Word Embeddings & Vector Spaces
Multimodal Computational Methods in Political Science

Tamara Grechanaya [1]

[1]LMU Munich — Computational Social Science MA Program

April, 2026

Course Roadmap

1. Text as Data: Foundations & Preprocessing
2. Classical Text Classification
3. **Word Embeddings & Vector Spaces** ← *Today*
4. Document Representations & Topic Models
5. Neural Networks & Sequence Models
6. Attention & the Transformer Architecture
7. Transfer Learning & Fine-Tuning BERT

So far we've represented words as **counts**. Today we represent words as **vectors in a continuous space** that capture meaning. This is the foundation for everything that follows in the course.

Today's Outline

Part I: From Sparse to Dense

- Limitations of bag-of-words
- One-hot encoding and its problems
- The distributional hypothesis
- Sparse vs. dense representations

Part II: Vector Spaces & Similarity

- Words as vectors
- Co-occurrence matrices
- Cosine similarity
- Geometric intuition

Part III: Word2Vec & GloVe

- Skip-gram and CBOW intuition
- How embeddings are learned
- GloVe: matrix factorization perspective
- Word analogies and semantic structure

Part IV: Politics & Practice

- Embeddings encode bias
- Measuring ideology with embeddings
- Visualizing with PCA
- Practical: political embeddings in R

Table of Contents

Recap: How We've Represented Text So Far

Lectures 1–2: We've used **sparse, count-based** representations.

Bag-of-Words representation

"The government must act on climate change"

$$\mathbf{x} = [\underbrace{0}_{\text{about}}, \underbrace{0}_{\text{after}}, \ldots, \underbrace{1}_{\text{climate}}, \ldots, \underbrace{1}_{\text{government}}, \ldots, \underbrace{0}_{\text{zero}}]$$

A vector of length $|V| = 50{,}000$ where almost every entry is 0.

This worked well for classification. But it has fundamental limitations that prevent us from doing more sophisticated things with text.

Today we ask: Can we find a representation where *words themselves* have meaning — where "climate" is somehow "close to" "environment" in the representation?

Three Limitations of Bag-of-Words

1. **No notion of word similarity.**
   In a BoW vector, "climate" and "environment" are as different from each other as "climate" and "chancellor". Each word is just an index in the vocabulary.
2. **High dimensionality and sparsity.**
   $|V| = 50{,}000$ dimensions for a vocabulary that is mostly zeros in any given document. This is computationally wasteful and statistically problematic — you need huge amounts of data to estimate models with so many parameters.
3. **No generalization across words.**
   If your training data has the word "climate catastrophe" but your test data has "climate crisis", a BoW classifier sees them as completely unrelated. Knowledge learned about one word does not transfer to similar words.

Key Concept

What we want: a representation in which **semantically similar words have similar vectors**. "climate" should be close to "environment", "king" close to "queen", "Berlin" close to "Paris".

One-Hot Encoding: The Naive Starting Point

One-hot encoding: represent each word as a vector of length $|V|$ with a single 1 at the word's index and 0 everywhere else.

A toy vocabulary

$$V = \{\text{climate}, \text{politics}, \text{Berlin}, \text{Paris}\},\ |V| = 4$$

$$\begin{aligned} \text{climate} &= [1,0,0,0] \\ \text{politics} &= [0,1,0,0] \\ \text{Berlin} &= [0,0,1,0] \\ \text{Paris} &= [0,0,0,1] \end{aligned}$$

The fatal flaw: *Every pair of words is equidistant.* The Euclidean distance between any two one-hot vectors is $\sqrt{2}$. The dot product between any two different one-hot vectors is 0. So according to this representation:

$$\text{distance}(\text{Berlin}, \text{Paris}) = \text{distance}(\text{Berlin}, \text{climate})$$

This is obviously wrong — Berlin and Paris are both capitals; they should be "closer" than Berlin and climate. We need a better idea.

The Distributional Hypothesis

The Foundational Idea (Harris 1954, Firth 1957)

"You shall know a word by the company it keeps." – J.R. Firth Words that appear in similar contexts tend to have similar meanings.

The intuition:

Example

Suppose you've never seen the word *Tesgüino* before. But you read these sentences:

- *A bottle of* **tesgüino** *is on the table.*
- *Everybody likes* **tesgüino**.
- **Tesgüino** *makes you drunk.*
- *We make* **tesgüino** *out of corn.*

You can infer: *tesgüino* is some kind of **alcoholic beverage** made from corn – without anyone telling you. You inferred this from the words that appeared *around* it.

This is the principle that underlies *every* modern word representation, from Word2Vec (2013) to BERT (2018) to today's largest language models. The contexts in which a word appears define its meaning.

Distributional Hypothesis: A Political Example

Inferring meaning from context

You see these sentences:

- The **climate crisis** threatens our future.
- We must tackle the **climate crisis** urgently.
- The scientific evidence on the **climate crisis** is clear.
- International cooperation against the **climate crisis**.

And separately:

- The **climate hysteria** of the left.
- This **climate hysteria** is destroying our economy.
- The exaggerated **climate hysteria** in the mainstream.

Both phrases refer to climate concerns, but their *contexts* encode their political valence. "Climate crisis" appears with words like "tackle", "evidence", "cooperation". "Climate hysteria" appears with "exaggerated", "destroying", "the left".
A word embedding trained on political text will place these two phrases in different regions of the vector space — because their contextual neighborhoods differ.

Sparse vs. Dense Representations

Sparse (one-hot, BoW)

$|V| = 50{,}000$ dimensions

$$\text{climate} = [0, 0, \underbrace{1}_{\text{idx}=2341}, 0, \ldots, 0]$$

Properties:

- Dimensions are interpretable (each = a word)
- Mostly zeros (sparse)
- Very high-dimensional
- Each word is its own dimension
- No notion of similarity

Dense (embedding)

$d = 300$ dimensions

$$\text{climate} = [0.21, -0.84, 0.13, \ldots, 0.05]$$

Properties:

- Dimensions are *not* interpretable individually
- All entries are non-zero (dense)
- Much lower dimensional
- Words share dimensions
- Similar words have similar vectors!

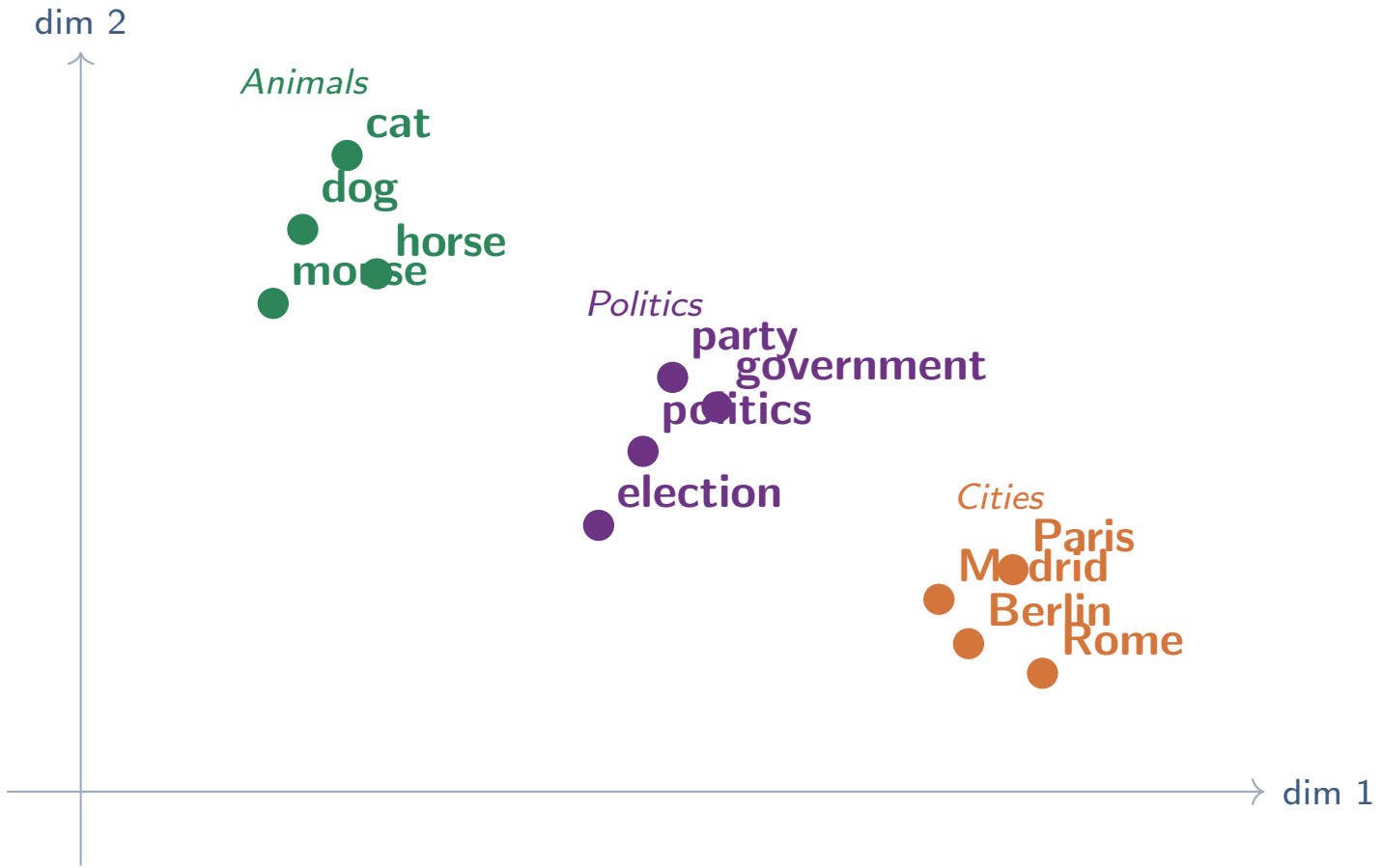
The trade-off

We give up *individual* interpretability of dimensions in exchange for *geometric* interpretability of relationships. The cost is real, but the benefit — being able to measure similarity between words — is enormous.

Table of Contents

Words as Points in Space

Imagine a 2D plane where each word is a point. Similar words cluster together.



Real word embeddings work the same way, but in $d = 300$ dimensions instead of 2. Words that mean similar things end up in similar regions of the space.

Building Word Vectors from Co-occurrence

The simplest way to build word vectors: count which words appear near each other. **Procedure:**

1. Define a **context window** (e.g., ± 2 words around the target word)
2. For each target word, count how often each other word appears within its window
3. This gives a **co-occurrence matrix** where rows = target words, columns = context words

Toy corpus

- "The government acted decisively."
- "The opposition criticized government policy."
- "Decisive government climate action is needed."
- "The government discussed the climate crisis."

With a window of ± 2 words, we count how often each pair of words co-occurs.

Note: for readability, we only show 6 content words in the matrix below. In practice, the matrix would include *all* words in the vocabulary (including "the", "acted", "crisis", "policy", etc.).

The Co-occurrence Matrix

	government	opposition	climate	action	criticized	discussed
government	0	1	1	1	1	1
opposition	1	0	0	0	1	0
climate	1	0	0	1	0	1
action	1	0	1	0	0	0
criticized	1	1	0	0	0	0
discussed	1	0	1	0	0	0

Each row is a word vector!

- government $= [0, 1, 1, 1, 1, 1]$
- action $= [1, 0, 1, 0, 0, 0]$
- discussed $= [1, 0, 1, 0, 0, 0]$

Notice: `action` and `discussed` have *identical* vectors in this tiny corpus. Both occur near `government` and `climate` but not the others. This is the distributional hypothesis at work — words with similar contexts get similar vectors.

Co-occurrence Matrices: Practical Issues

In practice, the simple co-occurrence matrix has problems:

1. **It's huge.** A vocabulary of 50,000 words gives a $50{,}000 \times 50{,}000$ matrix — 2.5 billion entries.
2. **It's still sparse.** Most word pairs never co-occur, so most entries are still zero.
3. **Frequent words dominate.** Words like "the" and "is" co-occur with everything, drowning out the meaningful signal. Solutions: use **Pointwise Mutual Information (PMI)** instead of raw counts, or apply $\log$ scaling.
4. **We need lower dimensions.** 50,000-dimensional vectors are still too large. *Solution:* use dimensionality reduction (e.g., Singular Value Decomposition — SVD).

Latent Semantic Analysis

Applying SVD to a co-occurrence (or document-term) matrix gives **Latent Semantic Analysis (LSA)** — the precursor to modern embeddings, dating back to Deerwester et al. (1990). LSA was the first method to produce dense word vectors with semantic structure.

Measuring Similarity: Two Approaches

Once we have vectors, how do we measure how "close" two words are?

Euclidean Distance

The straight-line distance between two points:

$$d(\mathbf{u}, \mathbf{v}) = \sqrt{\sum_{i=1}^{d} (u_i - v_i)^2}$$

Problem with text: sensitive to vector *magnitude*. A word that appears very often will have a large vector, even if its contexts are similar to a rarer word's. E.g., "the" appears 10,000 times; "despite" appears 50 times. Their vectors are very different lengths even if their contexts are similar.

Cosine Similarity

The cosine of the angle between two vectors:

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \cdot \|\mathbf{v}\|}$$

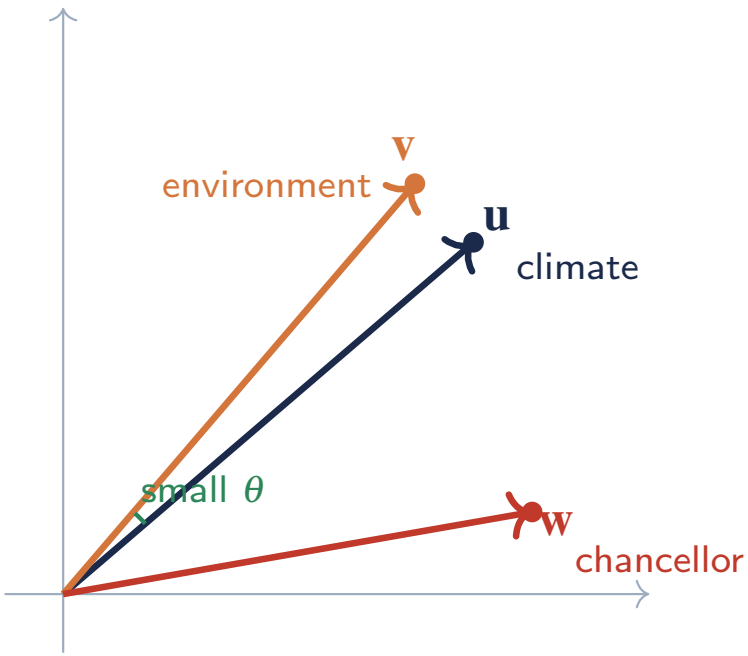
where the **magnitude** (norm) of a vector is:

$\|\mathbf{u}\| = \sqrt{u_1^2 + u_2^2 + \cdots + u_d^2} = \sqrt{\sum_{i=1}^{d} u_i^2}$

Why it's better: measures only the *direction*, not the magnitude. Two vectors pointing the same way have cosine $= 1$; perpendicular vectors have cosine $= 0$; opposite vectors have cosine $= -1$.

Word frequency cancels out — exactly what we want for measuring semantic similarity.

Cosine Similarity: Geometric Intuition



The angle tells the story:

$\cos(\mathbf{u}, \mathbf{v})$: very close to 1

$\rightarrow$ "climate" and "environment" are nearly the same direction

$\cos(\mathbf{u}, \mathbf{w})$: much smaller

$\rightarrow$ "climate" and "chancellor" point in different directions

Range:

- $\cos = 1$: identical direction
- $\cos = 0$: perpendicular (unrelated)
- $\cos = -1$: opposite (rare in practice)

Cosine similarity is the standard metric for comparing word embeddings. You'll use it constantly: finding nearest neighbors, measuring word-to-concept distances, evaluating embedding quality.

Cosine Similarity: A Numerical Example

Computing cosine by hand

Suppose we have two 4-dimensional word vectors:

$$\mathbf{u} = [2, 1, 0, 3], \quad \mathbf{v} = [1, 2, 1, 2]$$

Step 1: Dot product

$$\mathbf{u} \cdot \mathbf{v} = (2)(1) + (1)(2) + (0)(1) + (3)(2) = 2 + 2 + 0 + 6 = 10$$

Step 2: Magnitudes (norms)

$$\|\mathbf{u}\| = \sqrt{4 + 1 + 0 + 9} = \sqrt{14} \approx 3.742$$

$$\|\mathbf{v}\| = \sqrt{1 + 4 + 1 + 4} = \sqrt{10} \approx 3.162$$

Step 3: Cosine similarity

$$\cos(\mathbf{u}, \mathbf{v}) = \frac{10}{3.742 \times 3.162} = \frac{10}{11.832} \approx 0.845$$

Interpretation: Cosine ≈ 0.85 is high — these vectors point in very similar directions. The two words are likely semantically related.

Table of Contents

The Word2Vec Revolution (2013)

Mikolov et al. (2013) at Google introduced **Word2Vec**, which transformed NLP.

Key insight: Instead of building a co-occurrence matrix and reducing it with SVD, **train a small neural network** to predict context from words (or words from context). The network's hidden-layer weights *become* the word embeddings.

Why was Word2Vec a big deal?

- Trains efficiently on billions of words
- Produces high-quality dense vectors ($d = 100$ to 300)
- Captures rich semantic structure
- Enables word *analogies* (we'll see this!)
- Pre-trained vectors freely available for many languages

Two architectures:

1. CBOW (Continuous Bag of Words) Given context words, predict the center word.

2. Skip-gram Given the center word, predict the context words.

Skip-gram works better for rare words; CBOW is faster and works well for frequent words. In practice, Skip-gram is more commonly used.

CBOW: Predicting a Word from Its Context

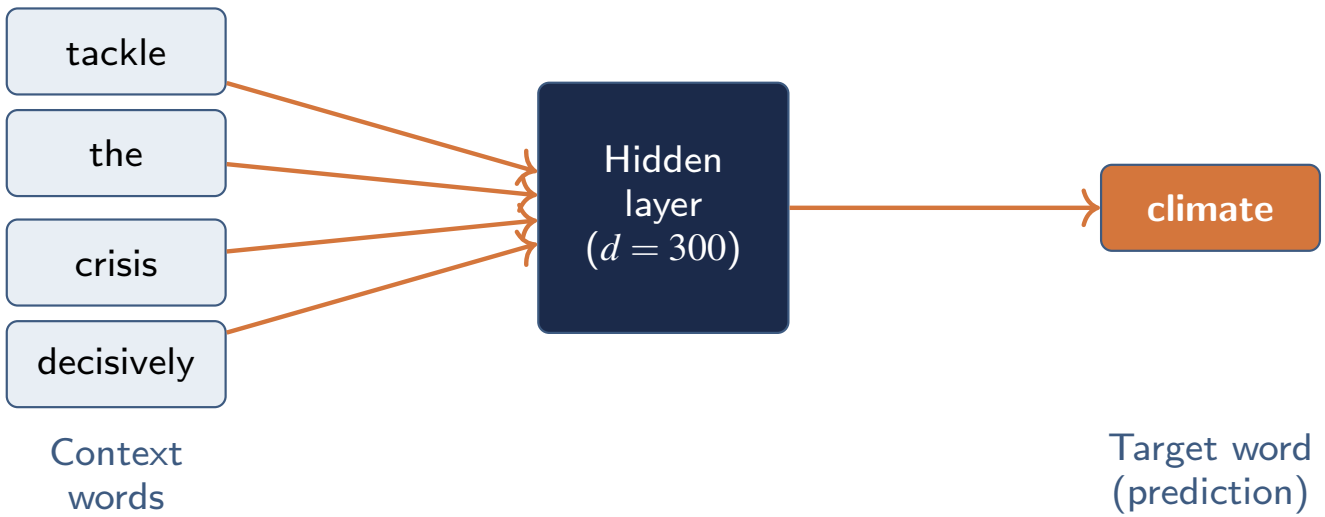
Idea: Take the words around a target word, and try to predict the target word.

A training example

Sentence: "The **government** must tackle the climate crisis decisively." With a window of ± 2:

- **Context** (input): `[tackle, the, crisis, decisively]`
- **Target** (output): `climate`

The model is trained to: *given these context words, output a high probability for "climate"*.



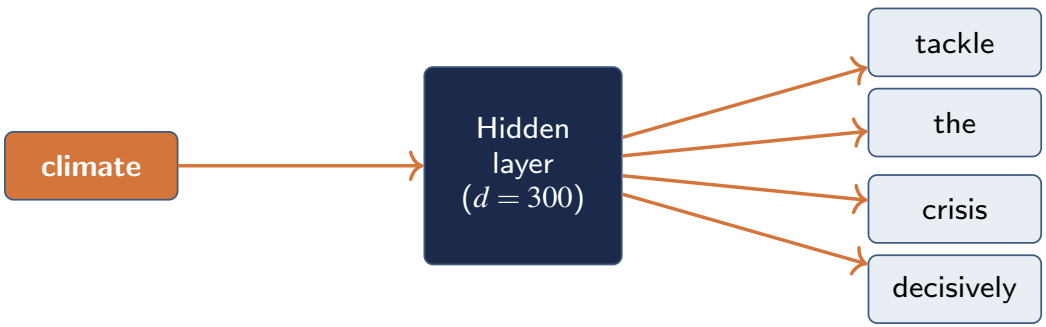
Skip-gram: Predicting Context from a Word

Skip-gram is the reverse of CBOW. Given a word, predict the words that surround it.

Same sentence, different framing

Sentence: "The government must tackle the **climate** crisis decisively." With a window of ± 2, the training pairs are:

Input	Predict
climate	tackle
climate	the
climate	crisis
climate	decisively



After training on millions of such pairs from a large corpus, the hidden-layer weights become the word vectors. Each word ends up with a 300-dimensional vector that encodes which contexts it tends to appear in.

Where Do the Embeddings Come From?

The trick: the embeddings ARE the weights of the neural network.

1. The network has an input layer of size $|V|$ (one-hot encoding of input word)
2. A hidden layer of size d (typically 100–300)
3. An output layer of size $|V|$ (predicted probability for each word in vocabulary)

The weights between input and hidden layer form a $|V| \times d$ matrix.
Row i of this matrix is the d-dimensional embedding of word i.

The Magic

We train the network on a **prediction task** (predicting context from word, or vice versa). But we don't actually care about the predictions! We care about the **hidden-layer weights** that the network learned in order to make good predictions. Those weights are the embeddings. The prediction task is just a *pretext* for forcing the network to learn useful representations.

This idea — using a "pretext task" to learn representations — is the foundation of **self-supervised learning**, which we'll see again in Lecture 7 with BERT.

GloVe: Another Approach (Stanford, 2014)

GloVe (Global Vectors for Word Representation) takes a different approach.
The intuition: Word2Vec only looks at *local* context windows. GloVe uses *global* co-occurrence statistics across the entire corpus.
The procedure (high-level):

1. Build a word-word co-occurrence matrix from the entire corpus (like the matrix we saw earlier, but huge).
2. Train word vectors so that the dot product of two word vectors approximates the log of their co-occurrence count:

$$\mathbf{u}_i \cdot \mathbf{v}_j \approx \log(\text{count}(w_i, w_j))$$

3. This is a **matrix factorization** approach: we're approximating the (log) co-occurrence matrix as a product of two smaller matrices, where the rows of the smaller matrices are the embeddings.

In practice, Word2Vec and GloVe produce embeddings of similar quality. GloVe is faster to train on small corpora; Word2Vec scales better to very large corpora. Both are widely used and have pre-trained vectors freely available for dozens of languages.

fastText: Subword Embeddings

fastText (Facebook, 2016) extends Word2Vec with one important addition: it learns embeddings for **character n-grams**, not just whole words.

Why does this matter?

Word2Vec/GloVe problem:

- Each word is a separate atomic unit
- Word *morphology* is ignored: "government" and "governing" share a root but get unrelated embeddings
- Cannot handle words not seen during training (out-of-vocabulary)

fastText solution:

- Each word is represented as a sum of its character n-gram embeddings
- "governing" = vec(<go) + vec(ove) + vec(ver) + ...
- Captures morphology: "govern" and "governing" share many n-grams
- Can produce embeddings for words it has *never seen*

Especially useful for morphologically rich languages

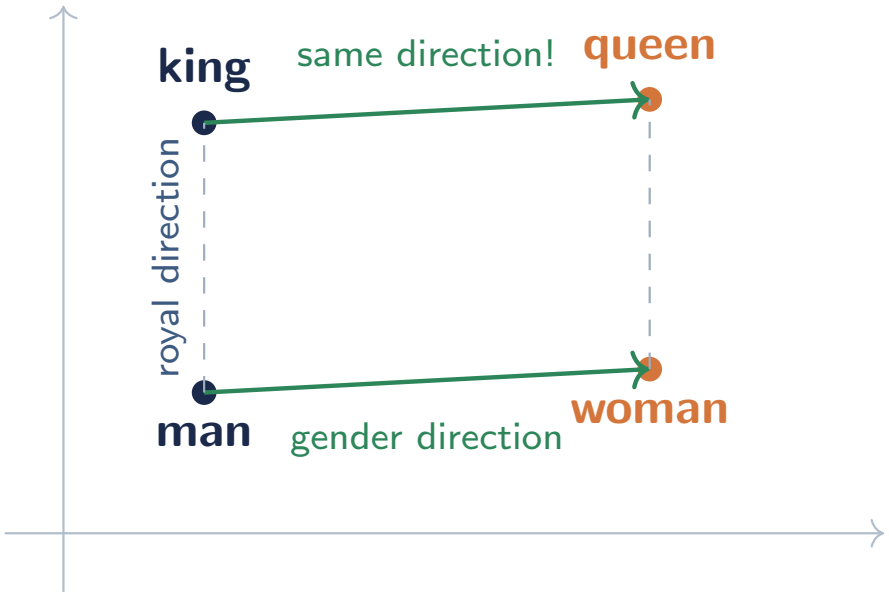
Languages like German, Turkish, or Finnish have extensive compounding and inflection. fastText handles these much better than Word2Vec, making it a strong default choice for multilingual political text analysis.

Word Analogies: The Surprising Discovery

One of the most striking findings about Word2Vec embeddings:

$$\mathbf{v}(\text{king}) - \mathbf{v}(\text{man}) + \mathbf{v}(\text{woman}) \approx \mathbf{v}(\text{queen})$$

In words: If you take the vector for "king", subtract the vector for "man", and add the vector for "woman", the resulting vector is closest to "queen".
This means embeddings encode relations as directions in the vector space!



The vector difference *king* − *man* captures "royalness". The difference *man* − *woman* captures "gender". These directions are consistent across the embedding space.

More Analogies: Patterns in the Embedding Space

Word2Vec captures many systematic relationships:

Analogy expression		Closest word
$\mathbf{v}(\text{Paris}) - \mathbf{v}(\text{France}) + \mathbf{v}(\text{Germany})$	$\approx$	Berlin
$\mathbf{v}(\text{Madrid}) - \mathbf{v}(\text{Spain}) + \mathbf{v}(\text{Italy})$	$\approx$	Rome
$\mathbf{v}(\text{quick}) - \mathbf{v}(\text{quickly}) + \mathbf{v}(\text{slowly})$	$\approx$	slow
$\mathbf{v}(\text{good}) - \mathbf{v}(\text{better}) + \mathbf{v}(\text{worse})$	$\approx$	bad
$\mathbf{v}(\text{Japan}) - \mathbf{v}(\text{sushi}) + \mathbf{v}(\text{Italy})$	$\approx$	pizza

What this tells us: The embedding space has *learned* that:

- Capital cities and their countries share a consistent direction
- Comparative/superlative forms follow a pattern
- Cultural associations (food $\leftrightarrow$ country) are encoded

Caution

Analogies are striking but **don't always work**. They work best for clean, well-attested relationships in large training corpora. For political concepts, analogies are noisier and should be used with caution. Don't over-interpret.

Pre-trained Embeddings: Standing on Giants

You usually don't train embeddings from scratch. Instead, you download **pre-trained embeddings** that someone else has trained on a massive corpus.
Popular pre-trained embeddings:

- **Google News Word2Vec** (English, 300d, trained on 100B words)
- **GloVe** (English, multiple sizes, trained on Wikipedia + Common Crawl)
- **fastText pre-trained vectors** (157 languages, trained on Common Crawl + Wikipedia) — https://fasttext.cc/docs/en/crawl-vectors.html

When to train your own:

- Your domain is very specialized (e.g., 18th-century parliamentary speech)
- You want embeddings that reflect a specific time period or community
- You're studying *how* word meanings differ across corpora

For most political science applications, pre-trained fastText or GloVe embeddings work very well as a starting point. You can also *fine-tune* pre-trained embeddings on your domain corpus to combine general knowledge with domain specificity.

Table of Contents

Why Should Political Scientists Care About Embeddings?

Embeddings open up entirely new ways to study political language quantitatively.

1. **Measuring concepts.** Define a concept (e.g., "populism") as a region in vector space. Measure how close any word is to that region.
2. **Tracking semantic change.** Train embeddings on different time periods. Has the meaning of "immigration" shifted over time? In which direction?
3. **Detecting bias.** Embeddings absorb social biases from their training data. We can measure these biases — and study their political consequences.
4. **Comparing discourses.** Train embeddings separately on left-leaning and right-leaning corpora. How do their semantic spaces differ?
5. **Improving classifiers.** Use embeddings as features for the classifiers from Lecture 2. Often outperforms TF-IDF.
6. **Scaling positions.** Project political texts onto an ideological dimension using embedding-based methods.

Embeddings Encode Bias (Caliskan et al. 2017)

A landmark finding published in *Science*: word embeddings reproduce human biases that are present in the training corpus.
The Word Embedding Association Test (WEAT): measures how strongly two sets of target words (e.g., *male/female names*) are associated with two sets of attribute words (e.g., *career/family*).

Examples of biases found in standard pre-trained embeddings

- *Male names* are closer to *career-related words* (executive, salary, professional); *female names* are closer to *family-related words*.
- *European-American names* are more strongly associated with *pleasant words*; *African-American names* with unpleasant words.
- *Female names* are more strongly associated with *arts and humanities*; *male names* with *math and science*.

Why this matters for political science:

- These are the *same* stereotypes documented in psychological research
- Embeddings learn them from text without anyone explicitly programming them
- If you use embeddings in a downstream classifier, the bias *transfers*
- Embeddings can be used as a *measurement tool* to study cultural bias in different corpora

Application: Measuring Concepts with Embeddings

The semantic projection method: measure how strongly any word is associated with a concept by projecting onto a defined direction.

Measuring economic vs. social orientation

Step 1: Define the concept axis using seed word pairs:

$$\mathbf{d}_{\text{econ}\to\text{soc}} = \frac{1}{|S|}\sum(\mathbf{v}_{\text{social}_i} - \mathbf{v}_{\text{economic}_i})$$

where seeds might be: (*taxation* $\to$ *solidarity*), (*market* $\to$ *justice*), (*business* $\to$ *welfare*), . . .

Step 2: Project any word onto this direction:

$$\text{score}(w) = \cos(\mathbf{v}_w, \mathbf{d}_{\text{econ}\to\text{soc}})$$

Result: A continuous score telling you how "social-policy oriented" vs. "economic-policy oriented" any word is, even words that weren't in your seed list.

This is a powerful idea: instead of building a classifier with thousands of labels, you define a concept with a handful of seed words and let the embedding space generalize.

Application: Rodriguez & Spirling (2022)

"Word Embeddings: What Works, What Doesn't, and How to Tell the Difference for Applied Research." *Journal of Politics* 84(1)
What they did:

- Compared multiple embedding methods (Word2Vec, GloVe, BERT) and pre-trained vs. custom-trained on political corpora
- Tested whether embeddings capture political concepts that political scientists care about
- Provided practical guidance for applied researchers

Key findings:

1. Pre-trained embeddings often work well for political tasks, but custom-trained on a relevant political corpus can be better.
2. Smaller, focused corpora can outperform larger general corpora for domain-specific tasks.
3. Different methods (Word2Vec vs. GloVe vs. BERT) give similar results on most tasks.
4. **Always validate your embeddings** on a task you care about, with a known answer.

Takeaway for your research: Don't blindly use pre-trained embeddings. Test whether they capture the concepts *you* care about.

Application: Tracing Semantic Change Over Time

Idea: Train embeddings on different time periods. Compare how a word's nearest neighbors change.

Has the meaning of "immigration" changed in U.S. Congressional speech?

Train embeddings on Congressional Record by decade:

1980s neighbors of "immigration"	**2010s neighbors of "immigration"**
labor	border
employment	illegal
quota	enforcement
naturalization	security
visa	deportation

Interpretation: The semantic neighborhood of "immigration" shifted from a labor/legal-process frame (1980s) toward a security/enforcement frame (2010s). This shift can be **quantified** (cosine similarity between the time-period vectors) and tested statistically.

Pioneering work in this area: Hamilton, Leskovec & Jurafsky (2016), "Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change." Methodological challenge: making embeddings from different time periods comparable (the spaces are not automatically aligned).

Visualizing Embeddings with PCA

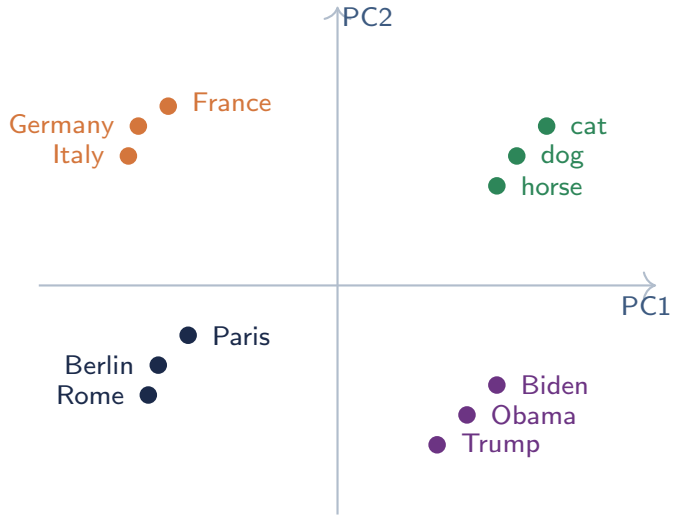
Problem: Embeddings live in 300 dimensions. We can't see them directly.
Solution: Principal Component Analysis (PCA) projects the high-dimensional vectors down to 2 or 3 dimensions for visualization.

What PCA does:

- Finds the directions ("principal components") along which the data varies the most
- Projects the data onto these directions
- Preserves as much of the original variance as possible

What it gives you:

- A 2D scatter plot of words
- Words that are close in the original 300-d space tend to be close in the 2D projection
- Useful for *exploration* and *communication* — not for precise measurement



Sketch: words form interpretable clusters in the 2D projection.

Caveats When Using Embeddings

Embeddings are powerful, but they have important limitations.

1. **Static, not contextual.** Word2Vec gives "bank" a single vector, regardless of whether it means a financial institution or a riverbank. *(Lecture 7 will fix this with BERT.)*
2. **Out-of-vocabulary words.** If a word wasn't in the training corpus, you have no embedding for it. (fastText partially solves this.)
3. **Domain mismatch.** Embeddings trained on Wikipedia may not capture how words are used in parliamentary speech.
4. **Embeddings inherit bias.** Whatever biases are in the training data are encoded in the embeddings — and will affect any downstream analysis.
5. **Hyperparameter sensitivity.** The embedding dimension, window size, training algorithm, and corpus size all affect results. Document your choices.
6. **Hard to interpret individual dimensions.** Unlike BoW, you can't say "dimension 47 means immigration." You can only interpret distances and directions.

Table of Contents

Practical Session: Overview

Exploring Pre-trained English Word Embeddings

What we'll do in the next 30 minutes:

1. Load pre-trained GloVe embeddings into R
2. Find nearest neighbors for political concepts
3. Compute cosine similarities between word pairs
4. Test analogies
5. Visualize political vocabulary with PCA
6. Measure "distance" from political concepts

Tools we'll use:

- `text2vec` for embedding operations
- Pre-trained GloVe vectors (download from https://nlp.stanford.edu/projects/glove/)

```
install.packages(c("text2vec", "ggplot2", "ggrepel"))
```

Table of Contents

Key Takeaways

1. **Word embeddings** represent words as dense vectors in a continuous space, where similar words have similar vectors. This is a fundamental shift from sparse, count-based representations.
2. **The distributional hypothesis** ("you shall know a word by the company it keeps") is the foundational idea that makes embeddings work — and it underlies every modern NLP method we'll see.
3. **Word2Vec, GloVe, and fastText** are the classical methods. Word2Vec uses local context windows, GloVe uses global co-occurrence statistics, and fastText adds subword information — crucial for morphologically rich languages.
4. **Cosine similarity** is the standard metric for measuring how similar two word vectors are. It measures angle, not magnitude.
5. **Embeddings encode bias.** This is both a problem (when they're used in classifiers) and a research opportunity (when we measure cultural attitudes).
6. **Political science applications are rich and growing**: scaling, semantic change, framing analysis, concept measurement. Rodriguez & Spirling (2022) is a great starting point.
7. **Embeddings are static**: each word has one vector, regardless of context. *We'll fix this in Lecture 7 with BERT.*

Home Assignment (Optional)

Task: Use pre-trained embeddings to study a political concept of your choice.
Instructions:

1. Choose a politically interesting word or concept (examples: populism, sovereignty, austerity, patriotism, climate, welfare, immigration).
2. Find its 20 nearest neighbors in the embedding space. Inspect them.
3. Define a meaningful axis using seed words (e.g., left vs. right, positive vs. negative, secular vs. religious). Project your concept and 10 related words onto this axis.
4. Visualize a set of $\sim$20 related words using PCA.
5. Test at least one bias dimension using WEAT-style seed words.
6. Write a **1.5-page interpretation**: What does the embedding tell you about how this concept is positioned in the linguistic landscape? What surprised you? What are the limitations of this analysis?

Submit: R script + report. Due before Lecture 4.

Readings

Required:

- Jurafsky, D. & Martin, J.H. (2025). *Speech and Language Processing*, Chapter 5: Embeddings. https://web.stanford.edu/~jurafsky/slp3/5.pdf

Optional reading

- Rodriguez, P.L. & Spirling, A. (2022). "Word Embeddings: What Works, What Doesn't, and How to Tell the Difference for Applied Research." *Journal of Politics*, 84(1), 101–115.
- Grimmer, J., Roberts, M. E., & Stewart, B.M. (2022). *Text as Data*, Chapters 6&7.

Other related readings:

- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). "Efficient Estimation of Word Representations in Vector Space." *arXiv:1301.3781*.
- Caliskan, A., Bryson, J.J., & Narayanan, A. (2017). "Semantics Derived Automatically from Language Corpora Contain Human-like Biases." *Science*, 356(6334), 183–186.
- Hamilton, W.L., Leskovec, J., & Jurafsky, D. (2016). "Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change." *ACL 2016*.

Next week: Document Representations & Topic Models

From word vectors to document vectors — and unsupervised discovery of latent themes