CSE2525 · DATA MINING · 2026–27

Embeddings

From sparse hand-crafted features to dense, semantic vectors — word2vec for text and random-walk embeddings (DeepWalk, node2vec) for graphs.

A. Anand · lecture 8 study deck

PART I

From raw data to vectors

Every learning algorithm eats vectors. How do we turn tables, text and graphs into them — and why the obvious way is a bad one.

THE ML PIPELINE NEEDS NUMBERS

How do we vectorize the input?

THE CLASSIC ML PIPELINE

raw data → **compute features** → learning algorithm → model → deploy on test → output. Everything downstream assumes each instance is already a **vector**.

— But raw data comes in many shapes: tabular, text, graph, …

— The first — often overlooked — step is choosing how each instance becomes a point in some feature space.

— This choice decides whether distances, similarity and learning will actually work.

ONE RECIPE PER DATA TYPE

Instances as vectors

Tabular

- Each **row** = an instance vector
- Each **column** = a dimension
- Values already numeric

Text

- Each **word** = a dimension
- Bag of words: value = word frequency
- Vocabulary-sized vectors

Graph

- Each **vertex** = a dimension
- Dim = 1 if an edge to that vertex
- A vertex = one row of the adjacency matrix

All three give the same thing: a long, mostly-zero vector in a vocabulary- or vertex-sized space.

TYPICAL HAND-CRAFTED FEATURE SPACE

What is the problem?

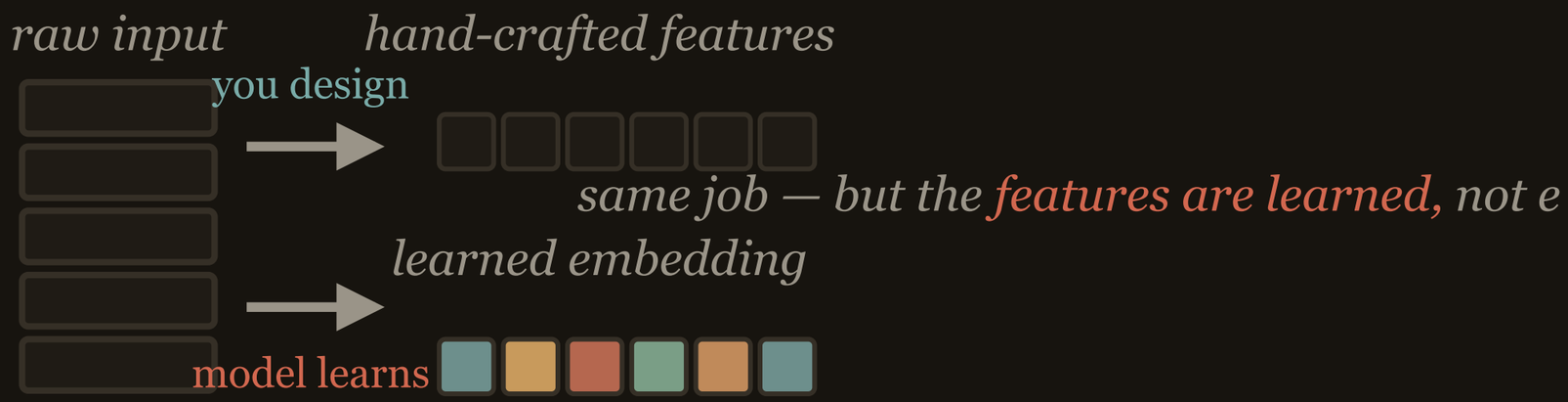
THESE REPRESENTATIONS ARE...

Sparse — almost all entries are zero · Large — one dimension per word / vertex · Non-semantic — dimensions carry no meaning · Hand-crafted — you choose the features by hand.

We want the opposite: dense, smaller, semantic, and automatically extracted.

THE GOAL OF THIS LECTURE

Automatic feature extraction



Raw text / tables / graphs first map to a sparse, large, hand-crafted space; embeddings add a second step to a dense, small, semantic space that is learned, not designed.

WHY ONE-HOT / BAG-OF-WORDS HURTS

Issues with sparse representations

- Distances are misleading — in very high dimensions everything is roughly equidistant.
- Vocabulary mismatch — *hotel* and *motel* share no dimension, so they look unrelated.
- Large space requirements — vectors as wide as the whole vocabulary.
- These are local (1-hot) representations: each word is its own isolated axis.

$$(w^{hotel})^T w^{motel} = (w^{hotel})^T w^{cat} = 0$$

Any two distinct one-hot words are orthogonal — the dot product is 0 whether the words are near-synonyms or unrelated.

ONE-HOT VS. DISTRIBUTED

Local vs. distributed representations

one-hot · 50,000 dims, a single 1



× E (lookup)

embedding · ~300 dims, every entry meaningful



Local (1-hot): one active dimension per word — no shared structure. Distributed: meaning is spread across many features (barks, fruit, has-tail, …), so *banana* and *mango* now overlap.

SAME DISEASE FOR GRAPHS

Sparsity in adjacency matrices

ADJACENCY ROWS ARE SPARSE TOO

A vertex's row in the adjacency matrix is sparse, large, non-semantic and hand-crafted — the number of edges is far smaller than the number of vertices (`#edges ≪ #vertices`).

So both text and graphs need the same cure: learn a dense, low-dimensional, semantic embedding.

QUESTION

With **one-hot** vectors, every pair of distinct words is **equally dissimilar** — “cat” is as far from “dog” as from “democracy”. How could a representation capture that some words are **related** — learned from text alone?

We want nearby vectors for words that behave alike.

Think — then → for the answer.

PART II

Word embeddings

Learn a vector per word so that words used in similar contexts land close together — word2vec.

THE DEFINITION

What is an embedding?

DEFINITION

An **embedding** is a **learned**, **dense**, low-dimensional vector that represents an object — a word, a node, a product — so that **similar objects get nearby vectors**.

— Learned, not hand-designed — the values come from training.

— Dense (~100–300 numbers), unlike a sparse one-hot of size |vocabulary|.

— Geometry carries meaning: distance / direction in the space stands for relationships.

THE OBJECTIVE IN ONE PICTURE

What we want from an embedding

different company → far apart



same company → close



Encode each word so that similarity in the embedding space (e.g. the dot product $z_u{}^T z_v$) approximates similarity in its textual context — many features → a few *semantic* ones.

THE DISTRIBUTIONAL HYPOTHESIS

What is “context”?

DISTRIBUTIONAL SIMILARITY

“You shall know a word by **the company it keeps**.” — J. R. Firth, 1957. Words that occur in **similar contexts** have similar meanings.

— *“We took two hours to reach the top of the hill.”*

— *“The top of the hill had the best view.”*

— *“This purple top will go well with my blue jeans.”*

The surrounding words tell you which sense of top is meant — context *is* the signal we learn from.

WORD2VEC · KEY IDEA

A word is known by the company it keeps

THE DUCK TEST

“If it walks like a duck and quacks like a duck, it must be a duck.” The blanks share a **context**: `like a __ and quacks` · `like a __ it must` · `be a __`.

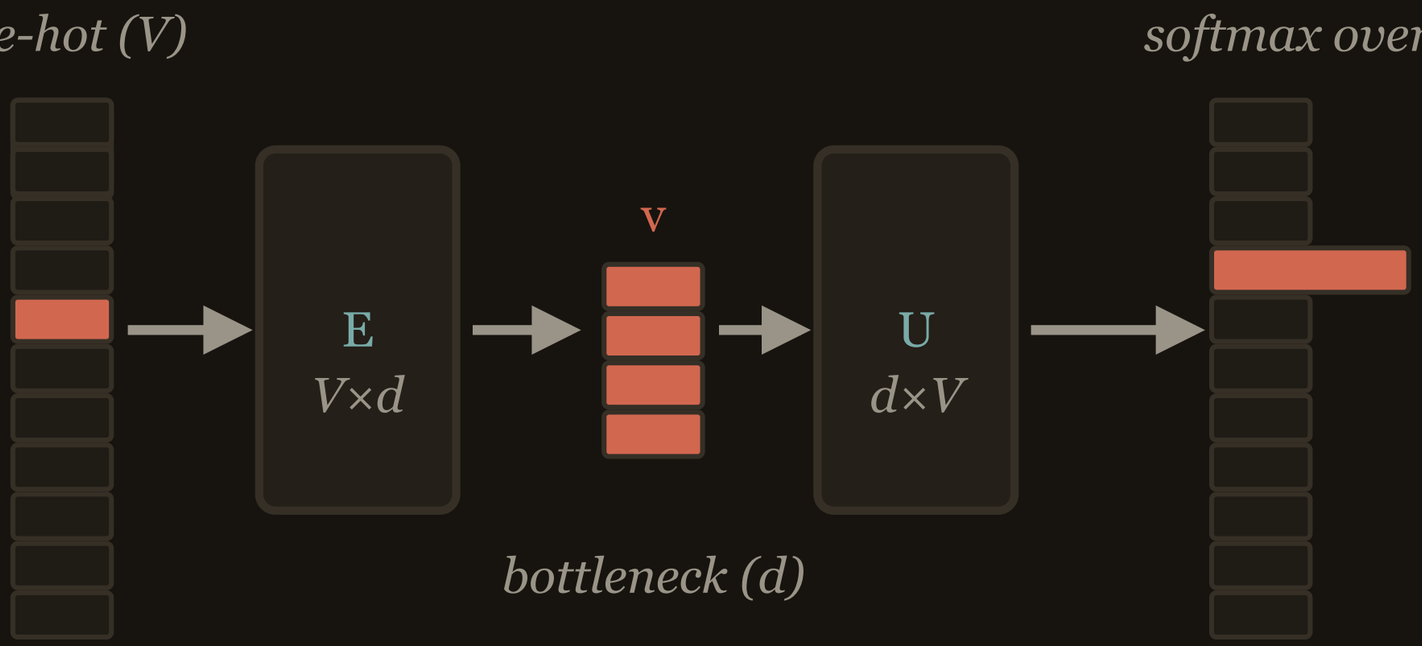
— Form ⟨word, context⟩ pairs by sliding a window over the text.

— Learn vectors so a word predicts the words around it.

— Q: how do we turn “predict the context” into something trainable?

THE ENCODER VIEW

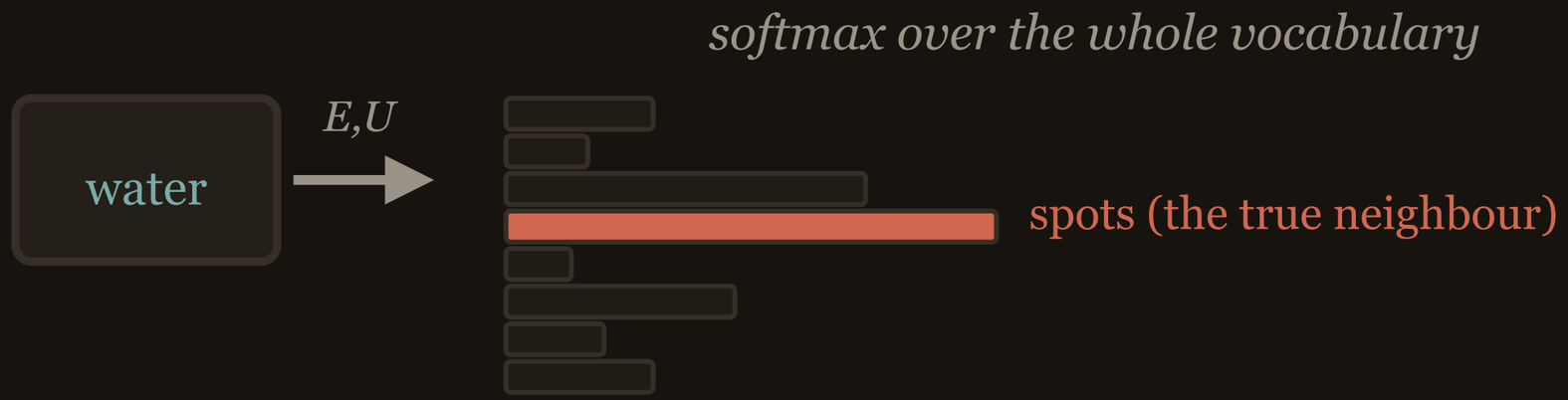
The word2vec model



Context vector × W_{in} → a small hidden representation × W_{out} → predicted center word. Input/output are vocabulary-sized (millions); the hidden layer is tiny (≈ 300).

READING IT AS CLASSIFICATION

Word2vec as multi-class classification



one center word → a probability for every vocabulary word

A shallow net: input layer $W_{V\times N}$ → hidden h → output $W'_{N\times V}$. Given a word we predict its context; each vocabulary word is one output class.

HOW THE PARAMETERS ARE LEARNED

Training with gradient descent

SOFTMAX OVER THE VOCABULARY

The output layer turns scores into probabilities with a **softmax**, trained against the true context word with **cross-entropy loss**.

$$\hat{y}_{ij} = softmax(w_j \cdot x_i) = \frac{e^{w_j \cdot x_i}}{\sum_{\ell=1}^{V} e^{w_\ell \cdot x_i}}$$

Scores = dot products of the center word with every vocabulary word; gradient descent nudges the vectors to raise the true context word's probability.

MAKING SOFTMAX AFFORDABLE · NOT EXAM MATERIAL

Negative sampling

WHY THE DENOMINATOR HURTS

The softmax denominator sums over all V words — with V in the millions, every update is far too expensive.

Negative sampling: instead of all N words, update only the true context word plus a few C random “negative” words. Cheap, and works well in practice.

Marked in the slides as not exam material — but it is exactly what makes word2vec (and node2vec) trainable at scale.

QUESTION

A trained word2vec model has learned good vectors. **Where do those vectors actually live** — which numbers in the model are the parameters you updated?

The model is "just" two matrices.

Think — then → for the answer.

THE ANSWER TO “WHERE ARE THE EMBEDDINGS?”

Parameters *are* the embeddings

embedding matrix E (V × d)



← row = “cat”

ıbers in these rows are the parameters θ — gradient descent updates them

We never use the predictions — we keep the weights. The hidden-layer vector for a word *is* its word embedding; the task was just a pretext to learn good vectors.

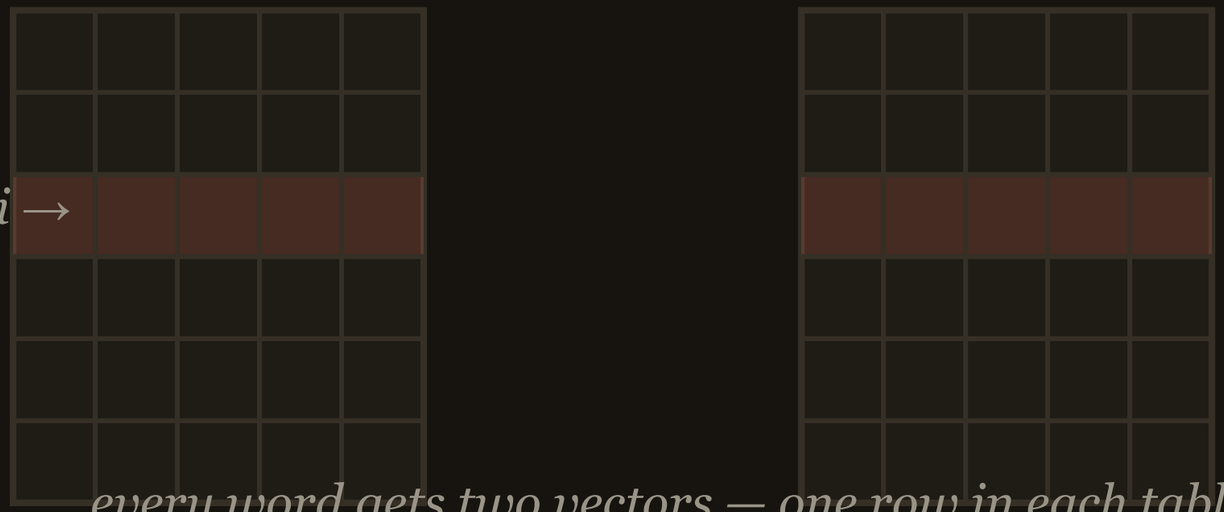
INPUT VS. OUTPUT EMBEDDINGS

Word2vec learns two matrices

E — input / center

U — output / context

word i →



every word gets two vectors — one row in each table

Both W_{in} (the Embedding matrix) and W_{out} (the Context matrix) are vocab × embedding-size. The input embeddings are usually taken as the final word vectors.

WHAT THE VECTORS CAPTURE

The geometry of embeddings



king − man + woman ≈ queen — a direction the model was never told

ANALOGIES BECOME ARITHMETIC

Directions in the space encode relationships, so analogies can be solved by adding and subtracting vectors.

vec(king) − vec(man) + vec(woman) ≈ vec(queen)

— The *man → woman* direction is roughly the same as *king → queen*.

— Good embeddings are often pre-computed and freely available (e.g. via gensim).

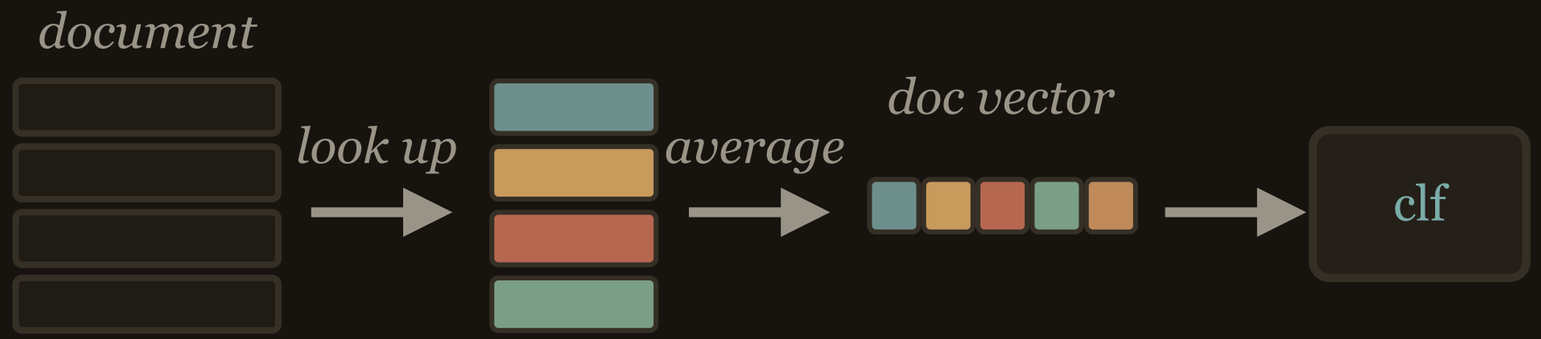
PART III

Applying embeddings to supervised tasks

Embeddings are learned without labels — then plugged into a supervised model as features.

DOCUMENTS → CLASSES

Text classification



a whole document becomes one vector — the average of its word vectors

Embed each document, then run the ordinary pipeline — train data → learning algorithm → model → predict the label (Physics · Chemistry · Maths) of a new document.

FROM WORDS TO DOCUMENTS

Representing a whole document

BAG OF WORD-VECTORS

Each word is a vector. Represent a document as the **average** (or **sum**) of the vectors of the words it contains.

- Simple, fast, and surprisingly strong as a baseline document representation.
- Order is ignored — it is still a *bag* — but each dimension is now semantic, not a raw word count.
- Libraries such as gensim provide word2vec and these aggregations out of the box.

PART IV

Graph embeddings

Reuse the word2vec idea on networks: random walks play the role of sentences — DeepWalk and node2vec.

THE OBJECTIVE, FOR NODES

What we want from node embeddings



nodes close in the graph should get nearby vectors

Encode each node so that similarity in the embedding space (the dot product $z_u{}^T z_v$) approximates similarity in the original network. Same recipe as words — different notion of context.

QUESTION

Word2vec needs a word's **context window**. A graph has **no sentences** and no left/right order. How could we reuse the very same method to embed **nodes**?

What plays the role of "a sentence" on a graph?

Think — then → for the answer.

NODE EMBEDDINGS · MOTIVATION FROM WORD2VEC

Context on a graph = random walks

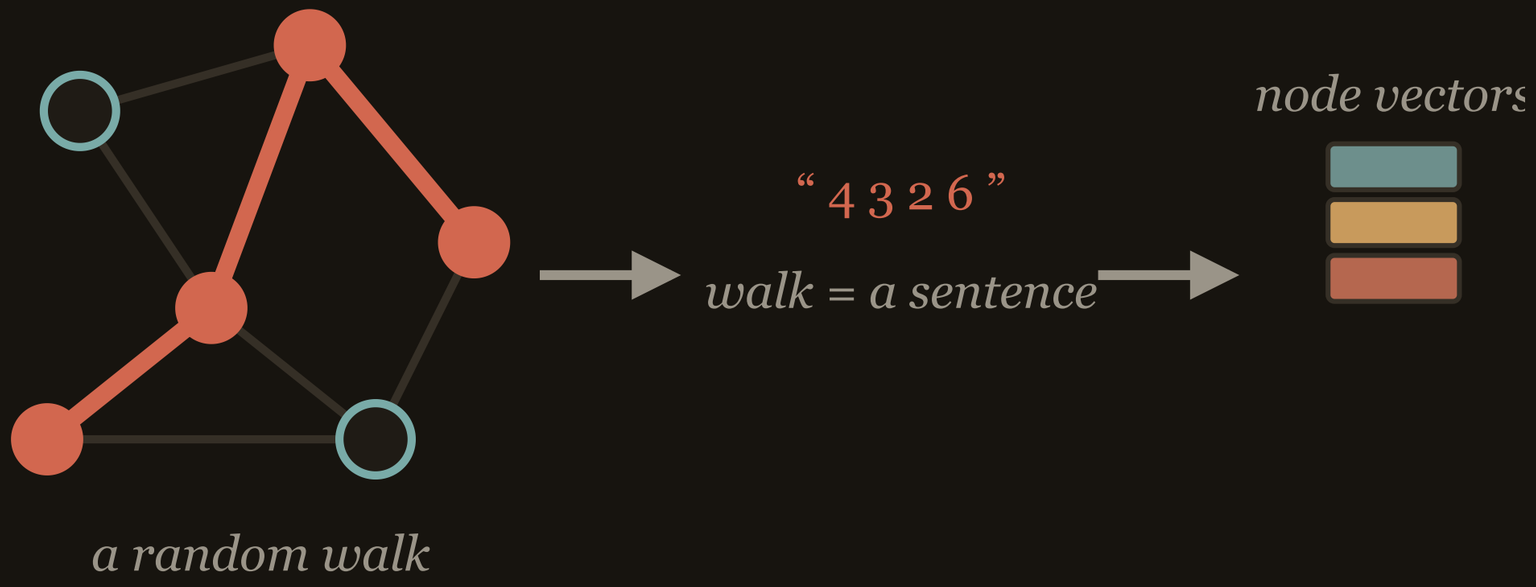
THE ANALOGY

In text, a sliding window gives each word its context. On a graph there is no linear order — so we **generate sequences by walking** the network.

- Random walks turn a graph into many short “sentences” of nodes.
- Nodes that co-occur on walks play the role of words that co-occur in context.
- Then run exactly the same skip-gram training on those sequences.

THE RANDOM-WALK OBJECTIVE

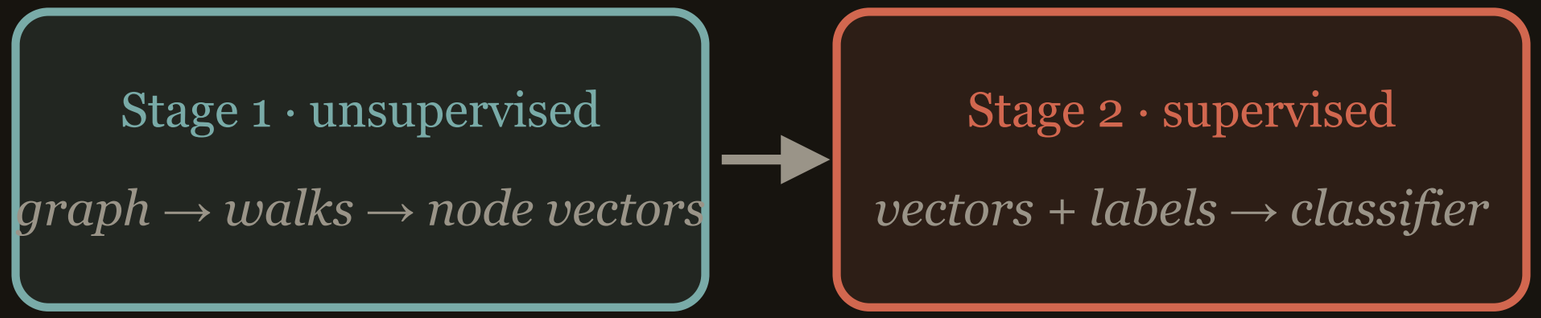
Random-walk embeddings



Make $z_u^T z_v$ approximate the probability that *u* and *v* co-occur on a random walk under strategy R: estimate $P_R(v \mid u)$, then optimize the vectors to encode it.

USED IN THE LAB

Putting it together: two stages



1 · Create training data — sample random walks (fixed-window or biased). 2 · Train / optimize — given an input node, predict its walk neighbours with skip-gram + negative sampling, then read off the embeddings.

FIXED-WINDOW WALKS

DeepWalk: unbiased fixed walks

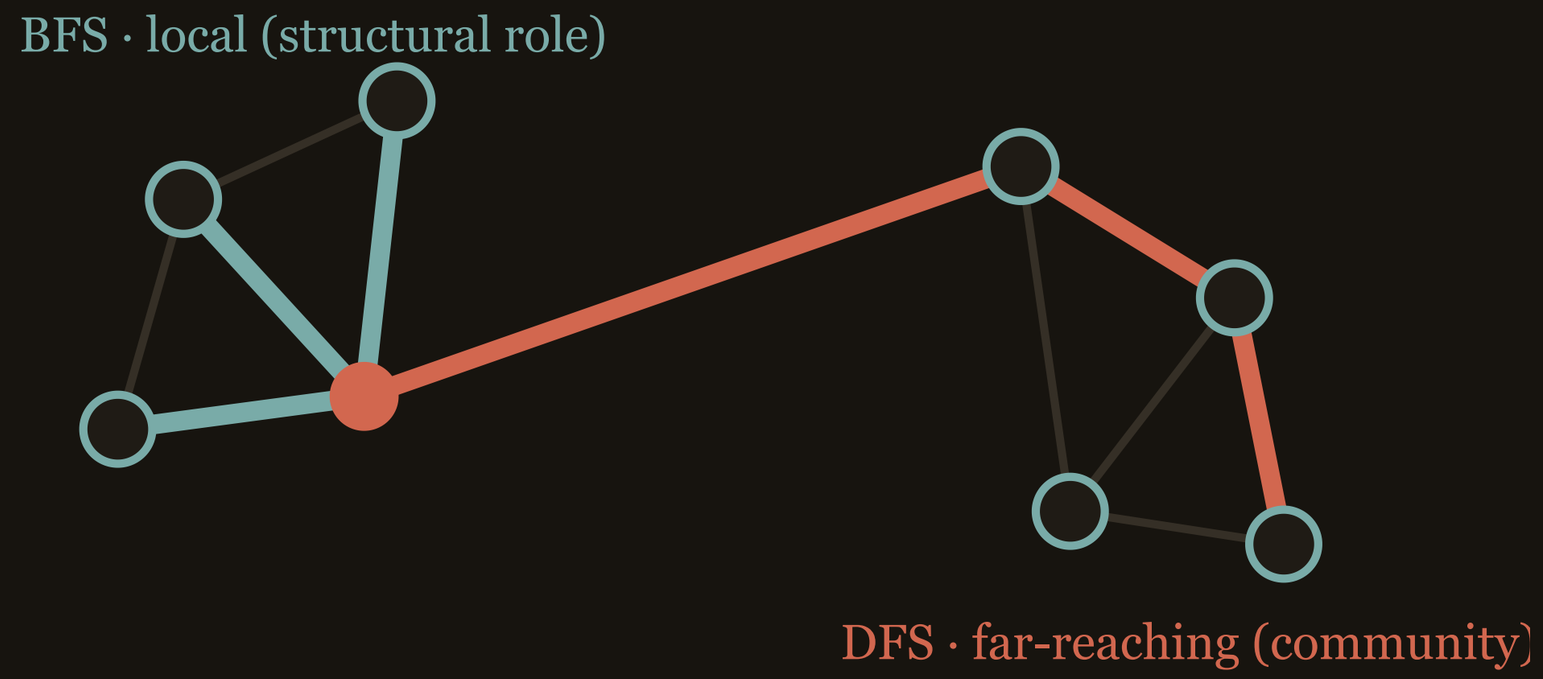


DeepWalk: take uniform fixed-length walks from every node → "sentences"

Start at a random node, move to each neighbour with equal probability, and stop after a fixed length.
The same node may be sampled several times in one walk.

BIASED WALKS · NODE2VEC

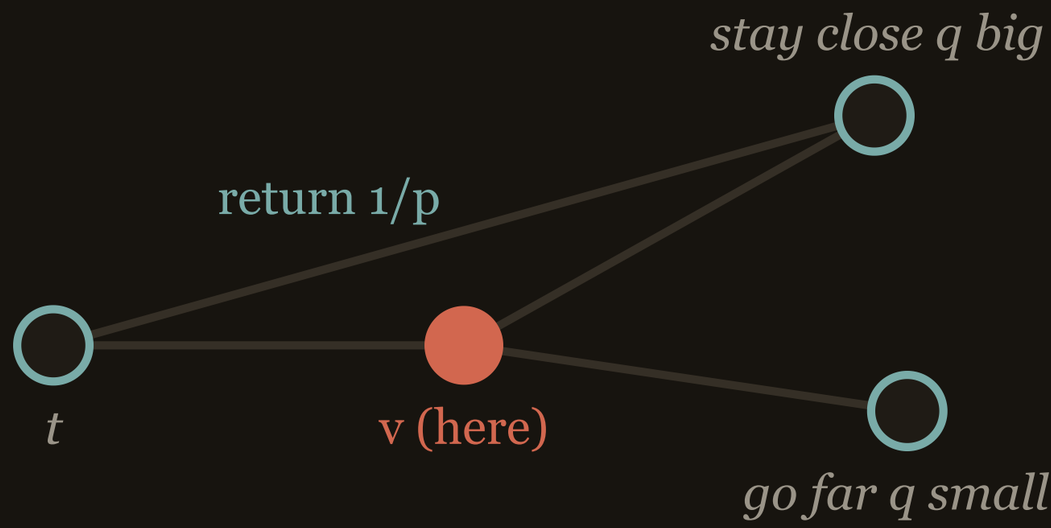
node2vec: BFS vs. DFS neighbourhoods



Two ways to define a neighbourhood $N_R(u)$: BFS stays close — a local, microscopic view of structure; DFS wanders far — a global, macroscopic view. node2vec interpolates between them.

THE RETURN AND IN-OUT PARAMETERS

node2vec: biasing the walk with p and q



p, q bias the walk between BFS-like and DFS-like exploration

From the current node: go back (weight 1/p), stay at the same distance (weight 1), or move outward (weight 1/q). p controls revisiting; q trades BFS-like vs DFS-like exploration.

THE LOSS, AND WHY IT IS HARD

Random-walk optimization

$$L = \sum_{u \in V} \sum_{v \in N_R(u)} -\log \frac{exp(z_u{}^T z_v)}{\sum_{n \in V} exp(z_u{}^T z_n)}$$

— Sum over all start nodes u, and over each node v seen on walks from u.

— The fraction is the predicted probability that u and v co-occur on a walk.

— Done naïvely the nested sum is $O(|V|^2)$ — too expensive.

$$L \approx log\ \sigma(z_u{}^T z_v) - \sum_{i=1}^{k} log\ \sigma(z_u{}^T z_{n_i}),\ n_i \sim P_V$$

Negative sampling again: normalise against k random negatives instead of all nodes (σ = sigmoid). not exam material

EMBEDDINGS FIRST, LABELS LATER

Downstream: node classification



ed every node, then a classifier labels the unknown from its neighbours' ve

Learn $f : u \to \mathbb{R}^d$ without labels, then train a classifier on a subset of labelled nodes to predict the rest (? → Physics / Chemistry / Maths).

LECTURE 8 · WRAP-UP

Summary

TAKE AWAY

Hand-crafted features are sparse, large, non-semantic. Embeddings replace them with dense, small, semantic vectors where similarity ≈ dot product.

word2vec learns word vectors by predicting context over ⟨word, context⟩ pairs; the hidden-layer weights are the embeddings, made trainable by negative sampling. Analogies become vector arithmetic.

Graph embeddings reuse the idea: random walks are sentences. DeepWalk uses fixed unbiased walks; node2vec biases them with p, q to blend BFS (local) and DFS (global). Same skip-gram objective.

WHERE TO READ MORE

References

— Mikolov et al., *Distributed Representations of Words and Phrases and their Compositionality*, NeurIPS 2013.

— Perozzi, Al-Rfou & Skiena, *DeepWalk: Online Learning of Social Representations*, KDD 2014.

— Grover & Leskovec, *node2vec: Scalable Feature Learning for Networks*, KDD 2016.

— Hamilton, *Graph Representation Learning* — grl_book. Code: gensim. Blog: *The Illustrated Word2vec* (jalammar).