UNIVERSITÀ
DEGLI STUDI
DI MILANO

LA STATALE

Master Degree in Computer Science

Information Retrieval

Introduction to neural language models

Prof. Alfio Ferrara

Department of Computer Science, Università degli Studi di Milano
Room 7012 via Celoria 18, 20133 Milano, Italia alfio.ferrara@unimi.it

sed noli modo

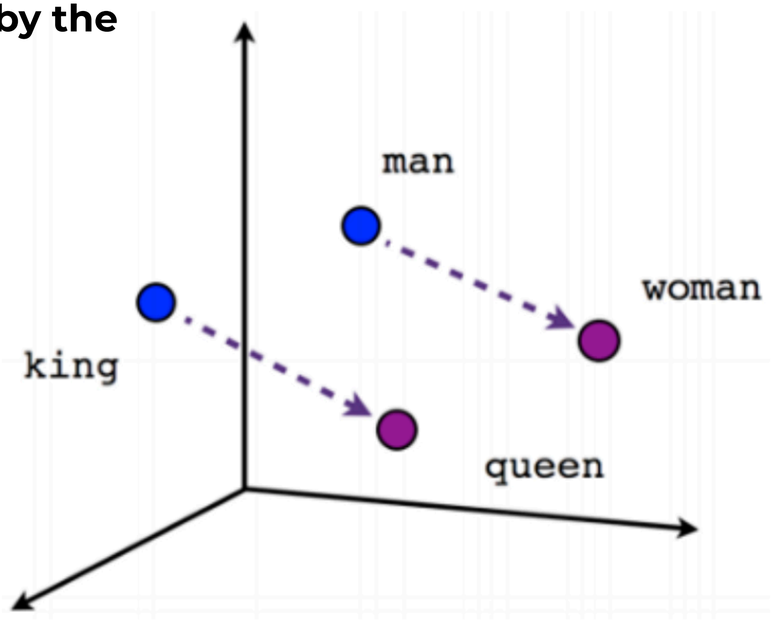
A language model in essentially a **probability distribution** over a **sequence of words** that can be modeled in terms of conditional probabilities as follows

$$p(w_n \mid w_1, w_2, \ldots, w_{n-1})$$

Using n-gram models to model this function does not solve some issues:

1. Generalize the meaning of words (n-gram models are still sensible to minimal variations in the observed words)
2. Deal with long sequences and distant dependencies

Representing **words as vectors** in the **multidimensional space defined by the other words** is a very effective way of embedding words in a vector space representing the word meaning.



Moreover, it makes it possible to:

- compute distances and similarity between words
- represent documents as regions of the feature (i.e., words) space
- providing a rich input for training neural language models

A natural but naive way of embedding a set of W_n words is to create an embedding matrix $E \in \mathbb{R}^{W_n \times W_n}$ where each entry $[e_{ij}]$ represents a relation between words w_i and w_j in a corpus.

Word relations may be

- **co-occurrence:** e_{ij} is the number of times w_j appear within w_j words from w_i in documents
- **pmi:** the pointwise mutual information associated with the pair (w_i, w_j)
- **context:** w_i and w_j appear in the same context according to a context model such as skip-gram models or continuous bag-of-words

However, this kind of embedding has two main limitations: i) **very large number of dimensions** and ii) **sparsity**

Linguistic and Philosophical Issues

In order to deal with the issues of **dimensionality** and **sparsity**, we aim at obtaining **dense word vectors**.
This can be done by **matrix factorization** or **machine learning.**
The goal of reducing the dimensionality is not only related to the cost of processing high dimensional and sparse data, but also to minimize the impact of zero and outliers

What is the semantics, the *meaning* of a word?

A common sense hypothesis is to say that the meaning of a word is the real object that word represents. In this framework, words that are not known to be mapped on real objects (e.g., new words for a reader or just random strings of characters like xvul) have no meaning at all.
However, this hypothesis is quite useless in a digital context, where we just have words, not real objects.

An alternative approach if the **Distributional Hypothesis** about language and word meaning that states that **words that occur in the same contexts tend to have similar meanings**. In other words, *you shall know a word by the company it keeps* .
According to this hypothesis, any random word (i.e., xvul) may have a meaning that we can infer from the other words in the context is appears.

One of the main advantages for us is that this way the meaning of words is quantifiable and measurable in terms of distances from the other words in a corpus.

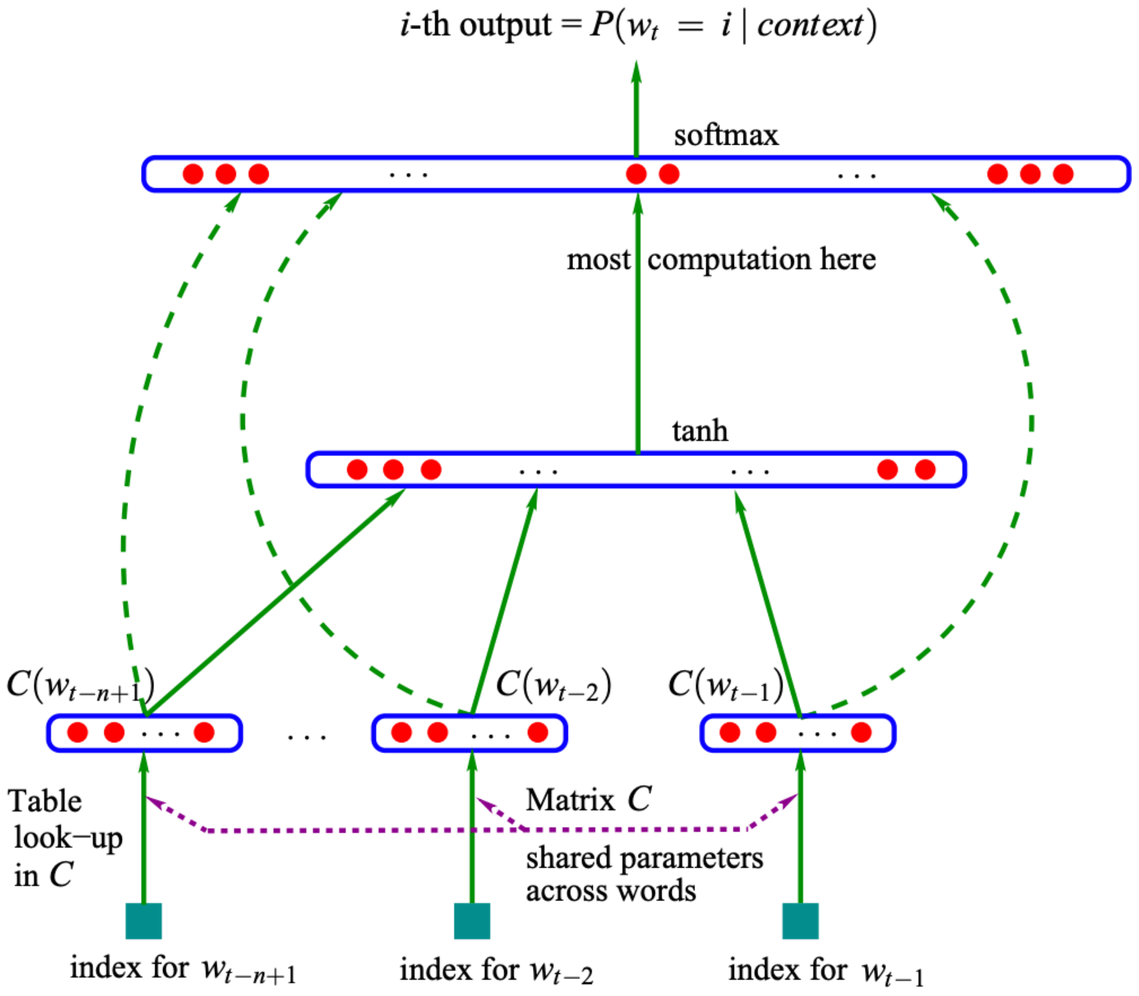
Harris, Z. S. (1954). Distributional structure. Word, 10(2-3), 146-162
John R. Firth. A synopsis of linguistic theory 1930–1955. In Studies in Linguistic Analysis, Special volume of the Philological Society, pages 1–32.
Firth, John Rupert, Haas William, Halliday, Michael A. K., Oxford, Blackwell Ed., 1957

Neural Language models

We exploit a neural network having as **input an n-gram of words** $w_{1:n}$ and having as **output a probability distribution over the next word**.

The **training set** is a sequence $w_1, \ldots, w_T$ of words, where w_t is a word in the corpus vocabulary V. The **objective function** is $f(w_t, \ldots, w_{t-n+1}) = \hat{P}(w_t \mid w_1, \ldots, w_{t-1})$ (for a n-gram), under the constraint that, for any $w_1, \ldots, w_{t-1}$

$$\sum_{i=1}^{V} f(w_i, w_{t-1}, \ldots, w_{t-n+1}) = 1, \textbf{ with } f > 0$$



Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. Journal of machine learning research, 3(Feb), 1137-1155

Neural Language models

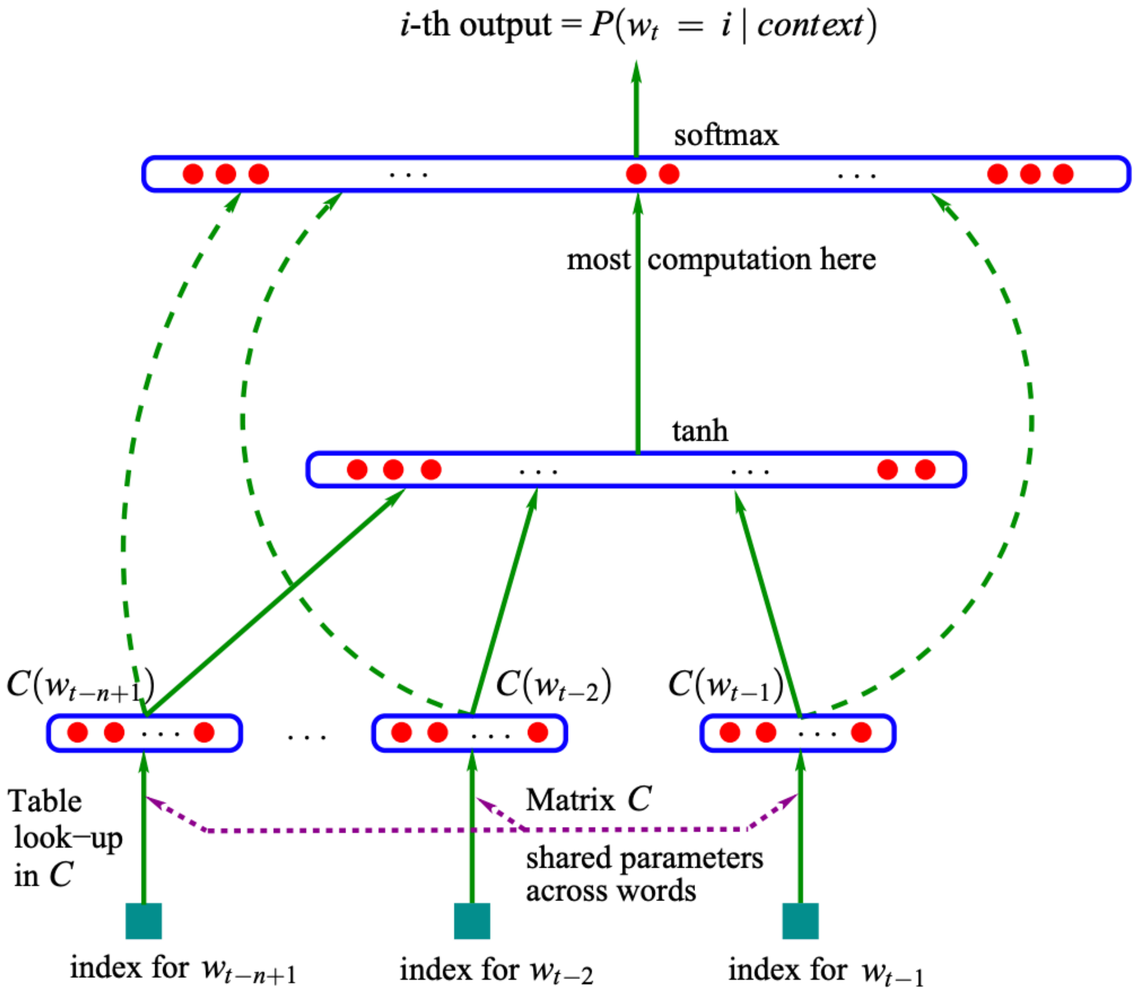
The function $f(w_t, \ldots, w_{t-n+1})$ is decomposed in two parts:

1. A mapping from any word $w_i \in V$ to a vector $\boldsymbol{w}_i \in \mathbb{R}^m$. The mapping is then a matrix $\boldsymbol{W} \in \mathbb{R}^{V \times m}$ of free parameters and represents the distributed feature vectors of each word in the vocabulary;
2. A probability function over words, that is a function g that maps and input sequence of word vectors $\boldsymbol{w}_{t-n+1}, \ldots, \boldsymbol{w}_{t-1}$ to a conditional probability distribution for the next word w_t. The output is then a vector $\boldsymbol{g} \in \mathbb{R}^V$ such that $\boldsymbol{g}_i = \hat{P}(w_t = i \mid w_1, \ldots, w_{t-1})$

According to the decomposition: $f(w_i, w_{t-1}, \ldots, w_{t-n+1}) = g(w_i, \boldsymbol{w_{t-1}}, \ldots, \boldsymbol{w_{t-n+1}})$

The function g is then implemented by a neural network that has its own parameters ω. The complete set of parameters is then $\theta = (\boldsymbol{X}, \omega)$. Learning is the performed by stochastic gradient ascent (with ϵ as learning rate) as:

$$\theta \leftarrow \theta + \epsilon \frac{\partial \log \hat{P}(w_t \mid w_{t-1,\ldots,w_{t-n+1}})}{\partial \theta}$$



Bengio, Y., Ducharme, R., Vincent, P., & Jauvin, C. (2003). A neural probabilistic language model. Journal of machine learning research, 3(Feb), 1137-1155

UNIVERSITÀ
DEGLI STUDI
DI MILANO

LA STATALE

Word2Vec

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:[1301.3781](https:// arxiv.org/abs/1301.3781).

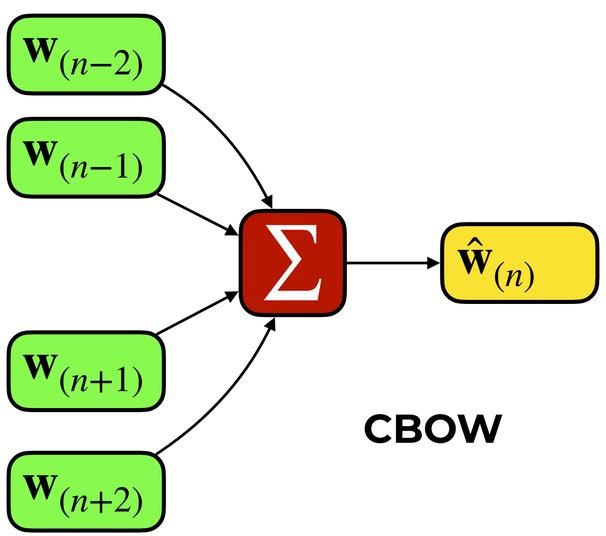
Word2Vec maps each word to a low-dimensional continuous vector, in a way that **words that have similar meaning have similar representations**.

Word2Vec is based on a simple but efficient **feed-forward neural architecture** which is trained with language modeling objective.

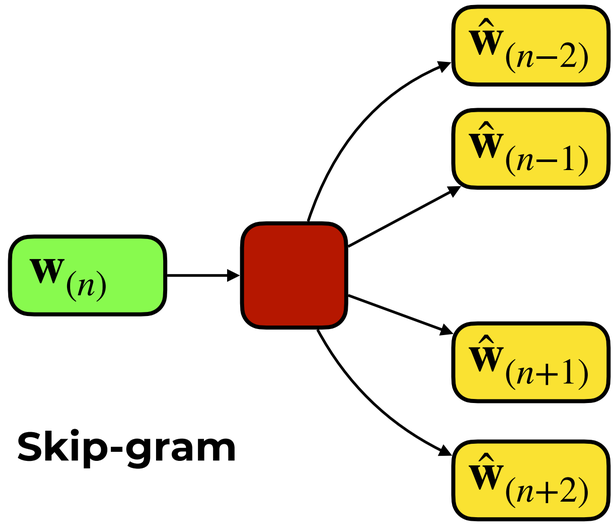
Word2Vec consists of two different architectures: **Skip-gram** and **Continuous Bag-of-Words(CBOW)**

The **CBOW** model aims at **predicting the current target word based on its context words** (surrounding words).

The **Skip-gram** model aims at **predicting the current context words (surrounding words) based on its target word**.



CBOW



Skip-gram

Suppose we want to train a **CBOW** model with the following text corpus:

`I like playing boardgames`

Example with n = 1 (equivalent to 2-gram models)

Construction of training examples

`I like playing boardgames`
`I like playing boardgames`
`I like playing boardgames`

Training set

Samples	Context	Target
1	I	like
2	like	playing
3	playing	boardgames

Example with n = 2 (equivalent to 3-gram models)

Construction of training examples

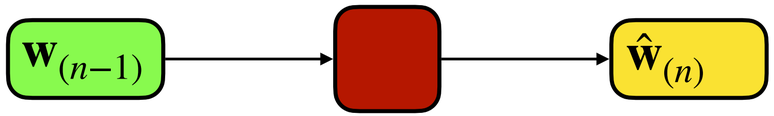
`I like playing boardgames`
`I like playing boardgames`
`I like playing boardgames`

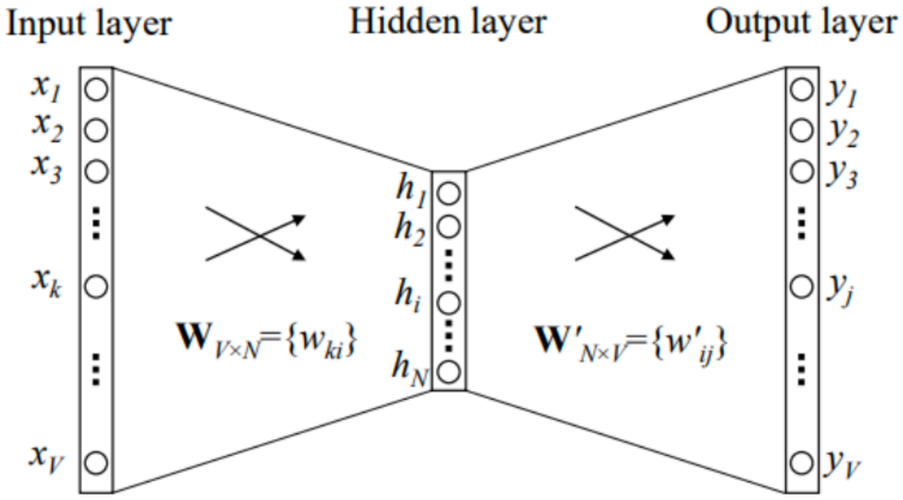
Training set

Samples	Context	Target
1	I	like
1	playing	like
2	like	playing
2	Boardgames	playing
3	playing	boardgames

Now we define a neural network to learn from the training set. Words are encoded by one-hot vectors and we define a hidden layer with a arbitrary number of dimensions (usually 100 or 300). Let's assume n=1.

CBOW





- The input is a one-hot encoded vector $\mathbf{x} \in \mathbb{R}^V$
- The weights between the input layer and the hidden layer can be represented by a matrix $\mathbf{W} \in \mathbb{R}^{V\times N}$
- Each row of $\mathbf{W}$ is the N-dimensional vector representation $\mathbf{v}_w$ of the associated word of the input layer
- The activation function of the hidden layer units is simply the Identity function (i.e., the weighted sum of inputs is directly passed to the next layer).
- The weights between the hidden layer and the output layer can be represented by a matrix $\mathbf{W}' \in \mathbb{R}^{N\times V}$

The network computes three scores:

$$\mathbf{h} = \mathbf{W}^T\mathbf{x} = \mathbf{W}^T_{(k,\cdot)} := \mathbf{v}^T_{w_i}$$

The hidden value **h** is like copying the *k*-th row of **W** to **h** assuming that the contextual word $x_k = 1$

$$u_j = \mathbf{v}'^T_{w_j}\mathbf{h}$$

Obtained by taking the j-th column of **W'** for each word *j*

$$P(w_j \mid w_i) = \hat{y}_j = \frac{\exp(u_j)}{\sum_{n=1}^{V} \exp(u_n)}$$

Activation function, a log-linear classification model to obtain the posterior distribution of words.

The final model is therefore

$$P(w_j \mid w_i) = \frac{\exp\left(\mathbf{v}'^{T}_{w_i}\mathbf{v}_{w_i}\right)}{\sum_{n=1}^{V}\exp\left(\mathbf{v}'^{T}_{n}\mathbf{v}_{w_i}\right)}$$

The loss function is then (given j^* as the index of the target word in the actual pair of words)

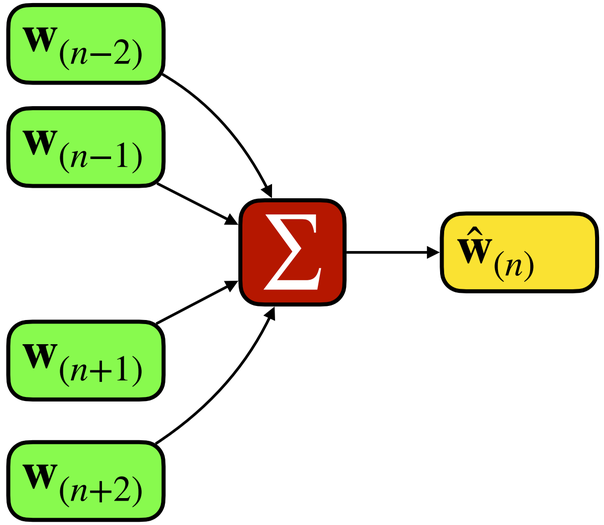
$$\mathscr{L} = \min \log \sum_{n=1}^{V}\exp\left(u_n\right) - u_{j^*}$$

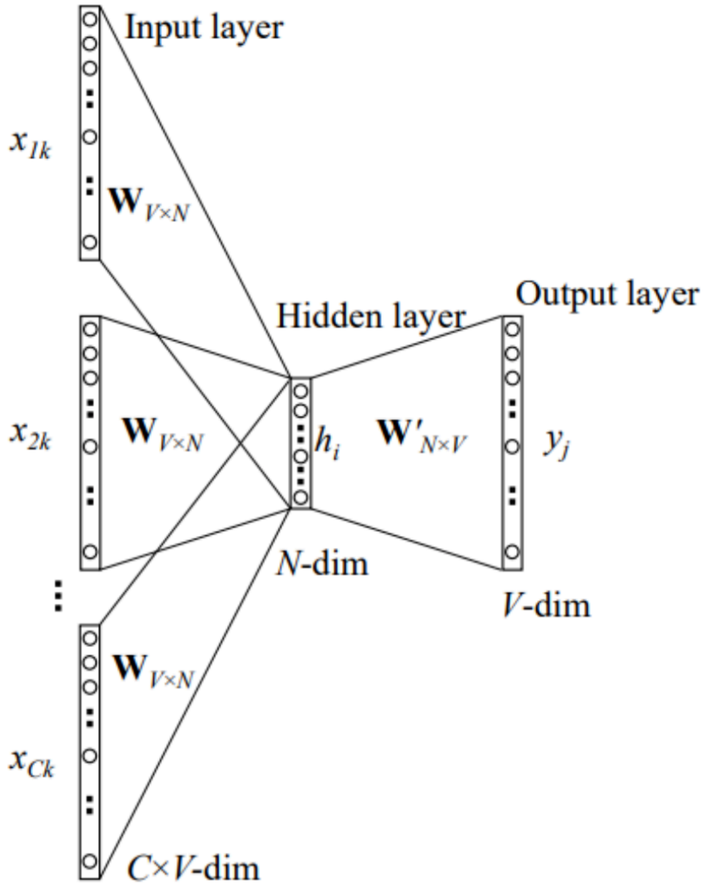
The optimization is performed by gradient descent

$$\mathbf{W}'^{(new)} = \mathbf{W}'^{(old)} - \eta\frac{\partial\mathscr{L}}{\partial\mathbf{W}'}$$

$$\mathbf{W}^{(new)} = \mathbf{W}^{(old)} - \eta\frac{\partial\mathscr{L}}{\partial\mathbf{W}}$$

In a multi-word context, i.e., with n > 1, when computing h_i instead of just coping the vector $\mathbf{v}_i$, we take the average of $v_1, \ldots, v_n$





Word2Vec in summary

The output layer is a softmax regression classifier that given the one-hot vector of a word and the weights in the hidden layer estimates the probability of each word w_j to be a word in the context of w_i.

The **main idea** of **Word2vec** is to **discard the input and the output layers and keep the hidden layer as the word embedding dense matrix of dimension** .

The main interesting property of Word2vec is that it assigns **similar vectors to words that have similar contexts**.

Why? Suppose to have (w_i, w_j) and (w_z, w_j) such that the word w_j appears in the contexts of both w_i and w_z. Since the target to estimate for w_i and w_z is the same, the only way the network has to assign the same prediction to w_i and w_z is to learn the same weights in h_i and h_z.

Another embedding model: GloVe

Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

GloVe stands for **Global vectors** and this summarizes them main idea of the algorithm. We have seen that Word2vec relies only on **local information** about a word because the word semantics is only affected by the surrounding words.

Glove starts instead from a **global co-occurence matrix** having the intuition that ratios of word- word co-occurrence probabilities have the potential for encoding some form of meaning. By learning the vectors that are good for predicting co-occurrences instead of single words (like in Word2vec) we inject more information in the final word embedding vectors.

In other terms, “the training objective of GloVe is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence. Owing to the fact that the logarithm of a ratio equals the difference of logarithms, this objective associates (the logarithm of) ratios of co-occurrence probabilities with vector differences in the word vector space. Because these ratios can encode some form of meaning, this information gets encoded as vector differences as well.”

Using word embeddings

Word vectors can be used in two ways: training a specific word embedding model (Word2vec, GloVe or others) for a corpus (assuming to have enough contents) or exploit a pre-trained model (trained usually over milions or even billions of data) to embed the corpus words

Word embedding is extremely useful for a variety of applications:

- Text search and retrieval
- Measuring the semantic distance among words
- Feeding a neural network language model
- Text classification, either supervised or unsupervised

Interesting properties of word vectors

Similarity among group of words: thanks to linearity, computing the average cosine similarity from a group of words to all other words can be calculated as

$$sim(w, (w_1, \ldots, w_k)) = \frac{1}{k} \sum_{i=1}^{k} sim_{cos}(\mathbf{w}, \mathbf{w}_i)$$

Analogy:

$$analogy(w_i : w_j = w_k = ?) =$$

$$= \arg \max_{w \in V \notin \{w_i, w_j, w_k\}} \cos(\mathbf{w}, \mathbf{w}_k - \mathbf{w}_i + \mathbf{w}_j)$$