Predictive Text Embedding

Hemant Verma (20162078)
Riddhi Patel (20162082)
Vikas Raghuwanshi (20162070)
Dhawnit Khurana (20162076)

- Paper studies the problem of **embedding** networks (graphs) into low-dimensional spaces, in which every vertex is represented as a low-dimensional vector.
- optimizes an objective function which preserves both the local and global network structures.
- local network structure: local pairwise proximity between two vertices - First Order Proximity – for (u, v), the weight on that edge, w_{uv} , indicates the first-order proximity between u and v).
- global network structure: similarity between their neighborhood network structures. , let $p_u = (w_{u,1}, \ldots, w_{u,|V|})$ denote the first-order proximity of u with all the other vertices, then the second-order proximity between u and v is determined by the similarity between p_u and p_v .

- First Order Proximity:
- for each undirected edge (i; j), joint probability between vertex vi and vj as follows:

$$p_1(v_i, v_j) = \frac{1}{1 + \exp(-\vec{u}_i^T \cdot \vec{u}_j)},$$

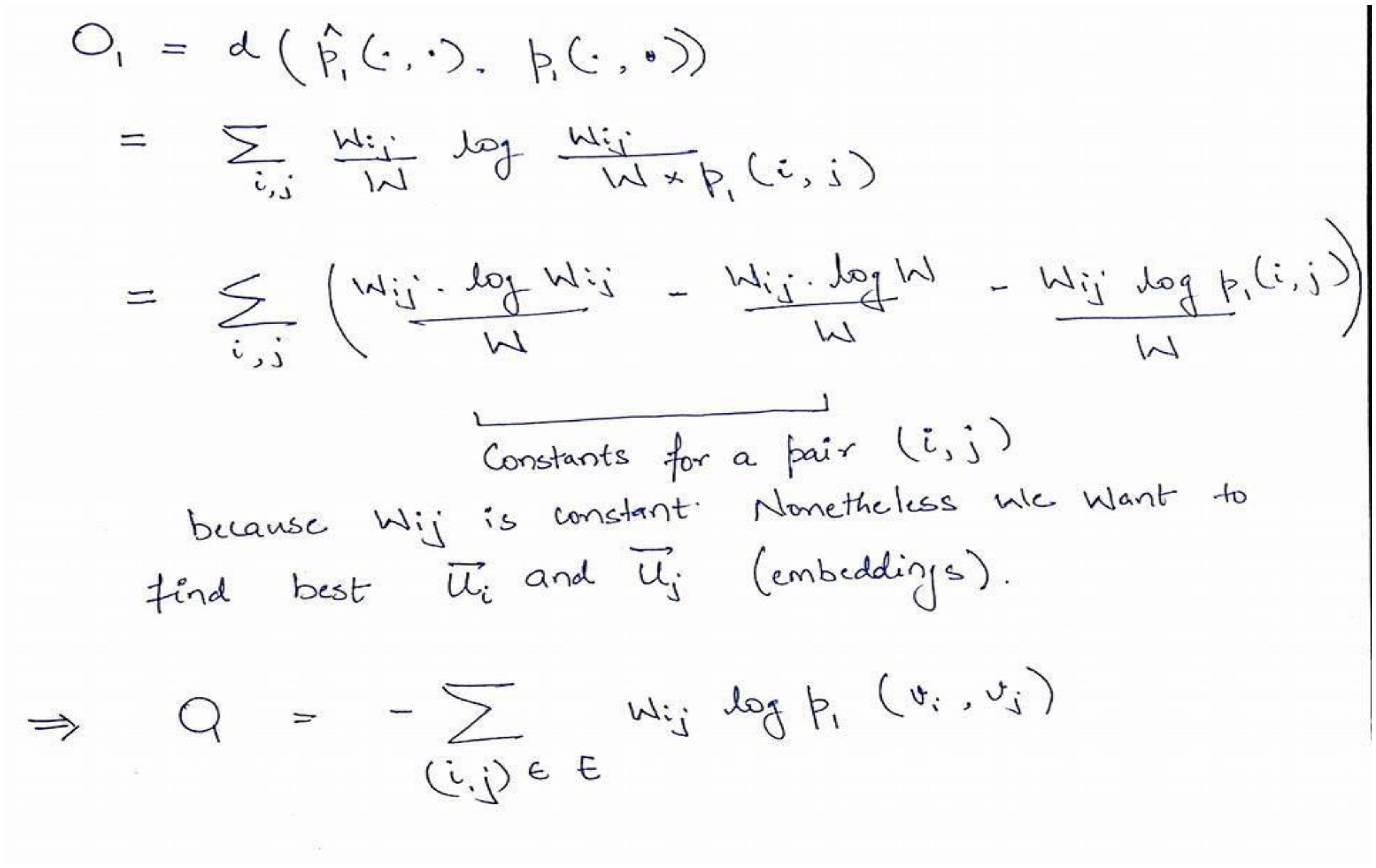
- where ui R^d is the low-dimensional vector representation of vertex vi.
- its empirical probability can be defined as

$$\hat{p}_1(i, j) = \frac{w_{ij}}{W}, \text{ where } W = \sum_{(i,j)\in E} w_{ij}.$$

- To preserve the first-order proximity, a straightforward way is to minimize the following objective function:

$$O_1 = d(\hat{p}_1(\cdot, \cdot), p_1(\cdot, \cdot)),$$

- Here d(*;*) is the K-L divergence between 2 distributions.



$$O_1 = d\left(\hat{p}_1(\cdot,\cdot),\ p_1(\cdot,\cdot)\right)$$

$$= \sum_{i,j} \frac{W_{ij}}{W} \log \frac{W_{ij}}{W \times p_1(i,j)}$$

$$= \sum_{i,j} \left(\underbrace{\frac{W_{ij} \cdot \log W_{ij}}{W} - \frac{W_{ij} \cdot \log W}{W}}_{\text{Constants for a pair } (i,j)} - \frac{W_{ij} \log p_1(i,j)}{W} \right)$$

because W_{ij} is constant. Nonetheless we want to find best $\vec{u}_i$ and $\vec{u}_j$ (embeddings).

$$\Rightarrow \quad O = - \sum_{(i,j) \in E} W_{ij} \log p_1(v_i, v_j)$$

- Second Order Proximity:
- It assumes that vertices sharing many connections to other vertices are similar to each other. In this case, each vertex is also treated as a specific “context” and vertices with similar distributions over the “contexts” are assumed to be similar.
- For edge (i,j), we define the probability of “context” vj generated by vertex vi as:

$$p_2(v_j|v_i) = \frac{\exp(\vec{u}_j'^T \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}_k'^T \cdot \vec{u}_i)},$$ which defines a conditional probability distribution over context

⊙ Vertices with similar probability distributions over the "Contexts" are assumed to be similar.

Therefore, for two vertices, v_i and $v_{i'}$, if $p_2(\cdot|v_i)$ is similar to $p_2(\cdot|v_{i'})$ then we assume that v_i and $v_{i'}$ has close second order proximity.

d_i = Out-degree of vertex v_i = $\sum_{k \in N(i)} w_{ik}$

Conditional probability distribution over contexts :

$$p_2(v_j | v_i) = \frac{\exp(\vec{u}_j \cdot \vec{u}_i)}{\sum_{k=1}^{|V|} \exp(\vec{u}_k \cdot \vec{u}_i)} \quad - ①$$

$|V|$ = total number of vertices.

eqn $\textcircled{1}$ is basically $p_2(\cdot \mid v_i)$

Empirical distribution $\hat{p}_2(\cdot \mid v_i)$

$$= \frac{w_{ij}}{d_i}$$

We try to keep conditional distribution close to empirical.

So, minimize:

$$O_2 = \sum_{i \in V} \lambda_i \, d\left(\hat{p}_2(\cdot \mid v_i), p_2(\cdot \mid v_i)\right)$$

λ_i = importance of vertex in the system = d_i

d = KL divergence.

$$\Rightarrow O_2 = - \sum_{(i,j) \in E} w_{ij} \log p_2(v_j \mid v_i)$$

PTE

- The labeled information and different levels of word co-occurrence information are first represented as a large-scale heterogeneous text network, which is then embedded into a low dimensional space.
- Three types of networks:
- Word-Word Network - denoted as $G_{ww} = (V, E_{ww})$, captures the word co-occurrence information in local contexts of the unlabeled data. V is a vocabulary of words and E_{ww} is the set of edges between words. The weight w_{ij} of the edge between word v_i and v_j is defined as the number of times that the two words co-occur in the context windows of a given window size.

- Word-Document Network - denoted as $G_{wd} = (V \cup D, E_{wd})$, is a bipartite network where D is a set of documents and V is a set of words. The weight w_{ij} between word v_i and document d_j is simply defined as the number of times v_i appears in document d_j.
- Word-Label Network - denoted as $G_{wl} = (V \cup L, E_{wl})$, is a bipartite network that captures category-level word co-occurrences. L is a set of class labels and V a set of words. The weight w_{ij} of P the edge between word v_i and class c_j is defined as: $w_{ij} = \sum_{(d:ld =j)} n_{di}$, where n_{di} is the term frequency of word v_i in document d, and ld is the class label of document d.

Heterogeneous Text Network

- Heterogeneous Text Network is the combination of word-word, word-document, and word-label networks constructed from both unlabeled and labeled text data.
- **Objective - to learn low dimensional representations of words by embedding the heterogeneous text network**

Bipartite Network Embedding

- Given a bipartite network $G = (V_A \cup V_B, E)$ where V_A and V_B are two disjoint sets of vertices of different types, and E is the set of edges between them. We first define the conditional probability of vertex vi in set V_A generated by vertex vj in set V_B as:

$$p(v_i|v_j) = \frac{\exp(\vec{u}_i^T \cdot \vec{u}_j)}{\sum_{i' \in A} \exp(\vec{u}_{i'}^T \cdot \vec{u}_j)},$$

- Objective function (as previously) comes out to be

$$O = \sum_{j \in B} \lambda_j d(\hat{p}(\cdot|v_j), p(\cdot|v_j)),$$

- Solving it becomes:

$$O = - \sum_{(i,j) \in E} w_{ij} \log p(v_j|v_i).$$

- The objective can be optimized with stochastic gradient descent using the techniques of edge sampling and negative sampling. In each step, a binary edge e = (i, j) is sampled with the probability proportional to its weight $w_{ij.}$

Heterogeneous Text Network Embedding

- composed of three bi-partite networks: word-word, word-document and word-label networks, where the word vertices are shared across the three networks.
- an intuitive approach is to collectively embed the three bipartite networks

$$O_{pte} = O_{ww} + O_{wd} + O_{wl},$$

where

$$O_{ww} = -\sum_{(i,j)\in E_{ww}} w_{ij} \log p(v_i|v_j)$$

$$O_{wd} = -\sum_{(i,j)\in E_{wd}} w_{ij} \log p(v_i|d_j)$$

$$O_{wl} = -\sum_{(i,j)\in E_{wl}} w_{ij} \log p(v_i|l_j)$$

Negative Sampling

- Idea from Skip-gram NN. Tremendous number of weights in the NN, all of which would be updated slightly by every one of billions of training samples. Negative sampling addresses this by having each training sample only modify a small percentage of the weights, rather than all of them.
- With negative sampling, we are instead going to randomly select just a small number of “negative” words (let’s say 5) to update the weights for. (In NN context, a “negative” word is one for which we want the network to output a 0 for). We will also still update the weights for our “positive” word

Learning Algorithm

Algorithm 1: Joint training.

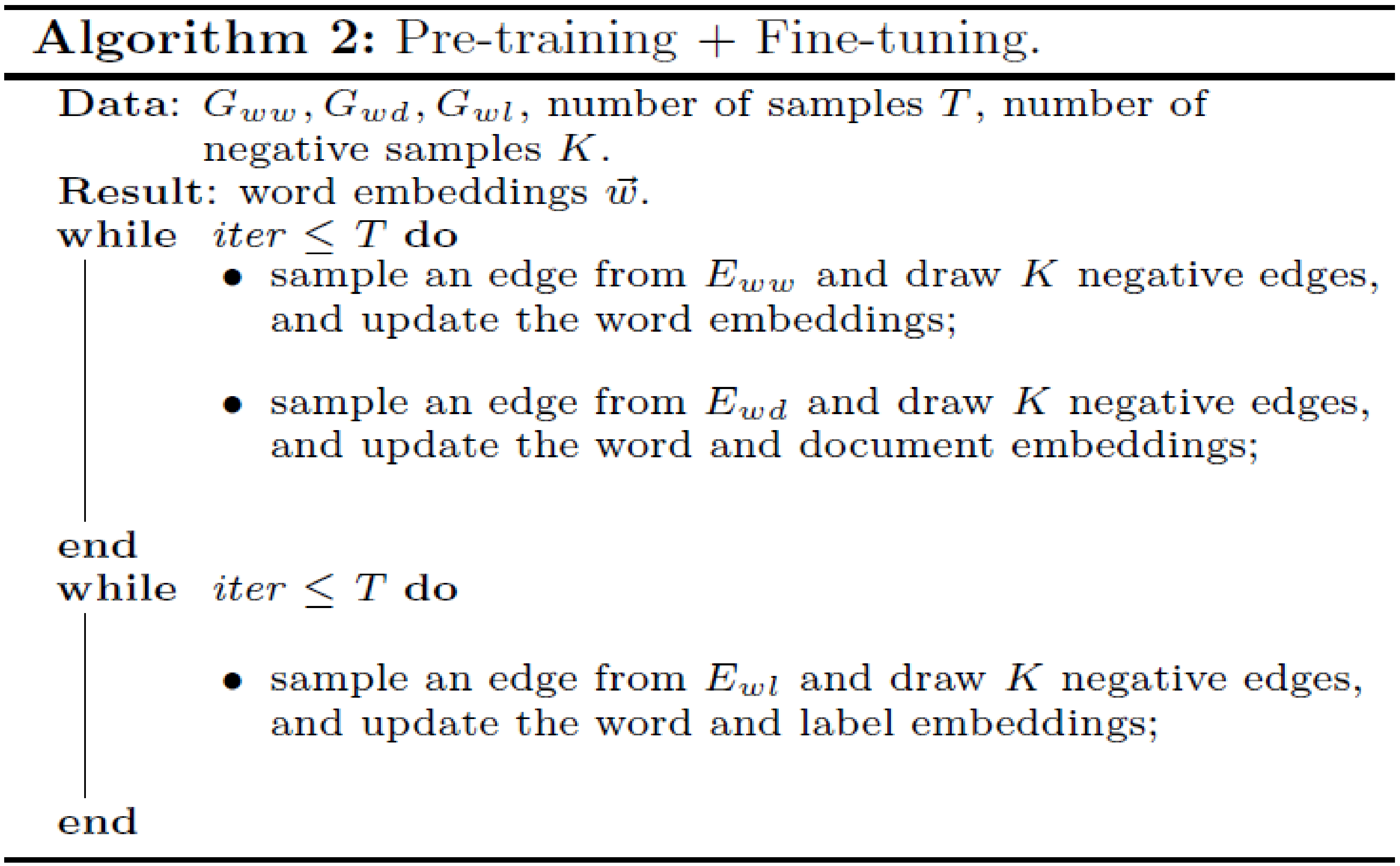
```
Data: $G_{ww}, G_{wd}, G_{wl}$, number of samples $T$, number of
      negative samples $K$.
Result: word embeddings $\vec{w}$.
while $iter \leq T$ do
    • sample an edge from $E_{ww}$ and draw $K$ negative edges,
      and update the word embeddings;

    • sample an edge from $E_{wd}$ and draw $K$ negative edges,
      and update the word and document embeddings;

    • sample an edge from $E_{wl}$ and draw $K$ negative edges,
      and update the word and label embeddings;

end
```

Learning Algorithm



Algorithm 2: Pre-training + Fine-tuning.

```
Data: $G_{ww}, G_{wd}, G_{wl}$, number of samples $T$, number of
      negative samples $K$.
Result: word embeddings $\vec{w}$.
while $iter \leq T$ do
    • sample an edge from $E_{ww}$ and draw $K$ negative edges,
      and update the word embeddings;
    • sample an edge from $E_{wd}$ and draw $K$ negative edges,
      and update the word and document embeddings;
end
while $iter \leq T$ do
    • sample an edge from $E_{wl}$ and draw $K$ negative edges,
      and update the word and label embeddings;
end
```

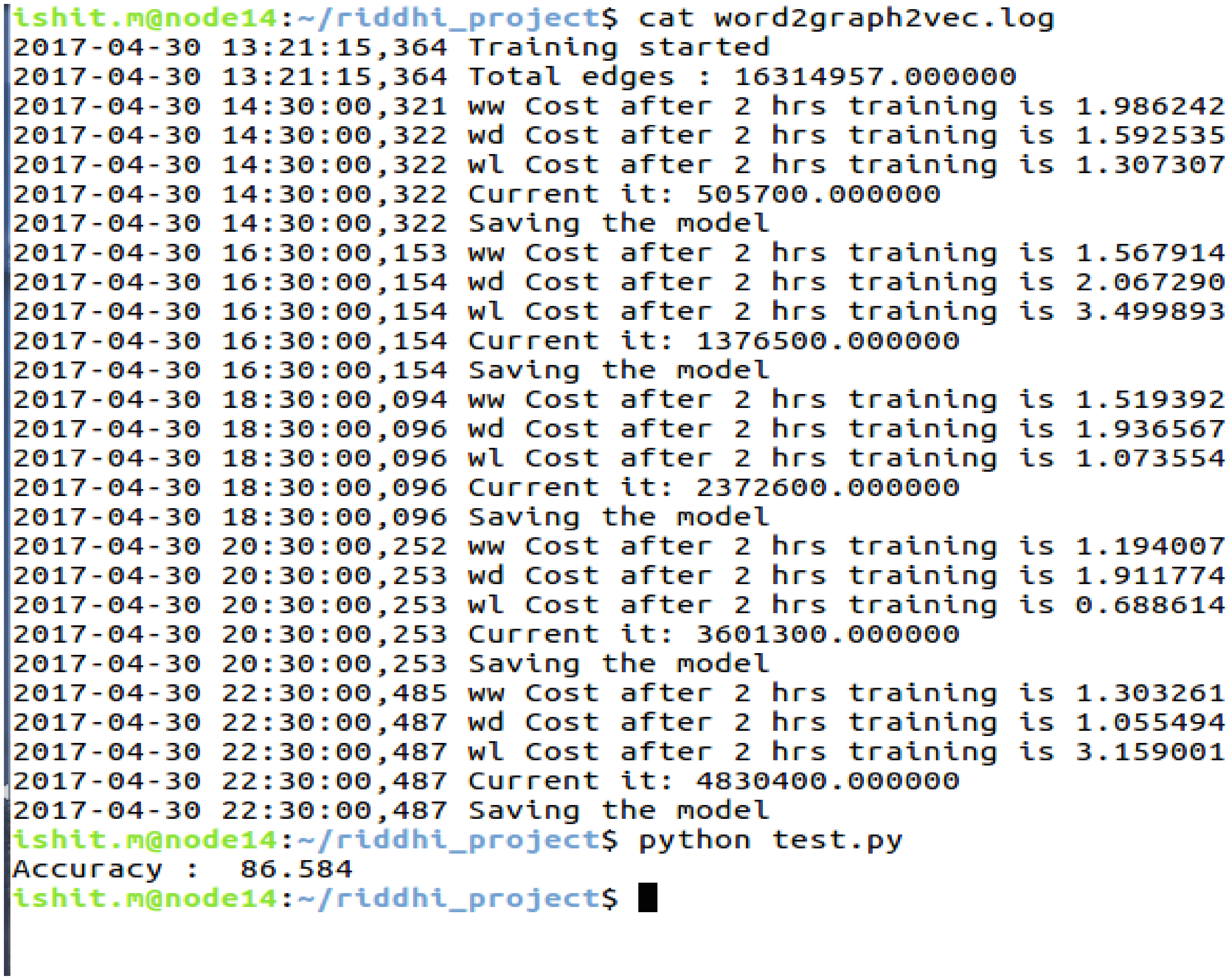
Text Embedding

- Once the word vectors are learned, the representation of an arbitrary piece of text can be obtained by simply averaging the vectors of the words in that piece of text.

$$\vec{d} = \frac{1}{n} \sum_{i=1}^{n} \vec{u}_i ,$$

Training-test Results

- Accuracy of ~87% on IMDB Movie Reviews Dataset



```
ishit.m@node14:~/riddhi_project$ cat word2graph2vec.log
2017-04-30 13:21:15,364 Training started
2017-04-30 13:21:15,364 Total edges : 16314957.000000
2017-04-30 14:30:00,321 ww Cost after 2 hrs training is 1.986242
2017-04-30 14:30:00,322 wd Cost after 2 hrs training is 1.592535
2017-04-30 14:30:00,322 wl Cost after 2 hrs training is 1.307307
2017-04-30 14:30:00,322 Current it: 505700.000000
2017-04-30 14:30:00,322 Saving the model
2017-04-30 16:30:00,153 ww Cost after 2 hrs training is 1.567914
2017-04-30 16:30:00,154 wd Cost after 2 hrs training is 2.067290
2017-04-30 16:30:00,154 wl Cost after 2 hrs training is 3.499893
2017-04-30 16:30:00,154 Current it: 1376500.000000
2017-04-30 16:30:00,154 Saving the model
2017-04-30 18:30:00,094 ww Cost after 2 hrs training is 1.519392
2017-04-30 18:30:00,096 wd Cost after 2 hrs training is 1.936567
2017-04-30 18:30:00,096 wl Cost after 2 hrs training is 1.073554
2017-04-30 18:30:00,096 Current it: 2372600.000000
2017-04-30 18:30:00,096 Saving the model
2017-04-30 20:30:00,252 ww Cost after 2 hrs training is 1.194007
2017-04-30 20:30:00,253 wd Cost after 2 hrs training is 1.911774
2017-04-30 20:30:00,253 wl Cost after 2 hrs training is 0.688614
2017-04-30 20:30:00,253 Current it: 3601300.000000
2017-04-30 20:30:00,253 Saving the model
2017-04-30 22:30:00,485 ww Cost after 2 hrs training is 1.303261
2017-04-30 22:30:00,487 wd Cost after 2 hrs training is 1.055494
2017-04-30 22:30:00,487 wl Cost after 2 hrs training is 3.159001
2017-04-30 22:30:00,487 Current it: 4830400.000000
2017-04-30 22:30:00,487 Saving the model
ishit.m@node14:~/riddhi_project$ python test.py
Accuracy :  86.584
ishit.m@node14:~/riddhi_project$ 
```