Lecture 3 — Word Embeddings & Vector Spaces

From Counting Words to Understanding Meaning

Tamara Grechanaya
Multimodal Computational Methods in Political Science

Summer Semester 2026

Contents

1 Learning objectives

By the end of this lecture, you should be able to:

1. Explain why bag-of-words representations are fundamentally limited and what the distributional hypothesis offers as an alternative.
2. Understand what a word embedding is — a dense vector that captures semantic meaning — and why similar words end up with similar vectors.
3. Compute cosine similarity between two vectors by hand and interpret the result.
4. Explain the intuition behind Word2Vec (both CBOW and Skip-gram), GloVe, and fastText at a conceptual level — what they optimize, how they learn, and where the embeddings come from.
5. Use pre-trained embeddings in R to find nearest neighbors, test analogies, and visualize political vocabulary.
6. Recognize how embeddings encode social and political biases, and understand both the risks and the research opportunities this creates.

---

2 Part I — Why we need a better representation

2.1 The limits of what we built in Lectures 1–2

In the first two lectures, we represented text as vectors of word counts (or TF-IDF weights). Each document became a row in a Document-Term Matrix with one column per word in the vocabulary. This worked well for classification: logistic regression and Naive Bayes could distinguish government from opposition speeches with reasonable accuracy.

But this representation has three fundamental limitations that become apparent once we try to do anything beyond classification.

Limitation 1: No notion of similarity between words. In a bag-of-words representation, every word is just an index — a column number. "Climate" is column 2,341 and "environment" is column 7,892 and "Brexit" is column 1,056. As far as the model is concerned, "climate" is just as different from "environment" as it is from "Brexit." There is no way to express the fact that "climate" and "environment" are semantically related. If a classifier learns that the word "climate" predicts Green Party speeches, that knowledge does not transfer to "environment" — the model has to learn the association with "environment" entirely from scratch, using separate training examples.

Limitation 2: Extreme dimensionality and sparsity. A vocabulary of 50,000 words means 50,000 dimensions. Each document uses perhaps 200 of them. The result is a 50,000-dimensional vector that is 99.6% zeros. This wastes memory, slows computation, and creates statistical problems: with so many dimensions relative to the number of training documents, the model is at constant risk of overfitting.

Limitation 3: No generalization across words. If your training data contains the word "healthcare" in many Labour speeches, the classifier learns that "healthcare" predicts Labour. But if the test data uses "NHS" instead (a synonym in context), the classifier has no idea that this should also predict Labour — because "healthcare" and "NHS" are completely unrelated in the bag-of-words representation. Each word is an island.

These limitations motivate the central question of this lecture: **can we find a representation where words that mean similar things have similar vectors?**

2.2 One-hot encoding: the baseline to improve upon

To make the problem precise, consider the simplest possible word representation: **one-hot encoding**. Each word is represented as a vector of length $|V|$ (the vocabulary size) with a single 1 at the word's index and 0 everywhere else.

For a tiny vocabulary of four words — {climate, policy, London, Paris} — the one-hot vectors would be:

- climate = [1, 0, 0, 0]
- policy = [0, 1, 0, 0]
- London = [0, 0, 1, 0]
- Paris = [0, 0, 0, 1]

The fatal flaw is that **every pair of words is equidistant**. The Euclidean distance between any two one-hot vectors is $\sqrt{2}$, and the dot product between any two different one-hot vectors is 0. According to this representation, London and Paris (both European capitals) are exactly as far apart as London and climate (completely unrelated concepts). This is obviously wrong, and no amount of downstream processing can fix it — the representation itself has lost the information.

What we want instead is a representation where London and Paris are *close* to each other in the vector space, because they share important properties (both are capitals, both are in Europe, both are cities). The distributional hypothesis gives us a way to build such a representation.

2.3 The distributional hypothesis

The foundational idea behind all modern word representations comes from linguistics, specifically from J.R. Firth (1957), who wrote:

> "You shall know a word by the company it keeps."

This is the **distributional hypothesis**: words that appear in similar *contexts* tend to have similar *meanings*. If two words regularly appear surrounded by the same neighboring words, they probably mean something similar.

Here is a concrete example. Suppose you encounter a word you have never seen before — *tesgüino* — in these sentences:

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

Without anyone telling you, you can infer that *tesgüino* is some kind of alcoholic drink made from corn. You inferred this entirely from the words that appeared *around* it — "bottle," "likes," "drunk," "corn." The context defined the meaning.

This principle is remarkably powerful, and it underlies every modern NLP method — from Word2Vec (2013) to BERT (2018) to today's largest language models. The difference between

these methods is *how* they exploit context, but they all rest on the same foundational insight: context defines meaning.

2.3.1 A political example

Consider two words: "climate crisis" and "climate hysteria." Both refer to climate concerns, but they carry very different political valences.

"Climate crisis" tends to appear in contexts like: "tackling the *climate crisis*," "scientific evidence on the *climate crisis*," "international cooperation on the *climate crisis*." These contexts are associated with urgency, evidence, and collective action.

"Climate hysteria" tends to appear in contexts like: "the so-called *climate hysteria*," "this *climate hysteria* threatens jobs," "media *climate hysteria*." These contexts are associated with dismissal, economic concern, and skepticism.

A word embedding trained on political text will place these two expressions in different regions of the vector space — not because anyone told the model that one is progressive and the other is conservative, but because their contextual neighborhoods differ. The model learns political valence from context alone.

2.4 Sparse vs. dense representations

The representations from Lectures 1–2 are **sparse**: 50,000 dimensions, mostly zeros, each dimension corresponding to a specific word. Word embeddings are **dense**: typically 100 to 300 dimensions, all entries are non-zero real numbers, and individual dimensions do not correspond to interpretable concepts.

What do we gain and what do we lose in this trade?

We gain **semantic structure**: similar words have similar vectors, which means similarity is now a meaningful and computable quantity. We gain **compactness**: 300 dimensions instead of 50,000, which is computationally efficient and reduces overfitting. We gain **generalization**: if the model learns something about "healthcare," that knowledge partially transfers to "NHS" because the two words have similar vectors.

We lose **individual interpretability**: in a bag-of-words vector, dimension 2,341 means "climate" — you can read it directly. In a 300-dimensional embedding, dimension 47 does not mean anything by itself. You can only interpret the embedding *geometrically* — through distances, directions, and relationships between vectors. This is a real trade-off, and it is worth being explicit about.

---

3 Part II — Measuring similarity in vector spaces

3.1 Words as points in space

Once we have word vectors, we can think of each word as a *point* in a high-dimensional space. Words that mean similar things cluster together. In an idealized 2D projection, you might see clusters of animals (dog, cat, horse), clusters of cities (London, Paris, Berlin), and clusters of political concepts (election, parliament, democracy).

The real embedding space has 300 dimensions, so we cannot visualize it directly. But the geometric intuition carries over: proximity in this high-dimensional space corresponds to semantic relatedness. We need a way to *measure* this proximity.

3.2 Two ways to measure distance

Euclidean distance is the straight-line distance between two points, computed by the Pythagorean theorem extended to many dimensions:

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

This is the most intuitive notion of distance, but it has a problem for text: it is sensitive to the *magnitude* (length) of the vectors. A word that appears 10,000 times in the training corpus will have a larger vector than a word that appears 50 times, even if they have similar contextual neighborhoods. Euclidean distance would say these two words are far apart because one vector is much longer than the other, even though their *directions* — which encode meaning — are similar.

Cosine similarity solves this by measuring only the *angle* between two vectors, ignoring their lengths:

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

Let me unpack each piece of this formula.

The **dot product** $\mathbf{u} \cdot \mathbf{v}$ is computed by multiplying corresponding entries of the two vectors and summing the results: $u_1 v_1 + u_2 v_2 + \cdots + u_d v_d$. The dot product is large when the two vectors point in the same direction and small (or negative) when they point in different directions.

The **norm** (or magnitude) $\|\mathbf{u}\|$ is the length of the vector, computed as $\sqrt{u_1^2 + u_2^2 + \cdots + u_d^2}$. Dividing by the product of the two norms normalizes the dot product so that the result is between -1 and $+1$, regardless of how long the vectors are.

The result is the cosine of the angle between the two vectors:

- Cosine = 1: the vectors point in exactly the same direction (identical meaning, as far as the embedding is concerned).
- Cosine = 0: the vectors are perpendicular (unrelated meanings).
- Cosine = -1: the vectors point in opposite directions (theoretically opposite meanings, but this is rare in practice for word embeddings).

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

3.3 A worked example

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.

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

Step 2: Norms.

$$\|\mathbf{u}\| = \sqrt{2^2 + 1^2 + 0^2 + 3^2} = \sqrt{4 + 1 + 0 + 9} = \sqrt{14} \approx 3.742$$

$$\|\mathbf{v}\| = \sqrt{1^2 + 2^2 + 1^2 + 2^2} = \sqrt{1 + 4 + 1 + 4} = \sqrt{10} \approx 3.162$$

Step 3: Cosine similarity.

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

A cosine similarity of 0.845 is high — these vectors point in very similar directions. If these were word embeddings, the two words would be considered semantically related.

For comparison, a typical cosine similarity between "climate" and "environment" in a well-trained embedding might be around 0.7–0.8. Between "climate" and "bicycle" it might be around 0.1–0.2. Between "king" and "queen" it might be around 0.6–0.7.

---

4 Part III — Word2Vec, GloVe, and fastText

4.1 The Word2Vec revolution

In 2013, Tomas Mikolov and colleagues at Google published a paper that transformed natural language processing: *Efficient Estimation of Word Representations in Vector Space*. The method, called **Word2Vec**, showed that you could train a simple neural network on a very large corpus of text to produce word vectors that captured rich semantic structure — including the famous analogy relationships like king − man + woman ≈ queen.

The key insight of Word2Vec is surprisingly indirect. The goal is to produce good word vectors, but the method achieves this by training a neural network on a *prediction task*. The prediction task is a pretext — a means to an end. The network is trained to predict words from their contexts (or contexts from words), and the word vectors emerge as a byproduct: they are the weights of the network's hidden layer. The network learns good word vectors *because* good word vectors are what it needs internally to make accurate predictions.

This idea — training a model on a pretext task and extracting the learned internal representations — is called **self-supervised learning**, and it is the foundational principle behind BERT, GPT, and all modern language models. We will see it again in Lecture 7.

4.2 CBOW: predicting a word from its context

Word2Vec comes in two variants. The first is **Continuous Bag of Words (CBOW)**, which works like this:

Given a sentence like "The government must take decisive action on climate change," and a window size of ± 2 words, consider the word "action" in the middle. The context is the surrounding words: ["decisive", "take", "on", "climate"]. CBOW trains a neural network that takes these context words as input and tries to predict the center word ("action").

The network has three layers. The input layer receives the one-hot encoded context words. The hidden layer is a dense layer of size d (typically 100–300 neurons). The output layer produces a probability distribution over the entire vocabulary — a prediction of which word is most likely to appear in this context.

The hidden layer weights form a $|V| \times d$ matrix. After training, row i of this matrix is the d-dimensional embedding of word i. This is where the embeddings come from: they are literally the weights that the network learned in order to make good context-to-word predictions.

Why does this produce meaningful embeddings? Because words that appear in similar contexts will need similar hidden-layer representations to predict those contexts correctly. If "government" and "administration" tend to appear in the same kinds of contexts (surrounded by similar words), the network will learn similar hidden-layer vectors for both — because similar vectors are what it needs to produce similar context predictions. The semantic similarity emerges from the shared contextual patterns.

4.3 Skip-gram: predicting context from a word

The second variant is **Skip-gram**, which reverses the direction. Instead of predicting the center word from its context, Skip-gram predicts the context words from the center word.

Given the same sentence and the center word "action," Skip-gram generates training pairs: (action $\rightarrow$ decisive), (action $\rightarrow$ take), (action $\rightarrow$ on), (action $\rightarrow$ climate). For each pair, the network takes the center word as input and tries to predict the context word.

Skip-gram tends to produce better embeddings for rare words (because each rare word generates multiple training pairs — one for each context word), while CBOW is faster and works better for frequent words. In practice, Skip-gram is more commonly used.

4.4 Where do the embeddings actually come from?

This is worth emphasizing because it is the conceptual heart of the method.

The neural network has an input layer, a hidden layer, and an output layer. The weights between the input layer and the hidden layer form a matrix of size $|V| \times d$ — vocabulary size times embedding dimension. Row i of this matrix is the d-dimensional embedding vector for word i.

We train the network to predict context from words (Skip-gram) or words from context (CBOW). **But we do not actually care about the predictions.** We care about the weight matrix — the embeddings — that the network learned *in order to* make good predictions. The prediction task is just a vehicle for forcing the network to organize words in a meaningful way.

After training is complete, we throw away the prediction machinery (the output layer) and keep only the hidden-layer weight matrix. That matrix *is* the set of word embeddings.

This idea — using a pretext task to learn useful representations — is the single most important concept in modern NLP. BERT does the same thing at a much larger scale: it trains a neural network to predict masked words in a sentence, and the learned representations turn out to be extraordinarily useful for all kinds of downstream tasks. We will return to this in Lecture 7.

4.5 GloVe: a matrix factorization perspective

GloVe (Global Vectors for Word Representation), developed at Stanford by Pennington, Socher, and Manning in 2014, takes a different approach to reaching the same destination.

Where Word2Vec looks at local context windows (one sentence at a time), GloVe starts by building a global co-occurrence matrix for the entire corpus. Entry (i, j) of this matrix counts how often word i appears near word j across the whole corpus. Then GloVe learns word vectors such that the dot product of two word vectors approximates the logarithm of their co-occurrence count:

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

This is a form of **matrix factorization**: we are approximating a large matrix (the log co-occurrence matrix) as the product of two smaller matrices (whose rows are the embeddings). If you have taken a course that covered Principal Component Analysis (PCA) or Singular Value Decomposition (SVD), the idea is closely related.

In practice, Word2Vec and GloVe produce embeddings of similar quality. GloVe can be faster to train on smaller corpora because it works with pre-computed co-occurrence statistics rather than iterating over the raw text. Both have freely available pre-trained vectors for many languages.

4.6 fastText: handling morphology

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

Standard Word2Vec treats each word as an atomic unit. "Government" gets one embedding, "governmental" gets a completely separate embedding, and the two are unrelated — the shared root "govern" is invisible to the model. Worse, if a word never appeared in the training corpus (an "out-of-vocabulary" or OOV word), Word2Vec has no embedding for it at all.

fastText solves both problems. It represents each word as the sum of its character n-gram embeddings. For example, "government" would be represented as the sum of its character trigrams: "<go", "gov", "ove", "ver", "ern", "rnm", "nme", "men", "ent", "nt>", plus the whole word "government" itself (the angle brackets mark word boundaries). Two words that share many character n-grams — like "government" and "governmental" — will have similar embeddings because they share most of their subword components.

fastText can also produce embeddings for words it has never seen during training: it simply sums the character n-gram embeddings for the new word. This is especially valuable for morphologically rich languages and for handling misspellings in social media text.

Pre-trained fastText vectors are freely available for 157 languages at https://fasttext.cc/docs/en/crawl-vectors.html, including excellent English models.

4.7 Word analogies: structure in the embedding space

One of the most striking findings about Word2Vec embeddings is that they encode *relationships as directions* in the vector space.

The famous example:

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

What this means: if you take the vector for "king," subtract the vector for "man," and add the vector for "woman," the resulting vector is closest (in cosine similarity) to the vector for "queen."

Why does this work? The vector difference "king − man" captures the concept of "royalness" — what you get when you remove the "maleness" from "king." Adding "woman" puts the "femaleness" back. The result should be a royal female — and indeed, the nearest neighbor in the embedding space is "queen."

This works because the embedding space has learned that the *direction* from "man" to "woman" is the same as the direction from "king" to "queen." The gender direction is a consistent axis in the space, and it operates independently of other properties like royalty, profession, or nationality.

Other analogies that typically work:

- Paris − France + Italy ≈ Rome (capital-country relationship)
- walked − walking + swimming ≈ swam (tense relationship)
- Japan − sushi + Germany ≈ bratwurst (cultural associations)

Analogies don't always work perfectly — they fail more often than the cherry-picked examples in papers suggest. But the fact that they work *at all* is remarkable: the embedding space has learned systematic semantic relationships from nothing but raw text.

4.8 Pre-trained embeddings: when to use them

You almost never need to train word embeddings from scratch. Pre-trained embeddings — vectors trained by someone else on a very large corpus — are freely available and work well for most tasks.

When to use pre-trained embeddings:

- You want high-quality vectors without spending days training on a massive corpus.
- Your corpus is too small to train reliable embeddings (you typically need at least tens of millions of words).
- You want your analysis to be reproducible using a standard resource.

When to train your own:

- Your domain is very specialized (e.g., 18th-century parliamentary language, or clinical medical text) and general-purpose embeddings don't capture the relevant meanings.
- You want embeddings that reflect a specific time period or community — for example, to study how the meaning of "immigration" differs between left-wing and right-wing media.
- You are studying semantic change over time and need separate embeddings for each period.

For most political science applications, pre-trained fastText embeddings are an excellent starting point.

5 Part IV — Embeddings for political science

5.1 Why political scientists should care

Embeddings open up research designs that are simply impossible with bag-of-words representations. Here are the main applications.

Measuring concepts as directions. Define a concept (say, the economic left-right spectrum) as a *direction* in embedding space using seed word pairs: (market, solidarity), (competition, equality), (enterprise, welfare). Average the differences to get a concept axis. Then project any word onto this axis to get a continuous score. Words like "privatization" will score high on the right-economic pole; words like "redistribution" will score high on the left-economic pole. This lets you measure abstract concepts without building a classifier — you define the concept with a handful of seed words and let the embedding space generalize.

Tracking semantic change. Train embeddings on text from different time periods (e.g., decade by decade). Compare the nearest neighbors of a word across periods. Has the meaning of "immigration" shifted from a labor-market frame in the 1980s to a security frame in the 2010s? You can answer this by checking whether the nearest neighbors changed from words like "workforce" and "labor" to words like "security" and "border." The shift can be quantified as the cosine distance between the same word's vectors in different time periods.

Detecting bias. Embeddings absorb whatever biases exist in their training data. Caliskan, Bryson, and Narayanan (2017) showed in *Science* that standard word embeddings reproduce human implicit biases: male names are closer to career words, female names are closer to family words; European-American names are closer to pleasant words, African-American names to unpleasant words. This is both a problem (if you use biased embeddings in a downstream system, the bias transfers) and a research opportunity (you can use embeddings as a measurement tool to quantify cultural biases in different text sources).

Comparing discourses. Train separate embeddings on left-leaning and right-leaning news corpora. Then compare: which words have the most different neighborhoods across the two spaces? A word like "freedom" might be surrounded by "market," "enterprise," "regulation" in the right-leaning embedding but by "equality," "rights," "oppression" in the left-leaning one. This tells you something about how the same concept is *framed* differently across ideological communities.

Improving classifiers. Instead of feeding a bag-of-words vector into your Lecture 2 classifier, feed the averaged embedding vector of the document. This is a 300-dimensional dense vector instead of a 10,000-dimensional sparse one. It captures semantic content, handles synonyms gracefully, and often improves classification accuracy — especially when training data is small.

5.2 The bias problem in detail

The Caliskan et al. (2017) paper deserves special attention because it demonstrates something profound: word embeddings, trained on ordinary text from the internet, systematically reproduce the same biases measured by the Implicit Association Test (IAT) in psychological research.

Their method, called the **Word Embedding Association Test (WEAT)**, works as follows. Define two sets of target words (e.g., male names vs. female names) and two sets of attribute words

(e.g., career words vs. family words). Measure how strongly each set of target words is associated with each set of attribute words in the embedding space, using cosine similarity. If male names are systematically closer to career words than female names are, the embedding encodes a gender-career bias.

They found that standard pre-trained embeddings (trained on Google News, Wikipedia, etc.) reproduce biases around gender, race, and age — with effect sizes comparable to those measured in human subjects using the IAT. This means:

1. **If you use pre-trained embeddings in a downstream classifier** (e.g., for hiring decisions, content moderation, or risk assessment), the classifier may perpetuate or amplify these biases.
2. **If you are a researcher**, you can use embeddings as a measurement tool to study how biases vary across text sources, time periods, or communities. For example: are gender biases in parliamentary language stronger in the 1980s than in the 2020s? Do left-leaning and right-leaning media encode different racial biases?

The dual nature of this finding — embeddings as both a source of risk and a tool for studying bias — is important for political scientists to understand.

5.3 Caveats and limitations

Embeddings are powerful, but they have important limitations that you should keep in mind:

Static representations. Word2Vec, GloVe, and fastText give each word a single, fixed vector regardless of context. The word "bank" gets the same vector whether it refers to a financial institution or a riverbank. This is a fundamental limitation: polysemy (words with multiple meanings) is lost. We will fix this in Lecture 7 with BERT, which produces *contextual* embeddings — a different vector for "bank" in every sentence, depending on the surrounding words.

Domain mismatch. Embeddings trained on Wikipedia or news text may not capture how words are used in parliamentary debates, legal documents, or social media. The word "motion" means something very specific in parliamentary procedure that is quite different from its general-English meaning. If your domain is specialized, test whether pre-trained embeddings actually capture the meanings you care about — and consider training your own.

Hyperparameter sensitivity. The quality of embeddings depends on the embedding dimension (100, 200, 300?), the context window size (2, 5, 10 words?), the training algorithm (CBOW or Skip-gram?), and the size of the training corpus. Different choices produce somewhat different embeddings. For published research, document your choices and test whether your results are robust to alternatives.

Validation is essential. Just because an embedding produces a nice-looking nearest-neighbor list does not mean it is capturing the concept you care about. Rodriguez and Spirling (2022) provide practical guidance on how to validate embeddings for political science applications — their paper is recommended reading for this lecture.

5.4 Visualizing embeddings with PCA

Embeddings live in 300 dimensions. We cannot see 300 dimensions. **Principal Component Analysis (PCA)** is a dimensionality reduction technique that projects the high-dimensional vectors down to 2 (or 3) dimensions for visualization.

PCA works by finding the directions along which the data varies the most. The first principal component (PC1) is the direction of greatest variance; the second (PC2) is the direction of greatest remaining variance, perpendicular to the first. The 2D projection preserves as much of the original structure as possible, but it necessarily loses most of the information (you are going from 300 dimensions to 2 — that is a 99.3% reduction in dimensionality).

The result is a scatter plot where each word is a point. Words that are close in the original 300-dimensional space tend to be close in the 2D projection (though not always — the projection can introduce distortions). You will typically see interpretable clusters: cities cluster together, animals cluster together, political concepts cluster together.

PCA plots are excellent for **exploration** and **communication** — they let you show your audience the structure of the embedding space in a single image. But they should not be used for precise measurement. For quantitative analysis, work with the full 300-dimensional vectors and use cosine similarity directly.

---

6 Part V — The practical session

The practical session for this lecture uses pre-trained English fastText embeddings to explore the semantic structure of political vocabulary. The complete R code is provided in a separate script (`Lecture3_Practical.R`).

The walkthrough proceeds through six steps.

Step 1: Load pre-trained embeddings. We load the fastText English vectors (the file `cc.en.300.vec` from https://fasttext.cc, about 5 GB for the full file — you can use a subset of the top 100,000 words to keep it manageable). Each word is associated with a 300-dimensional vector.

Step 2: Find nearest neighbors. For any word — "democracy," "immigration," "welfare," "sovereignty" — we find the 10 words whose vectors are closest (in cosine similarity). You will see immediately that the nearest neighbors are semantically sensible: the neighbors of "democracy" include "democratic," "freedom," "governance," "elections." This is the distributional hypothesis in action.

Step 3: Compute cosine similarities. We compare specific word pairs: how similar is "climate" to "environment" (high), vs. "climate" to "taxation" (low)? We build a small similarity matrix for a set of political concepts and see which concepts are close and which are distant in the embedding space.

Step 4: Test analogies. We compute "Paris − France + Italy" and check whether the nearest neighbor is "Rome." We test political analogies: "Labour − worker + business = ?" You will see that analogies sometimes work beautifully and sometimes fail — which is valuable insight for understanding the limits of the method.

Step 5: Visualize with PCA. We select 20–30 politically relevant words (from domains like climate, economy, immigration, social policy, and cities as a control group), extract their embeddings, run PCA, and plot them in 2D with ggplot2. The clusters that emerge are usually interpretable and make a strong visual impression.

Step 6: Measure concept distances. We define a "left-right" axis using seed word pairs

(e.g., solidarity vs. competition, equality vs. enterprise, welfare vs. market). We project political vocabulary onto this axis and see which words score as "left" and which as "right." Reflect whether the scores match their intuitions and what the limitations of this approach are.

---

7 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 the sparse, count-based representations of Lectures 1–2.
2. **The distributional hypothesis** — "you shall know a word by the company it keeps" — is the idea that makes embeddings work. It underlies every modern NLP system, from Word2Vec to BERT to GPT.
3. **Cosine similarity** is the standard metric for comparing word vectors. It measures the angle between vectors, ignoring their magnitude, and ranges from -1 (opposite) to $+1$ (identical).
4. **Word2Vec** learns embeddings by training a neural network on a word prediction task. The embeddings are the network's hidden-layer weights — a byproduct of learning to predict context. **GloVe** takes a matrix-factorization approach to the same goal. **fastText** adds character n-grams, enabling it to handle morphology and out-of-vocabulary words.
5. **Embeddings encode social and political biases** absorbed from the training data. This is a risk when using embeddings in downstream applications, and a research opportunity when using them to measure cultural attitudes.
6. **Political science applications** include measuring concepts as directions (ideological scaling), tracking semantic change over time, comparing discourses across communities, and detecting framing differences.
7. **Embeddings are static**: each word has one vector, regardless of context. A word like "bank" gets the same vector whether it means a financial institution or a riverbank. This is the fundamental limitation that motivates contextual embeddings (BERT), which we will cover in Lecture 7.

---

8 Required reading

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

9 Recommended 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 to 7.

10 Other related reading

- 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*.
- Pennington, J., Socher, R., & Manning, C. D. (2014). "GloVe: Global Vectors for Word Representation." *EMNLP 2014*.

11 Looking ahead

In **Lecture 4**, we move from individual words to entire documents. We will cover three things: how to build document-level vectors from word embeddings (by averaging, TF-IDF weighting, or using Doc2Vec), how to discover latent topics in a corpus using unsupervised methods (LDA and the Structural Topic Model), and how to validate and interpret the topics you find. Topic models are the single most widely used exploratory text method in political science, and Lecture 4 is where we learn how to use them properly.