Word Vectors - III

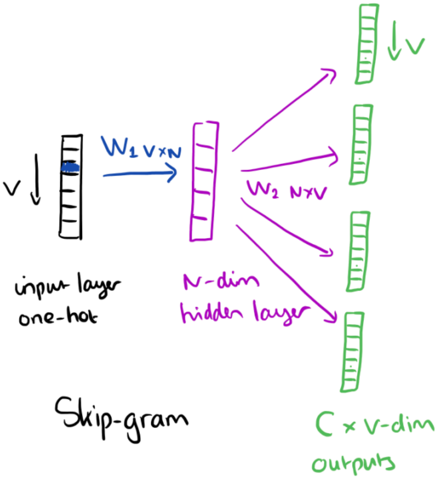
Pawan Goyal

CSE, IIT Kharagpur

February 1st, 2017

Skip-gram Model

The skip-gram model is the opposite of the CBOW model. It is constructed with the focus word as the single input vector, and the target context words are now at the output layer:



Skip-gram Model: Training

- The activation function for the hidden layer simply amounts to copying the corresponding row from the weights matrix W_1 (linear) as we saw before.
- At the output layer, we now output C multinomial distributions instead of just one.
- The training objective is to mimimize the summed prediction error across all context words in the output layer. In our example, the input would be "learning", and we hope to see ("an", "efficient", "method", "for", "high", "quality", "distributed", "vector") at the output layer.

Skip-gram Model

Details

Predict surrounding words in a window of length c of each word

Skip-gram Model

Details

Predict surrounding words in a window of length c of each word

Objective Function: Maximize the log probablility of any context word given the current center word:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} log\ p(w_{t+j} | w_t)$$

For $p(w_{t+j}|w_t)$ the simplest first formulation is

$$p(w_O|w_I) = \frac{exp({v'_{wO}}^T v_{WI})}{\sum_{w=1}^{W} exp({v'_w}^T v_{WI})}$$

Word Vectors

For $p(w_{t+j}|w_t)$ the simplest first formulation is

$$p(w_O|w_I) = \frac{exp({v'_{wO}}^T v_{WI})}{\sum_{w=1}^{W} exp({v'_w}^T v_{WI})}$$

where v and v' are "input" and "output" vector representations of w (so every word has two vectors)

Parameters θ

With $d-$dimensional words and V many words:

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ v'_{aardvark} \\ v'_a \\ \vdots \\ v'_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

Gradient Descent for Parameter Updates

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

Implementation Tricks

Batch update would take a very long time

Instead, parameters are updated after each window t.

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J_t(\theta)$$

Implementation Tricks

Batch update would take a very long time

Instead, parameters are updated after each window t.

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J_t(\theta)$$

Computing denominator in $p(w_O|w_I)$ is too computionally expensive

Negative Sampling

$$\log \sigma \left(v_{wI}^T v'_{wO} \right) + \sum_{i \sim P_n(w)} \log \sigma \left(-v_{wI}^T v'_{wi} \right)$$

Two sets of vectors

Best solution is to sum these up

$$L_{final} = L + L'$$

Two sets of vectors

Best solution is to sum these up

$$L_{final} = L + L'$$

A good tutorial to understand parameter learning:

https://arxiv.org/pdf/1411.2738.pdf

Two sets of vectors

Best solution is to sum these up

$$L_{final} = L + L'$$

A good tutorial to understand parameter learning:

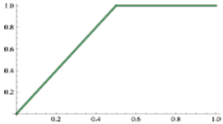
https://arxiv.org/pdf/1411.2738.pdf

An interactive Demo

https://ronxin.github.io/wevi/

Glove

$$J = \frac{1}{2}\sum_{ij} f(P_{ij})\left(w_i \cdot \tilde{w}_j - \log P_{ij}\right)^2 \qquad f \sim$$

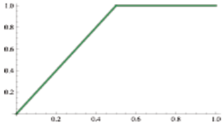


Combine the best of both worlds – count based methods as well as direct prediction methods

Glove

$$J = \frac{1}{2} \sum_{ij} f(P_{ij}) \left(w_i \cdot \tilde{w}_j - \log P_{ij} \right)^2$$

$f \sim$



Combine the best of both worlds – count based methods as well as direct prediction methods

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus, and small vectors

Code and vectors: http://nlp.stanford.edu/projects/glove/