SD TSIA 214 – Word embeddings

Chloé Clavel,
chloe.clavel@telecom-paristech.fr,

Telecom ParisTech, France

Outline of the course

Objectives of the Lecture

At the end of the lecture...

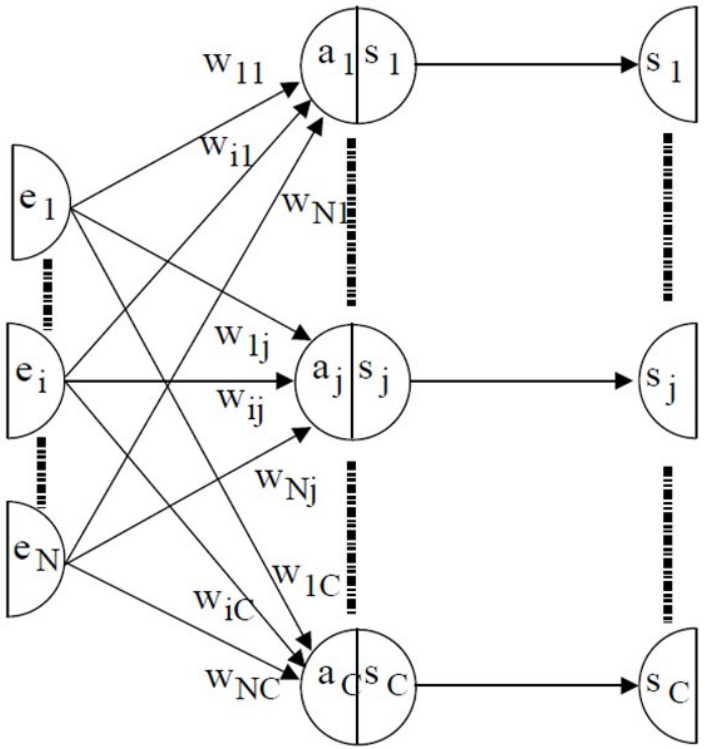
you will master :

- the underlying principles of word embeddings
- the algorithm of the most popular word embeddings tool word2vec

you will be able :

- to go further knowing the existing variants and tools

Prerequisites : Neural networks with one layer

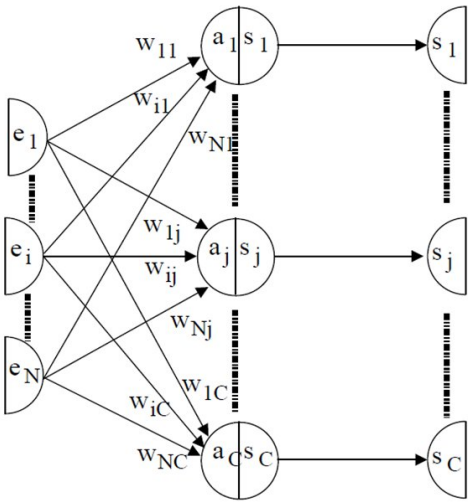


Prerequisites : Neural activation function

Activation function f

f calculates the output of the neural unit based on its inputs

$s_j = f(a_j)$, where $a_j = \sum_{i=1}^{N} w_{i,j} e_i \iff A = W^t * E$

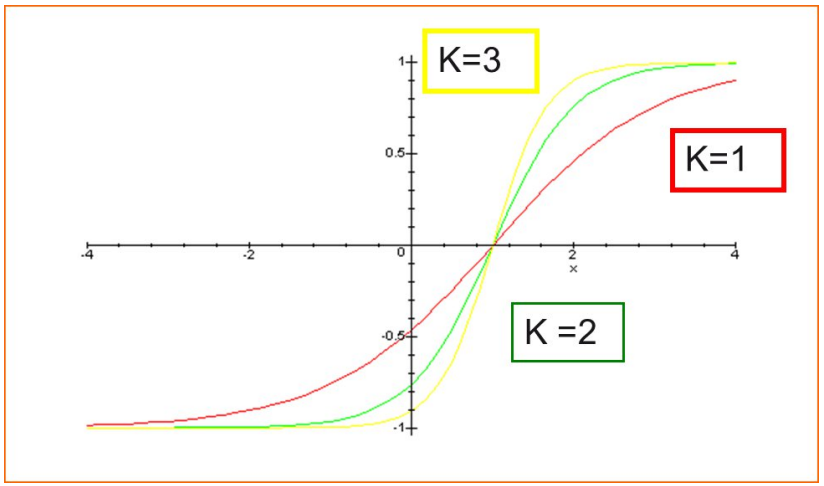


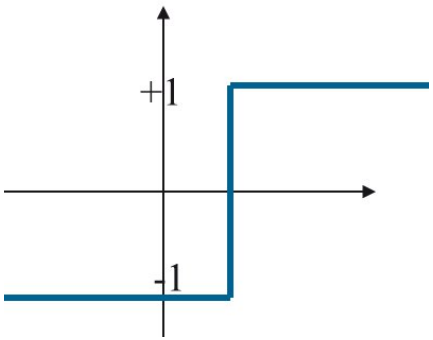
Prerequisites : Neural activation function

Activation function – choice

sigmoid functions

- $f_{K,\sigma}(x) = \frac{e^{K(x-\sigma)}-1}{e^{K(x-\sigma)}+1}$
- $f(x) \in]-1;1[$
- K : slope on $y = 0$
- differentiable function
- $K \to \infty$: threshold function

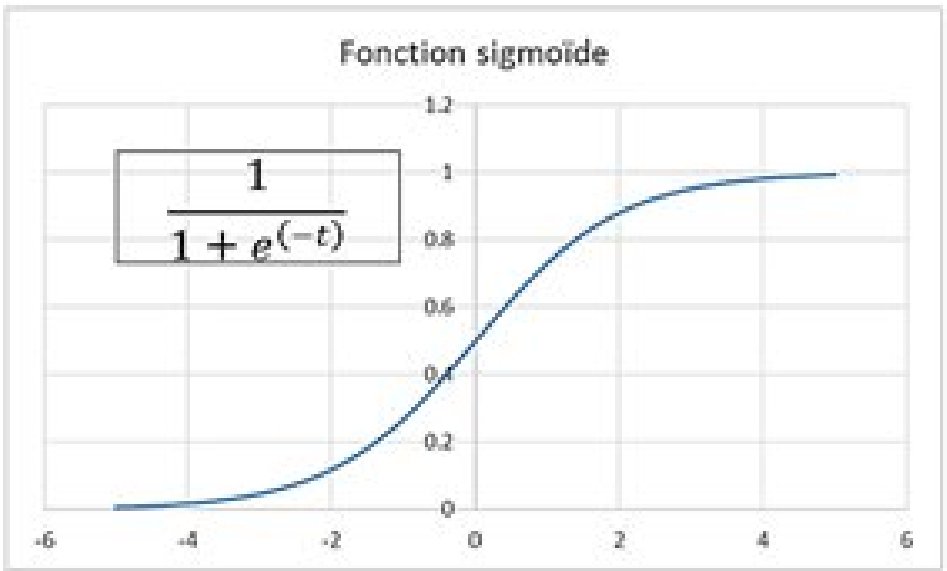




Prerequisites : Neural activation function

For a binary classification problem, the outputs $s_j \in \{0, 1\}$, we use the logistic function (special case of sigmoid function) also called **softmax** for vectors

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



Prerequisites : Neural activation function

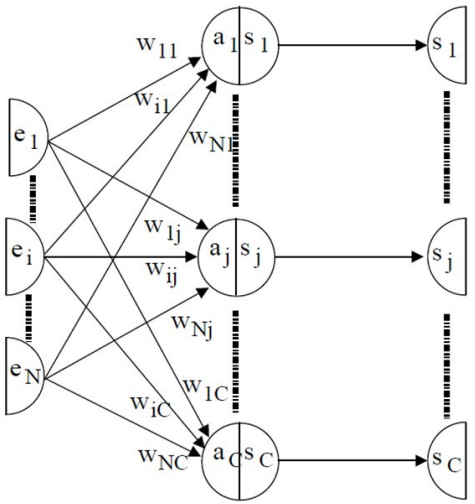
Softmax function or normalized exponential function is a generalization of the logistic function for a K-dimensional vector **z**
The function is given by

$$softmax : \mathbb{R}^K \rightarrow \left\{ z \in \mathbb{R}^K | z_i > 0, \sum_{i=1}^{K} z_i = 1 \right\}$$

$$softmax(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1, \ldots, K$$

Prerequisites : Optimization principle for the training

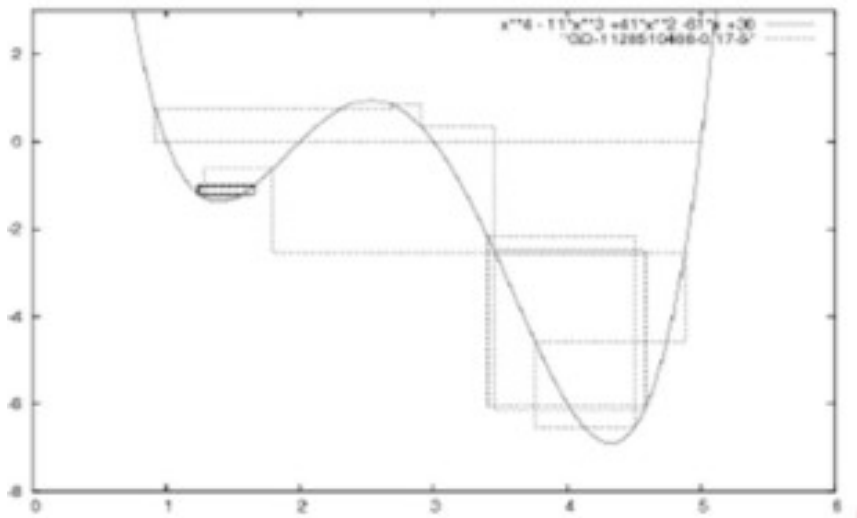
- Find the w_{ij} which minimize the learning error J on the training database
- Use Stochastic gradient descent for the minimization



Prerequisites : Optimization principle for the training

Use Stochastic gradient descent for the minimization of the learning error $J(W)$

- Iterative process $i \rightarrow i+1$: update W according to the gradient
- $w^{(i+1)} = w_{k,j}^{(i)} + \Delta w_{k,j}$ where $\Delta w_{k,j} = -\eta \frac{\partial J}{\partial w_{k,j}}$



General principle of word embeddings

- Feature learning : word $\rightarrow$ vector
- a vector ...
  - ... which embeds the semantic link between words

(Mikolov et al., NAACL HLT, 2013)

  - ... computed from the word context :

$$\overbrace{\text{My favorite fruit is an}}^{context}\ \textit{APPLE}.$$

General principle of word embeddings

Underlying principle : distributional semantics

- **Hypothesis :** 2 words occurring in a same context have a semantic proximity
- Example : $\overbrace{\textbf{My favorite fruit is an}}^{context}$ *APPLE*.

  $\overbrace{\textbf{My favorite fruit is an}}^{context}$ *ORANGE*.

  $\Rightarrow$ High semantic similarity between ORANGE and APPLE (both are fruits) because they tend to appear in a similar context in texts. $\Rightarrow$ $vect_{ORANGE} \sim vect_{APPLE}$

General principle of word embeddings

Inspiration

A simplification of Feedforward Neural net language models explained in *Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. ICLR Workshop, 2013*

Neural Probabilistic language models

they are trained using the maximum likelihood (ML) principle : maximize the probability of the next word w_t given the previous word h (for history) in terms of a softmax function.
See : *Y. Bengio, R. Ducharme, P. Vincent. A neural probabilistic language model. Journal of Machine Learning Research, 3 :1137-1155, 2003.*

Overview of word embeddings computation

What do you need in order to train word vectors ? a big dataset of texts

What will you obtain ? if V is the vocabulary of the dataset, for each word $w_i \in V$, a dense vector v is computed

Example : *linguistics* $=$

$$\begin{bmatrix} 15 \\ 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{bmatrix}$$

How ? using neural networks

Overview of word embeddings computation

These representations are very good at encoding dimensions of similarity !
Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

- syntactically : $x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$
- semantically : $x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$
  $x_{king} - x_{man} \approx x_{queen} - x_{woman}$

Definition - Context of a word

Context of a word :

- the context of a word is the set of the C surrounding words.
- Example : "In March and $\overbrace{\text{April 1959}}^{\text{left context}}$, **Davis** $\overbrace{\text{recorded what}}^{\text{right context}}$ many critics consider his greatest album, Kind of Blue"
- The $C = 2$ context of word **Davis** is $\{\textit{April}, 1959, \textit{recorded}, \textit{what}\}$
- Remark : the order the words in the context has no influence in word embeddings computation

Target :

DAVIS the word from which we want to compute the vector

Two types of NN architecture :

CBOW (Continuous Bag Of Words)

- Input of the network : context
- Output of the network : target
- learn how to predict a word according to its context
- learn W et W' so that if $INPUT = \{$**April, 1959, recorded, what**$\}$, $OUTPUT =$ **Davis**

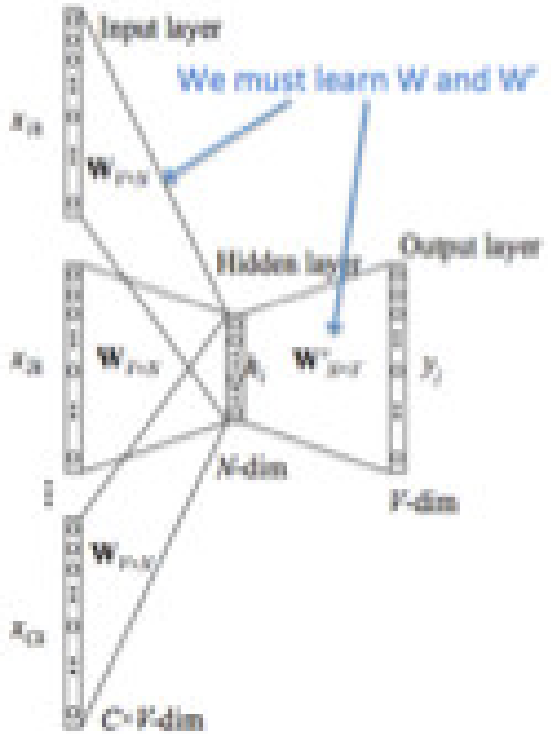


Figure 1: This image demonstrates how CBOW works and how we must learn the transfer matrices

Two types of NN architecture :

Skip-gram

- Input of the network : target
- Output of the network : context
- Learn how to predict a context given a word
- Learn W et W' so that if $INPUT =$ **Davis**, $OUTPUT =$ {**April, 1959, recorded, what**}

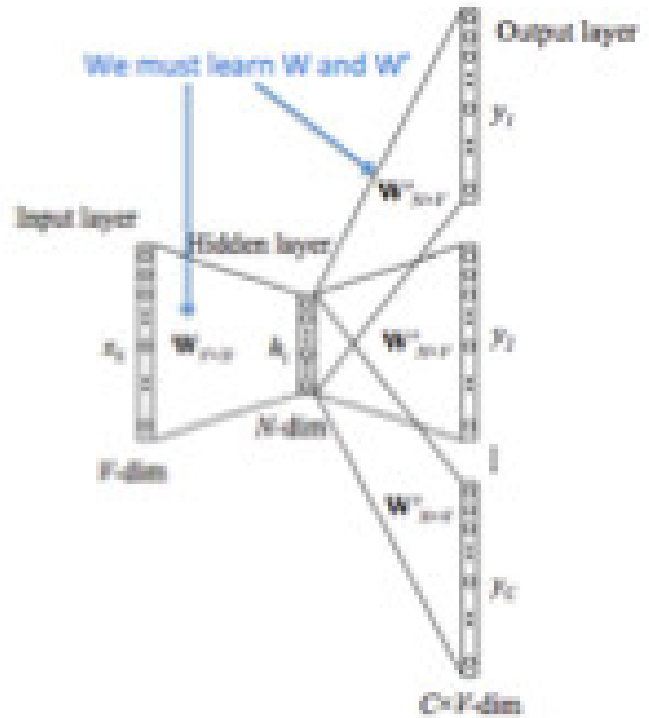


Figure 2: This image demonstrates how Skip-Gram works and how we must learn the transfer matrices

Inputs and outputs of NN architecture

- Network's inputs and outputs are words (target and context)
- words are represented by one-hot vectors.
- a **One-hot** vector represents every word as an $\mathbb{R}^{|V|}$ vector
- $|V|$ is the size of the vocabulary

- all words are indexed in the sorted English language (alphabetic order).
- the vector contains all 0 and one 1 at the index of that word

$$at \rightarrow \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

CBOW vs. Skip-gram

- Algorithmically : the two models are similar
- Statistically : the inversion (target-context) has an effect :
  - CBOW smoothes over a lot of the distributional information (by treating an entire context as one observation) $\Rightarrow$ better for smaller dataset
  - Skip-gram treats each target-context pair as a new observation $\Rightarrow$ better for larger datasets.

Reminder

What do we need to start learning the word vectors ?
What are the inputs/outpus of the NN in the CBOW model ?
How are we representing the words in these inputs/outputs ?

Training the NN for CBOW

CBOW Model

- Input of the network : one hot vectors of the *context* words : $(x^{c-m}, ..., x^{c-1}, x^{c+1}, ..., x^{c+m})$ (size m)
- Wanted outputs of the network : y one hot vector of the target
- The model learns W et W' so that the output of the NN, $\hat{y}$, matches y

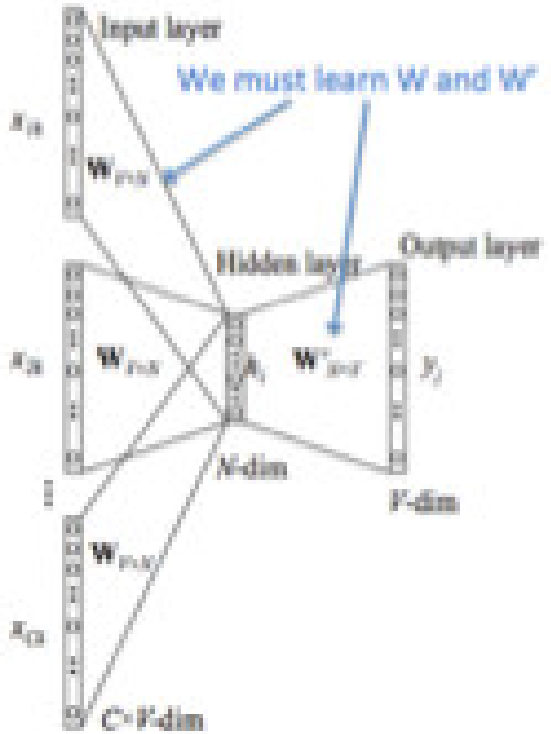


Figure 1: This image demonstrates how CBOW works and how we must learn the transfer matrices

Training the NN for CBOW

CBOW Model

- The model learns W et W' so that the output of the NN, $\hat{y}$, matches y
- Choice of the loss measure : cross-entropy
  $H(\hat{y}, y) = -\sum_{j=1}^{\|V\|} y_j log(\hat{y}_j)$
- As $y = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \dots \\ 0 \end{bmatrix}$, $H(\hat{y}, y) = -y_c log(\hat{y}_c) = -log(\hat{y}_c)$

Training the NN for CBOW

PRACTICE

- Compute the loss when the prediction is correct
- Compute the loss when the the prediction is bad *e.g.* $\hat{y}_c = 0.01$

Training the NN for CBOW

Remark

- If the prediction is correct $\hat{y}_c = 1$ and $H(\hat{y}, y) = -log(\hat{y}_c) = 0$
- If the prediction is bad *e.g.* $\hat{y}_c = 0.01$,
  $H(\hat{y}, y) = -log(\hat{y}_c) = -log(0.01) = 4.605$

Training the NN for CBOW

CBOW Model

- The model learns W et W' so that the output of the NN, $\hat{y}$, matches y
- $W \in \mathbb{R}^{Nx|V|}$ and $W' \in \mathbb{R}^{|V|xN}$, where N is the size of the embedding space
- The j^{th} column of W is the embedded vector for $word_j$, when it is an input of the model
- The i^{th} row of W' is the embedded vector for $word_i$, when it is an output of the model
- Note that we do in fact learn two vectors for every word

Training the NN for CBOW

CBOW Model : How to express $\hat{y}$?

$\hat{y}$ is computed according to

- the input of the network : one hot vectors of the *context* words : $(x^{c-m}, ..., x^{c-1}, x^{c+1}, ..., x^{c+m})$ (size m)
- $W \in \mathbb{R}^{N x |V|}$ and $W' \in \mathbb{R}^{|V| x N}$, where N is the size of the embedding space

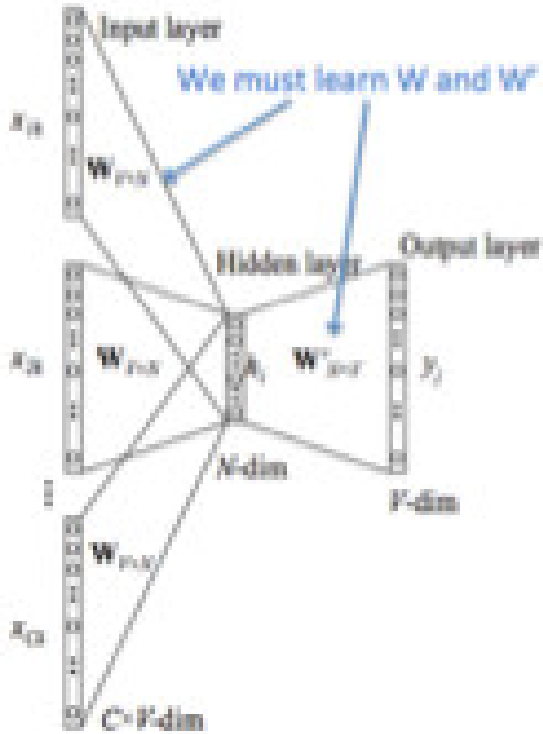


Figure 1: This image demonstrates how CBOW works and how we must learn the transfer matrices

Training the NN for CBOW

CBOW Model : How to express $\hat{y}$?

Given $(x^{c-m}, ..., x^{c-1}, x^{c+1}, ..., x^{c+m})$, W and W',

1) Express the word embeddings vectors of the inputs :

$(x^{c-m}, ..., x^{c-1}, x^{c+1}, ..., x^{c+m}) \rightarrow$

$v_{c-m} = W * x^{c-m}, ..., v_{c-1} = W * x^{c-1}, v_{c+1} = W * x^{c+1}, ..., v_{c+m} = W * x^{c+m}$

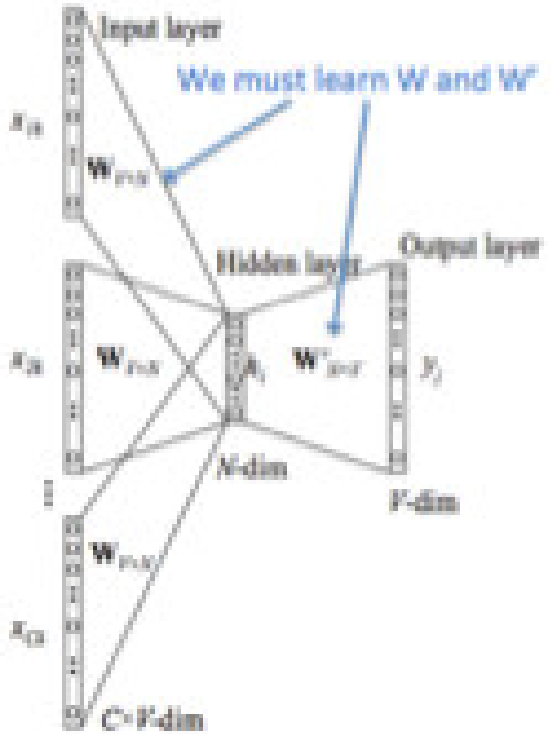


Figure 1: This image demonstrates how CBOW works and how we must learn the transfer matrices

Training the NN for CBOW

CBOW Model : How to express $\hat{y}$?

2) Average these vectors to get the vector $\hat{v} \in \mathbb{R}^N$ $\hat{v} = mean(v^{c-m}, ..., v^{c-1}, v^{c+1}, ..., v^{c+m})$

3) $\hat{y} = softmax(W' * \hat{v})$, $\hat{y} \in \mathbb{R}^{|V|} \Rightarrow$

$\hat{y}_j = \frac{e^{W'_j * \hat{v}}}{\sum_{k=1}^{|V|} e^{W'_k * \hat{v}}}$, where W'_j is the j^{th} line of W'

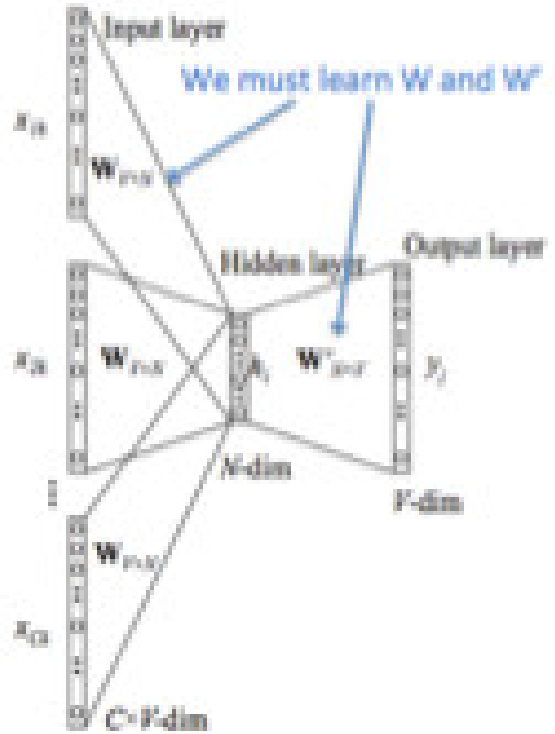


Figure 1: This image demonstrates how CBOW works and how we must learn the transfer matrices

Training the NN for CBOW

CBOW Model : how to compute the loss according to W and W'

- The model learns W et W' so that the output of the NN, $\hat{y}$, matches y
- $H(\hat{y}, y) = -log(\hat{y}_c)$
- $\hat{y}_c = softmax(W'_c * \hat{v})$,
- $H(\hat{y}, y) = -log(softmax(W'_c * \hat{v})) = -log(\frac{exp(W'_c * \hat{v}))}{\sum_{j=1}^{|V|} e^{W'_j * \hat{v}}})$

  $H(\hat{y}, y) = -W'_c * \hat{v} + log(\sum_{j=1}^{|V|} e^{W'_j * \hat{v}})$
- where :
  $\hat{v} = mean(W * x^{c-m}, ..., W * x^{c-1}, W * x^{c+1}, ..., W * x^{c+m})$
  and W'_j is the j^{th} line of W'

Training the NN for CBOW

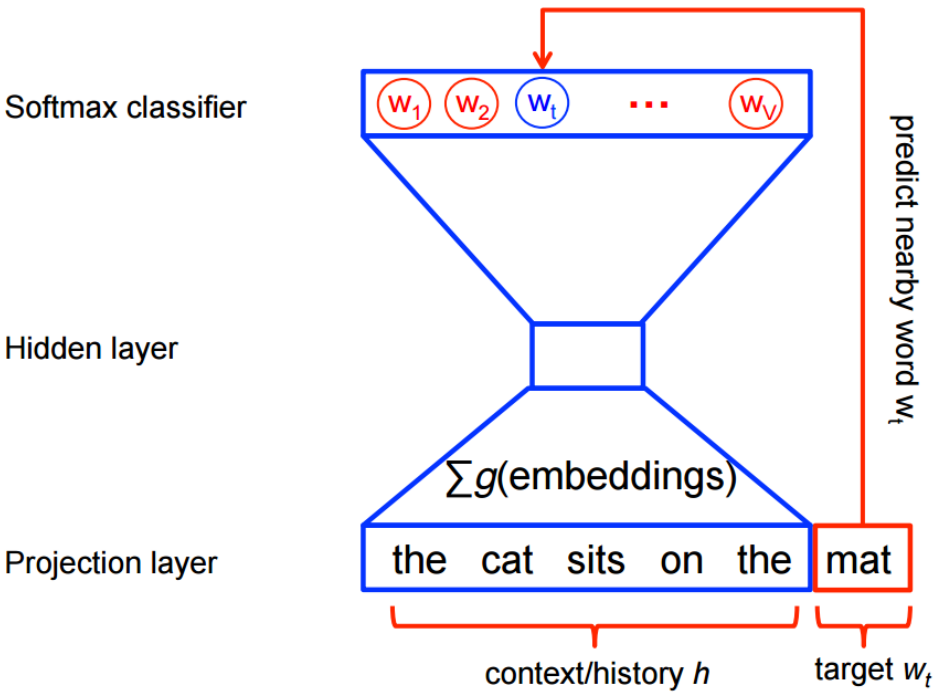
CBOW Model : how to find W and W' that minimize the loss

- $J(W, W') = (\hat{y}, y) = -W'_c * \hat{v} + log(\sum_{j=1}^{|V|} e^{W'_j * \hat{v}})$
- where :
  $\hat{v} = mean(W * x^{c-m}, ..., W * x^{c-1}, W * x^{c+1}, ..., W * x^{c+m})$
  and W'_j is the j^{th} line of W'
- use gradient descent to update W and W'

Training the NN for CBOW

Binary classification objective

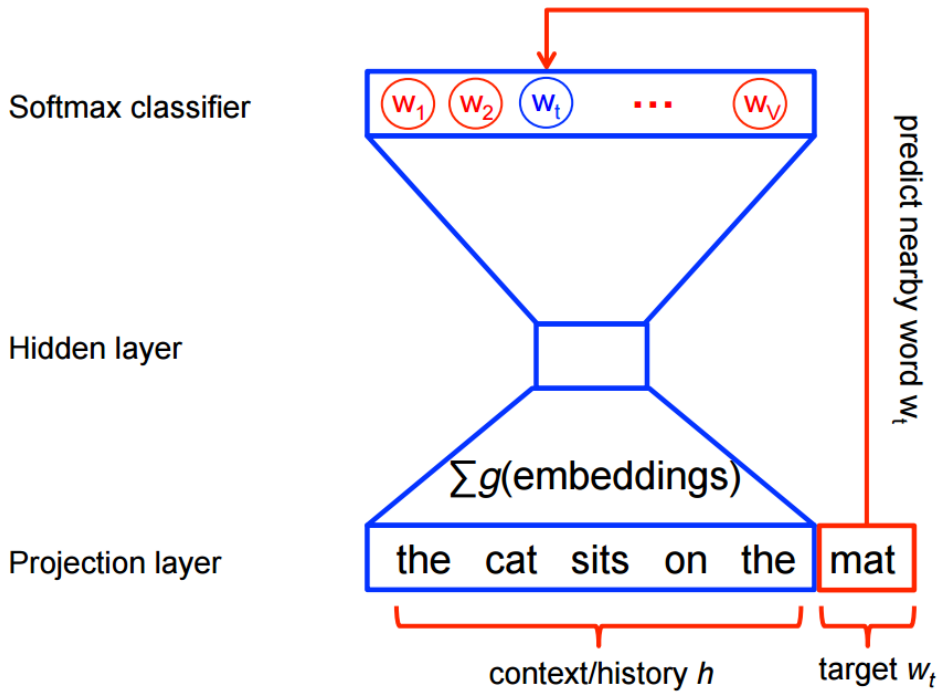
- learn to discriminate the target words w_t from k other (noise) words $w_1..w_k$ in the same context
- $\Rightarrow$ **Negative Sampling**



Training the NN for CBOW

Negative sampling

- the k constrative words are obtained from the noise distribution (Monte-Carlo average)



Training the NN for CBOW

This objective is maximized when the model assigns :

- high probabilities to the real words
- low probabilities to noise words

Scaling up Computing the loss function now scales only with the number of noise words (k) and not all words in the vocabulary(V) $\rightarrow$ much faster to train

Reminder

What do we need to start learning the word vectors ?
What are the inputs/outpus of the NN in the skip-gram model ?

Skip-gram in practice

Dataset : "The quick brown fox jumped over the lazy dog"
For **Skip-gram** , the task is to predict :

- 'the' and 'brown' from 'quick'
- 'quick' and 'fox' from 'brown'
- *etc.*

Skip-gram in practice

Step 1 : form a dataset of (context, target) pairs with $C = 1$

Step 2 : Training iteration (i.e. : iterative word vector computation)

- let's imagine at training step t we observe the first training case above, where the goal is to predict "the" from "quick".
  - we select k noisy (contrastive) examples by drawing from some noise distribution, typically the unigram distribution. For simplicity let's say $k = 1$ and let's select "sheep" as a noisy example.
  - we compute the loss for this pair of observed and noisy examples at time step t $J_{\text{NEG}}^{(t)}$

Skip-gram in practice

- - The goal is to make an update to the embedding parameters θ to improve (in this case, maximize) this objective function $J_{\text{NEG}}^{(t)} \rightarrow$ derive the gradient of the loss $\frac{\partial}{\partial \theta} J_{\text{NEG}}$
  - perform an update to the embeddings by taking a small step in the direction of the gradient
- When this process is repeated over the entire training set, this has the effect of 'moving' the embedding vectors around for each word until the model is successful at discriminating real words from noise words.

References for this lecture

Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.

Lecture from Stanford
`http://cs224d.stanford.edu/lecture_notes/notes1.pdf`
Tools : word2vec from Google
`https://code.google.com/p/word2vec/` tutorial from tensorflow `https://www.tensorflow.org/tutorials/word2vec`

To go further

Glove `http://nlp.stanford.edu/projects/glove/`
DOC2VEC `https://arxiv.org/pdf/1607.05368.pdf` and gensim tool : `https://radimrehurek.com/gensim/lrec2010_final.pdf`
SENTENCE EMBEDDINGS Article de Felix Hill : "Learning Distributed Representations of Sentences from Unlabelled Data"