Word Embedding Using Neural Networks

Pengyu Hong
COSI 101A

What is Word Embedding?

Assume the dictionary has V words

- One-Hot representation of the i-th word

$$\vec{x}_i = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}_{V\times 1} \longleftarrow \text{the } i\text{-th element}$$

Hard to compute word similarity

- Word-embedding representation of the i-th word

$$\vec{x}_i = \begin{bmatrix} 0.03 \\ \vdots \\ 0.52 \\ \vdots \\ 0.12 \end{bmatrix}_{d\times 1}$$

Typically $50 \leqq d \leqq 1000 << V$

Word Embedding

- Measure the semantic similarity between two words are by calculating the cosine similarity between their corresponding word vectors.
- Similar words should have similar embeddings.

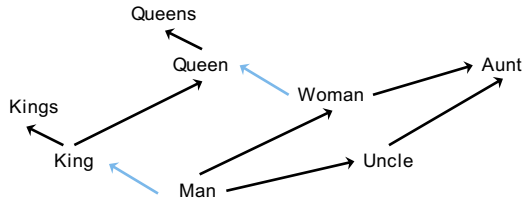
FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Word Embedding

- Tend to obey the laws of analogy.

For example, *"Woman is to queen as man is to king"*.

$$\vec{x}_{queen} - \vec{x}_{women} + \vec{x}_{man} \approx \vec{x}_{king}$$

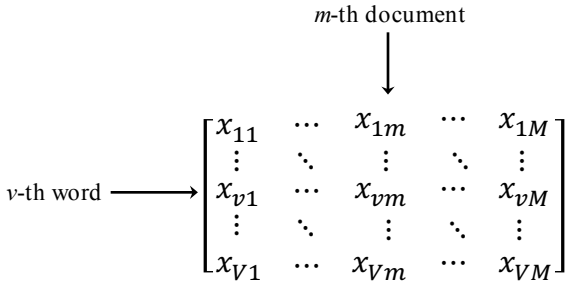


Word Embedding

- Embeddings of phrases, sentences, …
- Learn embedding from samples

Learn Word Embedding by Counting
– Latent Semantic Analysis

Construct a word-document matrix **X**



Each column vector represent a document in words

Each row vector represent a word in documents

Latent Semantic Analysis

Singular Value Decomposition of $\mathbf{X}=\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}$

$$\underset{\mathbf{X}}{\begin{bmatrix} x_{11} & \cdots & x_{1M} \\ \vdots & \ddots & \vdots \\ x_{V1} & \cdots & x_{VM} \end{bmatrix}} = \underset{\mathbf{U}}{\begin{bmatrix} \begin{bmatrix} \\ \vec{u}_1 \\ \\ \end{bmatrix} & \cdots & \begin{bmatrix} \\ \vec{u}_L \\ \\ \end{bmatrix} \end{bmatrix}} \underset{\mathbf{\Sigma}}{\begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_L \end{bmatrix}} \underset{\mathbf{V}^{\mathbf{T}}}{\begin{bmatrix} [& &] \\ & \vec{v}_1 & \\ & \vdots & \\ [& &] \\ & \vec{v}_L & \\ & &] \end{bmatrix}}$$

Choose $K < L$ and get $\hat{\mathbf{X}}$ as the best rank K approximation of $\mathbf{X}$

$$\hat{\mathbf{X}} = \underset{\mathbf{U}_K}{\begin{bmatrix} \begin{bmatrix} \\ \vec{u}_1 \\ \\ \end{bmatrix} & \cdots & \begin{bmatrix} \\ \vec{u}_K \\ \\ \end{bmatrix} \end{bmatrix}} \underset{\mathbf{\Sigma}_K}{\begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_K \end{bmatrix}} \underset{\mathbf{V}_K^{\mathbf{T}}}{\begin{bmatrix} [& &] \\ & \vec{v}_1 & \\ & \vdots & \\ [& &] \\ & \vec{v}_K & \\ & &] \end{bmatrix}}$$

Latent Semantic Analysis

Singular Value Decomposition of $\mathbf{X}=\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}$

$$\underset{\mathbf{X}}{\begin{bmatrix} x_{11} & \cdots & x_{1M} \\ \vdots & \ddots & \vdots \\ x_{V1} & \cdots & x_{VM} \end{bmatrix}} = \underset{\mathbf{U}}{\begin{bmatrix} \begin{bmatrix} \\ \vec{u}_1 \\ \\ \end{bmatrix} & \cdots & \begin{bmatrix} \\ \vec{u}_L \\ \\ \end{bmatrix} \end{bmatrix}} \underset{\mathbf{\Sigma}}{\begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_L \end{bmatrix}} \underset{\mathbf{V}^{\mathbf{T}}}{\begin{bmatrix} [& &] \\ & \vec{v}_1 & \\ & \vdots & \\ [& &] \\ & \vec{v}_L & \\ & &] \end{bmatrix}}$$

Choose $K < L$ and get $\hat{\mathbf{X}}$ as the best rank K approximation of $\mathbf{X}$

$$\hat{\mathbf{X}} = \underset{\mathbf{U}_K}{\begin{bmatrix} \begin{bmatrix} \\ \vec{u}_1 \\ \\ \end{bmatrix} & \cdots & \begin{bmatrix} \\ \vec{u}_K \\ \\ \end{bmatrix} \end{bmatrix}} \underset{\mathbf{\Sigma}_K}{\begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_K \end{bmatrix}} \underset{\mathbf{V}_K^{\mathbf{T}}}{\begin{bmatrix} [& &] \\ & \vec{v}_1 & \\ & \vdots & \\ [& &] \\ & \vec{v}_K & \\ & &] \end{bmatrix}}$$

Latent Semantic Analysis

$$\widehat{\mathbf{X}} = \begin{bmatrix} \begin{bmatrix} \\ \vec{u}_1 \\ \\ \end{bmatrix} & \cdots & \begin{bmatrix} \\ \vec{u}_K \\ \\ \end{bmatrix} \end{bmatrix} \begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_K \end{bmatrix} \begin{bmatrix} [& \vec{v}_1 &] \\ & \vdots & \\ [& \vec{v}_K &] \end{bmatrix}$$

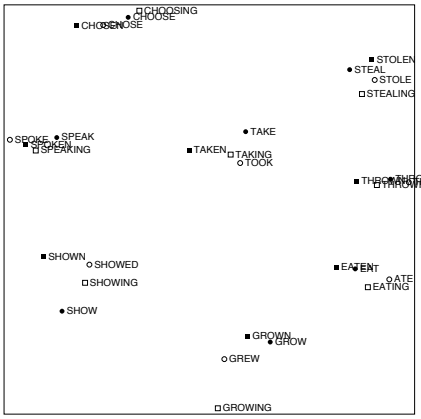
$\mathbf{U}_K$ $\quad$ $\mathbf{\Sigma}_K$ $\quad$ $\mathbf{V}_K^{\mathbf{T}}$

$\vec{t}_v^T$: v-th row of $\mathbf{U_K}$ → $\begin{bmatrix} \begin{bmatrix} \\ \vec{u}_1 \\ \\ \end{bmatrix} & \cdots & \begin{bmatrix} \\ \vec{u}_K \\ \\ \end{bmatrix} \end{bmatrix}$

The represenation of v-th word: $\vec{w}_v = \Sigma_K \vec{t}_v$

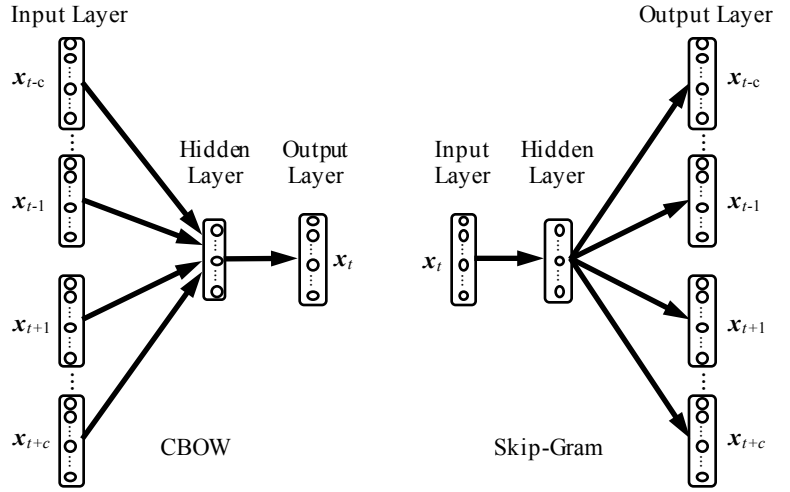
Semantics of Words by Context

"You shall know a word by the company it keeps" – J.R. Firth (1957)



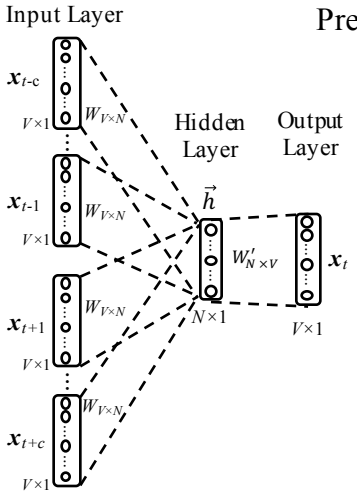
Rohde et al. 2005. An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence

Learn Word Embedding by Predicting



Word2Vec (Mikolov et al. 2013)

CBOW (Continues Bags of Words)



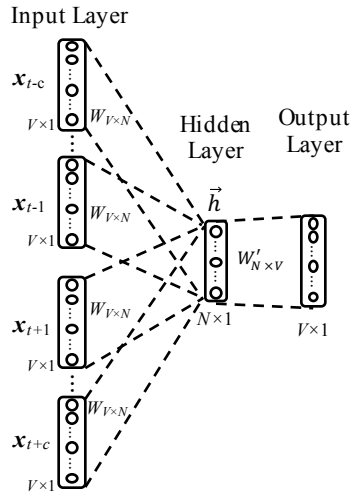
Predict the current word x_t using its context

$$\mathbb{C}_t = \{x_{t-c}, \dots, x_{t-1}, x_{t+1}, \dots, x_{t+c}\}$$

- The input layer consists of the *one-hot* encoded input context words
- Hidden layer $\vec{h} = \frac{1}{|\mathbb{C}_t|} W_{V\times N} \sum_{\vec{x}_i \in \mathbb{C}_t} \vec{x}_i$

The *k*-th row of $W_{V\times N}$ is the word embedding vector of the *k*-th word in the dictionary.

CBOW cont.



- The input to the *j*-th output neuron $v_j = \vec{h} \cdot \vec{w}'_j$

  $\vec{w}'_j$ is the *j*-th column of $W'_{N\times V}$

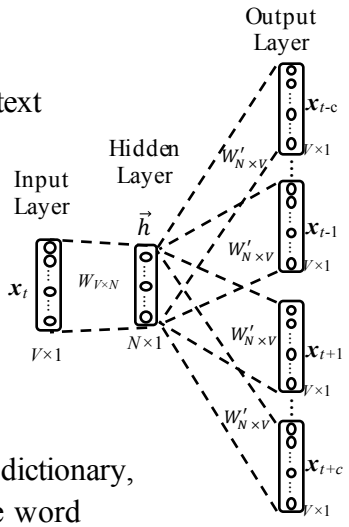
- The output of the *j*-th neuron

$$y_j = \frac{\exp(v_j)}{\sum_{s=1}^{V}\exp(v_s)}$$

$$= P(o = j\text{—th word}|\mathbb{C}_t, W_{V\times N}, W'_{N\times V})$$

Skip-Gram

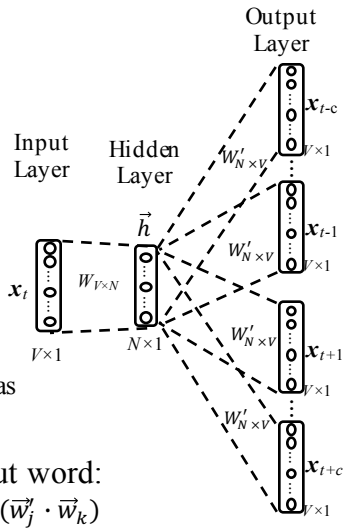
Given the current word x_t predict its context $\mathbb{C}_t = \{x_{t-c}, \dots, x_{t-1}, x_{t+1}, \dots, x_{t+c}\}$



- The input layer consists of the *one-hot* encoded input word
- The desired output vectors are *one-hot* encoded.
- Hidden layer $\vec{h} = \vec{x}_t^T W_{V\times N}$

  If the input word is the *k*-th word in the dictionary, $\vec{h} = \vec{w}_k$, i.e., the *k*-th row of $W_{V\times N}$ is the word embedding vector of the *k*-th word in the dictionary.

Skip-Gram cont.



- The input to the j-th neuron of the n-th output word

$$v_{nj} = \vec{w}'_j \cdot \vec{h} = \vec{w}'_j \cdot \vec{w}_k$$

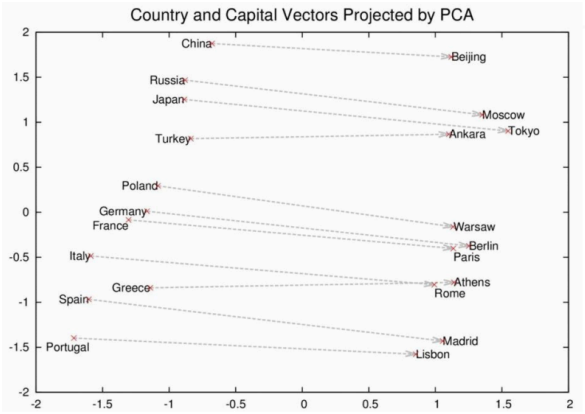
$\vec{w}'_j$ is the j-th column of $W'_{N\times V}$ or the output representation of the j-th word in the dictionary

In fact the j-th neuron of every output word has the same input v_{nj}.

- The output of the j-th neuron of an output word:

$$P(o = j\text{—th word}|k, W_{V\times N}, W'_{N\times V}) = \frac{\exp(\vec{w}'_j \cdot \vec{w}_k)}{\sum_{s=1}^{V} \exp(\vec{w}'_s \cdot \vec{w}_k)}$$

Example: Learn the Concept of Capital Cities



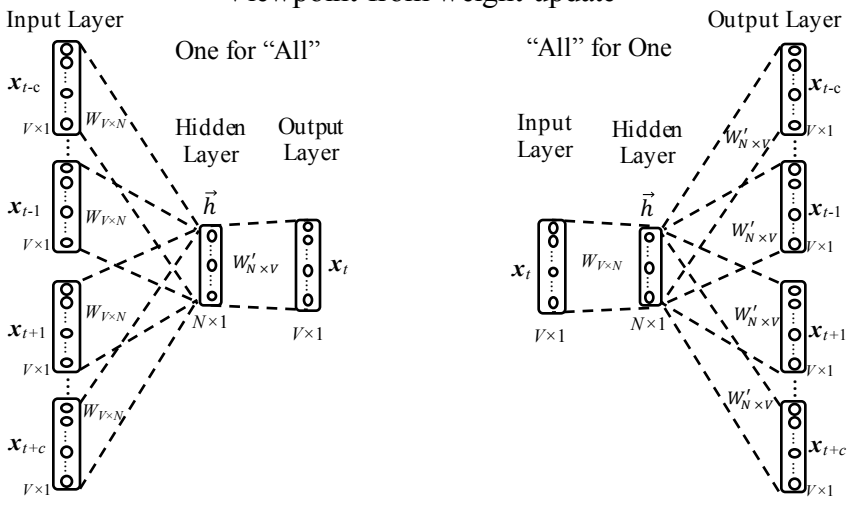
Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. During the training, no supervised information about what a capital city means is provided.

CBOW *vs* Skip-Gram

- Skip-gram produces better word vectors for infrequent words
- CBOW is faster by a factor of window size – more appropriate for larger corpora

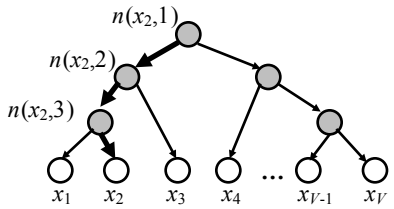
CBOW *vs* Skip-Gram

Viewpoint from weight update



Hierarchical Softmax

The hierarchical softmax model uses a tree (e.g. a Huffman binary tree) to represent all words in the dictionary as leaves in the tree.



An example path from root to the word x_2 is highlighted.

The length of the path $L(x_2) = 4$.

$n(x, j)$ indicates the j-th node on the path from root to the word x.

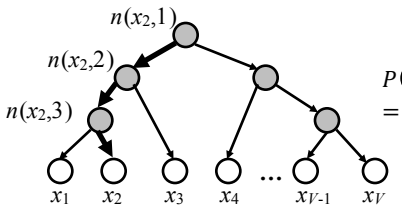
Hierarchical Softmax

There is no output vector representation for words. Each of the V-1 inner nodes has an output vector $w'_{n(x,j)}$.

The probability of a word being the output word is dened as:

$$P(x = o) = \prod_{j=1}^{L(x)-1} \sigma\left(s(x, j+1, j)\vec{w}'_{n(x,j)} \cdot \vec{h}\right)$$

$$s(x, j+1, j) = \begin{cases} 1 & n(x, j+1) \text{ is a left child of } n(x, j) \\ -1 & otherwise \end{cases}$$



$$P(x_2 = o) = \sigma\left(\vec{w}'_{n(x_2,1)} \cdot \vec{h}\right)\sigma\left(\vec{w}'_{n(x_2,2)} \cdot \vec{h}\right)\sigma\left(-\vec{w}'_{n(x_2,3)} \cdot \vec{h}\right)$$

Negative Sampling

In order to deal with there being too many nodes to compute for one iteration, we can just sample some of the nodes which we compute scores for.

For each "positive" sample in the training set, sample K "negative" samples. Optimize the following

$$E = -\log \sigma(\vec{w}_o' \cdot \vec{h}) - \sum_{i=1}^{K} \log \sigma(-\vec{w}_i' \cdot \vec{h})$$

W2V Applications

- Word segmentation
- Word cluster
- Find synonym
- Part-of-speech tagging
- Machine translation
- Question answering
- … …

Different Versions of W2V

Google code: http://word2vec.googlecode.com/svn/trunk/

400 lines C++11 version: https://github.com/jdeng/word2vec

Python version: http://radimrehurek.com/gensim/models/word2vec.html

Java: https://github.com/ansjsun/word2vec_java

Parallel java version: https://github.com/siegfang/word2vec

CUDA version: https://github.com/whatupbiatch/cuda-word2vec

References

- Xiaodong He, Jianfeng Gao. Microsoft Research DSSM for Text Processing. 2014
- Jason Weston, Antoine Bordes. Embedding Methods for NLP. Facebook AI research.2014.
- Richard Socher.Simple Word Vector Representations: word2vec. 2015
- Yoav Goldberg. Word embeddings what, how and whither
- J Turian, L Ratinov, Y Bengio. Word representation: A simple and general method for semi-supervised learning. ACL'10 Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics. 2010.
- R.Collobert, J.Weston, L.Botton, M.Karlen, K.Kavukcuoglu and P.Kuksa. Natural Language Processing from Scratch. Journal of Machine Learning Research. 2011
- M.Baroni, G.Dinu, G.Kruszewski. Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. ACL. 2014