.
Noise-contrastive estimation: A new estimation principle for unnormalized statistical models.
Journal of Machine Learning Research, 135(35):13096–13106, 2013.

Deep Neural Networks for Learning Graph Representation

Julina Maharjan, jmaharja@kent.edu

Kent State University – KSU

April 2, 2021