slidenumbers: true
footer: A. Ferrara. **Language models**. *Part 4: Word embeddings*. email, corse website, slack, github

Information Retrieval

[fit]Language models

Part 4: Word Embeddings. Prof. Alfio Ferrara

Master Degree in Computer Science

Master Degree in Data Science and Economics

Words as vectors

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 (see Part 5 of these lectures)

Naive embeddings

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 t 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**

Dense embeddings

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

---

[fit] Linguistic and Philosophical Issues (I)

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.

Linguistic and Philosophical issues (II)

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. [1] In other words, *you shall know a word by the company it keeps* . [2]

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. **Distributional semantics** is the research are interested in quantifying semantic similarities between linguistic items according to their distributional properties in large text corpora.

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.

Weight contextual relations

Given a word w_i and a word w_j in the context of w_i, we need thus to **quantify** the relation (w_i, w_j) and define the score $[e_{ij}]$ in the emebedding matrix.

We can do this by just counting the occurrences of (w_i, w_j) (i.e., how many times w_j appears in the context of w_i) or taking the normalized count:

$$[e_{ij}] = \frac{count(w_i, w_j)}{\sum\limits_{w_i^{'}, w_j^{'} \in D} count(w_i^{'}, w_j^{'})}$$

The drawback of this solution is to overestimate frequent words. So pairs like 'the apple' will have higher scores than 'red apple', although the last one is more informative.

Pointwise Mutual Information

A different option is to evaluate the relation between the words joint probability and their marginal probability through the **Pointwise Mutual Information (PMI)**

$$[e_{ij}] = PMI(w_i, w_j) = \log \frac{P(w_i,w_j)}{P(w_i)P(w_j)} = \log\left[\left(\sum_{w_i^{'},w_j^{'}\in D} count(w_i^{'},w_j^{'})\right)\frac{count(w_i,w_j)}{count(w_i)count(w_j)}\right]$$

or, to avoid negative values, **positive PMI**, $PPMI(w_i, w_j) = \max(PMI(w_i, w_j), 0)$

A drawback of PMI is that it tends to assign high value to rare events. It is therefore advisable to apply a count threshold before using the PMI metric, or to otherwise discount rare events.

Distributed word representation

Count-based methods (such as PMI) represent a word as a vector $\vec{w} \in \mathbb{R}^n$ where each dimension corresponds to a word in the corpus dictionary, so that $\vec{w}_i$ represents the score of the relation between w and a word w_i (for example, $\vec{w}_i = PMI(w, w_i)$). Such vectors are typically **sparse** and n is usually large.

On the opposite, the **distributed representation** of words meaning associates each word w with a **dense** vector $\vec{w} \in \mathbb{R}^d$ with $d \ll n$. The vector dimensions **do not** represent words nor concepts, and we are not allowed to interpret them as concepts.

The semantics of words is completely represented by the mutual position of words in the vector space. In particular, we want to preserve the **proximity assumption** for which if two word vectors are close one to the other, then the two words have a similari meaning.

Neural network models (I)

An example is given in the work of Bengio at al. [3] 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.

Preliminary set up: The training set is a sequence $w_1, \dots, w_T$ of words, where w_t is a word in the corpus vocabulary V.

The objective function is $f(w_t, \dots, w_{t-n+1}) = \hat{P}(w_t \mid w_1, \dots, w_{t-1})$ (for a n-gram), under the constraint that, for any $w_1, \dots, w_{t-1}$

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

Neural network models (II)

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:

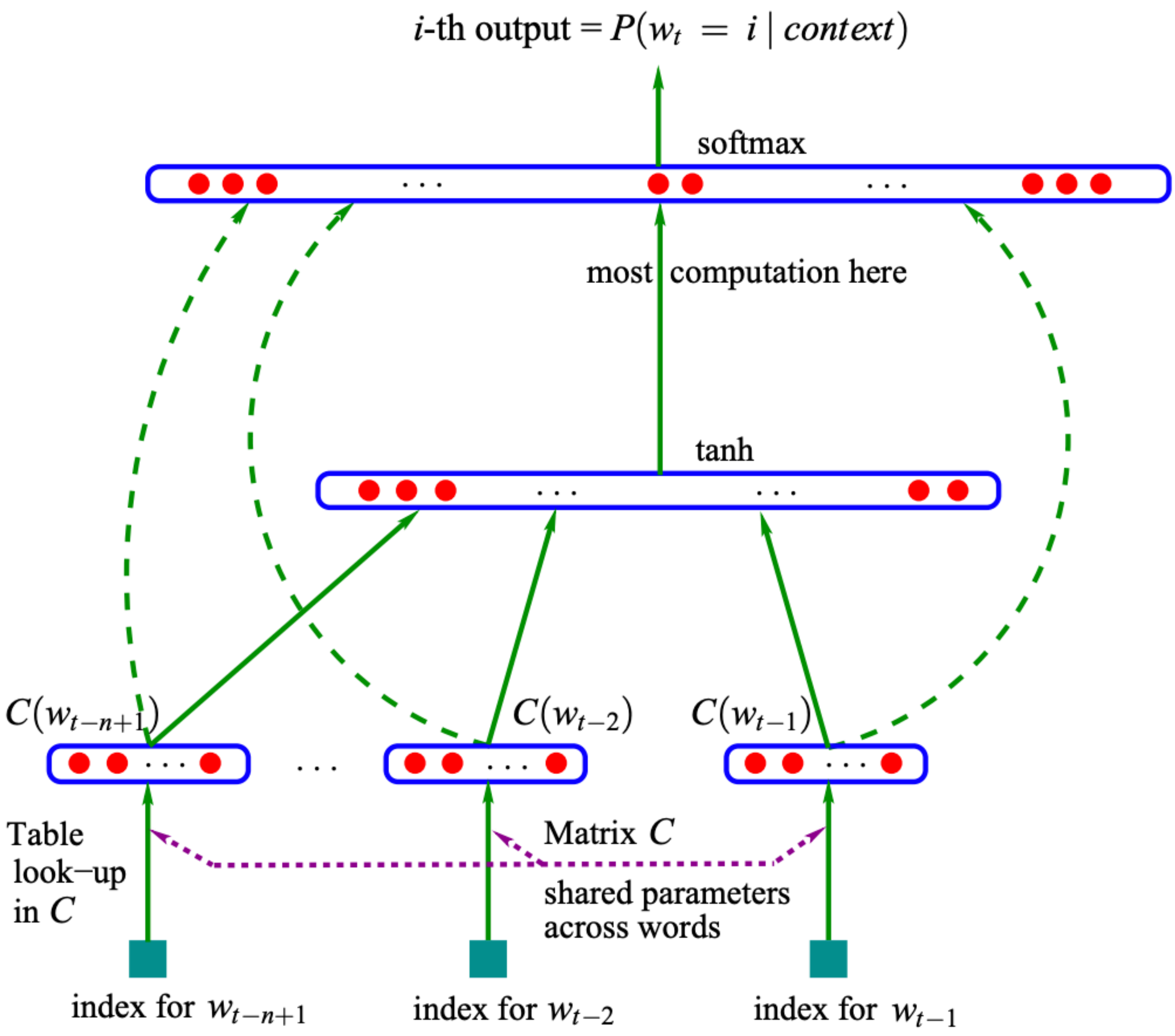
$$f(w_i, w_{t-1}, \ldots, w_{t-n+1}) = g(w_i, \boldsymbol{w_{t-1}}, \ldots, \boldsymbol{w_{t-n+1}})$$

Neural network models (III)

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:

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

For an example on from scratch implementation see GitHub



Neural language models for word embeddings

As we have seen, the neural language model 'learns' the embedding as part of its parameter estimation process.

From the work of Bengio, other models have been proposed in particular by relaxing the probabilistic output requirement.

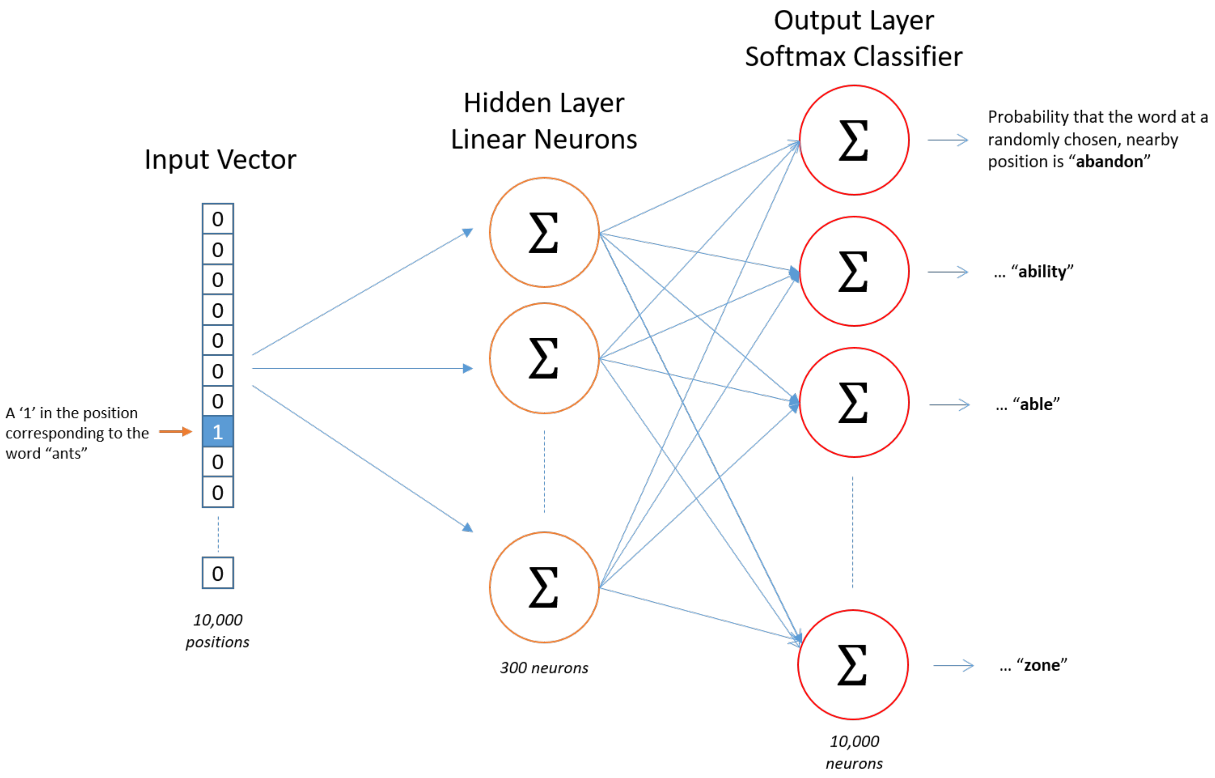
Instead of computing a probability distribution over target words given a context, the Collobert and Weston model only attempts to assign a score to each word, such that the correct word scores above incorrect ones. [4]

Word2vec

Word2vec is a method for efficiently represent words in vector space. [5]

It uses either the **Continuous Bag of Words (CBOW)** or the **Skip-gram** models for representing the word context and two different optimization objectives that are **Negative-Sampling** and **Hierarchical Softmax**. We will se Negative-Sampling.

The image is taken from the online Word2Vec Tutorial (link) where it is possible too find a intuitive view of the Word2vec main ideas.



Word2vec

Consider a set D of *correct* word pairs (w_i, w_j) (e.g., valid bigrams) and a set $\overline{D}$ of *bad* word pairs. The goal of the algorithm is to estimate $P(D = 1 \mid w_i, w_j)$. The objective is to maximize the log-likelihood of the pairs in $D \cup \overline{D}$

$$\mathcal{L}(\Theta; D; \overline{D}) = \sum_{w_i, w_j \in D} \log P(D = 1 \mid w_i, w_j) + \sum_{w_i, w_j \in \overline{D}} \log P(D = 0 \mid w_i, w_j),$$

where, given the score $s(w_i, w_j)$

$P(D = 1 \mid w_i, w_j) = \frac{1}{1+e^{-s(w_i, w_j)}}$

Word2vec

The positive samples D are taken from the corpus, while the negative ones are sampled according to the word frequency in the corpus.

CBOW: the scoring function is defined as $\sum_{j=1}^{k} \boldsymbol{w}_i \cdot \boldsymbol{w}_j$ for a word context of k words

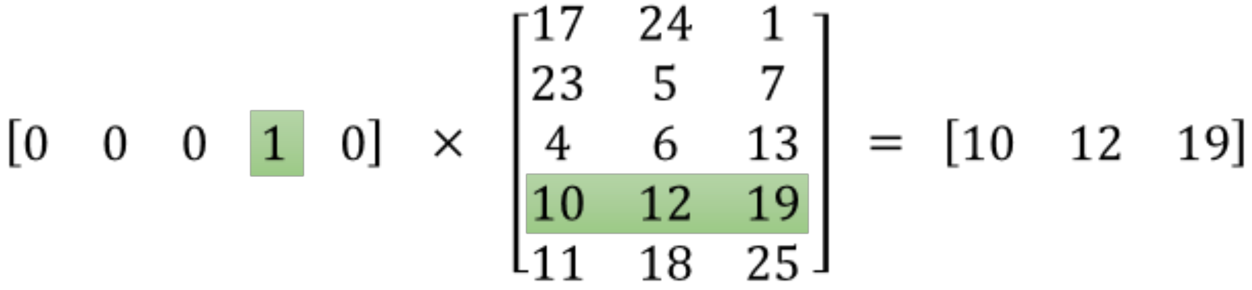
Skip-gram: in the skip-gram variant, assumes that the elements of the context are all independent, such that

$$P(D = 1 \mid w_i, w_{1:k}) = \sum_{j=1}^{k} \log \frac{1}{1+e^{-w_i \cdot w_j}}$$

Word2vec

In practice, Word2vec is implemented by a neural network composed by a $\boldsymbol{W}^{V \times V}$ input layer, a $\boldsymbol{H}^{V \times m}$ hidden layer, and a $\boldsymbol{w}^{V}$ output layer, where V is the size of the vocabulary ($\boldsymbol{W}^{V \times V}$ is the one-hot encoding representation of words) and m is the dimension of the embedding vectors.

The hidden layer stores the weights that are used to feed the output layer and that are learned by the network. See the example in the word2vec tutorial:



$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}$$

Word2vec

The output layer is a softmax regression classifier that given the one-hot vector of a word w_i 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 $V \times m$.

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.

GloVe

GloVe [6] 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 (see the overview on the GloVe web page).

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.

GloVe

The main intuition of GloVe is represented by its objective function that is

$$\boldsymbol{w}_i \cdot \boldsymbol{w}_j + \boldsymbol{b}_i + \boldsymbol{b}_j = \log p(w_i, w_j),$$

where $\boldsymbol{b}_i$ $\boldsymbol{b}_j$ represent bias parameters that are learned together with the word 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.* link

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 bilions 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 (see next lecture)
- Text classification, either supervised or unsupervised

Some interesting properties of word embedding 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

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

Analogy

$$analogy(w_i : w_j \rightarrow w_k : \ ?) = argmax_{w \in V \notin \{w_i,w_j,w_k\}} cos(\boldsymbol{w}, \boldsymbol{w}_k - \boldsymbol{w}_i + \boldsymbol{w}_j)$$

Example

As an example, we will train our word embedding model on the Movie Dialogs dataset. However, a single movie does not provide enough text to have meaningful results. Thus, we will train a different model for each of the 7 genres having at least 50 000 lines of text, by aggregating movies of that genre into a unique corpus.

Text sequences have been generated using 2-skip-3-grams. See the complete example on GitHub

	1	2	3	4	5	6	7
genre	crime	mystery	romance	thriller	action	drama	comedy
lines	75435	51662	77518	124082	70257	172557	94575
sequences	1313534	875790	1300360	2060898	1142600	3042069	1557657

[1] Harris, Z. S. (1954). Distributional structure. *Word*, *10*(2-3), 146-162. ↩

[2] 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. ↩

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

[4] Collobert, R., & Weston, J. (2008, July). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning* (pp. 160-167). ↩

[5] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*. ↩

[6] 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). ↩