Word Embedings - Part II

Adapted from Prof. Pawan Goyal's Slides

Somak Aditya

CSE, IIT Kharagpur

Lecture 15

CBOW

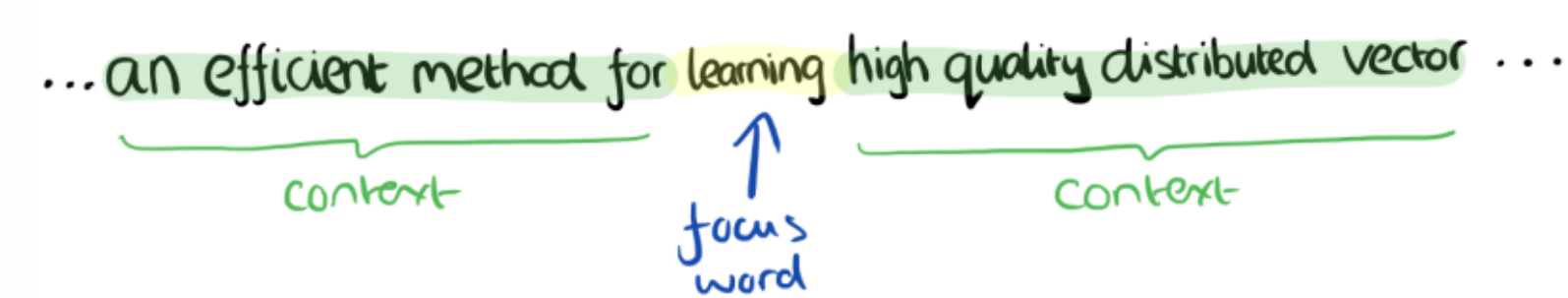
- Consider a piece of prose such as:
  "The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of syntatic and semantic word relationships."

CBOW

- Consider a piece of prose such as:
  "The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of syntatic and semantic word relationships."
- Imagine a sliding window over the text, that includes the central word currently in focus, together with the four words that precede it, and the four words that follow it:

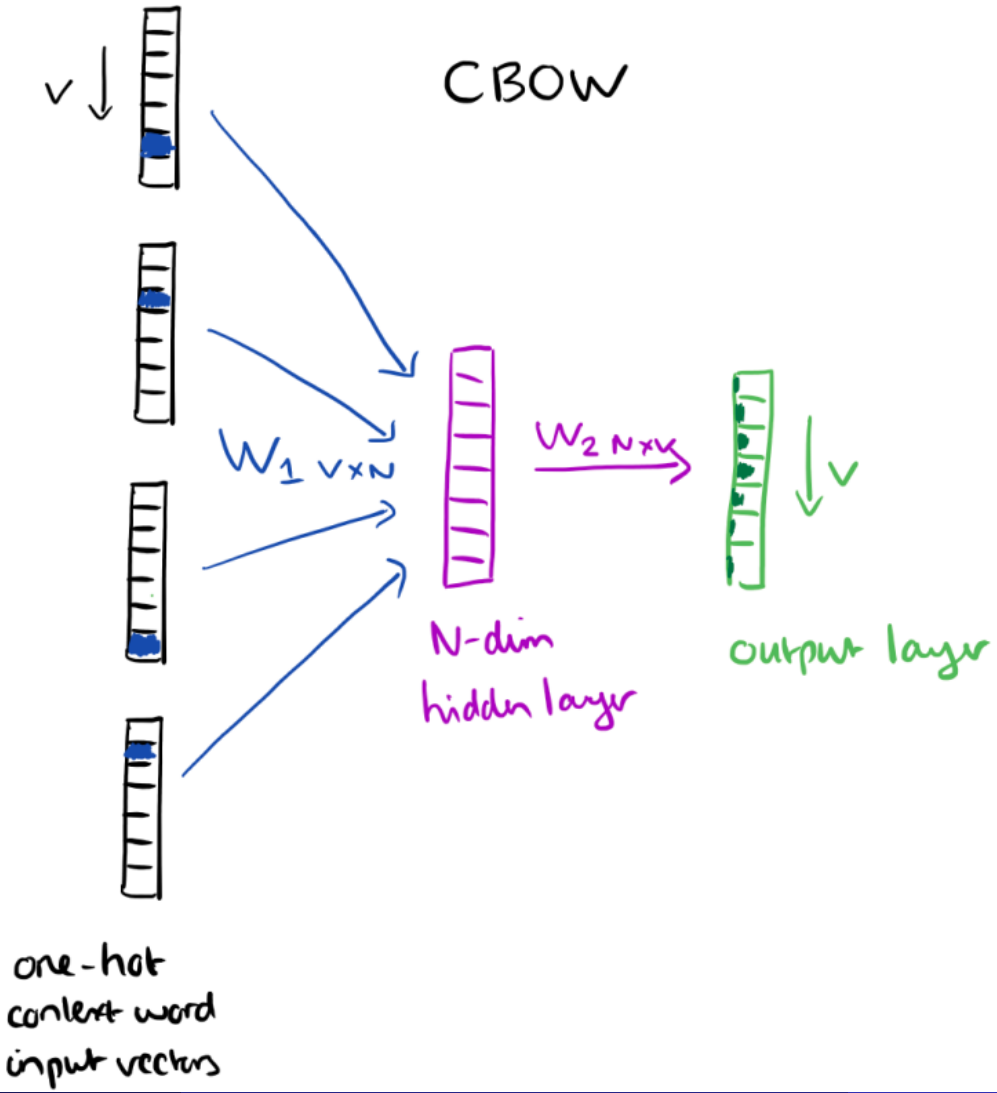
CBOW

- Consider a piece of prose such as:
  "The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of syntatic and semantic word relationships."
- Imagine a sliding window over the text, that includes the central word currently in focus, together with the four words that precede it, and the four words that follow it:



CBOW

The context words form the input layer. Each word is encoded in one-hot form. A single hidden and output layer.

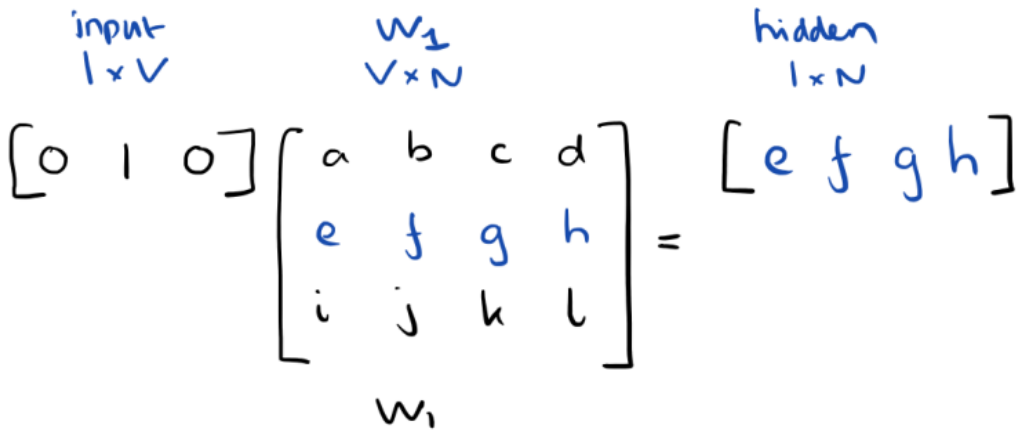


CBOW: Training Objective

- The training objective is to maximize the conditional probability of observing the actual output word (the focus word) given the input context words, with regard to the weights.
- In our example, given the input (“an”, “efficient”, “method”, “for”, “high”, “quality”, “distributed”, “vector”), we want to maximize the probability of getting “learning” as the output.

CBOW: Input to Hidden Layer

Since our input vectors are one-hot, multiplying an input vector by the weight matrix W_1 amounts to simply selecting a row from W_1.



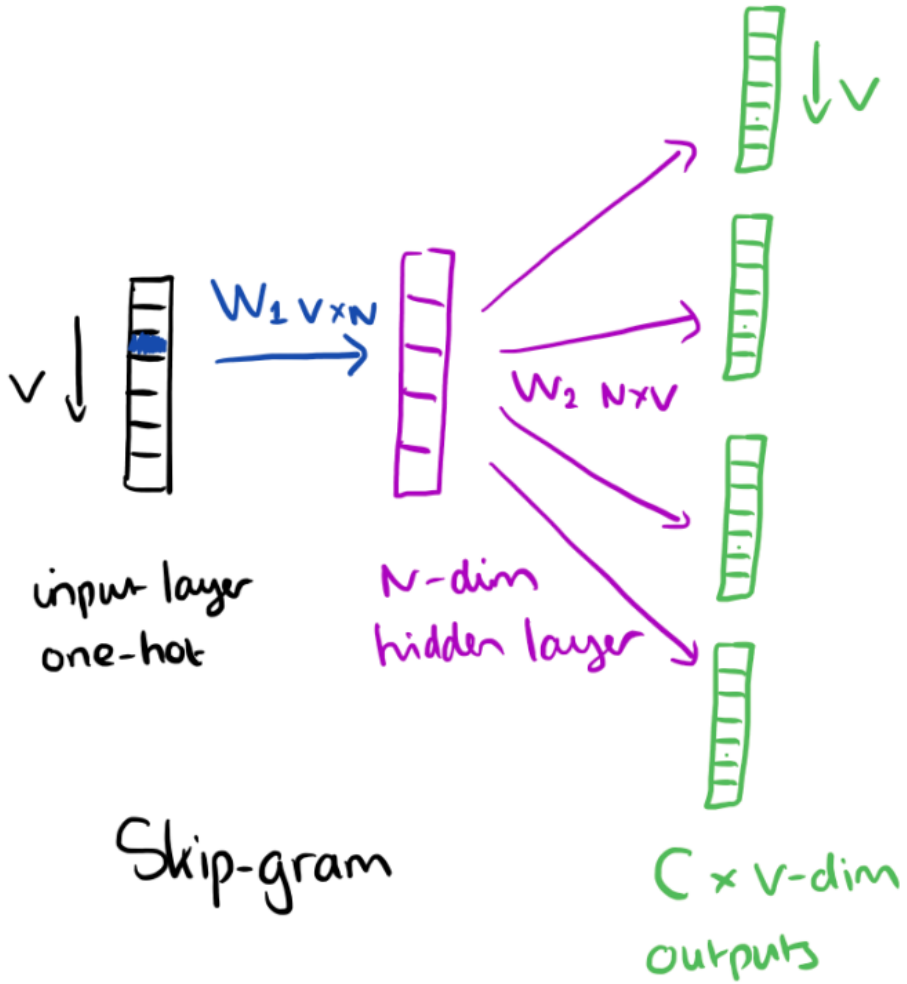
Given C input word vectors, the activation function for the hidden layer h amounts to simply summing the corresponding 'hot' rows in W_1, and dividing by C to take their average.

CBOW: Hidden to Output Layer

From the hidden layer to the output layer, the second weight matrix W_2 can be used to compute a score for each word in the vocabulary, and softmax can be used to obtain the posterior distribution of words.

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

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

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

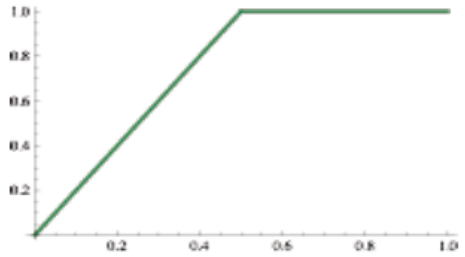
An interactive Demo

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

Glove

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

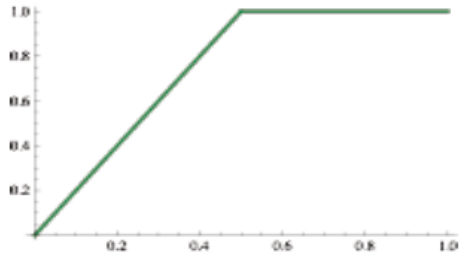
$$f \sim$$



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/