Dimension Reduction
From theory to application

Peiyan Li
Multi-Label Learning & Network Mining
14.04.2017

Outline

- Motivation
- Dimension reduction
  - A brief introduction
  - From SNE to T-SNE
- Word2Vec
- LargeVis
- Lookalike (an application of Node2Vec)
- Conclusion

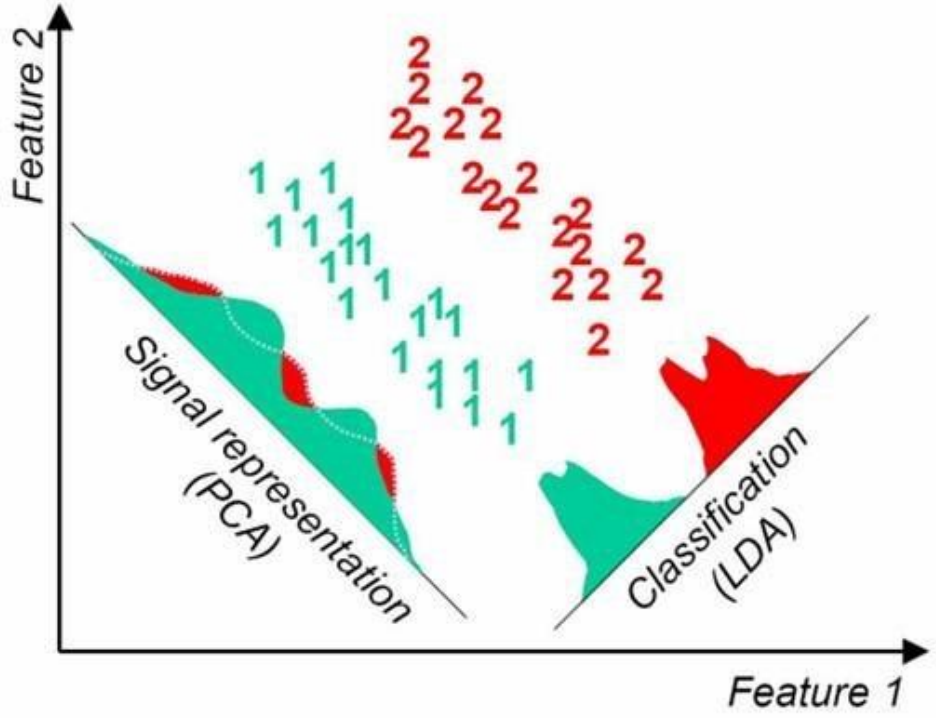
Motivation

- **Why choose those topics ??**

- Dimension reduction is a fundamental research field in data mining.
- **T-SNE** and **LargeVis** are dimension reduction algorithms in **data visualization**. Maybe we can learn something.
- **Word2Vec** is the basis of NLP. Many other fields are inspired by it.
- I'm trying to figure out **the connection** between those parts.
- Personally, I want to introduce **some interesting applications**, real applications, not in a theoretical way.

Dimension Reduction: A brief intro.

- Linear methods
  - PCA
  - LDA
  - MDS
  - etc.
- Nonlinear methods
  - Isomap
  - LLE
  - LE
  - etc.

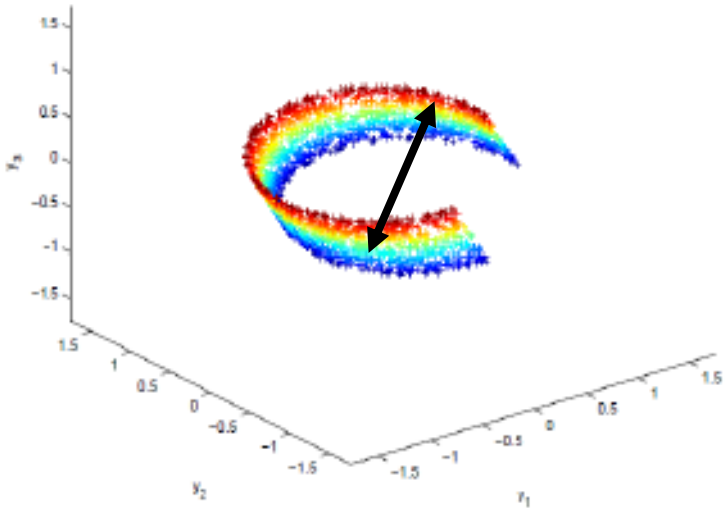


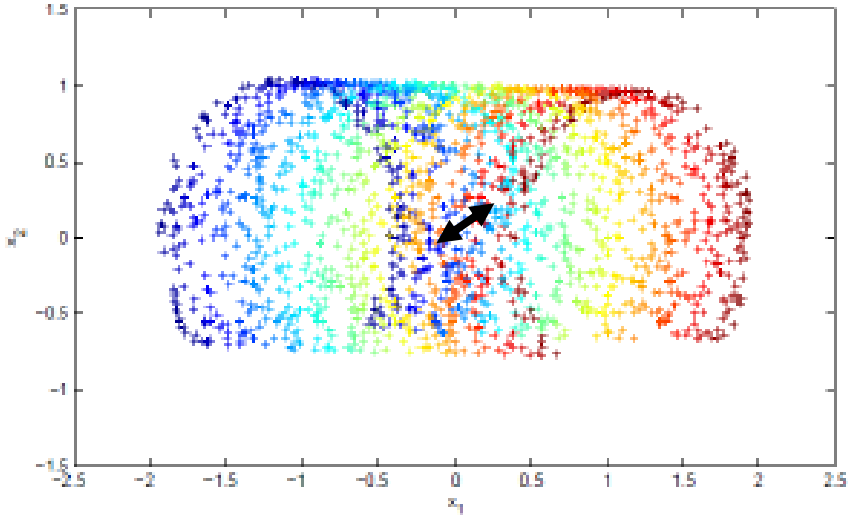
- PCA: selects the projection where the sample point has the largest variance (Unsupervised)
- LDA: after projection, interclass variance is the smallest, and the variance among classes is the largest (Supervised)

Dimension Reduction in Data Visualization

我需要
一块二向箔，
清理用。
—歌者

- **Goal**
  - To visualize data living in a d-dimensional space (d > 3)
- **Limitations of linear projections**
  - Even simple manifolds can be poorly projected
  - Points originally far from each other are projected close: this is an intrusion





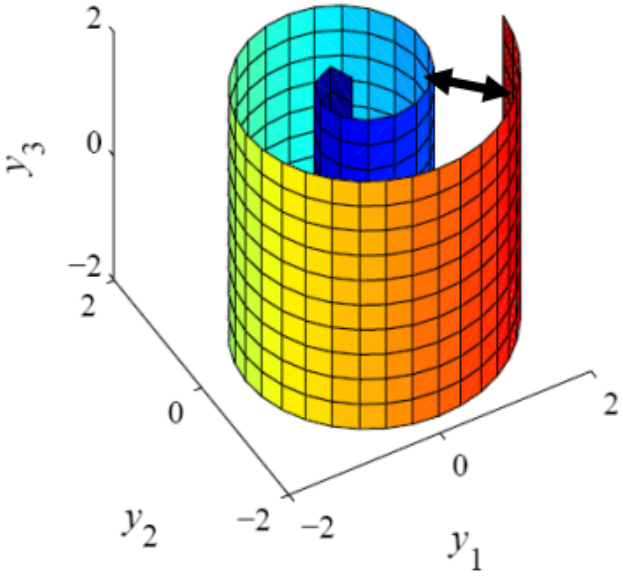
Dimension Reduction in Data Visualization

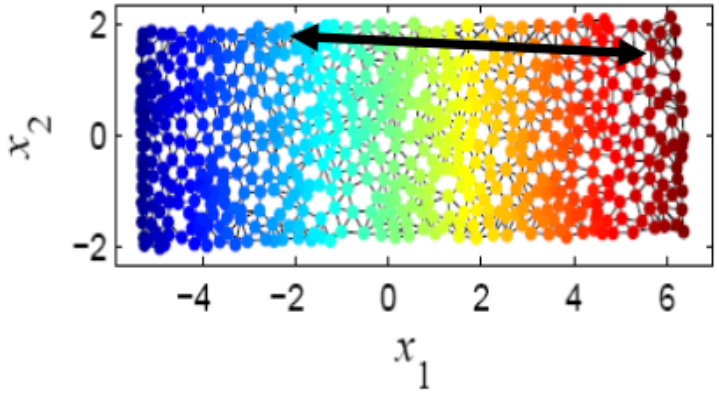
- Nonlinear projections
- Goal: to unfold, rather than to project (linearly)
- Drawbacks: intrusions can be hopefully decreased, but extrusions could appear

A general solution

$$w_{ij} = \lambda f\left(\frac{d_{ij}}{\sigma}\right) + (1-\lambda) f\left(\frac{\delta_{ij}}{\sigma}\right)$$

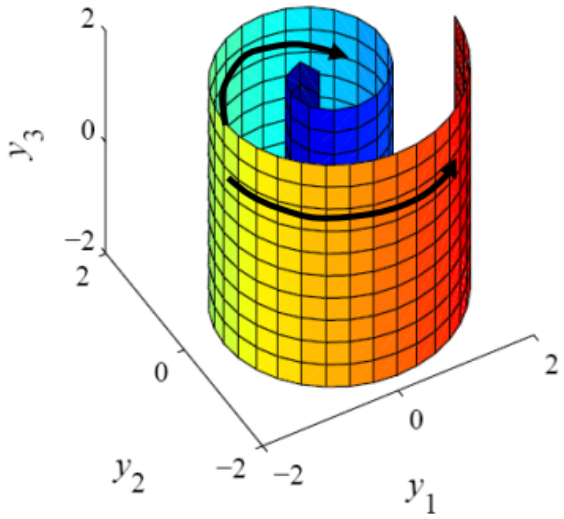
allows intrusions allows extrusions

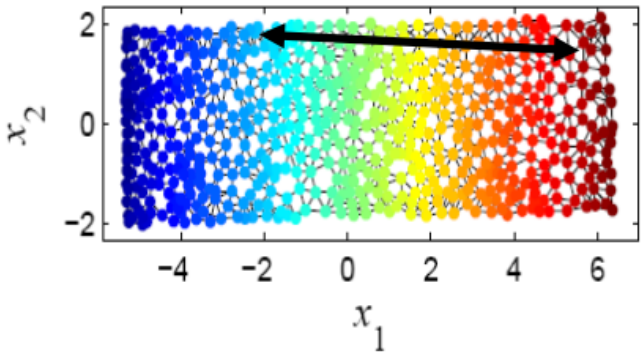




Dimension Reduction in Data Visualization

- Geodesic distances
- Goal: to measure distances along the manifold
- Breakthrough: such distances are more easily preserved

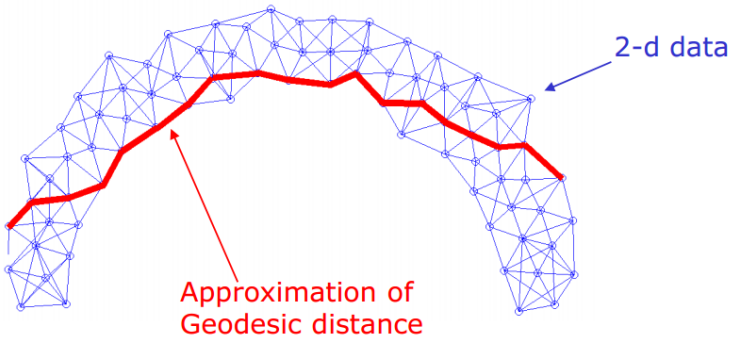




The manifold is unknown (ノ ﹏ ヽ)

Approximate it by a graph (⊙ o ⊙)

The graph usually be built by KNN.



From SNE to T-SNE: SNE

Goal: reduce the dimension of data while keeping information content

Basic Idea: similarity in the input space should be consistent with that in the output space; use conditional probability to measure this similarity

Consider $(x_i, x_j) \rightarrow (y_i, y_j)$, a mapping from the input space to the output space

$p_{j|i}$: the probability that x_i choose x_j as its' neighbor.

x_i is the center of a Gaussian, $x_j \sim N(0, \sigma_i)$

$\sigma = 1/\sqrt{2}$

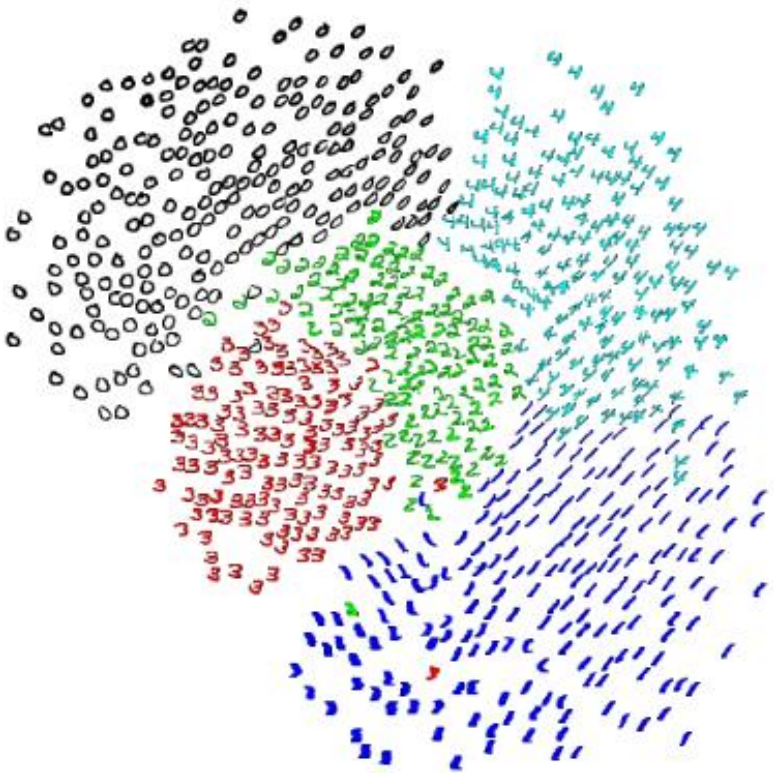
$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma_i^2)} \qquad q_{j|i} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} \exp(-\|y_i - y_k\|^2)}$$

Finally, a KL divergence is used:

$$C = \sum_i KL(P_i || Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

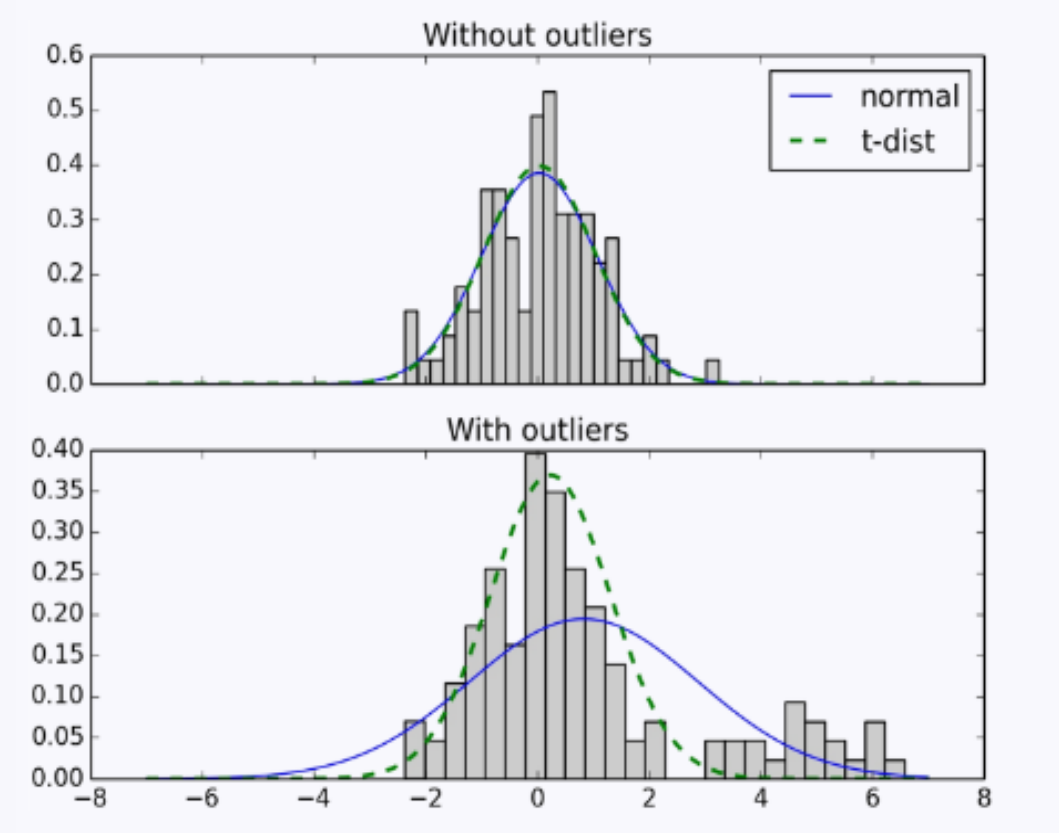
Symmetry SNE – one more step $\quad p_{ij} = \dfrac{p_{j|i} + p_{i|j}}{2n}$

The Crowding Problem



T-SNE

- Advances of T-distribution
  - Robust to outliers
  - Better capturing the overall characteristics

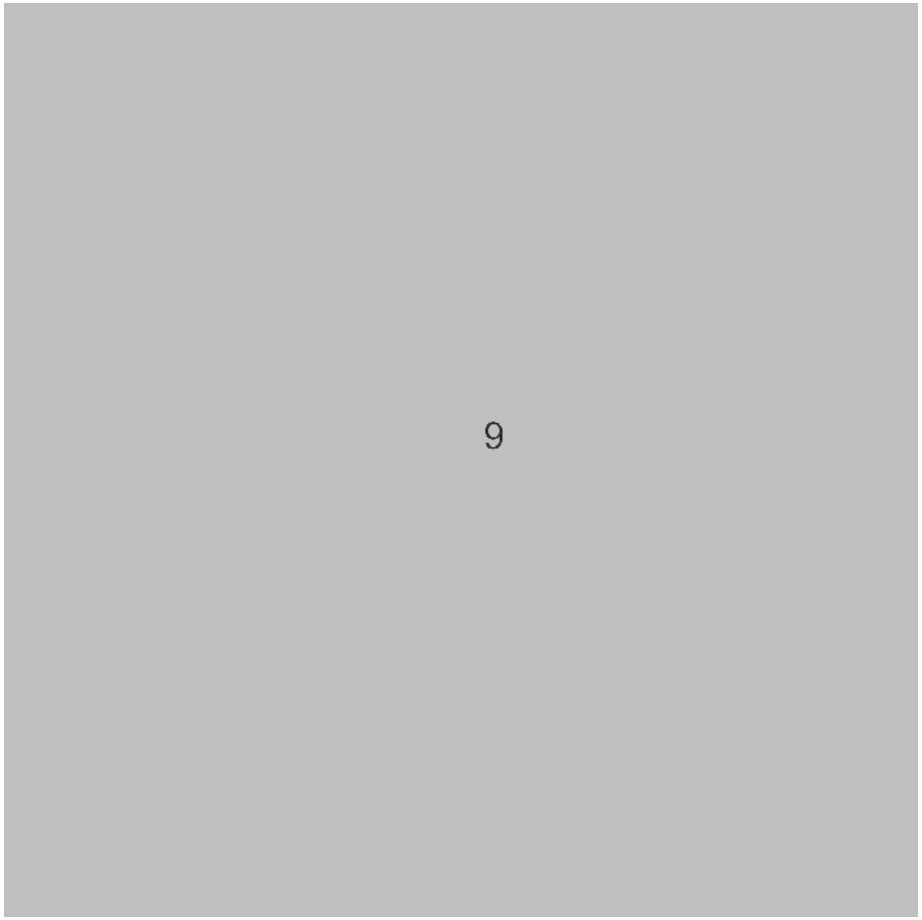


$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1}}$$

Distance between objects with low similarity should be further.

T-SNE



Word2Vec

“A word is known by the company it keeps”

- N-gram Model

- Problem Definition: a sentence with T words, $\mathrm{W} = (w_1, w_2, w_3, \ldots, w_T)$, find out the probability of this sentence.

- Bayes Chain Rule: too much parameters, $N + N^2 + \cdots + N^T$ ✖

- Markov Assumption ○

$$p(w_k | w_1^{k-1}) \approx p(w_k | w_{k-n+1}^{k-1}) \qquad \text{N=3}$$

$$\approx \frac{count(w_1^k)}{count(w_{k-n+1}^{k-1})}$$

- N-gram:

$$Context(w_i) = w_{i-n+1}^{i-1}$$

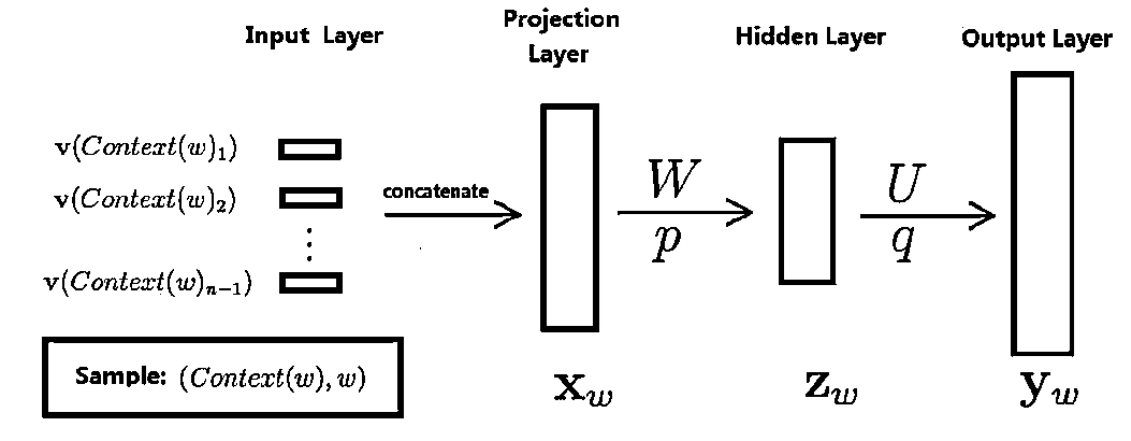
- Neural probability language model

- Input layer: $(Content(w), w)$, also $v(w)$
- Projections layer: combine word vectors together, thus get X_w
- Hidden layer:

$$Z_w = tanh(WX_w + p)$$

- Output layer: $dim(y_w) = |D|$, D is the dictionary.

$$y_w = Uz_w + q$$



- Too complex

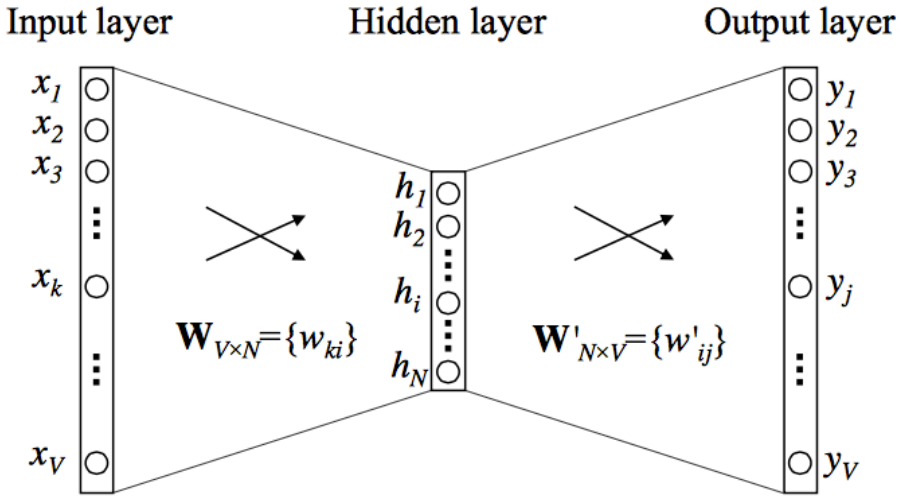
Traditional Method - Bag of Words	Word Embedding
• Uses one hot encoding • Each word in the vocabulary is represented by one bit position in a HUGE vector. • For example, if we have a vocabulary of 10000 words, and "Hello" is the 4th word in the dictionary, it would be represented by: 0 0 0 1 0 0 . . . . . 0 0 0 0 • Context information is not utilized	• Stores each word in as a point in space, where it is represented by a vector of fixed number of dimensions (generally 300) • Unsupervised, built just by reading huge corpus • For example, "Hello" might be represented as : [0.4, -0.11, 0.55, 0.3 . . . 0.1, 0.02] • Dimensions are basically projections along different axes, more of a mathematical concept.

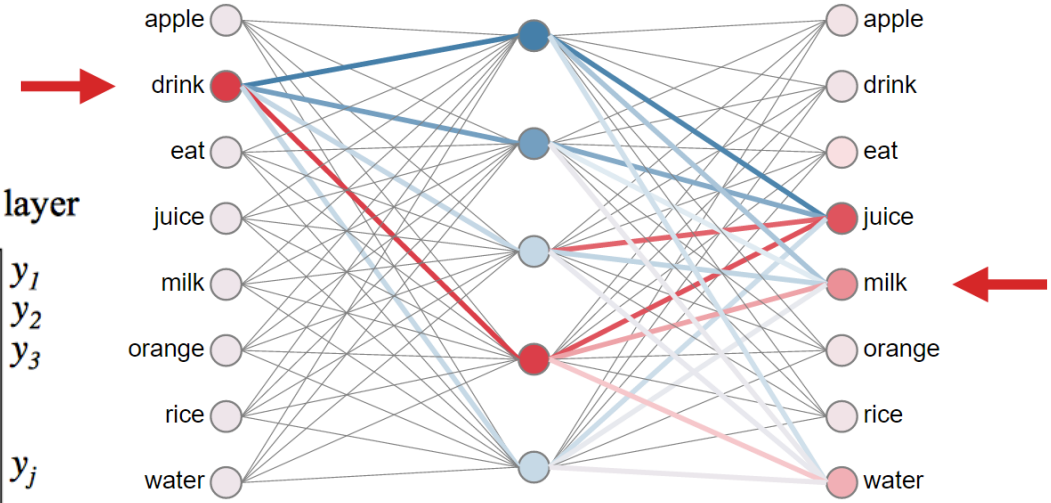
Intuitive Idea of Word2Vec

- Some things are better explained using a blackboard and chalk !

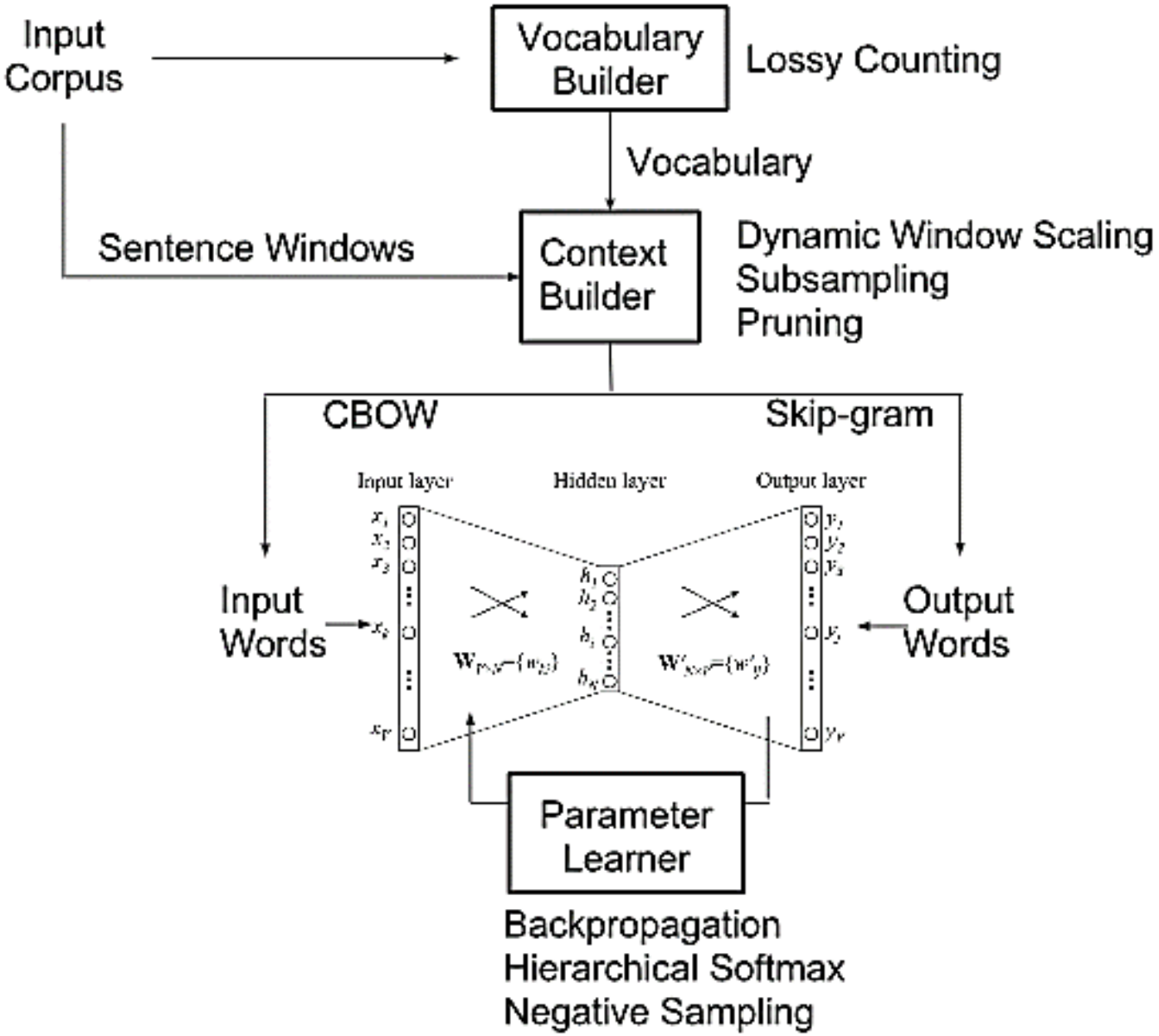
- **Input layer → Projection layer: change "JOINT" to "SUM"**
- **Remove hidden layer**

$$y_i = \sum_{j=1}^{n} x_j w_{ij} = \overrightarrow{w_i}^T x$$





Main Frame of Word2Vec



- Optimization of Softmax
  - Hierarchical Softmax (Huffman tree)
    - CBOW: Continuous Bag-of-Words Model
    - Skip-gram Model
  - Negative Sampling
    - CBOW: Continuous Bag-of-Words Model
    - Skip-gram Model

CBOW: Continuous Bag-of-Words Model (Hierarchical Softmax)

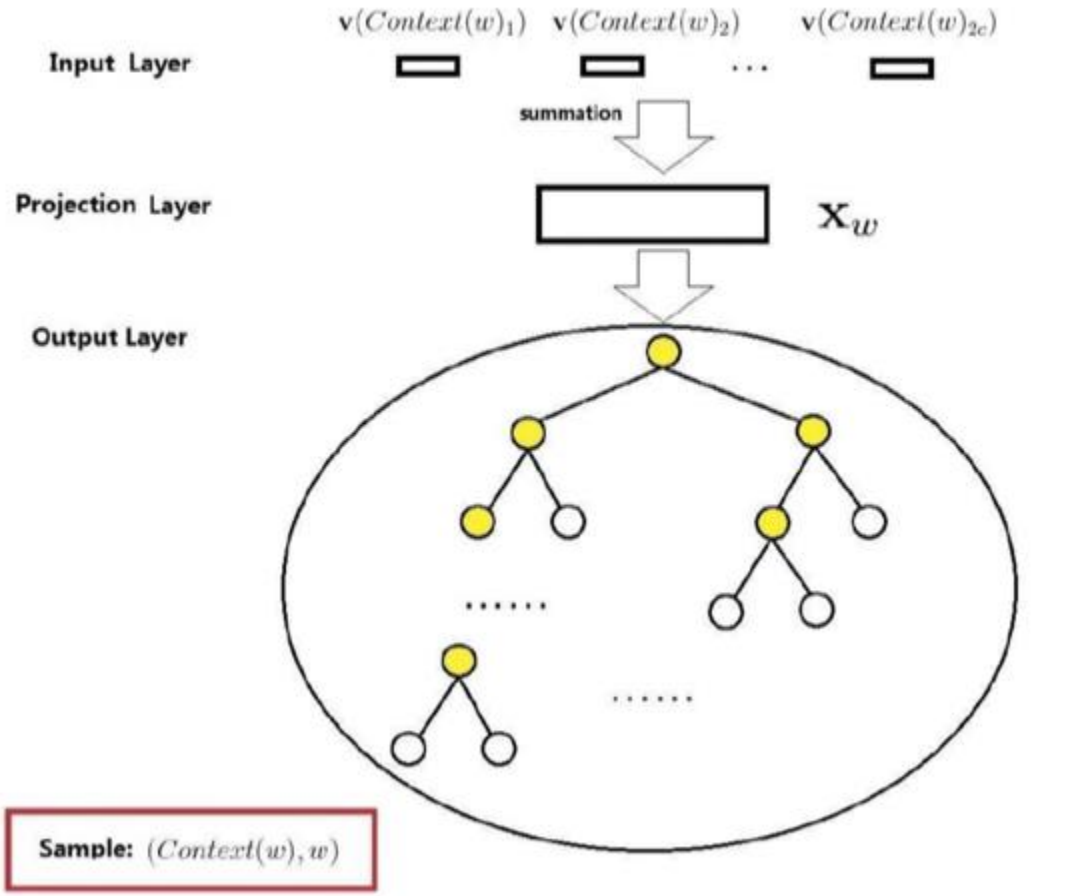
- Context(w): c words before w and c words after w.
- Input layer: a word vector containing 2c words
- Projection layer: summarization
- Output layer: a Huffman tree

$$X_w = \sum_{i=1}^{2c} v(Context(w_i)) \in \mathbb{R}^m$$

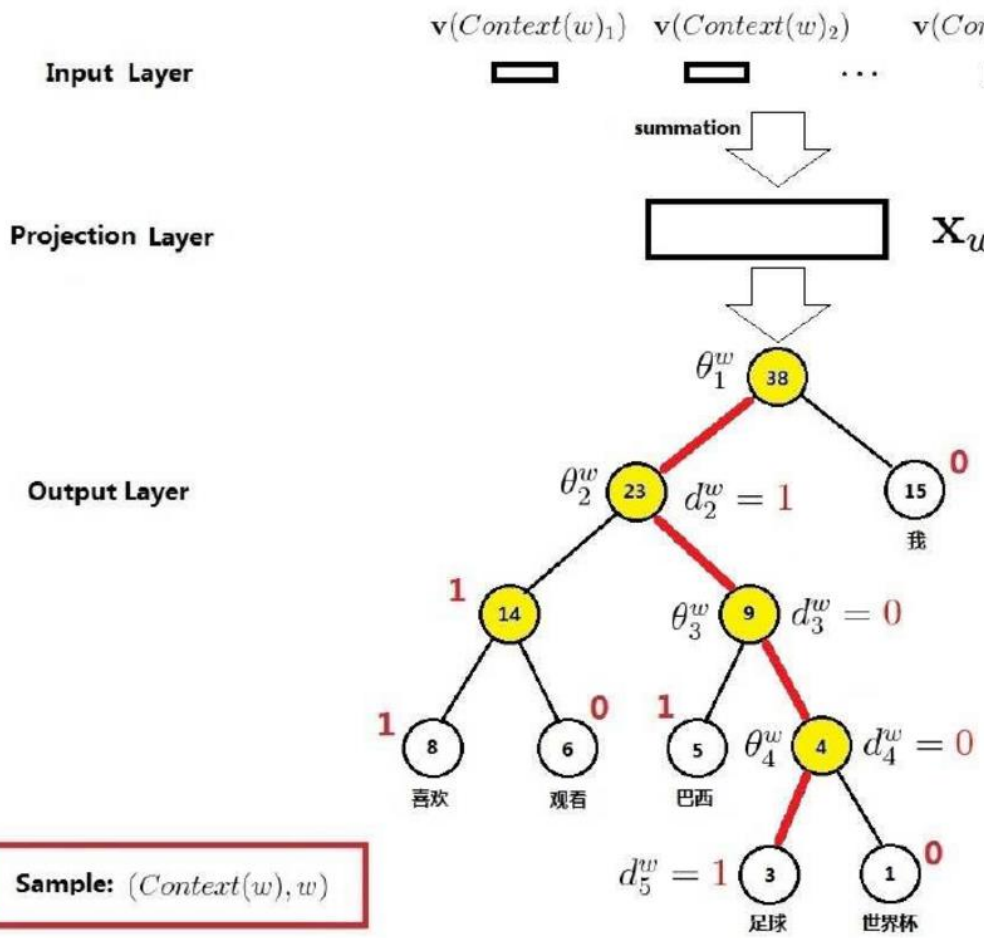
- Predicting the current word based on the context

$$\mathcal{L} = \sum_{w \in C} \log p(w|Context(w))$$

- order of words in the history does not influence the projection
- faster & more appropriate for larger corpora



Analysis of CBOW (Hierarchical Softmax)



$$\mathcal{L} = \sum_{w \in C} \log p(w|Context(w))$$

$$p(d_j^w|X_w, \theta_{j-1}^w) = \begin{cases} \sigma(X_w^T \theta_{j-1}^w) & d_j^w = 0 \\ 1 - \sigma(X_w^T \theta_{j-1}^w) & d_j^w = 1 \end{cases}$$

$$= [\sigma(X_w^T \theta_{j-1}^w)]^{1-d_j^w} \cdot [1 - \sigma(X_w^T \theta_{j-1}^w)]^{d_j^w}$$

1. 第 1 次: $p(d_2^w|\mathbf{x}_w, \theta_1^w) = 1 - \sigma(\mathbf{x}_w^\top \theta_1^w)$;
2. 第 2 次: $p(d_3^w|\mathbf{x}_w, \theta_2^w) = \sigma(\mathbf{x}_w^\top \theta_2^w)$;
3. 第 3 次: $p(d_4^w|\mathbf{x}_w, \theta_3^w) = \sigma(\mathbf{x}_w^\top \theta_3^w))$;
4. 第 4 次: $p(d_5^w|\mathbf{x}_w, \theta_4^w) = 1 - \sigma(\mathbf{x}_w^\top \theta_4^w)$,

$$p(\text{足球}|Contex(\text{足球})) = \prod_{j=2}^{5} p(d_j^w|\mathbf{x}_w, \theta_{j-1}^w).$$

Sample: $(Context(w), w)$

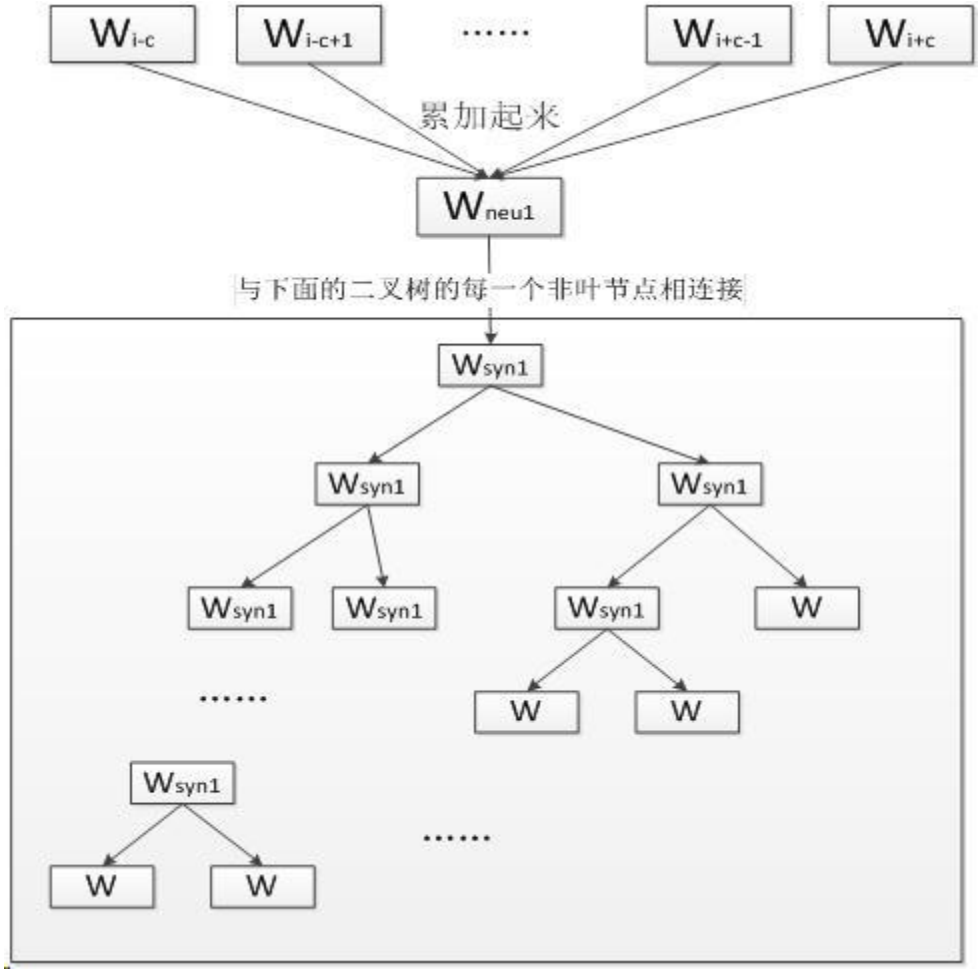
- **Positive** $\sigma(\mathbf{x}_w^\top \theta) = \dfrac{1}{1 + e^{-\mathbf{x}_w^\top \theta}}$
- **Negative** $1 - \sigma(\mathbf{x}_w^\top \theta)$

$$\mathcal{L} = \sum_{w \in C} log \prod_{j=2}^{l^w} \{[\sigma(X_w^T \theta_{j-1}^w)]^{1-d_j^w} \cdot [1 - \sigma(X_w^T \theta_{j-1}^w)]^{d_j^w}\}$$

$$= \sum_{w \in C} \sum_{j=2}^{l^w} \{(1 - d_j^w) \cdot log[\sigma(X_w^T \theta_{j-1}^w)] + d_j^w \cdot log[1 - \sigma(X_w^T \theta_{j-1}^w)]\}$$

CBOW + Hierarchical Softmax

网络结构-输入层与隐层



输入层：输入的是若干个词的词向量。

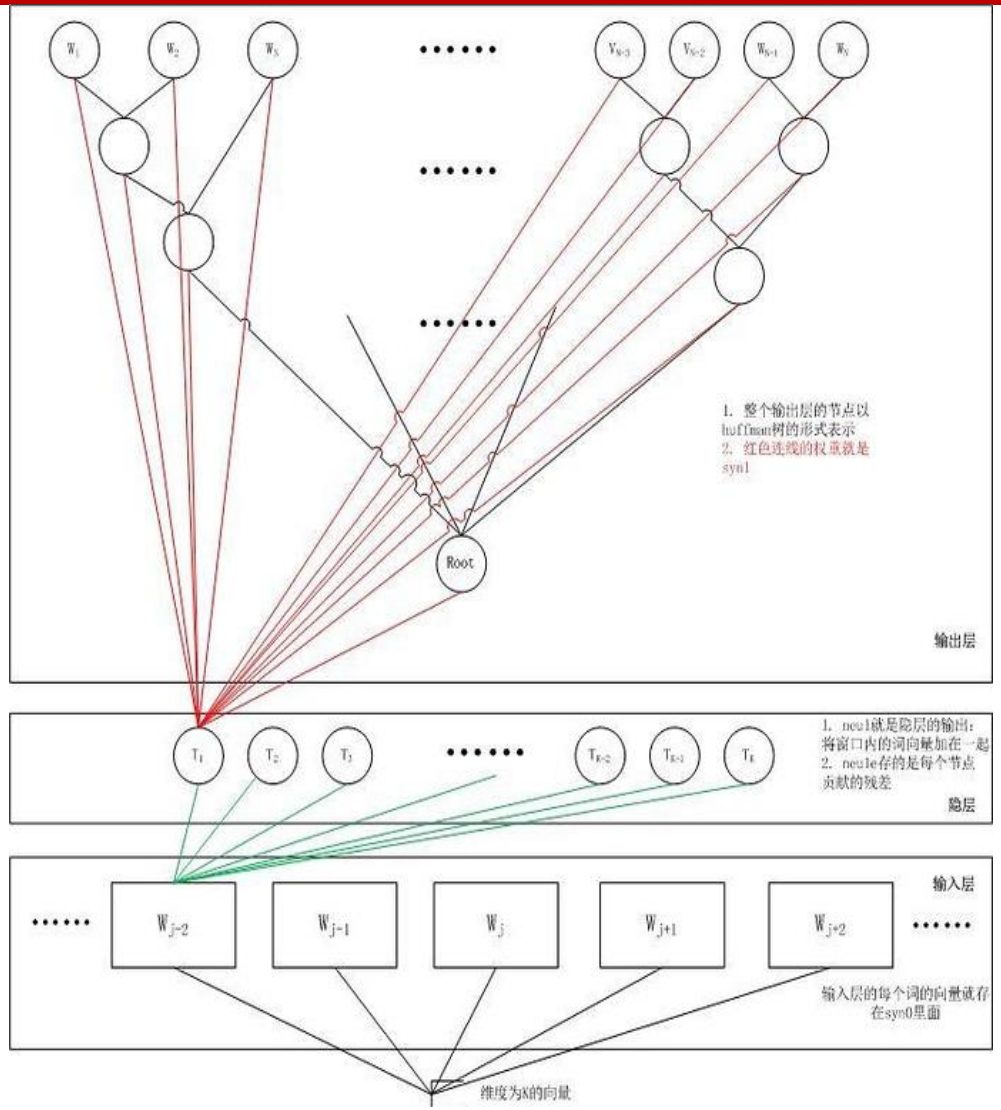
起初，每个单词都是一个随机 K 维向量。经过训练之后，该算法利用 CBOW 或者 Skip-gram 的方法获得了每个单词的最优向量。

隐层，是输入的若干个词向量的累加和，注意是向量的累加和，结果是一个向量。

取一个适当大小的窗口当做语境，输入层读入窗口内的词，将它们的向量（K维，初始随机）加和在一起，形成隐藏层K个节点。

霍夫曼树是根据词频建立的，word2vec做的是优化 **从根节点出发到每个叶子节点的路线概率值（边的概率的乘积）**。

网络结构-输出层



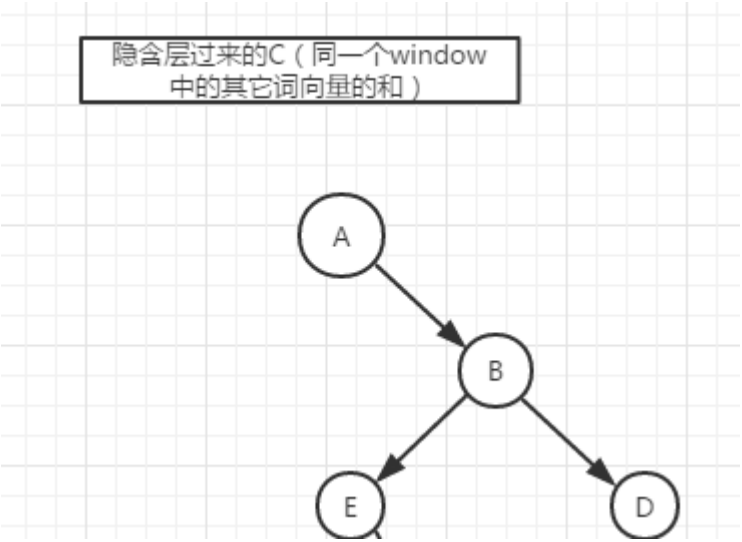
输出层是一个巨大的霍夫曼树，叶节点代表语料里所有的词（语料含有V个独立的词，则二叉树有|V|个叶节点）。

这样，对于叶节点的 每一个词，就会有一个全局唯一的编码，形如"010011"，不妨记左子树为1，右子树为0。接下来，隐层的每一个节点都会跟二叉树的内节点有连边，于是 对于二叉树的每一个内节点都会有K条连边，每条边上也会有权值。

对于语料库中的某个词W_t，对应着二叉树的某个叶子节点，因此它必然有一个二进制编码，如"010011"。在训练阶段，当给定上下文，要预测后面的词W_t的时候，我们就从二叉树的根节点开始遍历，这里的目标就是预测这个词的二进制编号的每一位。即对于给定的上下文，我们的目标是使得预测词的二进制编码概率最大。

输入的那几个词向量跟它们最终输出的到的那个词向量未必是同一个，而且基本不会是同一个词，只是它们往往有语义上的关系。这个网络最重要的是输出层的霍夫曼树的叶子节点上的那些词向量 。

网络结构-如何去做预测



举例：假设“吃”这个词在霍夫曼树中是的最右边那一个叶子节点D，根节点对应的词向量命名为A，根节点的右孩子节点对应的词向量命名为B，另外再假设“大家”、“喜欢”、“好吃”和“的”这四个词的词向量的和为C.
上面的那句话的每个词都计算后连乘起来得到联合概率，这个概率如果大于某个阈值，就认为是正常的话；否则就认为不是自然语言.

$$p\left(\text{吃}\middle|Context_{\text{吃}}\right)=(1-\sigma(A\cdot C))\cdot(1-\sigma(B\cdot C))$$

网络训练完成后，假如给定一句话s，这句话由词w1,w2,w3,…,wT组成，就可以利用**计算这句话是自然语言的概率**，计算的公式是下面的公式

$$\mathrm{p(s)}=\mathrm{p}(w_1,w_2,\cdots w_T)=\prod_{i=1}^{T}p(w_i|Context_i)$$

其中的Context表示的是该词的上下文，也就是这个词的前面和后面各若干个词，这个“若干”（后面简称c）一般是随机的，也就是一般会从1到5之间的一个随机数；每个 $p(w\mid context(w))$ 代表的意义是前后的c个词分别是那几个的情况下，出现该词的概率。举个例子就是：“大家 喜欢 吃 好吃 的 苹果”这句话总共6个词，假设对“吃”这个词来说c随机抽到2，则“吃”这个词的context是“大家”、“喜欢”、“好吃”和“的”，总共四个词，这四个词的顺序可以乱，这是word2vec的一个特点。

从霍夫曼树到条件概率的计算

word2vec的计算这个条件概率的方法是利用神经网络的能量函数，因为在能量模型中，能量函数的功能是把神经网络的状态转化为概率表示。
能量模型有个特别大的好处，就是能拟合所有的指数族的分布。那么，如果认为这些条件概率是符合某个指数族的分布的话，是可以用能量模型去拟合的。总之word2vec就认为这个条件概率可以用能量模型来表示了。
既然是能量模型，那么就需要能量函数，word2vec定义了一个非常简单的能量函数

$$E(A,C) = -(A \cdot C)$$ 这个式子非常关键，是后面推导的基础!!

其中A可以认为是某个词的词向量，C是这个词的上下文的词向量的和（向量的和），基本上就可以认为C代表Context；中间的点号表示两个向量的内积。
然后根据能量模型（这个模型假设了温度一直是1，所以能量函数没有分母了），就可以表示出词A的在上下文词向量C下的概率了。

$$\mathrm{p(A|C)} = \frac{e^{-E(A,C)}}{\sum_{v=1}^{V} e^{-E(wv,C)}}$$

其中V表示语料库里面的的词的个数，这个定义的意思是在上下文C出现的情况下，中间这个词是A的概率。

但这个概率其实并不好统计，**为了算一个词的的概率，得算上这种上下文的情况下所有词的能量，然后还计算指数值再加和。**

因此考虑用树的结构！！！

从霍夫曼树到条件概率的计算

考虑一种简单的情况：

假如把语料库的所有词分成两类，分别称为G类和H类，每类一半，其中词A属于G类，那么下面的式子就可以成立了

$$p(A \mid C) = p(A|G,C)p(G|C)$$

这个式子的的含义算明确的了，词A在上下文C的条件下出现的概率，与后面的这个概率相等——在上下文C的条件下出现了G类词，同时在上下文为C，并且应该出现的词是G类词的条件下，词A出现的概率。

这说明了一个问题，**计算一个词A在上下文C的情况下出现的概率，可以先对语料库中的词分成两簇，然后能节省计算**。

假设G，H这两类的簇中心也用G和H表示，那么上面那个式子中的$p(G|C)$可以用下面的式子计算

$$\mathrm{p(G|C)} = \frac{e^{-E(G,C)}}{e^{-E(G,C)}+e^{-E(H,C)}} = \frac{1}{1+e^{-\left(-(H-G)^{C}\right)}} = \frac{1}{1+e^{-E(H-G,C)}}$$

也就是说，可以不用关系那个是簇中心，只要利用一个$F = H - G$的类词向量的一个向量就可以计算 $P(G|C)$ 了。（转换为了向量的加减运算！！！）

另外一步：

$$\mathrm{p(A|G,C)} = \frac{e^{-E(A,C)}}{\sum_{W \in G} e^{-E(W,C)}}$$

由于在G内的词数量只有$V/2$个，也就是说计算分母的时候只要计算$V/2$个词的能量就可以了。

从霍夫曼树到条件概率的计算

来考虑复杂的情况：

把G类词再分成两个簇GG，GH，A在GH里面，然后

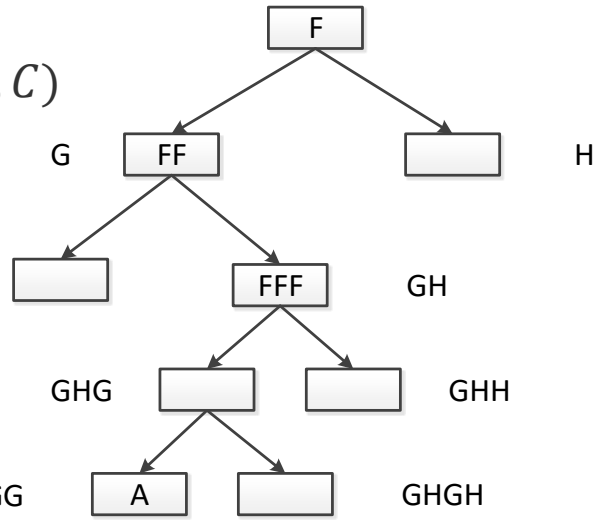
$$p(A \mid G, C) = p(A|GH, G, C)p(GH|G, C)$$

同样有

$$p(GH|G, C) = \frac{1}{1 + e^{-E(GG-GH,C)}}$$

和

$$p(A|GH, G, C) = \frac{1}{\sum_{W \in H} e^{-E(W,C)}}$$



同样可以把$GG - GH$用一个类词向量表达，这时候

$$p(A \mid C) = p(A|GH, G, C)p(GH|G, C)p(G|C)$$

继续下去假设继续分到GHG簇的时候只剩两个词了，再分两簇为GHGG和GHGH，其中的簇GHGG就只有一个词A，那么$p(A \mid C)$可以用下面的式子算

$$p(A \mid C)$$
$$= p(A \mid GHGG, GHG, GH, G, C)p(GHGG|GHG, GH, G, C)p(GHG|GH, G, C)p(GH|G, C)p(G|C)$$

其中$p(A|GHGG, GHG, GH, G)$是1，因为只有一个单词。那么就有

$$p(A \mid C) = p(GHGG|GHG, GH, G, C)p(GHG|GH, G, C)p(GH|G, C)p(G|C)$$

也就是

$$\mathrm{p(A|C)} = \frac{1}{1+e^{-E(GHH-GHG,C)}} \cdot \frac{1}{1+e^{-E(GG-GH,C)}} \cdot \frac{1}{1+e^{-E(H-G,C)}}$$

假设再令$FFF = GHH - GHG$，$FF = GG - GH$，$F = H - G$，那么$p(A|C)$只要算这三个词与上下文C的能量函数了，确实比原来的要节省很多计算的。

从霍夫曼树到条件概率的计算

接上面的例子：
对于上面的霍夫曼树来说假设G表示向右，H表示向左。
但是一个词总是会一会向左，一会向右的，也就是在根节点那里，一会是p(G|C)那么F=H-G，一会又是p(H|C)那么F=G-H，如果F在每个节点都是唯一一种表示，就可以直接用一次词向量表示这个非叶节点了。令F一直是等于H-G，那么一直有：

$$p(H|C) = \frac{1}{1 + e^{-E(F,C)}}$$

并且：$p(G|C) = 1 - p(H|C)$
总结下来，就是这个式子：

$$\mathrm{p(w|Context)} = \prod_{k=1}^{K} p(d_k|q_k,C) = \prod_{k=1}^{K} \left(\left(\sigma(q_k \cdot C) \right)^{1-d_k} \cdot \left(1 - \sigma(q_k \cdot C) \right)^{d_k} \right)$$

其中C表示上下文的词向量累加后的向量，q_k 表示从根节点下来到叶子节点的路径上的那些非叶节点，d_k 就是编码了，也可以说是分类，因为在霍夫曼树的每个非叶节点都只有两个孩子节点，那可以认为当 w_i 在这个节点的左子树的叶子节点上时$d_k = 0$，否则$d_k = 1$。这样的话每个词都可以用一组霍夫曼编码来表示，就有了上面的那个式子中间的那个d_k，整个$p(w \mid Context)$就可以用霍夫曼树上的若干个非叶节点和词w的霍夫曼编码来计算了。

从根节点出发到叶子节点的路线中，每条边都对应着一个概率值，这个概率序列即构建了叶子节点所代表的词的词向量。

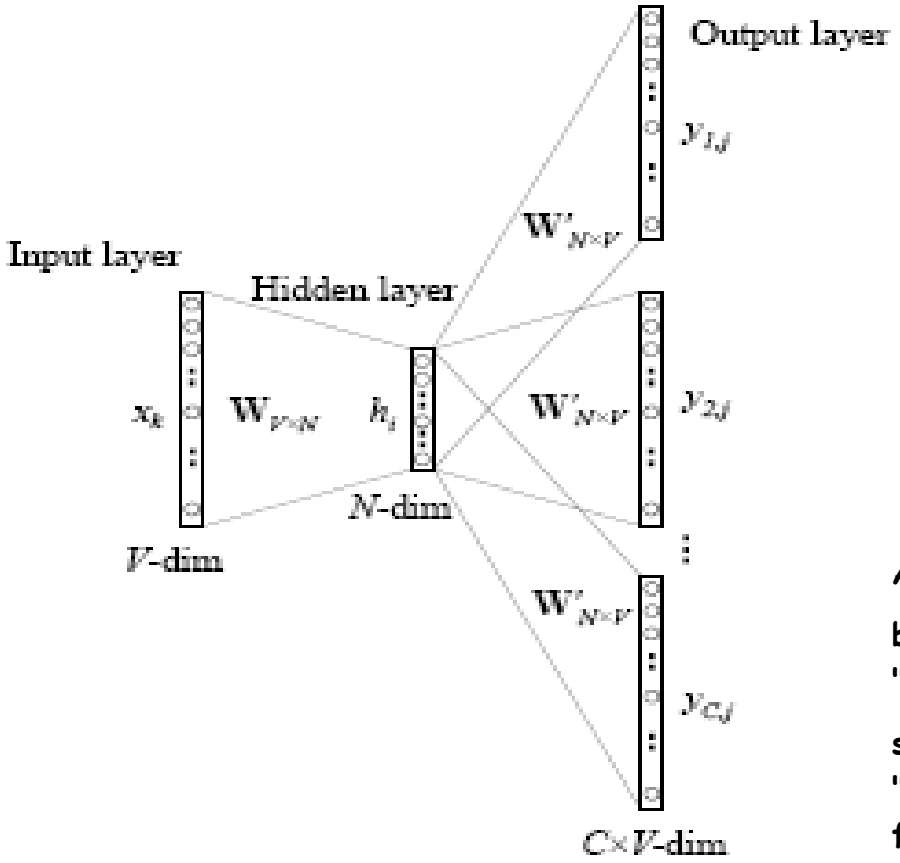
小结：

虽然hierarchical softmax一般被认为只是用于加速，但是仍然可以感性地理解一下为啥它会奏效。

二叉树里面的每一个内节点实际上是一种隐含概念的分类器（二元分类器，因为二进制编码就是0/1），它的输出值的大小预示着当前上下文能够表达该隐含概念的概率，而一个词的编码实际上是一堆隐含概念的表达（注意，这个隐含概念的表达和词向量的维度所表达的隐含概念是不一样的）。我们的目标就在于找到这些当前上下文对于这些概念分类的最准确的那个表达（即目标词向量）。由于概念之间实际上是有互斥关系的（二叉树保证），即在根节点如果是”1″，即可以表达某一概念，那么该上下文是绝对不会再有表达根节点是”0″的其他情况的概念了，因此就不需要继续考虑根节点是”0″的情况了。因此，整个hierarchical softmax可以被看作完全不同于传统softmax的一套。

Skip-gram model (Negative Sampling)

- Context(w): c words before w and c words after w.
- Input layer: a word vector containing 2c words
- Projection layer: summarization
- Output layer: a Huffman tree



- maximize classification of a word based on another word in the same sentence

$$\mathcal{L} = \sum_{w \in C} \log p(Context(w)|w)$$

- better word vectors for frequent words, but slower to train

An example: "Insurgents killed in ongoing fighting"

bigrams = ["insurgents killed", "killed in", "in ongoing", "ongoing fighting"]

skip_2_bigrams = ["insurgents killed", "insurgents in", "insurgents ongoing", "killed in", "killed ongoing", "killed fighting", "in ongoing", "in fighting", "ongoing fighting"]

z = context(w)

LR:

$$p(z|w) = \begin{cases} \sigma\left(v(w)^T \theta^z\right), & L^u(z) = 1 \\ 1 - \sigma\left(v(w)^T \theta^z\right), & L^u(z) = 0 \end{cases}$$

The likelihood of positive sample:

$$\prod_{(w,z) \in D} p(L^u(z) = 1|w)$$

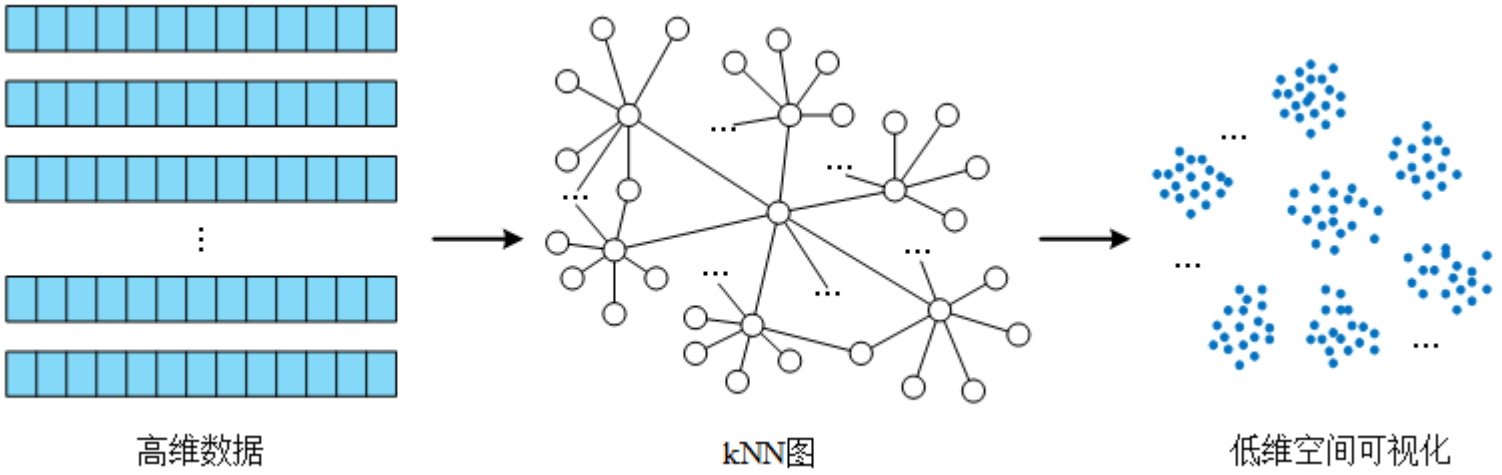
The likelihood of negative sample:

$$\prod_{(w,z) \notin D} p(L^u(z) = 1|w)$$

So:

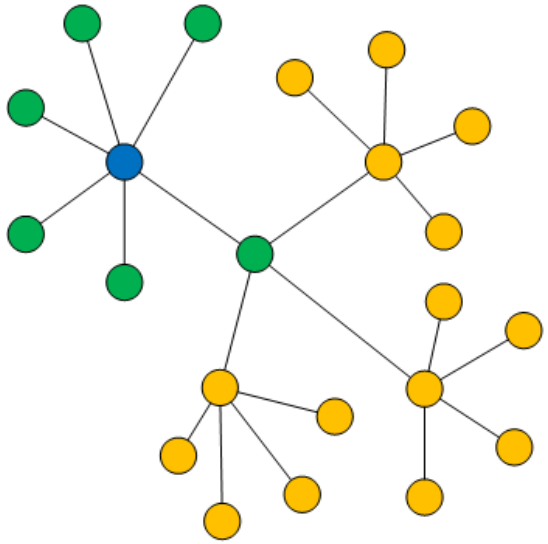
$$argmax\, \mathcal{L} = \log\left(\prod_{(w,z) \in D} p(L^u(z) = 1|w) \prod_{(w,z) \notin D} (1 - p(L^u(z) = 1|w)) \right)$$

- You may consider it as an improved version of t-SNE.



- Drawbacks of t-SNE
  - Efficiency decrease sharply when dealing with massive data.
  - Sensitive to parameters.

- Optimization for KNN graph construction.



the center point

( ) positive

( ) negative

- How to project this to a low-dimensional space???

- The distance between positive samples should be close, vice versa.

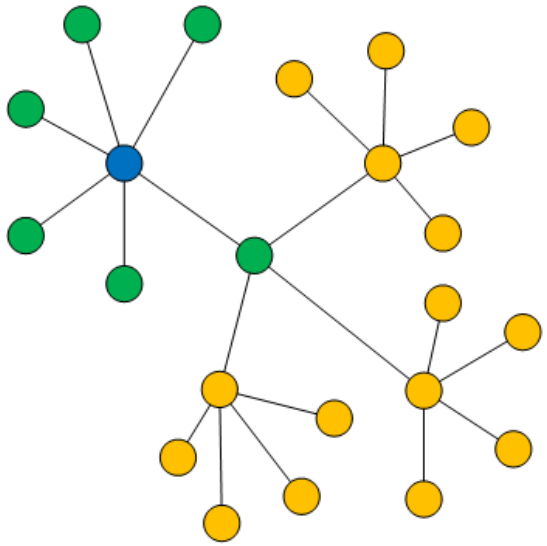
- Case 1: unweighted KNN network

$$y_i, y_j \in \mathbb{R}^{low}$$

$$p(e_{ij} = 1) = f(\|y_i - y_j\|^2)$$

and:

$$f(x) = \frac{1}{1 + x^2}$$



- Case 2: weighted KNN network , w_{ij}

$$p(e_{ij} = w_{ij}) = p(e_{ij} = 1)^{w_{ij}}$$

positive E, negative $\overline{E}$, so:

$$Object\ function = \prod_{(i,j)\in E} p(e_{ij} = 1)^{w_{ij}} \prod_{(i,j)\in \overline{E}} \left(1 - p(e_{ij} = 1)\right)^{\gamma}$$

γ is the weight we set for negative samples.

- **Case 2:**

$$Object\ function = \sum_{(i,j)\in E} w_{ij}\, p(e_{ij}=1) + \sum_{(i,j)\in \overline{E}} \gamma(1 - p(e_{ij}=1))$$

Notice: $\overline{E}$ is too large

→ Negative Sampling

Select M nodes, combine them with i to form negative samples

$$Object\ function = \sum_{(i,j)\in E} w_{ij}\, p(e_{ij}=1) + \sum_{k=1}^{M} \gamma \cdot \log\left(1 - p\left(e_{ij_k}=1\right)\right)$$

Similar to Skip-gram (Negative Sampling)

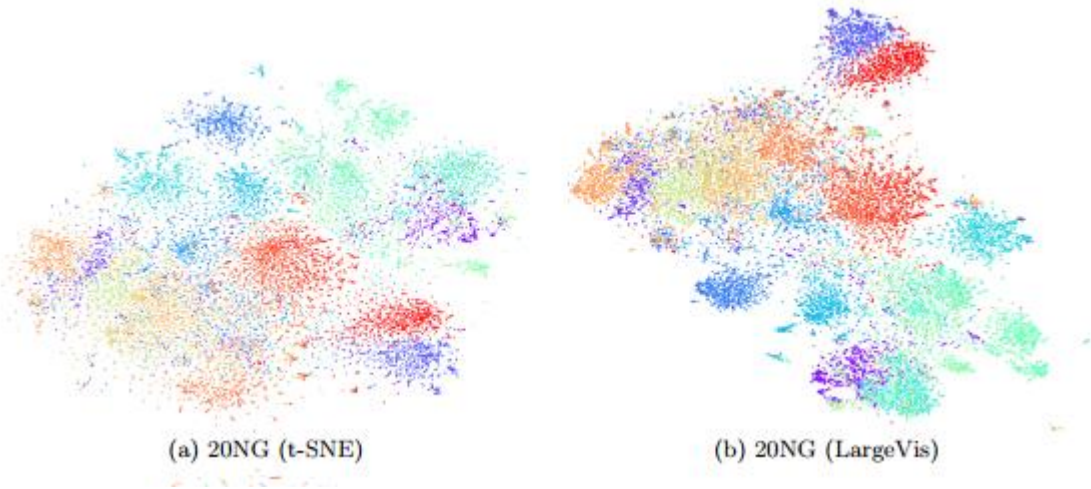
w_{ij} still exists, the range varies a lot. → gradient explosion and vanishing problem （梯度剧增与消失问题）

- **Case 2 :**

  Introduce edge sampling (LINE) to LargeVis

Why:

- **After sampling, weights are the same, thus we get a smooth gradient.**
- **This sampling follows the weighted sampling strategy, edge with larger weight transfer to more edges.**



(a) 20NG (t-SNE)

(b) 20NG (LargeVis)

Summary

- SNE
- t-SNE
- LargeVis
- word2Vec
- Skip-gram
- Negative Sampling
- Dimension Reduction
- Linear Methods
- Nonlinear Methods
- Manifold Learning
- Node2Vec

Lookalike

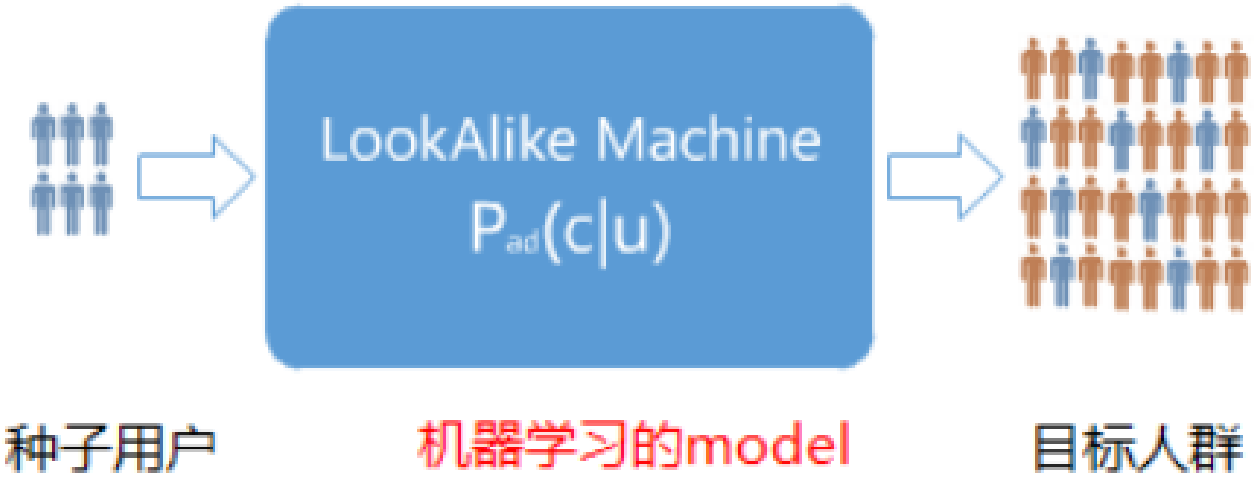
A real application of Node2Vec

社交 Lookalike 算法

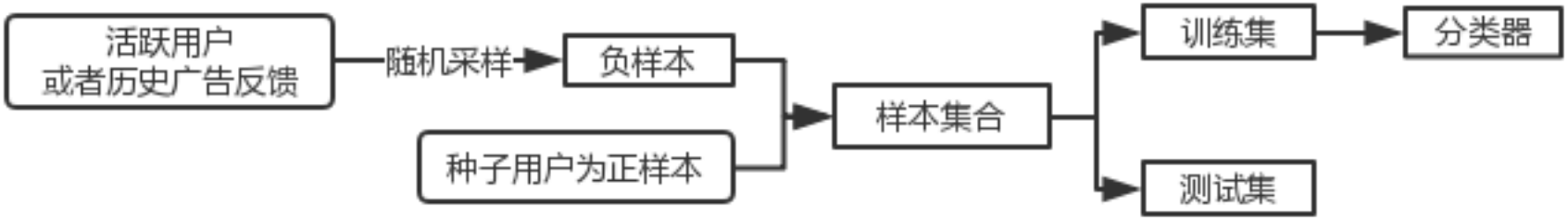
- 问题背景： 如何帮广告主找到潜在客户？

- 解决方案：Lookalike系统

- 转化为机器学习的模型
  - 正样本：广告主提供的客户名单（称为种子用户）
  - 负样本：从活跃用户（非种子用户）随机抽样
  - 打分结果：利用model对活跃用户进行打分排序，取出目标数量



- **Lookalike最常用模型：LR model**

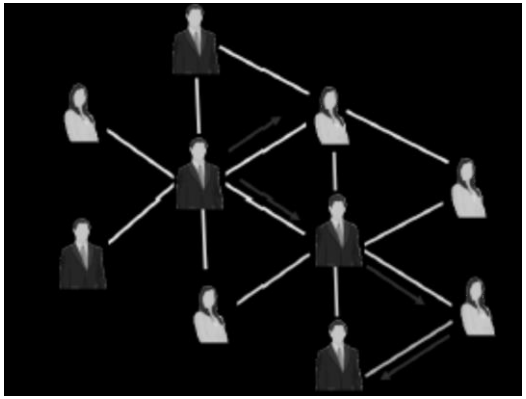


Lookalike模型	google	facebook	yahoo
建模方法	Predict model	Predict model	Linear SVM, GBDT, LR
主要特征	近30天的网页浏览行为，app行为，搜索行为，网页分类，query分类	社交行为，人口学标签	人口学标签，网页浏览行为，app行为

- **Idea：同样转为 预估模型？**

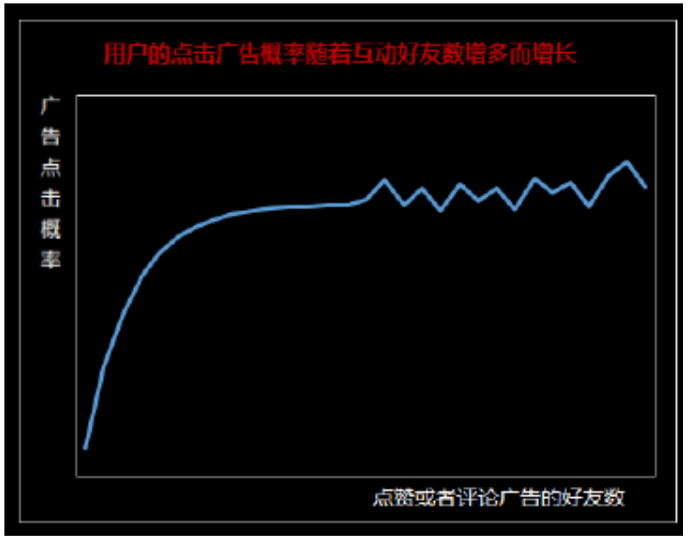
- **具体应用场景：微信朋友圈的广告投放**

微信朋友圈是一种社交场合



- 好友评论说不错...买买买
- Who点赞了...我也点开看看

- 用户以 **社交网络** 进行链接
- 行为以 **社交传播** 相互感染



数据分析结果

- 用户点击广告的概率与点赞评论好友数正相关

社交关系链的两个核心价值

社交同质性：同质性是社交网络结构性的基本外部原因 – 物以类聚人以群分

对于朋友圈广告，种子用户喜欢某广告产品，那么他的好友也有可能喜欢

社交影响力：指社交个体的思想、态度、情绪、习惯乃至价值观会受到其所在社群的其他人的影响对于朋友圈广告，因为好友点赞评论了某个广告，用户也进行点赞评论

对于朋友圈广告，因为好友点赞评论了某个广告，用户也进行点赞评论

社交 Lookalike 算法的 intuition

扩散种子用户的好友作为潜在用户

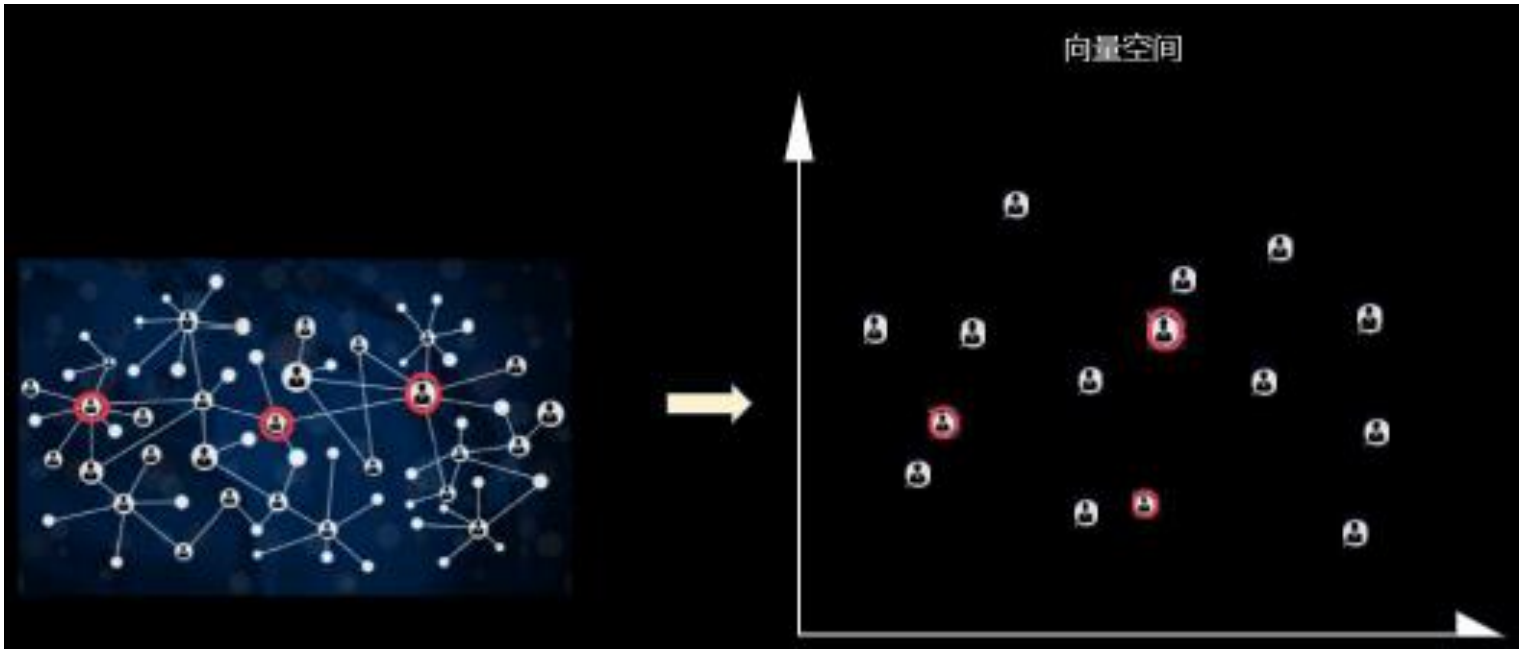
社交相似度模型

根据社交行为数据，预测广告喜好的相似度

数据都是以 **网络** 的形式组织的

社交特征工程

- **表达学习：Network Embedding**
- **将网络节点表达为一个向量：Node2Vec**



社交特征工程

- **Word2Vec：将一个词 embedding为一个向量**
- **单词的语义相似性**

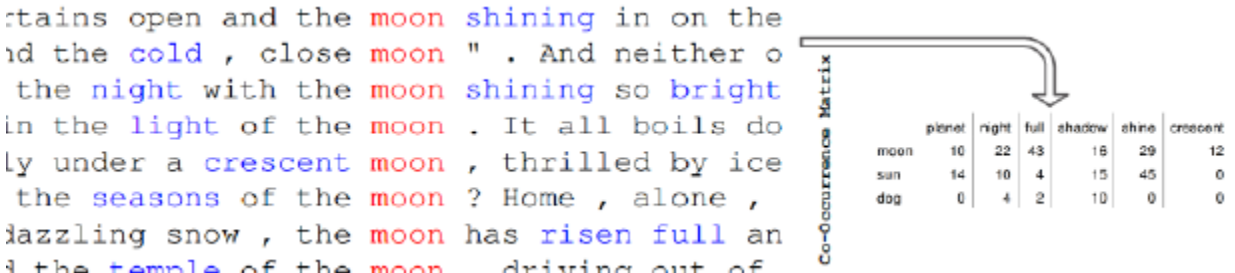
Words closest to “Sweden”

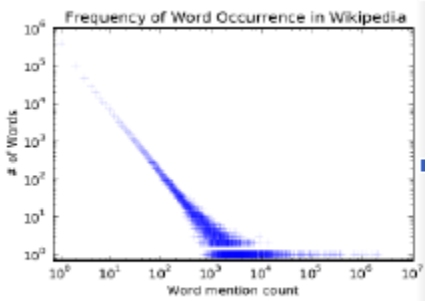
Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

社交特征工程

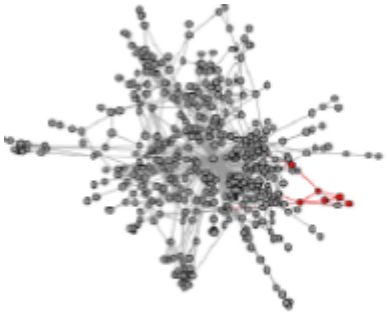
- From Word2Vec to Node2Vec
- 相似之处：幂律分布

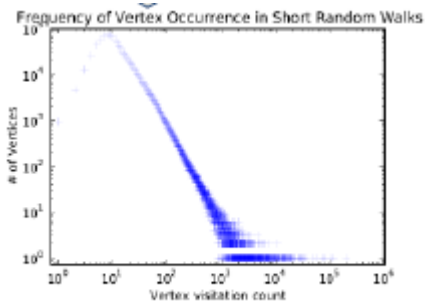
还有吗？？





(b) Wikipedia Article Text

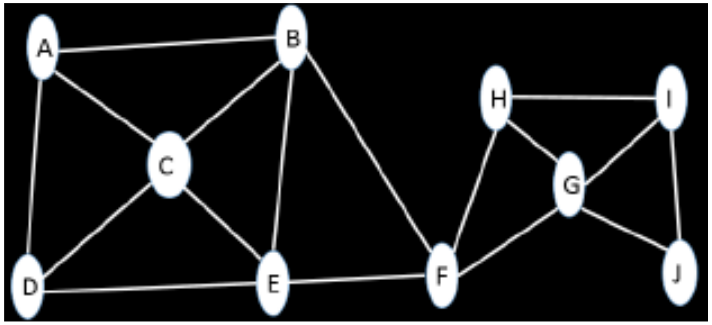

Scale Free Graph



(a) YouTube Social Graph

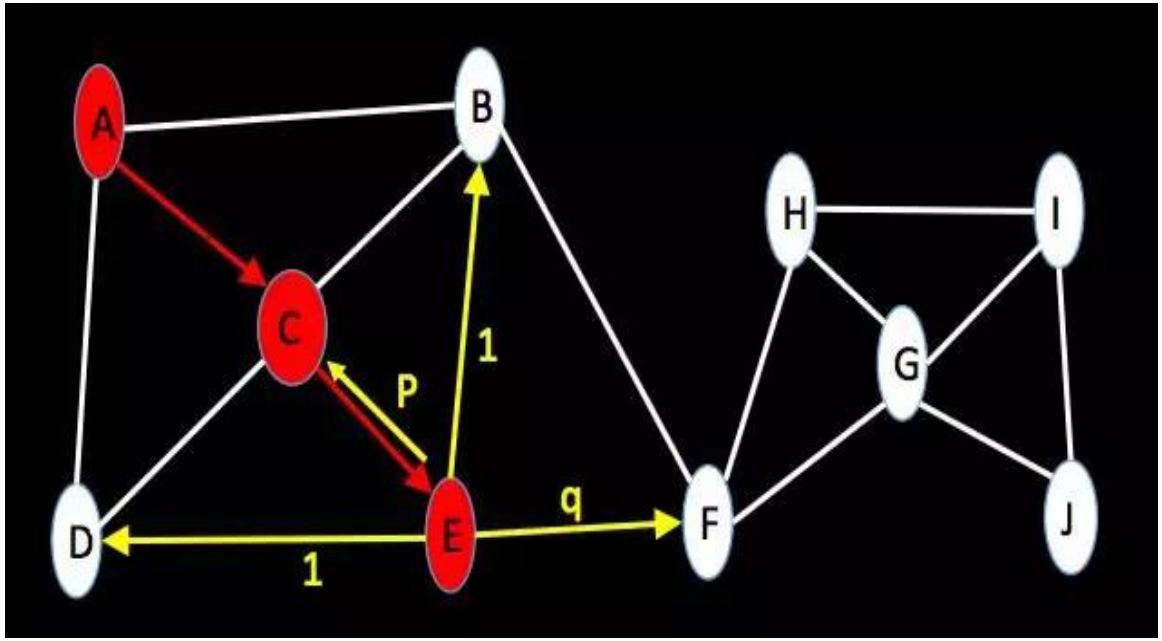
- **From Word2Vec to Node2Vec：**
  - **Node2Vec = Random Walk + Word2Vec**

- **Node2vec的关键点：**
  - **Random walk的算法：选择邻居节点非常关键！**
  - **大规模计算**



Random walk算法

- 选择邻居节点
- 抽样，减少计算量

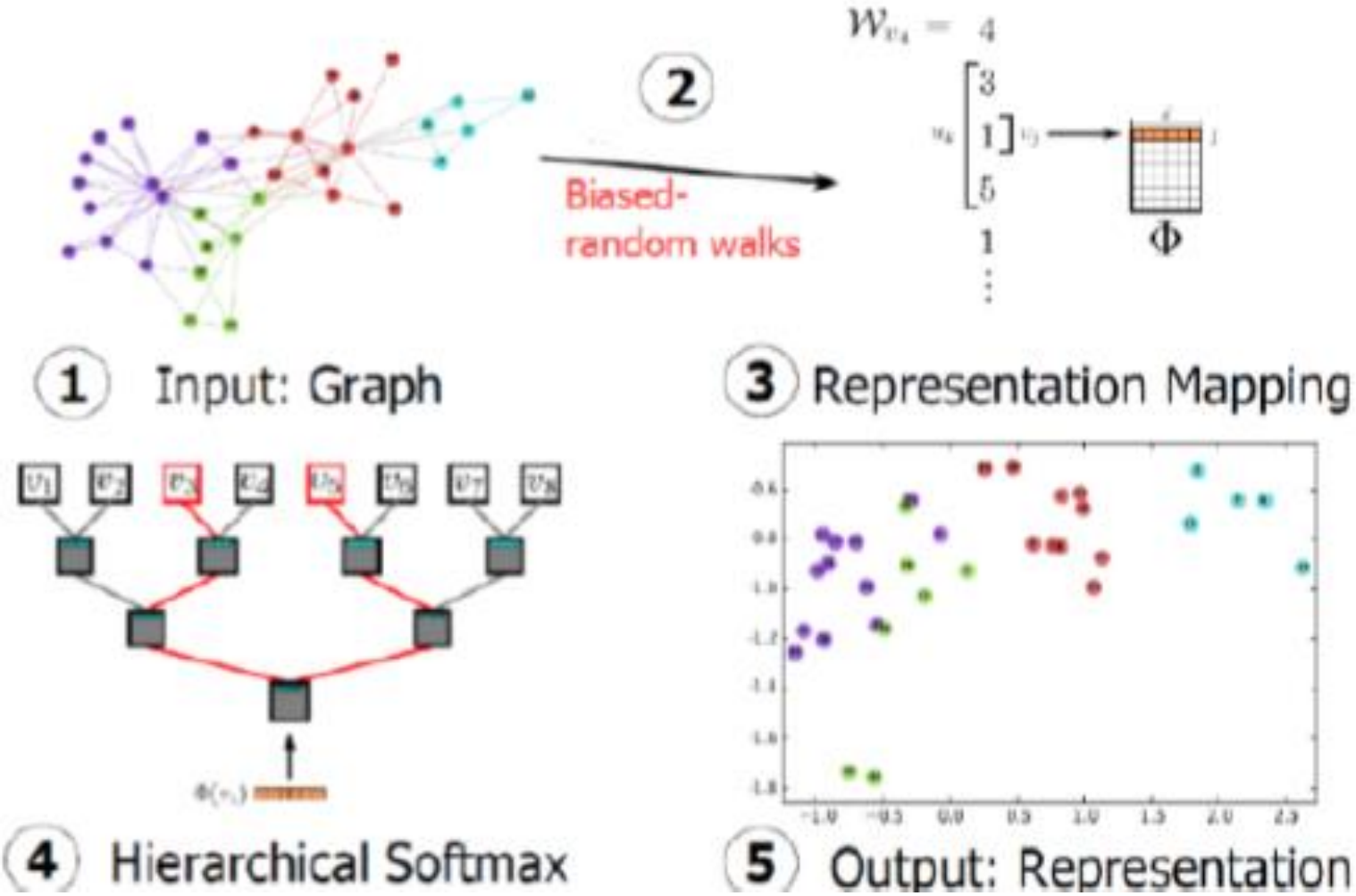


Biased-Random walk

- 考虑社团结构
- 设置两个参数，p，q

- **p 越大，越容易往社团内部走，体现同质性**
- **q 越大，越容易往社团外部走，体现结构相似（比如桥节点）**

- **Node2Vec=Biased Random walk+ word2vec**



相关资源

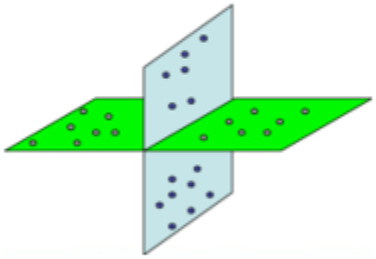
博客：

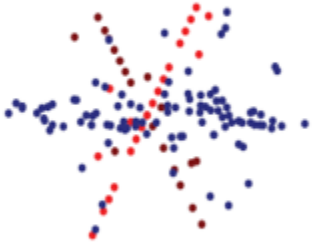
1. 从SNE到t-SNE再到LargeVis
2. Word2Vec-知其然知其所以然（这篇忽略太多过程了，建议结合后面的一起看）
3. 深度学习word2vec学习笔记【DOC】.（讲的最好的一篇）
4. word2vec源码解析之word2vec.c
5. word2vec 中的数学原理详解（四）基于 Hierarchical Softmax 的模型
6. word2vec 中的数学原理详解（五）基于 Negative Sampling 的模型
7. 当机器学习遇上复杂网络——社交lookalike算法实践

论文：

- Hinton G E, Roweis S T. Stochastic neighbor embedding[C]//Advances in neural information processing systems. 2002: 833-840.
- Van der Maaten L, Hinton G. Visualizing data using t-SNE[J]. Journal of Machine Learning Research, 2008, 9(2579-2605): 85.
- Mikolov T, Sutskever I, Chen K, et al. Distributed representations of words and phrases and their compositionality[C]//Advances in neural information processing systems. 2013: 3111-3119.
- Goldberg Y, Levy O. word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method[J]. arXiv preprint arXiv:1402.3722, 2014.

Less is More…





Thank you ☺