Word2Vec

Niklas Weber

LMU Munich

October 21, 2020

Outline

Introduction

- Word embeddings occure in Natural language processing (NLP)
- The idea is, to map words from a vocabulary to a continuous vector space, with lower dimension.
- Why could this be helpful?

Example

Consider a dictionary containing thousands of words. How can we represent one word to a computer program?

- A vector of zeros and a single one? $v \in [0,1]^N, |v| = 1$
- An array of characters? $\{c, a, r\}$?

Distance between words

- In NLP, especially in the context of neural networks and producing output, we often need a notion for the distance between words. This distance should reflect the semantic meaning of words.
- This is not the case in the two cases mentioned above:
  - $\forall v, v' \in \{0,1\}^N, v \neq v', |v| = |v'| = 1 : |v - v'| = 1$
  - "car" and "cat" are similar strings, with different meanings.
  - "cat" and "dog" are different strings, with similar meanings
- In a good version of a word embedding, "cat" and "dog" would probably be closer then "cat" and "car".

Models

- There exist models that "generate" such word embeddings
- Two famous ones, proposed by Mikolov et. al. (2013) (Google)
- Simple architecture, but work fast and surprisingly well.

Assumption

Words are similar, when they are used in similar contexts

- What is the definition of a context?
- Is this true?

Idea

Example

Consider a simple neural network with one hidden layer with N neurons. Input is one word from a text corpus $\mathbb{C}$. The word is represented as $x_{in} \in \{0,1\}^V, |x_{in}| = 1$, i.e. the entry of an dictionary $\mathbb{V}$. Output is a discrete probability distribution over the dictionary $x_{out} \in [0,1]^V, |x_{out}| = 1$. The aim is to predict the next occurring word in the text corpus $\mathbb{C}$, using only the previous words as input.

If such a trained neural network was good at the task of predicting the next word, and if the hidden layer is smaller then the input and output layers, then the model must be **good in compressing meaningful input information** and transforming in into accurate output information.

Bigram

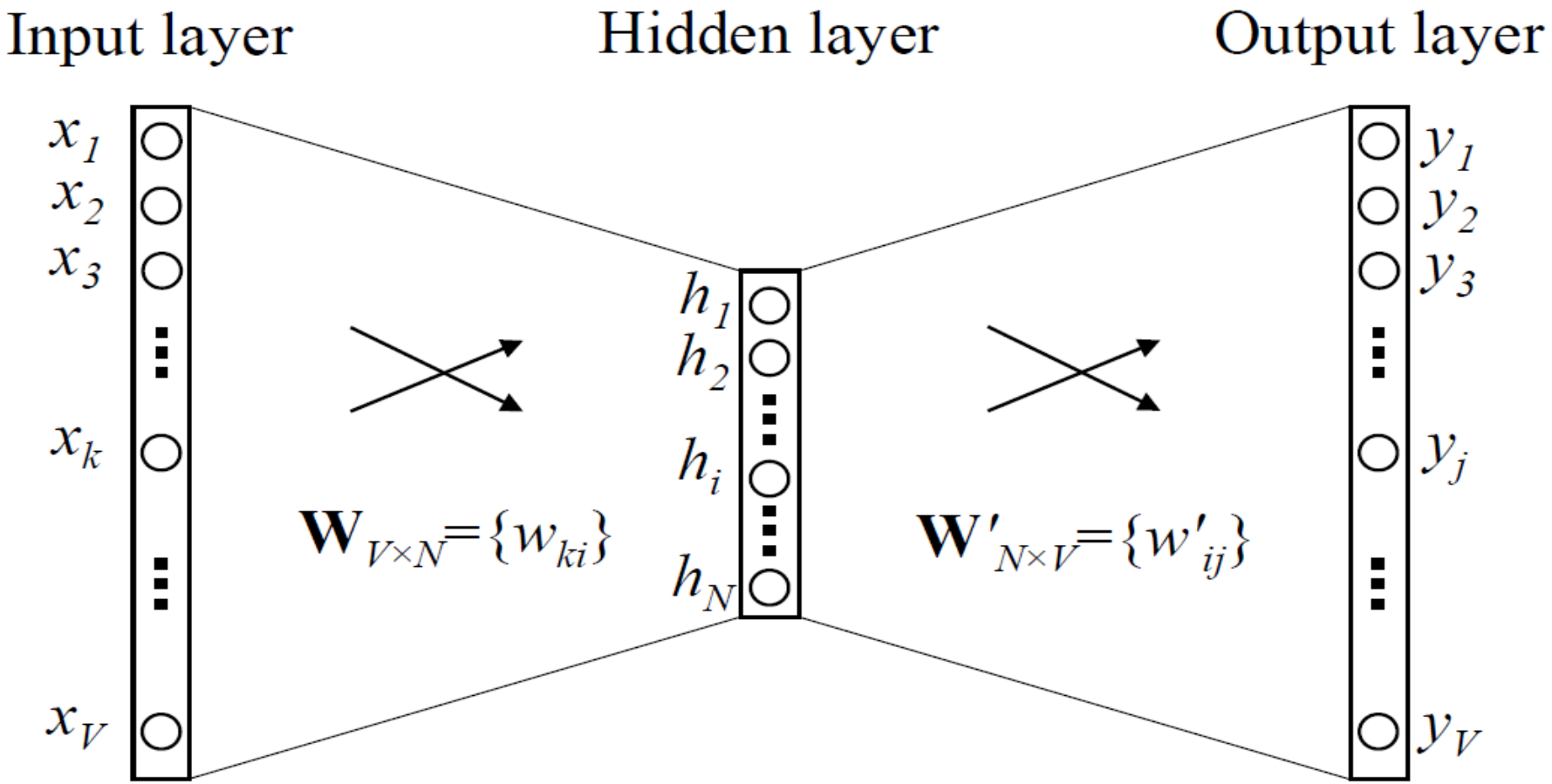

Figure: Bigram model

Bigram

- Input/vocabulary of dimension V.
- Weight matrix $W = \{w_{ki}\} \in \mathbb{R}^V \times \mathbb{R}^N$ connecting input and hidden layer linear.
- Since only one entry $k \in \{1, ..., V\}$ of x_{in} is nonzero , for h we get a row of the weight matrix W, which can be interpreted as a vector representation $v_k \in \mathbb{R}^N$ of the input word w_k:

$$h = W^T x_{in} = {W_{(k,.)}}^T = v_k^T \tag{1}$$

- A different weight matrix $W' \in \mathbb{R}^N \times \mathbb{R}^V = \{w'_{ij}\}$ maps from the hidden layer to the output layer, where we get a score u_j for every word w_j:

$$u_j = {W'_{(.,j)}}^T h = {v'_j}^T h \tag{2}$$

where v'_j is the j-th column of W', which constitutes a second vector representation of a word w_j.

Bigram

- At the output layer we transform the scores u_j into a multinomial probablity distribution y using the softmax function:

$$p(w_j = w_O | w_k) = y_j = \frac{exp(u_j)}{\sum_{i=1}^{V} exp(u_i)} \tag{3}$$

By plugging (1) and (2) into (3) this leads to:

$$y_j = \frac{exp({v'_j}^T v_k)}{\sum_{i=1}^{V} exp({v'_i}^T v_k)} \tag{4}$$

Bigram loss function

- The training objective (for one training sample) is to maximize (4) the conditional probability of observing the actual output word w_O (denote its index in the output layer as j^*) given the input word w_I with regard to the weights.
- Since we want to compare y to an idealized distribution (0,...,1,...0) a natural loss function to minimize is the cross-entropy loss L:

$$L = L(w_O, w_I) = -log(y_{j^*}) \tag{5}$$

$$= -u_{j^*} + log\left(\sum_{i=1}^{V} exp(u_i)\right) \tag{6}$$

Bigram derivative with respect to W'

Hidden $\rightarrow$ output weights:

$$\begin{aligned}\frac{\partial L}{\partial u_j} &= \frac{\partial log(y_{j^*})}{\partial u_j} \\ &= -1_{j=j^*} + \frac{exp(u_j)}{\sum_{i=1}^{V} exp(u_i)} \\ &= y_j - t_j := e_j \end{aligned} \tag{7}$$

And using the chain rule:

$$\frac{\partial L}{\partial w'_{ij}} = \frac{\partial L}{\partial u_j}\frac{\partial u_j}{\partial w'_{ij}} = e_j h_i \tag{8}$$

Bigram update rules

Stochastic gradient descent with learning rate $\eta > 0$ yields the following update rule:

$$w_{ij}'^{new} = w_{ij}'^{old} - \eta e_j h_i \tag{9}$$

or in terms of the word vectors v_j':

$$v_j'^{new} = v_j'^{old} - \eta e_j \mathbf{h} \tag{10}$$

Intuition

Remember that h $= v_k^T$ and $e_j = y_j - t_j$

Bigram derivative with respect to W

Input $\rightarrow$ hidden weights:

$$\frac{\partial L}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial L}{\partial u_j} \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j w'_{ij} := LH_i \tag{11}$$

Recall that:

$$h_i = \sum_{k=1}^{V} x_k w_{ki} \tag{12}$$

Finally we have:

$$\frac{\partial L}{\partial w_{ki}} = \frac{\partial L}{\partial h_i} \frac{\partial h_i}{\partial w_{ki}} := LH_i x_k \tag{13}$$

Bigram update rule W

Since only one component x_{j^*} of $x_{in} = (x_1, ..., x_V)$ is 1 and not 0, only one row of W will be updated following the rule:

$$W_{(k,.)}{}^{new} = v_k^{new\,T} = v_k^{old\,T} - \eta LH^T \tag{14}$$

Intuition

$$LH = \sum_{j=1}^{V} e_j v_j' \tag{15}$$

Continuous bag of words (CBOW) model

In the same setting as the Bigram model, now instead of using only one word, the input will be the averaged input of multiple words. From here on it makes sense to speak of a context.
Almost everything stays the same, when plugging in the new h into the previous formulas:

$$\begin{aligned} h &= \frac{1}{C} W^T (x_1 + ... + x_C) \\ &= \frac{1}{C} (v_1^T + ... + v_C^T) \end{aligned} \tag{16}$$

Intuition

The Bigram model is usually called a one-word-context CBOW model

CBOW

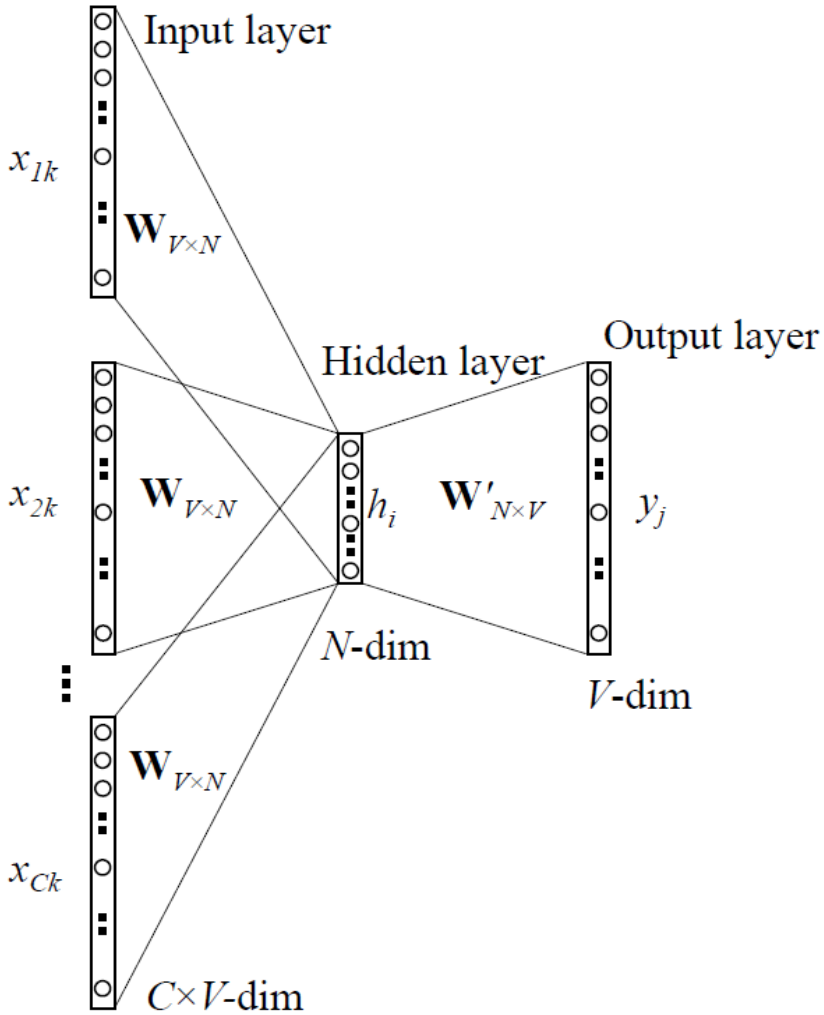

Figure: CBOW model

Update rules CBOW

The update of W' stays the same (with new **h**):

$$v_j'^{new} = v_j'^{old} - \eta \cdot e_j \cdot \mathbf{h} \tag{17}$$

The update of W changes slightly. Since the input context consists of multiple words, all corresponding rows of W are updated:

$$v_{I,c}^{new} = v_{I,c}^{old} - \frac{1}{C} \cdot \eta \cdot LH \tag{18}$$

for every c=1,...,C.

Skip-Gram model

The Skip-gram model is the opposite of the CBOW model. Input is merely one word w_I. Output are C words $w_{O,1}, ..., w_{O,C}$ and the goal is to predict the context, in which the word occurred. On the Input side everything equals the Bigram model, e.g.:

$$h = W^T_{(k,.)} = v^T_{w_I} \tag{19}$$

The output layer generates C multinomial distributions. Each output is computed using:

$$p(w_{c,j} = w_{O,c}|w_I) = y_{c,j} = \frac{exp(u_{c,j})}{\sum_{i=1}^{V} exp(u_{c,i})} \tag{20}$$

Skip-Gram model

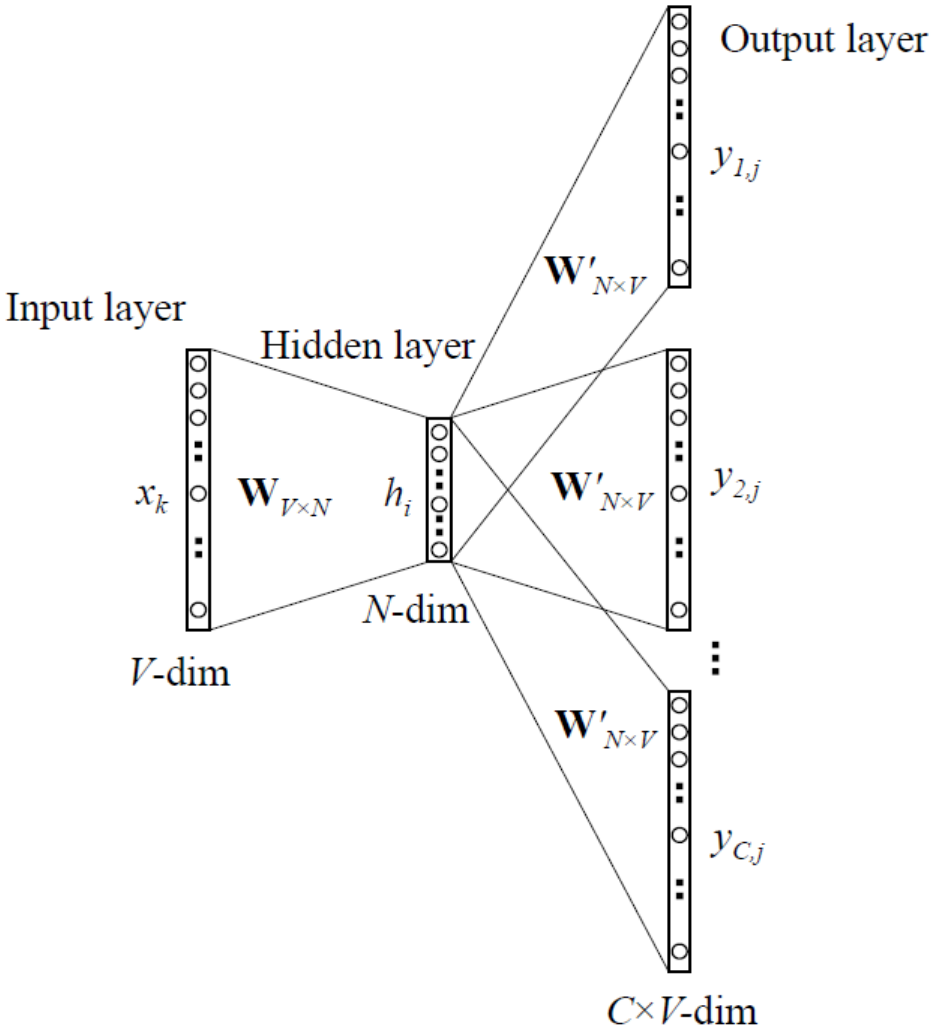

Figure: Skip-Gram model

Skip-Gram loss function

Since all C distributions use the same Weights W', input h and activation function, they are identical. The score $u_{c,j}$ for word w_j in every panel c=1,...,C is:

$$u_{c,j} = u_j = v_j'^T \cdot h \tag{21}$$

The loss function is adjusted slightly:

$$L = -log(p(w_{O,1}, ..., w_{O,C}|w_I)) \tag{22}$$

$$= -log\left(\prod_{c=1}^{C} \frac{exp(u_{c,j_c^*})}{\sum_{i=1}^{V} exp(u_{c,i})}\right) \tag{23}$$

$$= -\sum_{c=1}^{C} u_{j_c^*} + C \cdot log\left(\sum_{i=1}^{V} exp(u_i)\right) \tag{24}$$

where j_c^* is the index of the actual c-th context word.

Skip-Gram derivative with respect to W'

We take the derivative of L with respect to the score of every unit (j=1,...,V) of every panel (c=1,...,C):

$$\frac{\partial L}{\partial u_{c,j}} = y_{c,j} - t_{c,j} := e_{c,j} \tag{25}$$

For notation simplicity write $EI = (EI_1, ..., EI_V)$ the sum of errors for every word across all c=1,...,C panels:

$$EI_j = \sum_{c=1}^{C} e_{c,j} \tag{26}$$

and then the derivative with regard to W':

$$\frac{\partial L}{\partial w'_{ij}} = \sum_{c=1}^{C} \frac{\partial L}{\partial u_{c,j}} \cdot \frac{\partial u_{c,j}}{\partial w'_{ij}} = EI_j \cdot h_i \tag{27}$$

Skip-Gram update rule for W'

We obtain the following update rule for the matrix W' connecting hidden and output layer:

$$w_{ij}'^{new} = w_{ij}'^{old} - \eta \cdot EI_j \cdot h_i \tag{28}$$

Or in vector notation, recalling that a column in W' is an output representation vector v' for a word:

$$v_j'^{new} = v_j'^{old} - \eta \cdot EI_j \cdot h \tag{29}$$

Derivative with respect to W

For the derivative with respect to W, we can go back to the Bigram model. The calculation is identical after replacing e_j with EI_j: Bigram model

Computational costs of updating

In the models discussed, there are two vector representations per word. v_j and v'_j in W and W' respectively. Recall the update rules:

Update of output vectors

$$v_j'^{new} = v_j'^{old} - \eta \cdot EI_j \cdot h \tag{30}$$

Every word: compute score/prediction/error

Update of input vectors

$$v_k^{new} = v_k^{old} - \eta LH \tag{31}$$

$$LH = \sum_{j=1}^{V} e_j v'_j \tag{32}$$

Hierarchical Softmax

Idea:

Instead of representing every word by an output vector, represent them as leaves in a binary tree.

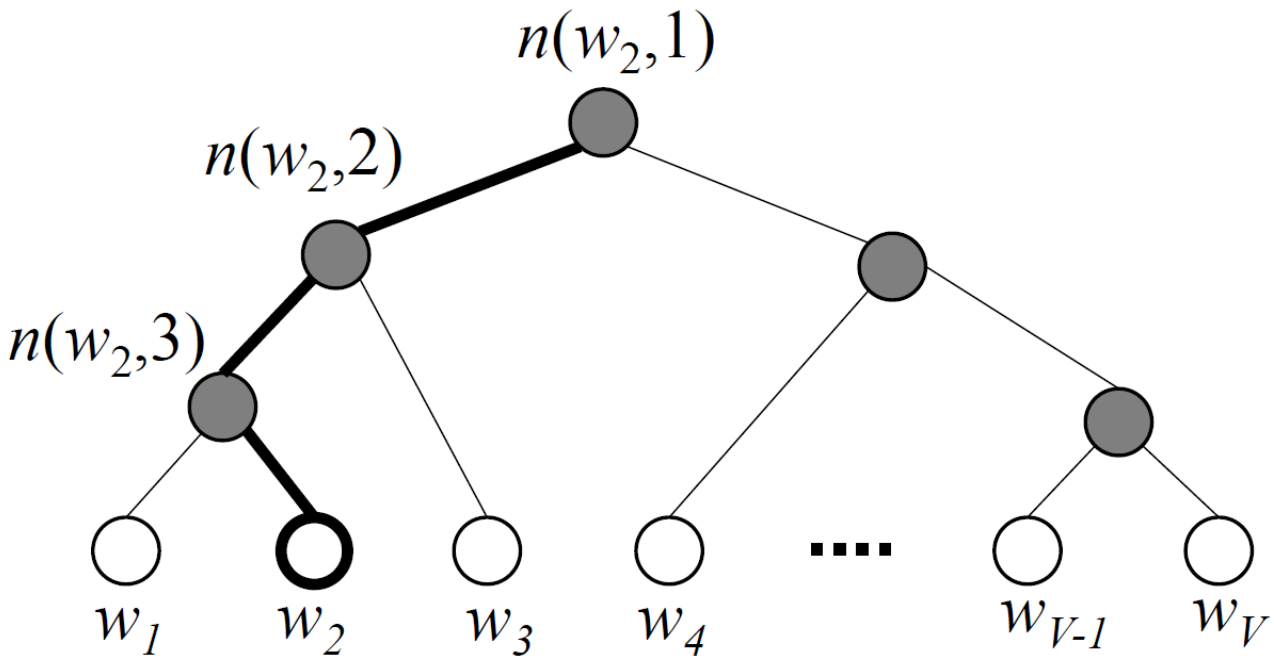

Figure: Example binary tree for hierarchical softmax

Hierarchical Softmax

- V words must be leave units
- V-1 inner units
- No output vector representation for every word
- Instead every inner unit has an output vector $v'_{n(w,j)}$

Before proceeding to the probabilities in this tree, recall the logistic function:

$$\sigma(x) := \frac{1}{1+e^{-x}} \in (0,1) \tag{33}$$

with the handy properties:

$$\sigma(-x) = 1 - \sigma(x) \tag{34}$$

$$\frac{\partial}{\partial x}\sigma(x) = \sigma(x)\sigma(-x) \tag{35}$$

Hierarchical Softmax - probabilities

For navigating through the tree we assign probabilities. The probability to go left at an inner unit n:

$$p(n, left) = \sigma\left({v'_n}^T \cdot h\right) \tag{36}$$

where h is the output of the hidden layer (e.g. v_{w_I} in Bigram or Skip-Gram) and ${v'_n}^T$ is the vector representation of the inner unit n. The corresponding probability to go right is:

$$p(n, right) = 1 - \sigma\left({v'_n}^T \cdot h\right) = \sigma\left(-{v'_n}^T \cdot h\right) \tag{37}$$

Hierarchical Softmax - probabilities

The probability for choosing a word w being the output word w_O is simply defined as the probability of a random walk ending up in leaf w:

$$p(w = w_O|w_I) = \prod_{j=1}^{M(w)-1} \sigma\left([\![n(w, j+1) == ch(n(w, j))]\!] \cdot {v'_{n(w,j)}}^T \mathbf{h}\right) \tag{38}$$

where:

- ch(n) is the left child of unit n
- $v_{n(w,j)}$ is the vector representation of inner unit n(w,j)
- h output of hidden layer (e.g. v_{w_I} in Bigram/Skip-Gram)
- $[\![x]\!] = \begin{cases} 1 & \text{if x is true,} \\ -1 & \text{otherwise.} \end{cases}$
- M(w) is the length of the path of word w

Hierarchical Softmax Training

The loss function L is defined as:

$$L = -log\left(p(w = w_O|w_I)\right) = -\sum_{j=1}^{M(w)-1} log\left(\sigma([\![\cdot]\!]{v'_j}^T h)\right) \tag{39}$$

We take the derivative of L with respect to the score provided by ${v'_j}^T h$:

$$\frac{\partial L}{\partial {v'_j}^T h} = -\sigma(-[\![\cdot]\!]{v'_j}^T h)[\![\cdot]\!] \tag{40}$$

$$= \left(\sigma([\![\cdot]\!]{v'_j}^T h) - 1\right)[\![\cdot]\!] \tag{41}$$

$$= \sigma({v'_j}^T h) - t_j \tag{42}$$

where $t_j = 1$ if $[\![\cdot]\!] = 1$ and $t_j = 0$, otherwise.

Hierarchical Softmax Training

The derivative with respect to the vector v'_j, which is the representation of the inner unit n(w,j) leads to:

$$\frac{\partial L}{\partial v'_j} = \frac{\partial L}{\partial {v'_j}^T h} \frac{\partial {v'_j}^T h}{\partial v'_j} = (\sigma({v'_j}^T h) - t_j) \cdot h \tag{43}$$

As a result the update equation for the vectors representing the unit vectors is:

$${v'_j}^{new} = {v'_j}^{old} - \eta \cdot \left(\sigma({v'_j}^T h) - t_j\right) \cdot h \tag{44}$$

Improvement

Only the vectors corresponding to the inner units n(w,1),...,n(w,M(w)-1) must be updated.

Hierarchical softmax training

Of course we also want to update the input vector v_I. First consider:

$$\frac{\partial L}{\partial h} = \sum_{j=1}^{M(w)-1} \frac{\partial L}{\partial {v'_j}^T h} \frac{\partial {v'_j}^T h}{\partial h} \tag{45}$$

$$= \sum_{j=1}^{M(w)-1} \left(\sigma\left({v'_j}^T h\right) - t_j\right) \cdot v'_j \tag{46}$$

$$:= LH \tag{47}$$

This LH can be plugged into the update of the vector representation of the input word, since the derivations are identical:

$$v_I^{new} = v_I^{old} - \eta \cdot LH \tag{48}$$

Negative Sampling

Another possibility to improve computing time is to reduce the number of updated vectors. The idea of negative sampling is, that instead of updating all output vectors, we **only update a sample** of them.

- Keep actual output word w_O
- Take negative samples $\{w_j\}_j = \mathbb{W}_{neg}$ from the text corpus with some distribution $\mathbb{P}$.
- Minimize new loss function

A loss function capable of producing high-quality word embeddings according to [2] is:

$$L = -log\left(\sigma\left({v'_{w_O}}^T h\right)\right) - \sum_{w_j \in \mathbb{W}_{neg}} log\left(\sigma\left(-{v'_{w_j}}^T h\right)\right) \tag{49}$$

Negative sampling training

Using the above loss function the derivatives of L with respect to v' will only be nonzero for v'_{w_O} and the v'_{w_j}.
Instead of updating V or $log_2 V$ parameters, only e.g. 10 vectors are updated.

Update of output vectors

$$v_j'^{new} = v_j'^{old} - \eta \cdot \left(\sigma\left(v_j'^T h\right) - t_j\right) \cdot h \tag{50}$$

Update of input vectors

$$v_k^{new} = v_k^{old} - \eta LH \tag{51}$$

$$LH = \sum_{w_j \in \{w_O\} \cup \mathbb{W}_{neg}} \left(\sigma\left(v_j'^T h\right) - t_j\right) \cdot v_j' \tag{52}$$

References

The content of this slides was created relying on at least the following sources:

- Rong, X. (2016).word2vec Parameter Learning Explained, arXiv:1411.2738v4
- Goldberg, Y. and Levy, O. (2014). word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. arXiv:1402.3722 [cs, stat]. arXiv: 1402.3722
- Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.