TEXT ANALYTICS

Static Embeddings

Francesco Spinnato

* francesco.spinnato@unipi.it
University of Pisa



Where We Are

In the lecture on representing text we gave **documents** a vector: one coordinate per word type, filled with counts or tf-idf weights.

We ended with a limit: a query for *car* scores 0 on a document that only contains *automobile*. The two words are different coordinates, and one-hot vectors of different words are orthogonal.

We promised to give **words their own vectors**, learned from the contexts they appear in, so that *car* and *automobile* end up close.

These vectors are called **embeddings**.

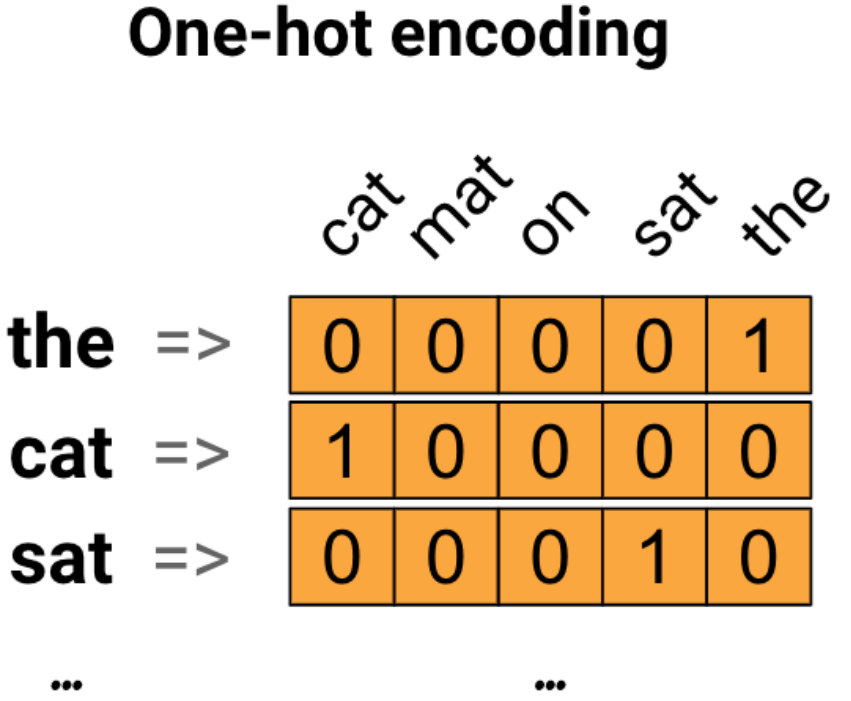
Recap, One-Hot Vectors and Bags of Words

Let $V = \{w_1, \ldots, w_{|V|}\}$ be the vocabulary. Each word $w \in V$ is a **one-hot vector** with a single 1 at its own position:

$$\mathbf{o}_w = [0, \ldots, 0, 1, 0, \ldots, 0] \in \mathbb{R}^{|V|}$$

A **bag of words** describes a document $d = (w_1, \ldots, w_T)$ as the sum of the one-hot vectors of its words:

$$\mathbf{x}_d = \sum_{t=1}^{T} \mathbf{o}_{w_t}$$



Every Word Is Equally Far

Take a tiny vocabulary $V = (\text{cat}, \text{dog}, \text{car}, \text{bicycle})$:

$$\mathbf{o}_{\text{cat}} = [1, 0, 0, 0] \quad \mathbf{o}_{\text{dog}} = [0, 1, 0, 0] \quad \mathbf{o}_{\text{car}} = [0, 0, 1, 0] \quad \mathbf{o}_{\text{bicycle}} = [0, 0, 0, 1]$$

Any two different words are **orthogonal**, and at the same distance:

$$\text{cosine}(\mathbf{o}_{\text{cat}}, \mathbf{o}_{\text{dog}}) = \text{cosine}(\mathbf{o}_{\text{cat}}, \mathbf{o}_{\text{car}}) = 0$$

- the representation carries **identity** only, no meaning: *cat* is as far from *dog* as from *car*
- it is **high-dimensional** ($|V|$ is 10^5 or more) and **sparse**: almost all entries are zero
- a model that has seen *cat* learns **nothing** about *dog*: related words share no statistical strength

Distributional Semantics

Words as vectors of contexts

The Distributional Hypothesis

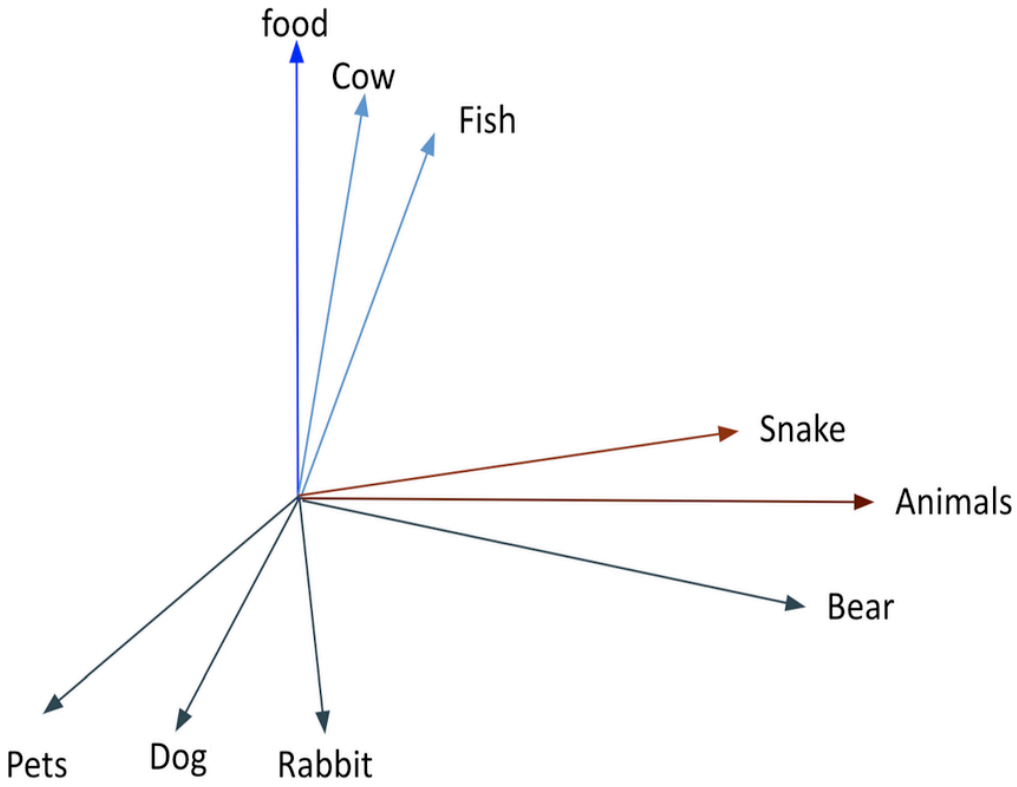
"You shall know a word by the company it keeps." (Firth, 1957)

The **distributional hypothesis**: words that occur in **similar contexts** tend to have similar meanings.

- "I drank a cup of **tea** this morning"
- "I drank a cup of **coffee** this morning"

Tea and *coffee* are close because their contexts are nearly interchangeable.

Meaning is learned from **raw text**: no dictionary, no annotation.



The Word-Context Matrix

We turn the hypothesis into numbers with the Word-Context Matrix.

- rows are **words**, columns are **contexts**,
- cells in the matrix $\mathbf{Z}$ count how often a word appears with another
- context: words within a window of $\pm m$ tokens. The matrix is $|V| \times |V|$.

Corpus: *"the dusty road ends nowhere"*; *"the dusty track ends there"*, $m = 2$:

	the	**dusty**	**ends**	**nowhere**	**there**
road	1	1	1	1	0
track	1	1	1	0	1
...	...	...	...	...	...

Each row is the **usage signature** of a word. *road* and *track* never occur together, but their rows are close.

The Word-Context Matrix

	the	dusty	ends	nowhere	there
road	1	1	1	1	0
track	1	1	1	0	1
...	...	...	...	...	...

If we now try to compare the representations of words "road" and "track":

$$\text{cosine}(\mathbf{z}_{\text{road}}, \mathbf{z}_{\text{track}}) = \frac{\mathbf{z}_{\text{road}} \cdot \mathbf{z}_{\text{track}}}{\|\mathbf{z}_{\text{road}}\| \|\mathbf{z}_{\text{track}}\|} = \frac{3}{2 \cdot 2} = 0.75$$

Weighting Counts with PPMI

Raw counts are dominated by words that are frequent **everywhere** (*the*, *of*). Weighting can fix it: replace each count with how much **more** often w and c co-occur than chance predicts:

$$\mathrm{PMI}(w, c) = \log_2 \frac{P(w, c)}{P(w)\,P(c)} \qquad \mathrm{PPMI}(w, c) = \max\big(0,\ \mathrm{PMI}(w, c)\big)$$

In a corpus of $10{,}000$ word pairs, *data* occurs 200 times, *science* 300 times, together 50 times:

$$\mathrm{PMI} = \log_2 \frac{0.005}{0.02 \cdot 0.03} = \log_2 8.33 \approx 3.06$$

The pair co-occurs about 8 times more often than chance.

The same formula that ranked **collocations** now gives **every** cell of the matrix its value.

Syntagmatic and Paradigmatic Associations

Syntagmatic (first-order)

Words that occur **together**, in the same context.

- *drink*, *water*; *play*, *game*; *commit*, *crime*
- read **directly** from the matrix: a high count or PPMI
- combine in sequence: verb and object, adjective and noun

Paradigmatic (second-order)

Words that occur in **similar contexts**, so one can replace the other.

- *drink*, *eat*; *water*, *juice*; *road*, *track*
- obtained by **comparing rows** of the matrix
- they may **never** occur together

Embeddings aim at the second: *car*, *automobile* are paradigmatic neighbors.

Limits of Word-Context Matrices

Rows of a word-context matrix are a good signature of a word, but:

- **high-dimensional**: one column per context word, so $|V|$ columns (more, if contexts are n-grams)
- **extremely sparse**: most cells are zero
- **unreliable for rare words**: few occurrences, noisy counts
- **brittle**: patterns change a lot on small corpora

We want **dense, low-dimensional** vectors that keep the similarities of the rows, and share them across related words.

Dense Embeddings

The embedding matrix

Dense Representations

An **embedding** maps a discrete object, here a word, into a **dense vector**:

$$w \longrightarrow \mathbf{e}_w \in \mathbb{R}^d \qquad \text{with} \qquad d \ll |V|$$

	One-hot	**Embedding**
Dimensions	$\lvert V\rvert$, tens or hundreds of thousands	d, a few hundred
Entries	almost all zero	all real numbers
Similar words	orthogonal	close

In a one-hot vector, each coordinate stands for one word. In an embedding, **no single coordinate has a meaning on its own**: the meaning of a word is spread over all its coordinates. This is a **distributed** representation.

Embedding Intuition

The goal is not just compression. It is to learn a space where **useful relationships become geometric relationships.**

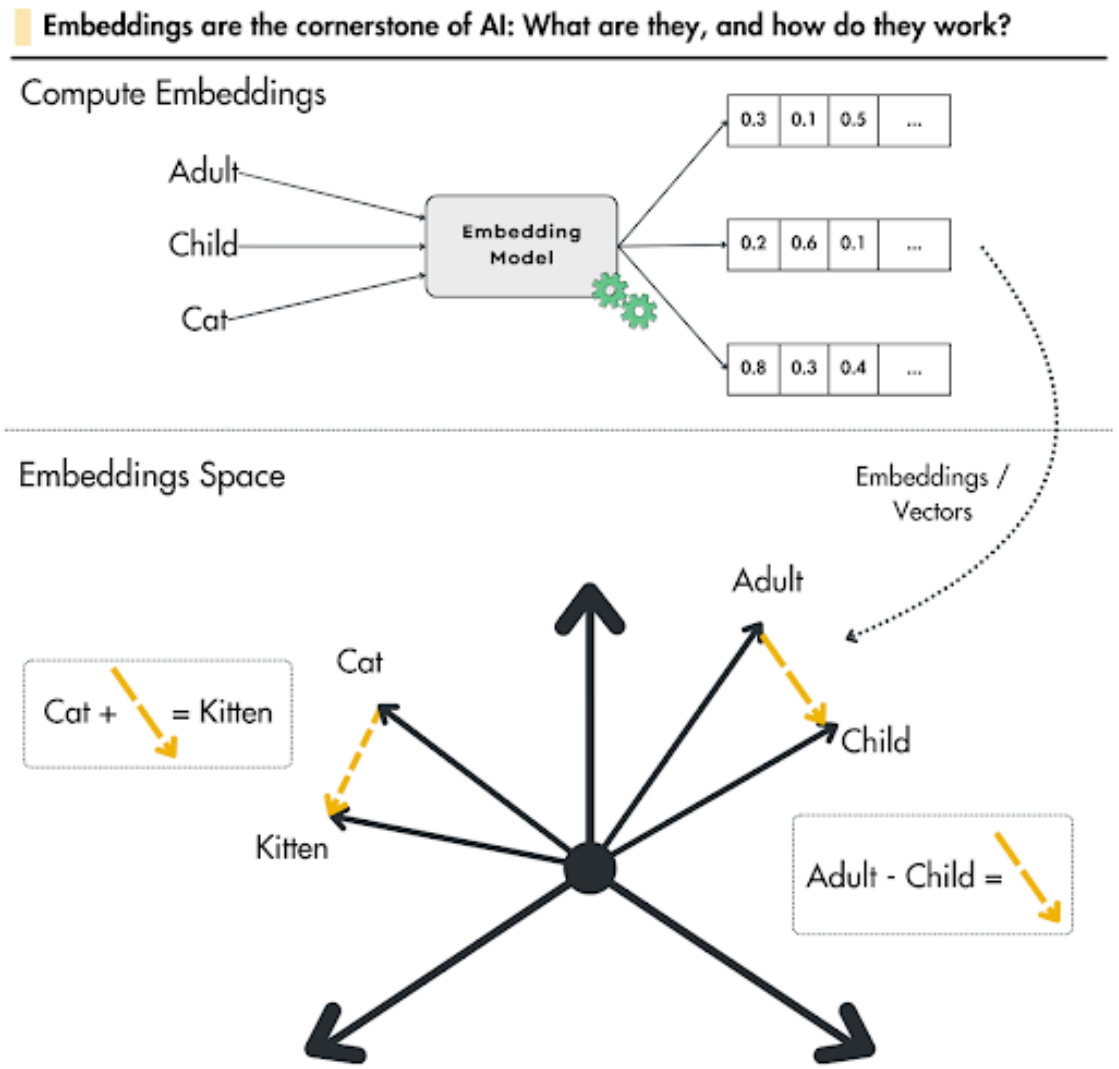
Similar words have nearby vectors:

$$\mathbf{e}_{\text{cat}} \approx \mathbf{e}_{\text{kitten}}$$

Different words have distant vectors:

$$\mathbf{e}_{\text{cat}} \not\approx \mathbf{e}_{\text{child}}$$

Closeness is measured with the **cosine**, as in lecture 6.



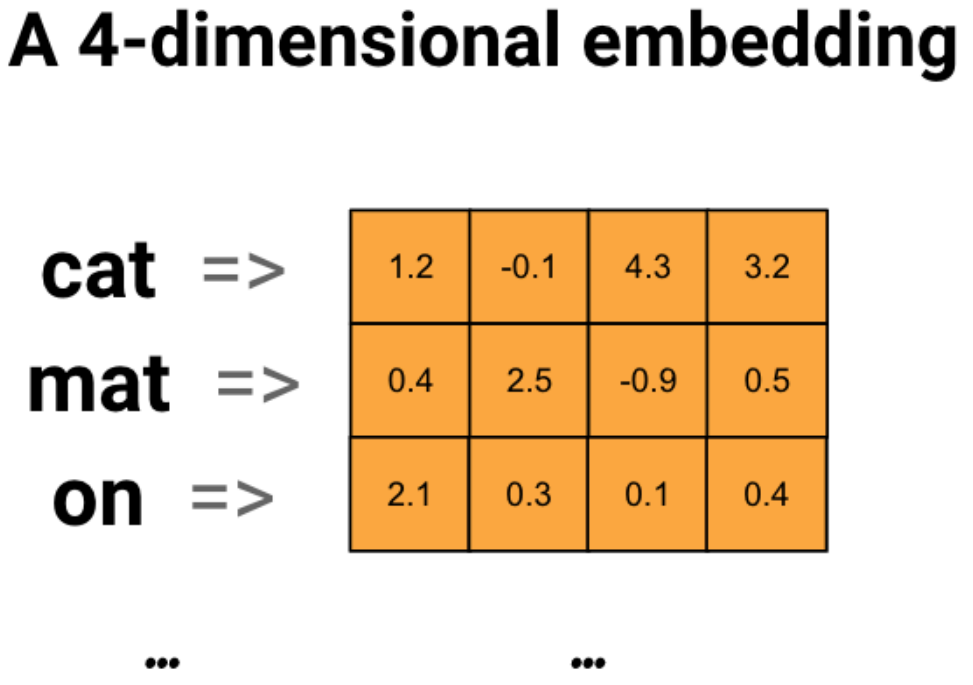
The Embedding Matrix

An embedding layer is just a **matrix**:

$$\mathbf{E} \in \mathbb{R}^{|V| \times d}$$

- $|V|$ is the vocabulary size
- d is the embedding dimension
- row i is the embedding of the word with ID i

$$\mathbf{e}_w = \mathbf{E}_{i,:}$$



Embedding Lookup

Given the word w with ID i, the embedding layer returns the i-th row of $\mathbf{E}$:

$$\mathbf{e}_w = \mathbf{E}_{i,:}$$

This operation is called **embedding lookup**.

It is equivalent to multiplying the one-hot vector by the embedding matrix:

$$\mathbf{o}_w \mathbf{E} = \mathbf{E}_{i,:}$$

but selecting a row is **much cheaper** than a product with a matrix of $|V| \times d$ numbers, almost all multiplied by zero.

Embedding Lookup, Example

Suppose bicycle is encoded as $[0, 0, 0, 1]$ and: $\mathbf{E} = \begin{bmatrix} 0.2 & 1.1 \\ -0.4 & 0.8 \\ 1.5 & -0.3 \\ 0.7 & 0.1 \end{bmatrix}$

Naive view: one-hot multiplication

$$\begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.2 & 1.1 \\ -0.4 & 0.8 \\ 1.5 & -0.3 \\ 0.7 & 0.1 \end{bmatrix} = \begin{bmatrix} 0.7 & 0.1 \end{bmatrix}$$

Efficient view: row lookup

We directly select row 4:

$$\mathbf{E}_{4,:} = \begin{bmatrix} 0.7 & 0.1 \end{bmatrix}$$

Embeddings Are Parameters

The embedding matrix is **learned from data**. It is part of the parameters of a neural network:

$$\theta = \{\mathbf{E}, \mathbf{W}_1, \mathbf{W}_2, \ldots\}$$

- during training, the gradients update the rows of $\mathbf{E}$ of the words observed in the batch
- an embedding is **not designed by hand**: it is learned to be useful for the training objective
- for text, words that appear in **similar contexts** end up with **similar** vectors

How do we choose the objective? Two families of methods.

Two Ways to Learn Embeddings

1. Factorize the word-context matrix. Compress $\mathbf{Z}$ with a truncated **SVD**:

$$\mathbf{Z} \approx \mathbf{U}_d \mathbf{\Sigma}_d \mathbf{V}_d^\top$$

The rows of $\mathbf{U}_d \mathbf{\Sigma}_d$ are d-dimensional word vectors (the idea behind latent semantic analysis). It needs the **whole matrix** and a new factorization when data changes: expensive for large vocabularies.

2. Predict the context. Train a small neural network to predict words from their contexts. Its **parameters are the embeddings**. This scales to billions of tokens.

Word2Vec

Learning embeddings by predicting context

Word2Vec

Word2Vec is a family of **shallow neural models** that learn word embeddings by solving a simple **prediction task** on a large corpus.

1. choose a prediction task: context from a word, or word from its context
2. measure the prediction error with a cost function
3. adjust the parameters by **gradient descent**
4. the **embedding matrix** of the network is the result

No labels are needed: the text supplies its own supervision (**self-supervised**), and the assumption behind it is the distributional hypothesis.

Context Windows

Given the tokens of the corpus $w_1, w_2, \ldots, w_T$ (with $T = |C|$), we define a **context window** of size m around each token. For $m = 2$, the context of w_t is the two tokens before and the two after:

$$w_{t-2},\ w_{t-1},\ w_t,\ w_{t+1},\ w_{t+2}$$

For the center word **fox** in *"the quick brown fox jumps over"*, the context is $\{\text{quick}, \text{brown}, \text{jumps}, \text{over}\}$:

- the **quick brown** fox **jumps over**

Word2Vec learns from these **local co-occurrence patterns**, the same ones counted in the word-context matrix.

Two Word2Vec Formulations

CBOW (continuous bag of words)

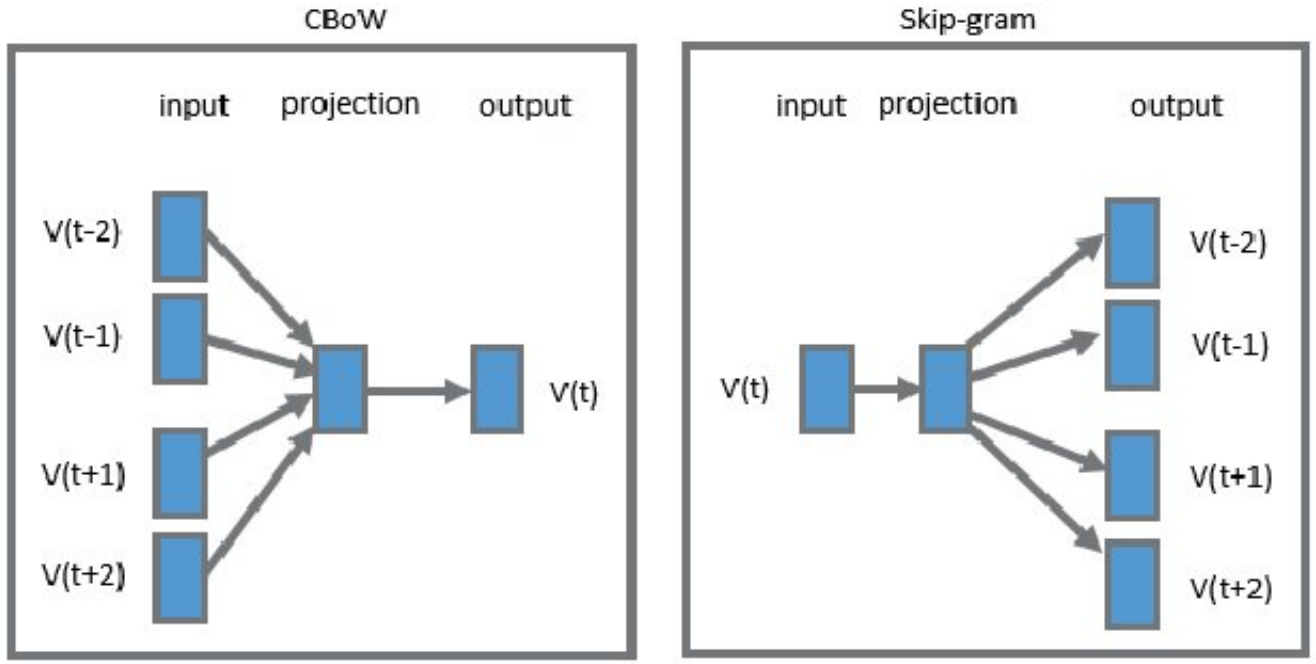
Predict the center word from its context.

$$\text{context} \rightarrow \text{center word}$$

Skip-gram

Predict the context words from the center word.

$$\text{center word} \rightarrow \text{context}$$



Skip-gram, the Model

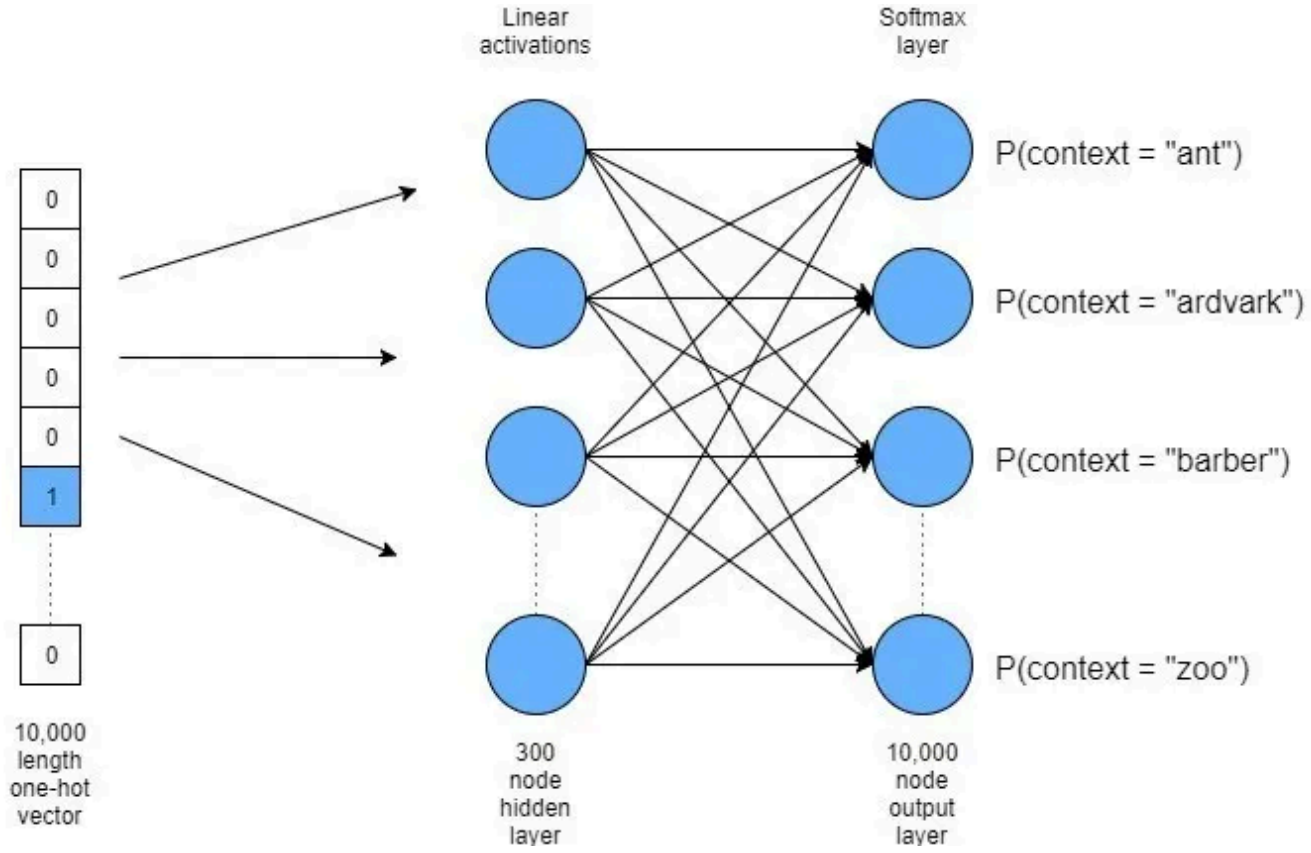
Given many observed token sequences, the model estimates its parameters so that **real center-context pairs become likely**.

For **skip-gram**, it learns probabilities

$$P(w_o \mid w_c)$$

- w_c: the **center** word
- w_o: an observed **context** word

The learned parameters are the **embedding vectors**.



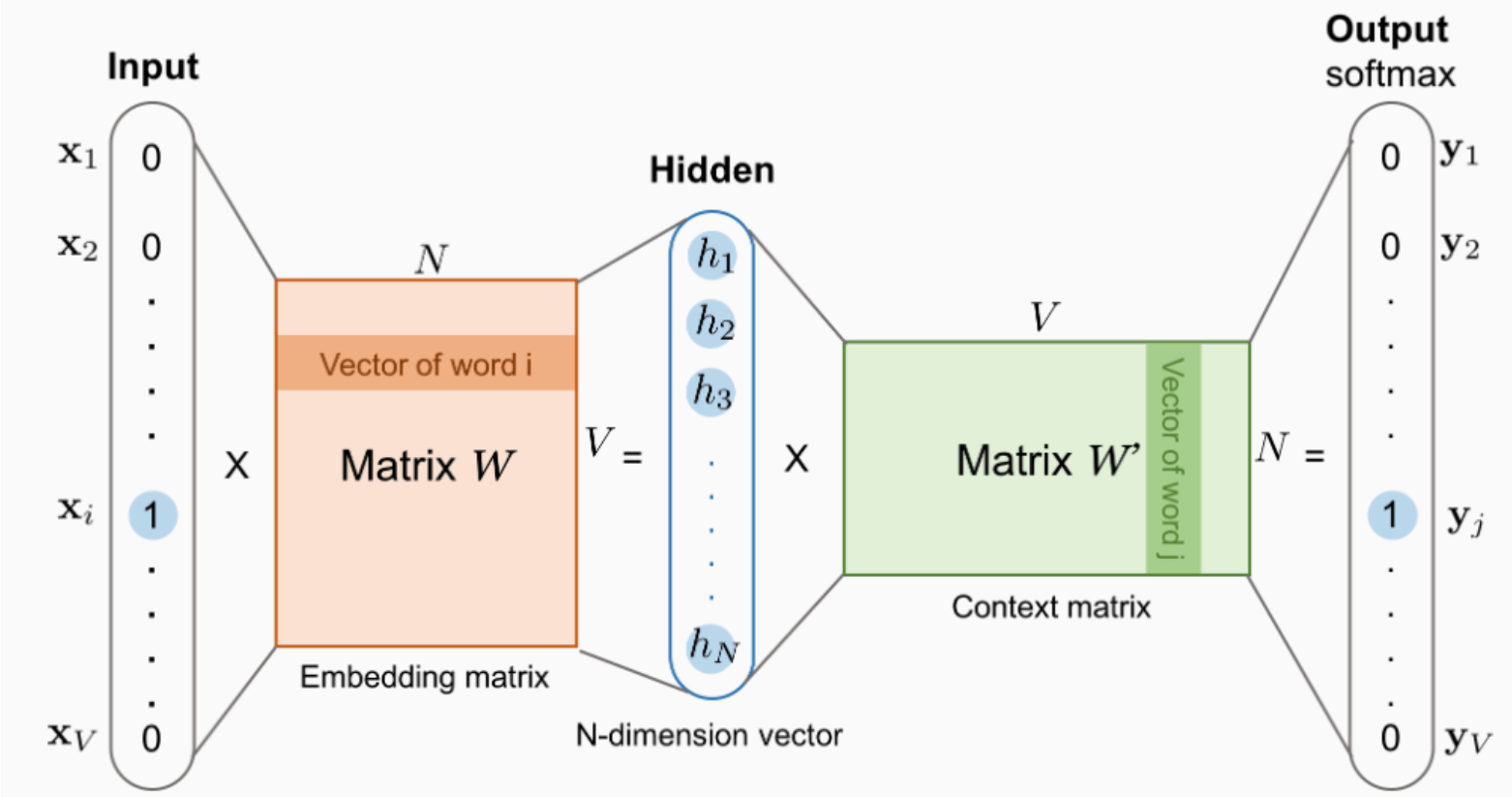
Skip-gram, from One-Hot to the Embedding

The first step multiplies the one-hot vector of the center word by the embedding matrix $\mathbf{E} \in \mathbb{R}^{|V| \times d}$ ($\mathbf{W}$ in the picture).

It **selects the row** of w_c:

$$\mathbf{o}_{w_c}\mathbf{E} = \mathbf{e}_{w_c}$$

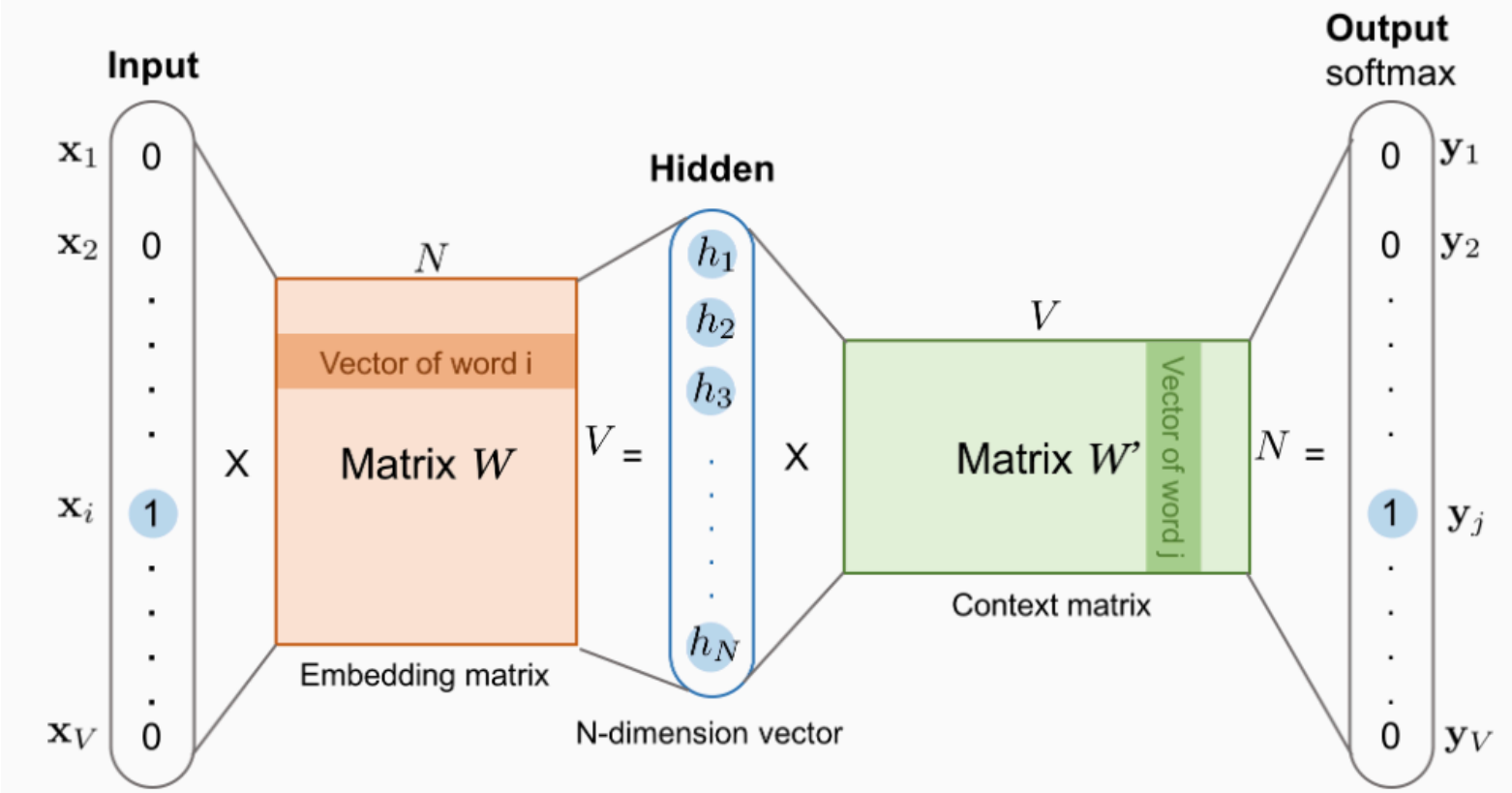
The result **is** the embedding of the center word.



Skip-gram, from the Embedding to Probabilities

The embedding $\mathbf{e}_{w_c}$ is multiplied by a second matrix $\mathbf{W}' \in \mathbb{R}^{d \times |V|}$, giving one **score** (a **logit**) for each word of the vocabulary. A softmax turns the scores into probabilities: $P(\cdot \mid w_c) = \text{softmax}(\mathbf{e}_{w_c}\mathbf{W}')$

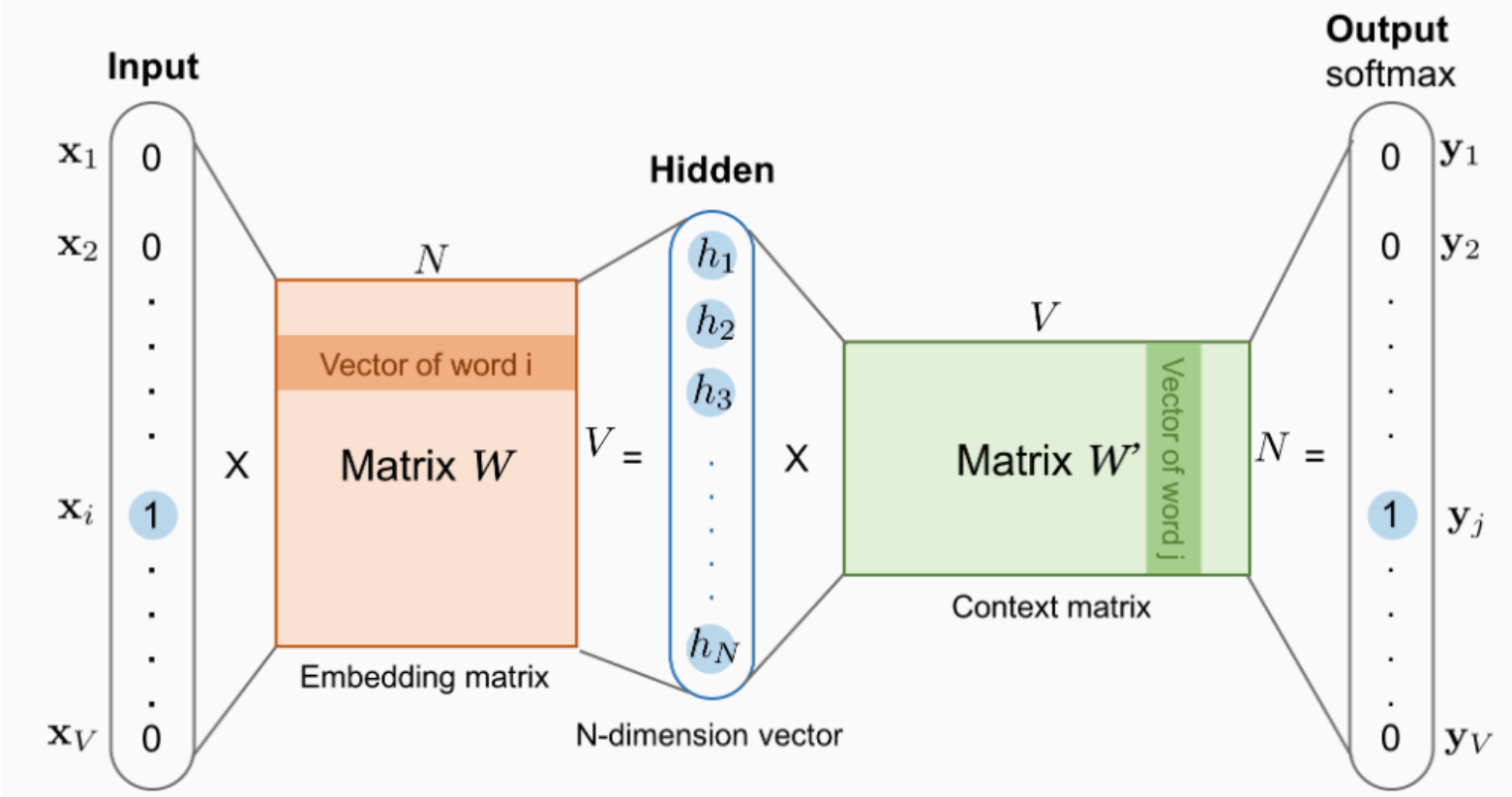
- $P(w_o \mid w_c)$ is the probability that softmax gives to the word w_o.
- The score of a word w is $\mathbf{u}_w \cdot \mathbf{e}_{w_c}$, where $\mathbf{u}_w$ is the column of $\mathbf{W}'$ of w.



Skip-gram, from the Embedding to Probabilities

So every word has **two vectors**:

- an embedding $\mathbf{e}_w$ as a center,
- and a **context vector** $\mathbf{u}_w$.



For a center word w_c, the output is a **probability distribution over** V: for each word w, $P(w \mid w_c)$ is the probability that w is the **context word**.

Skip-gram Objective

For a center token w_t, the model predicts the nearby tokens w_{t+j}. We **maximize the likelihood** of the observed center-context pairs:

$$\prod_{t=1}^{T} \prod_{-m \leq j \leq m,\ j \neq 0} P(w_{t+j} \mid w_t)$$

We learn the embeddings so that, given w_t, the tokens that **actually appear around it** in the corpus get a high probability.

Equivalently, we **minimize the negative log-likelihood**. In practice, the optimization is done with SGD, on center-context pairs sampled from the corpus.

CBOW

CBOW is the **mirror image** of skip-gram: given the context words, predict the center word.

1. each context word w_{t+j} is mapped to its embedding
2. their **average** is the input of the output layer:

$$\bar{\mathbf{e}} = \frac{1}{2m} \sum_{-m \le j \le m,\ j \neq 0} \mathbf{e}_{w_{t+j}}$$

3. the same output layer gives $P(w_t \mid \text{context}) = \text{softmax}(\bar{\mathbf{e}}\mathbf{W}')$, a distribution over V

The order of the context words is lost, hence *bag of words*. In practice, CBOW is **faster** to train, while skip-gram usually works better on **rare words** and small corpora.

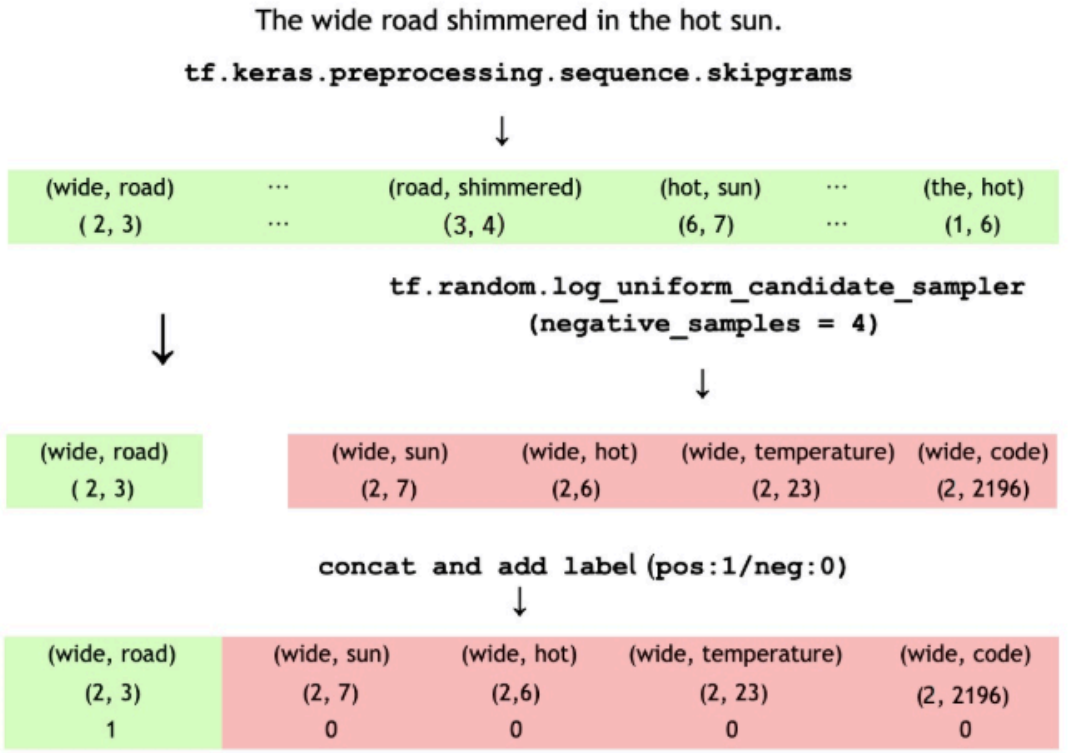
Negative Sampling, Positive and Negative Pairs

In practice, computing a softmax over the whole vocabulary needs a logit for **every word**, and updates $|V|$ vectors at each step: too slow.

Negative sampling turns it into a **binary classification**:

- **positive pair**: a real center-context pair
  - $(\text{wide}, \text{road}) \rightarrow 1$, a **high** score
- **negative pair**: a randomly sampled fake pair
  - $(\text{wide}, \text{code}) \rightarrow 0$, a **low** score

For each positive pair we sample a few negatives.



Negative Sampling, the Score

For each pair, we take the dot product of the **center word's input embedding** and the **candidate context word's output embedding**. This gives a single logit. A sigmoid converts it into the model's estimated probability that the pair comes from the observed text rather than the noise sampler.

Suppose $d = 2$ and these (illustrative) vectors:

$$\text{wide} \to [0.2, 0.7] \qquad \text{road} \to [0.8, 0.6] \qquad \text{code} \to [-0.6, 0.1]$$

Positive pair $(\text{wide}, \text{road}) \to 1$

$$[0.2, 0.7] \cdot [0.8, 0.6] = 0.2 \cdot 0.8 + 0.7 \cdot 0.6 = 0.58$$

$$\sigma(0.58) \approx 0.64$$

Negative pair $(\text{wide}, \text{code}) \to 0$

$$[0.2, 0.7] \cdot [-0.6, 0.1] = 0.2 \cdot (-0.6) + 0.7 \cdot 0.1 = -0.05$$

$$\sigma(-0.05) \approx 0.49$$

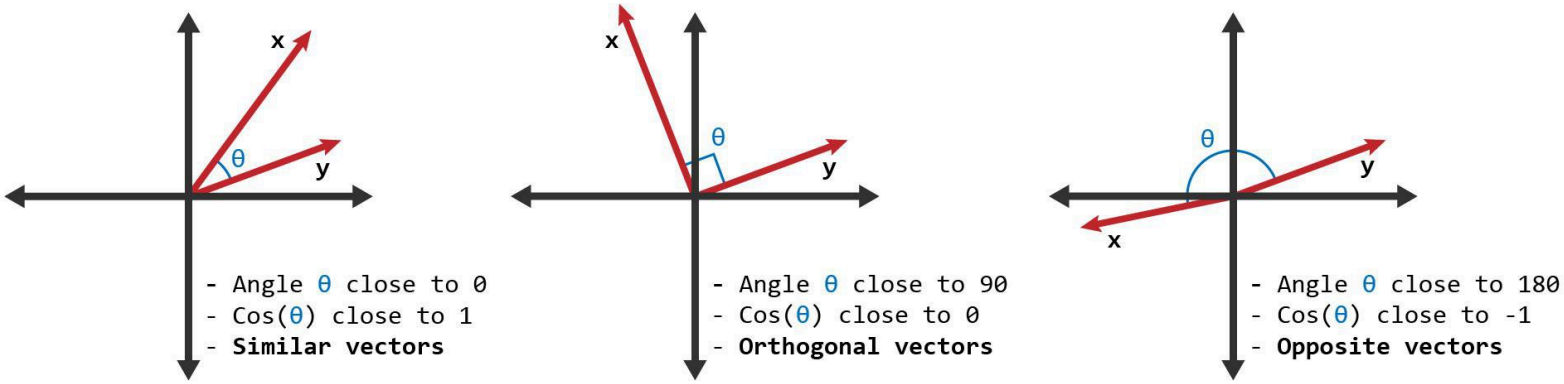
Training raises the first probability toward 1 and lowers the second toward 0.

After Training, Cosine Similarity

To **compare** two embeddings, we use the cosine similarity:

$$\text{cosine}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}$$

Unlike count vectors, embeddings have **negative** entries, so the full range $[-1, 1]$ is possible. Close to 1: similar direction; close to 0: unrelated; close to -1: opposite.

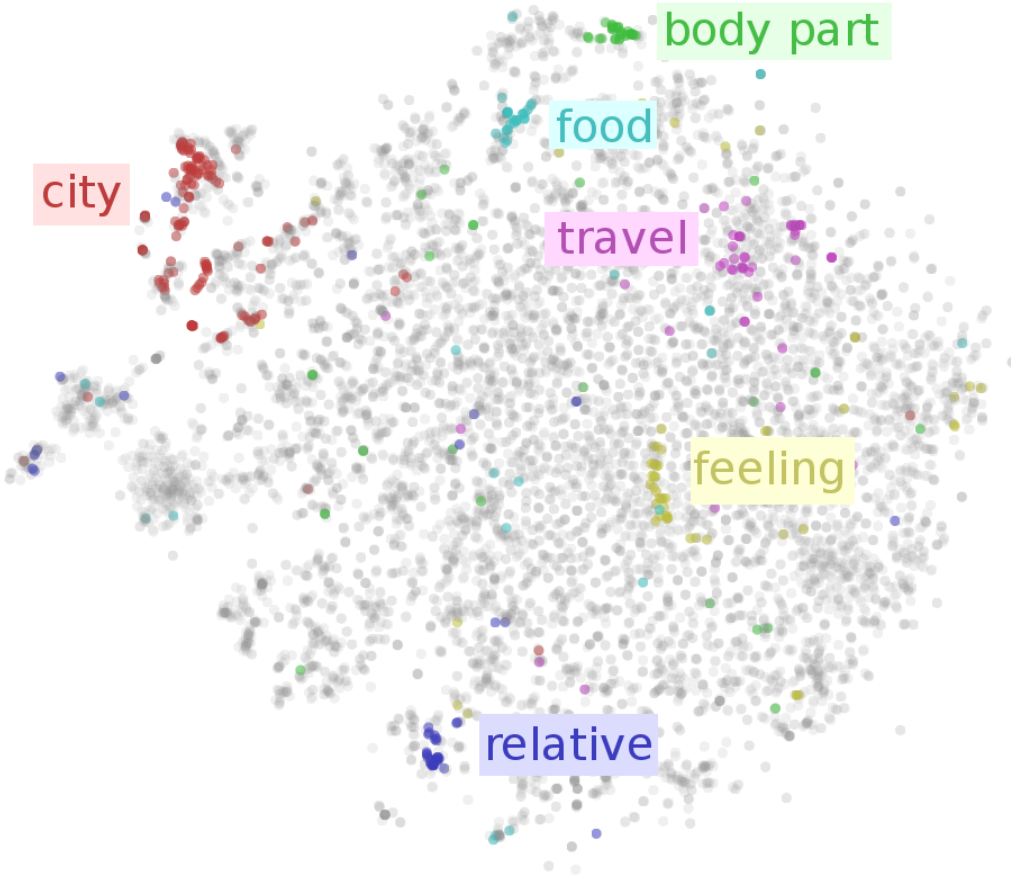


What Word2Vec Gives Us

After training, **each word has its vector representation.**

These vectors can be used for:

- **similarity search**: the nearest neighbors of a word
- visualization (PCA, T-SNE, UMAP...)
- clustering
- input features of neural networks
- initialization of downstream models



Beyond Word2Vec

Subwords, global statistics and documents

Limits of Word2Vec

- **out-of-vocabulary words**: one vector per word **seen in training**. A neologism, a technical term or a misspelling has **no vector**
- **no morphology**: *walk*, *walked*, *walking* are unrelated vectors, as are *eat* and *eaten*. Data is sparse in morphologically rich languages
- **static**: one vector per word, whatever the context. *Bank* is the same vector for a river and for money

FastText addresses the first two. Contextual models, later in the course, address the third.

FastText

FastText keeps the skip-gram model and adds **subword information**: every word is a bag of its **character n-grams** (typically 3 to 6), with boundary symbols.

goodbye with 3-grams: `<go` , `goo` , `ood` , `odb` , `dby` , `bye` , `ye>`

- every n-gram has its **own vector**
- the vector of a word is the **sum** of its own vector and those of its n-grams

A word **never seen** in training still gets a vector: the sum of its n-grams.

FastText

- **unseen words**: the vector is computed from the n-grams
- **morphology**: *walk*, *walked*, *walking* share many n-grams, so their vectors are close; so are *happy* and *happiness*
- **misspellings**: *accomodation* shares most of its n-grams with *accommodation*
- **pretrained** vectors are available for many languages, which helps low-resource and morphologically rich ones

The price is more vectors to store: one per word and one per n-gram.

GloVe

GloVe (Global Vectors) is another method to learn **static** embeddings. It differs from Word2Vec in **how** it learns them.

Word2Vec

- reads the corpus **one window at a time**
- the dot product becomes a **probability** (softmax): given *ice*, predict *cold*
- a frequent pair is seen **many times**, once per occurrence

GloVe

- first **counts** the whole corpus once: the word-context matrix
- the dot product has to **match the count** (in logarithm): a regression on the word-context matrix
- each pair is used **once**, and very frequent pairs are down-weighted

In both, words that appear **near the same words** (*coffee* and *tea*, both near *cup*, *hot*, *drink*) end up with **similar vectors**.

Pretrained GloVe vectors, trained on large corpora, are widely available.

From Words to Documents, the Average

Many tasks need a vector for a **sentence or a document**, not only a word. The most naive solution: **average** the embeddings of its words.

$$\mathbf{e}_{\text{doc}} = \frac{1}{T} \sum_{t=1}^{T} \mathbf{e}_{w_t}$$

- a **fixed-size** vector in $\mathbb{R}^d$, whatever the length of the text
- the same as multiplying the bag of words with the embedding matrix
- a surprisingly strong baseline, and the input of a classifier

But it **forgets the order**: *dog bites man* and *man bites dog* get the same vector, and *not good* is only the average of *not* and *good*. Every word weighs the same, the useful ones and stopwords.

Doc2Vec

Instead of averaging, **Doc2Vec** (Le & Mikolov, 2014) extends Word2Vec and learns the document vector directly:

- every document d gets its own vector $\mathbf{e}_d$, **learned like a word vector**
- it is used as an **extra context** in the prediction task
- two variants: **PV-DM** (like CBOW) and **PV-DBOW** (like skip-gram)

After training, $\mathbf{e}_d$ can be used to compare documents with the cosine, to cluster them, or as **features** of a classifier.

It remains a static model: it does not capture context or polysemy.

Static Embeddings and Their Limit

All the models of this lecture give **one vector per word type**, whatever the sentence:

- in *"I sat on the river **bank**"* and *"the **bank** raised its rates"*, *bank* has the **same** vector
- a polysemous word, one with several senses, gets **one vector that mixes them all**, dominated by the most frequent one.
  - In GloVe, trained on Wikipedia and news, the nearest neighbors of *bank* are *banks*, *securities*, *banking*, *investment*, *exchange*, *financial*, *credit*, *lender*: all about **money**. *River* is not even among the first 1,000
- the vector knows nothing of the **order** of the words

A model that reads a sequence should produce vectors that **depend on the context**.

Evaluating Embeddings

Do the vectors capture meaning?

Evaluation, Word Similarity

Intrinsic evaluation. Compare the similarity of the model with the judgments of **humans**:

1. for each word pair of a benchmark, compute the **cosine** of their embeddings
2. compute the **Spearman correlation** ρ with the human ratings
3. a higher correlation means a better semantic structure

- **WordSim-353** (Finkelstein et al., 2002): 353 noun pairs rated from 0 to 10, e.g. $\text{sim}(plane, car) = 5.77$
- **SimLex-999** (Hill et al., 2015): 999 pairs annotated for **true similarity**, not mere association

Coffee and *cup* are associated but not similar: SimLex rates them low.

Evaluation, Analogies

If embeddings encode **relations**, relations should behave **linearly**:

$$\mathbf{e}_{\text{king}} - \mathbf{e}_{\text{man}} + \mathbf{e}_{\text{woman}} \approx \mathbf{e}_{\text{queen}}$$

To answer *"Paris is to France as Rome is to ?"*:

1. compute $\mathbf{e}_{\text{France}} - \mathbf{e}_{\text{Paris}} + \mathbf{e}_{\text{Rome}}$
2. find the **nearest** embedding, **excluding** the three input words
3. the top word should complete the analogy: *Italy*

Semantic relations (*Paris*, *France*) and syntactic ones (*cat*, *cats*, *mouse*, *mice*) are tested. It works for **some** relations and fails on others.

What the Corpus Puts in the Vectors

Embeddings reflect the **corpus** they are trained on and its **biases**.

- the **nearest neighbors** of a word change with the corpus: the same method trained on Wikipedia or on a collection of books gives different neighbors for the same word
- the analogy method shows **stereotypes**: in embeddings trained on news, the closest word to *computer programmer* $-$ *man* $+$ *woman* is **homemaker**
- associations measured in embeddings **mirror** the ones found in human tests

Consequences:

- embeddings are **not neutral**: a model built on them inherits and amplifies biases
- removing them is **partial**: it needs an evaluation of the downstream system, not only of the vectors

Sources

Textbooks

- Jurafsky & Martin, *Speech and Language Processing*, 3rd ed. (Jan 2026 draft), ch. 5, *Embeddings*: lexical semantics and vector semantics (5.1 to 5.3), cosine (5.4), Word2Vec with negative sampling, FastText and GloVe (5.5), visualization (5.6), semantic properties and analogies (5.7), bias (5.8), evaluating vector models (5.9)

Papers

- Bengio et al. (2003), *A Neural Probabilistic Language Model*.
- Mikolov et al. (2013), *Efficient Estimation of Word Representations in Vector Space*.
- Levy & Goldberg (2014), *Neural Word Embedding as Implicit Matrix Factorization*.
- Pennington et al. (2014), *GloVe*.
- Bojanowski et al. (2017), *Enriching Word Vectors with Subword Information*.
- Le & Mikolov (2014), *Distributed Representations of Sentences and Documents*.
- Bolukbasi et al. (2016), Caliskan et al. (2017)