Word Embeddings

BA2: Digital Korea

Aron van de Pol

Korean Studies · Leiden University

2026-03-30

$$\text{King} - \text{Man} = \ ?$$

$$\text{King} - \text{Man} + \text{Woman} = \ ?$$

$$\text{김치} - \text{한국} + \text{일본} = \ ?$$

$$\text{노태우} - \text{보수} + \text{진보} = \ ?$$

Today's Agenda

1. Review: from counting to clustering
2. The limits of counting words
3. What are word embeddings?
4. Word2Vec: learning from context
5. Word arithmetic
6. Visualizing embeddings with t-SNE
7. From Word2Vec to BERT
8. Concept review & break
9. Hands-on: Embeddings in Orange

Review & Connection

Where We Are

Weeks 1–5: Description

- Preprocessing text
- Bag of Words / TF-IDF
- Word clouds, bar charts
- Concordance, subsetting

You learned to **count** words.

Weeks 7–10: Discovery

- Clustering ✓
- **Word embeddings** (today)
- Classification
- Topic modeling

Now the computer learns what words **mean**.

The Limits of Counting

The Problem with BoW

BoW and TF-IDF treat **every word as completely independent.**

Word pair	**BoW says**	**We know**
경제 / 경기	unrelated	closely related
대통령 / 정부	unrelated	connected
king / queen	unrelated	same domain

BoW has **no concept of meaning.** Every word is just a separate column.

Why This Matters

Imagine two presidential speeches:

- **Speech A** uses “경제 성장” (economic growth)
- **Speech B** uses “경기 발전” (economic development)

In BoW, these share **zero** vocabulary. Cosine similarity ≈ 0.

But they are about the **same topic.**

What Are Word Embeddings?

The Core Idea

> "You shall know a word by the company it keeps."
>
> — J.R. Firth (1957)

What Does That Mean?

- "The **economy** grew by 3% this year."
- "The **GDP** grew by 3% this year."

"Economy" and "GDP" keep appearing with the same neighbors.

A model can learn they are related.

From Sparse to Dense

BoW vector (sparse)

- One dimension per word
- Thousands of dimensions
- Most values are 0

`경제 → [0, 0, 1, 0, ..., 0]`

Embedding vector (dense)

- Fixed small size (100–300)
- Every value is filled
- Encodes **meaning**

`경제 → [0.21, −0.55, ..., 0.44]`

What Could Dimensions Mean?

Imagine we describe words with a few simple properties:

	alive	**furry**	**legs**	**royal**	**human**
cat	0.93	0.91	0.82	0.04	0.08
dog	0.91	0.78	0.88	0.06	0.12
fish	0.84	0.03	0.05	0.02	0.03
king	0.87	0.11	0.85	0.97	0.95
queen	0.88	0.12	0.84	0.96	0.94
car	0.06	0.08	0.14	0.11	0.22

What Can We See?

- **Cat** and **dog** are similar — both alive, furry, have legs
- **Fish** is alive but very different from cat
- **King** and **queen** are almost identical — we need **more dimensions** to tell them apart
- **Car** is nothing like any of the others

From 5 Dimensions to 300

With 5 dimensions we can already see patterns. But 5 is not enough — king and queen look the same.

Real embeddings use **100–300 dimensions**, learned automatically from text.

More dimensions = more nuance = the model can capture gender, formality, topic, era, and relationships we might not even have words for.

What Real Vectors Look Like

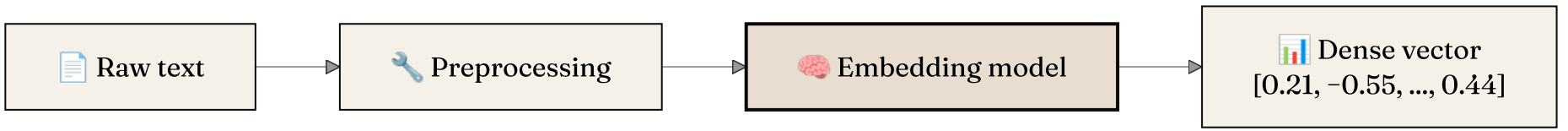
▶ Show code

```
info = json.load(open(f"{CACHE}/en_vector_info.json"))
print(f"Vocabulary size: {info['vocab_size']:,} words")
print(f"Vector dimensions: {info['vector_size']}")
for w, v in info['samples'].items():
    print(f"\n  '{w}' → [{v['first3'][0]:.2f}, {v['first3'][1]:.2f}, {v['first3'][2]:.2f}, ..., {v['last']:.2f}]")
```

```
Vocabulary size: 400,000 words
Vector dimensions: 100

  'economy' → [-0.19, 1.02, 1.08, ..., -0.20]

  'politics' → [-0.54, 0.45, 0.65, ..., 0.65]

  'soccer' → [0.84, 0.52, 0.64, ..., 0.23]
```

No single dimension means “furry” or “royal.” The meaning is spread across **all** dimensions together.

The Big Picture



The **embedding model** is the new piece.

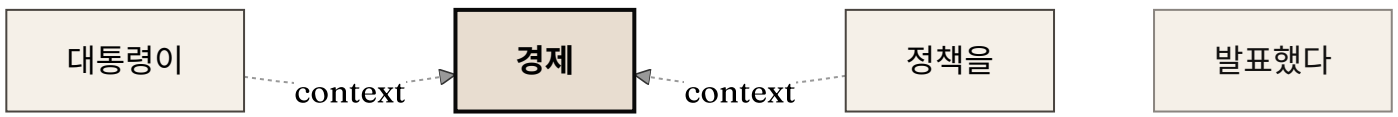
You **download** it — someone else already read billions of words to create it.

Word2Vec

How Does the Model Learn?

The model reads text and looks at which words appear **near** each other.

It slides a **context window** across every sentence:



Same neighbors → similar vectors.

The Training Process

If two words always hang out with the same crowd, the model decides they must be similar.

You Don't Train These Yourself

Researchers already trained models on massive text collections.

We use their **pre-trained embeddings** — like downloading a dictionary someone else compiled.

Model	Trained on	Words
GloVe (English)	Wikipedia + news	400,000
fastText (Korean)	Web crawl	2,000,000
Word2Vec (Google)	Google News	3,000,000

You download the model. Then you look up any word → get its vector.

Skip-gram vs. CBOW

For your reference — you do not need to memorize this

Skip-gram	**CBOW**
Given the target, predict the context.	Given the context, predict the target.
“경제” → predict “대통령이”, “정책을”	“대통령이 ___ 정책을” → predict “경제”
Better for **rare** words.	**Faster** to train.

Both produce the same output: a lookup table where every word maps to a dense vector.

Finding Similar Words

▶ Show code

```
similar = json.load(open(f"{CACHE}/en_similar_economy.json"))
print("Words closest to 'economy':\n")
for word, score in similar:
    bar = '' * int(score * 40)
    print(f"  {word:<14} {score:.3f}  {bar}")
```

```
Words closest to 'economy':

  economic       0.828  ████████████████████████████████
  growth         0.795  ███████████████████████████████
  recession      0.769  ██████████████████████████████
  economies      0.755  ██████████████████████████████
  recovery       0.749  ████████████████████████████
  downturn       0.746  ████████████████████████████
  slowdown       0.737  ████████████████████████████
  slowing        0.728  ████████████████████████████
```

Nobody told the model what “economy” means. It learned from **context alone.**

Word Arithmetic

The Famous Example

Word embeddings can do something BoW never could: **analogy by arithmetic**.

$$\overrightarrow{\text{king}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}} \approx \ ?$$

The Model's Answer

▶ Show code

```
result = json.load(open(f"{CACHE}/en_king_analogy.json"))
print("king − man + woman = ?\n")
for word, score in result:
    bar = '' * int(score * 40)
    print(f"  {word:<14} {score:.3f}  {bar}")
```

```
king − man + woman = ?

  queen          0.770  ████████████████████████████████
  monarch        0.684  ████████████████████████████
  throne         0.676  ████████████████████████████
  daughter       0.659  ███████████████████████████
  princess       0.652  ███████████████████████████
```

The model was **never taught** this rule. It discovered it from patterns in text.

It Works for Other Relationships Too

▶ Show code

```
1  analogies = json.load(open(f"{CACHE}/en_more_analogies.json"))
2  for a in analogies:
3      print(f"  {a['label']:<32} = {a['word']:<12} ({a['score']:.3f})")
```

```
paris − france + germany          = berlin       (0.885)
walking − walked + swam           = swimming     (0.707)
japan − france + paris            = tokyo        (0.899)
```

Why Does Arithmetic Work?

The vectors encode **relationships as directions** in space.

- The direction from “man” → “king” = the **royalty** direction
- Apply that same direction to “woman” → you land near “queen”

Korean Analogies

Does the same work in Korean?

서울 – 한국 + 일본 = **?**

아버지 – 남자 + 여자 = **?**

Korean Analogies: The Results

▶ Show code

```
ko_data = json.load(open(f"{CACHE}/ko_analogies.json"))
for item in ko_data:
    print(f"  {item['label']} =")
    for word, score in item['results']:
        bar = '█' * int(score * 30)
        print(f"    {word:<8} {score:.3f}  {bar}")
    print()
```

```
서울 - 한국 + 일본 =
  도쿄       0.632  ██████████████████
  東京       0.471  █████████████
  강북       0.463  ████████████

아버지 - 남자 + 여자 =
  어머니      0.667  ███████████████████
  외할아버지    0.490  ██████████████
  할아버지     0.484  ██████████████
```

One More Analogy

$$\overrightarrow{\text{doctor}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}} \approx \ ?$$

A Note on Bias

$$\overrightarrow{\text{doctor}} - \overrightarrow{\text{man}} + \overrightarrow{\text{woman}} \approx \overrightarrow{\text{nurse}}$$

The model learned this from text written by humans — including our stereotypes.

Bias in the Numbers

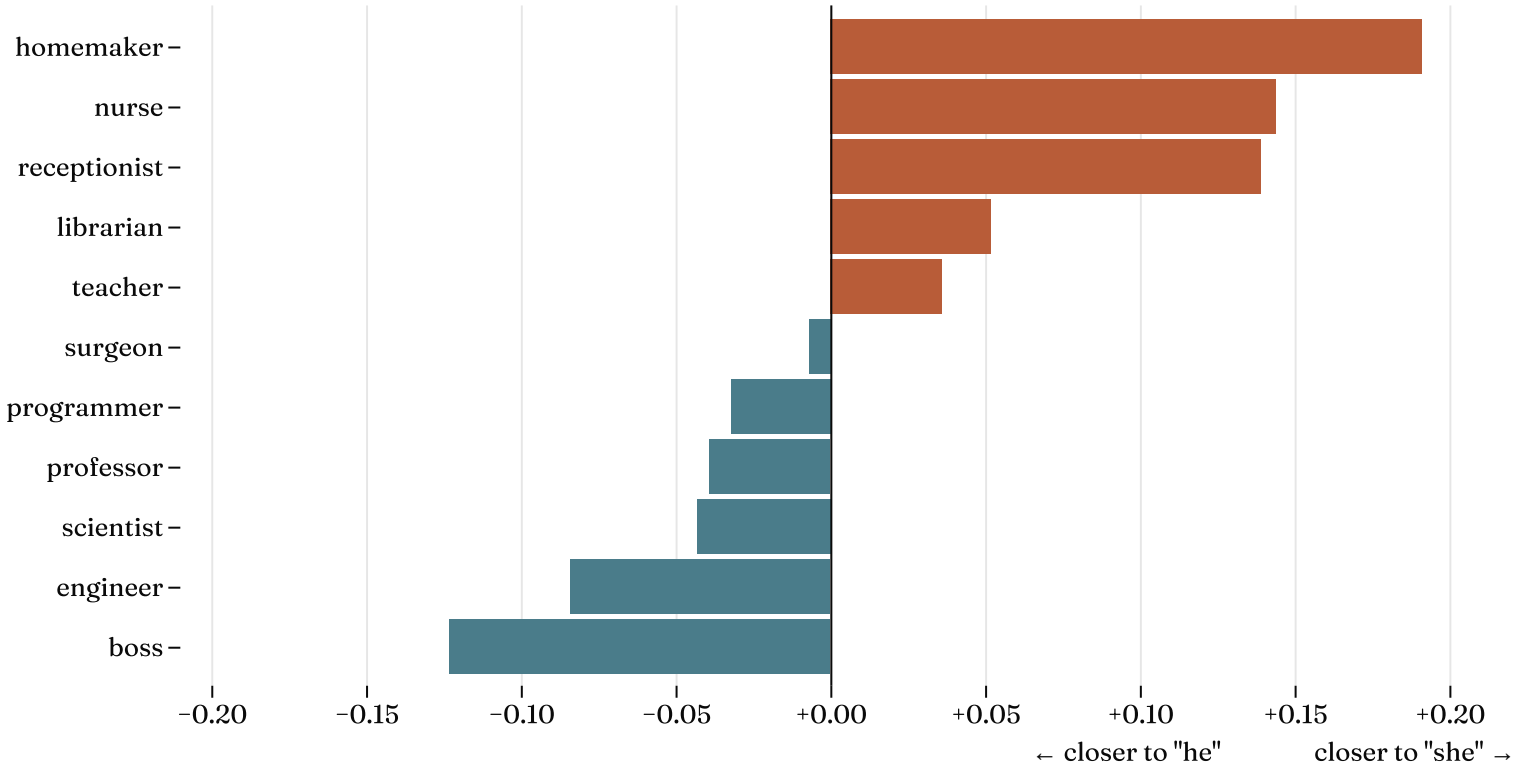


Figure 1

The Same in Korean

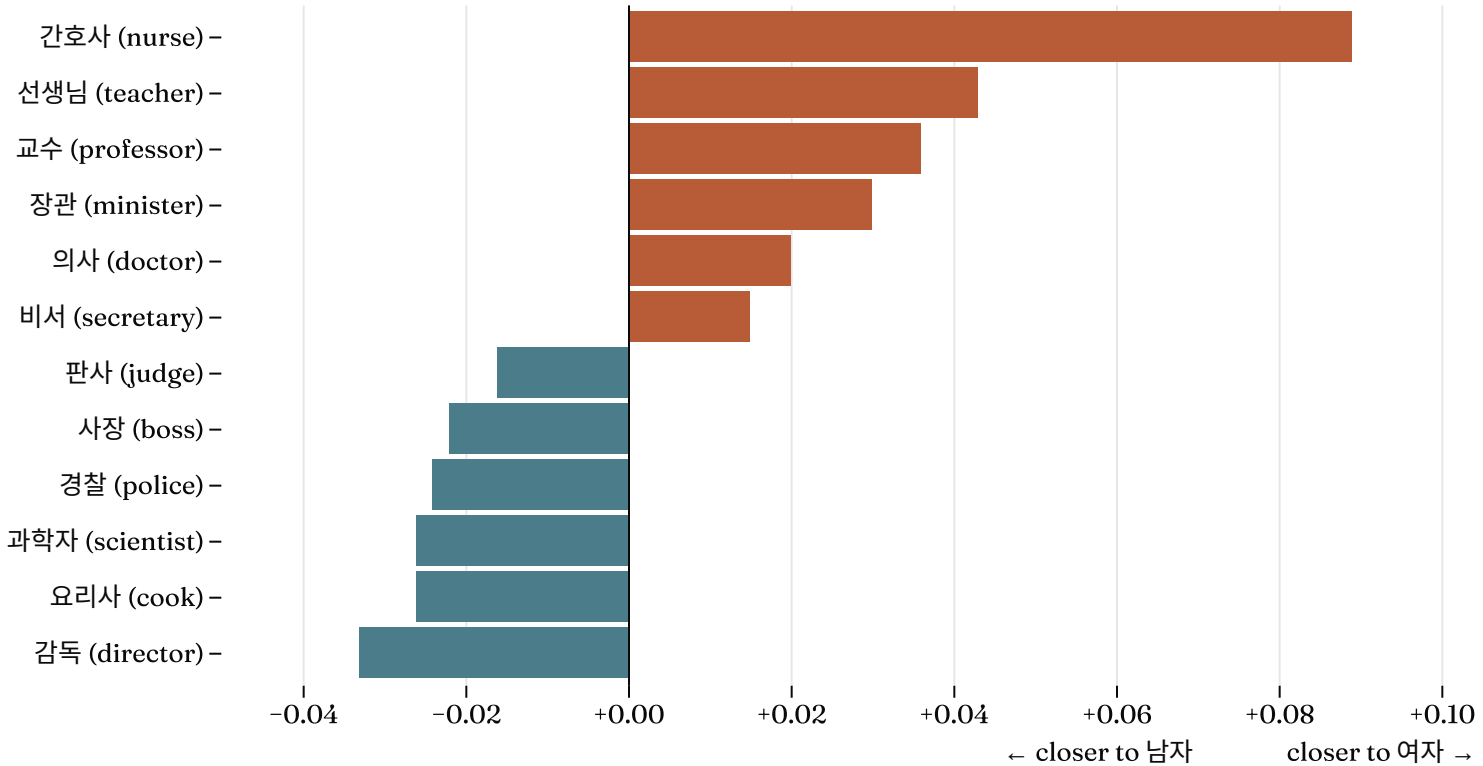


Figure 2

Why Bias Matters

The models were never told “nurses are women” or “engineers are men.”

They **discovered** these patterns from how words are used in text.

Embeddings are powerful, but they are **not neutral**.

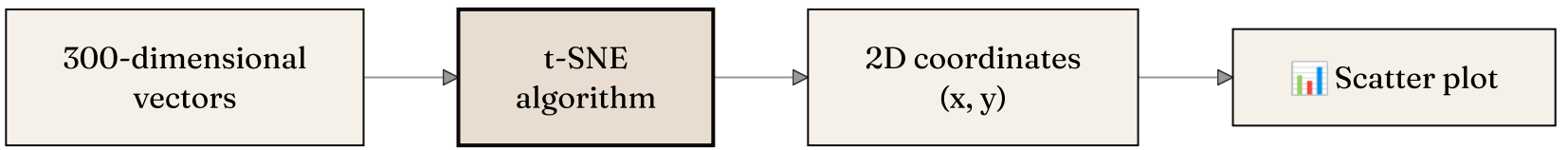
They absorb the biases present in their training data.

Visualizing Embeddings

The Problem: Too Many Dimensions

Each word is a vector with 100–300 numbers. We cannot picture that.

t-SNE projects them down to **2 dimensions** so we can plot them.



How to Read t-SNE

- **Nearby** points = similar words
- **Distant** points = different words
- The **groupings** matter — the exact axis numbers do not

English Words in 2D

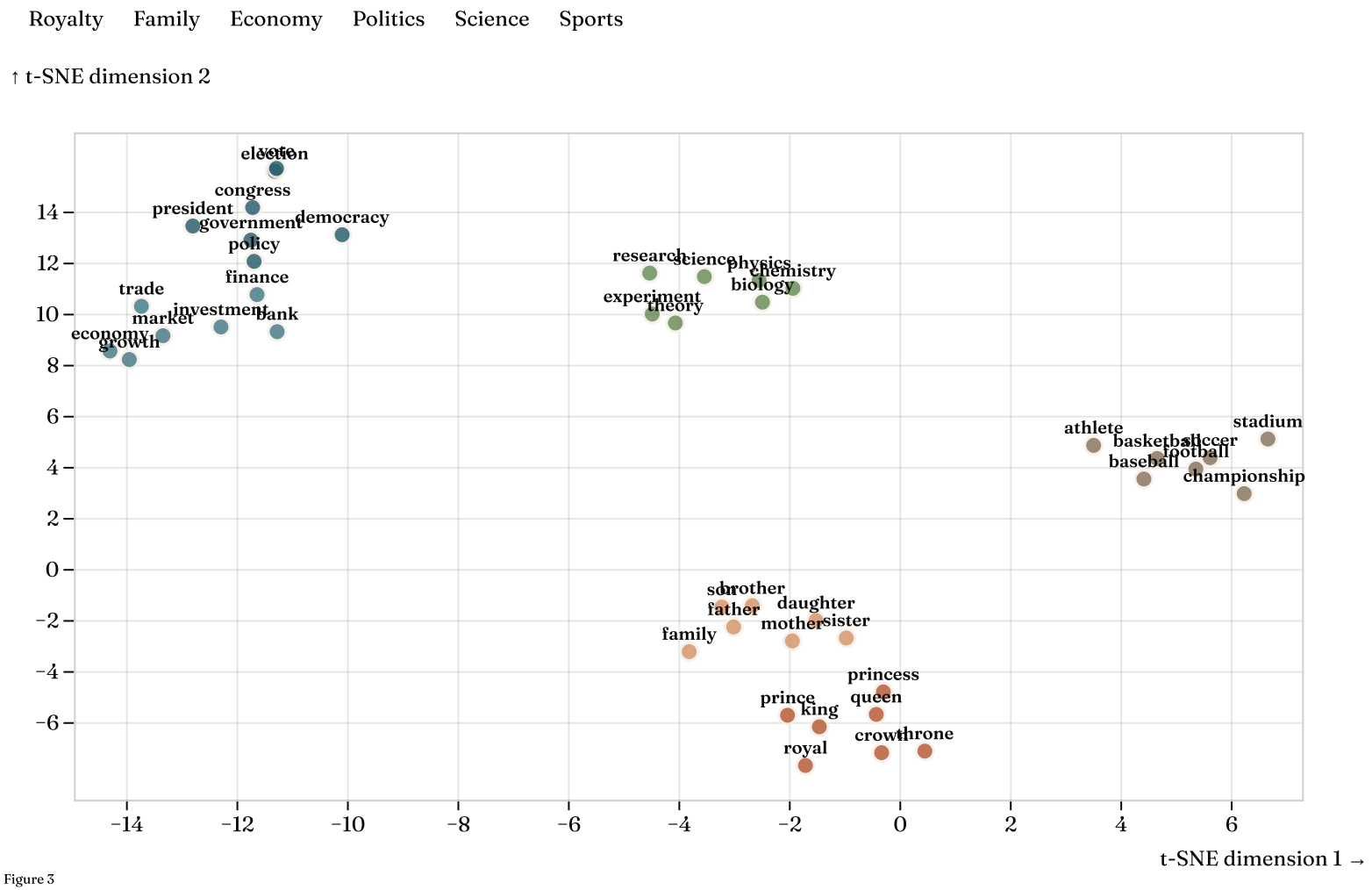


Figure 3

What Do You See?

The model **discovered** these groupings from context — nobody told it what “sports” or “economy” means.

Does This Remind You of Something?

Last week we clustered **documents** by vocabulary similarity.

Now we are clustering **words** by meaning similarity.

What is the same? What is **different**?

Analogy in 2D

Royalty Family Geography

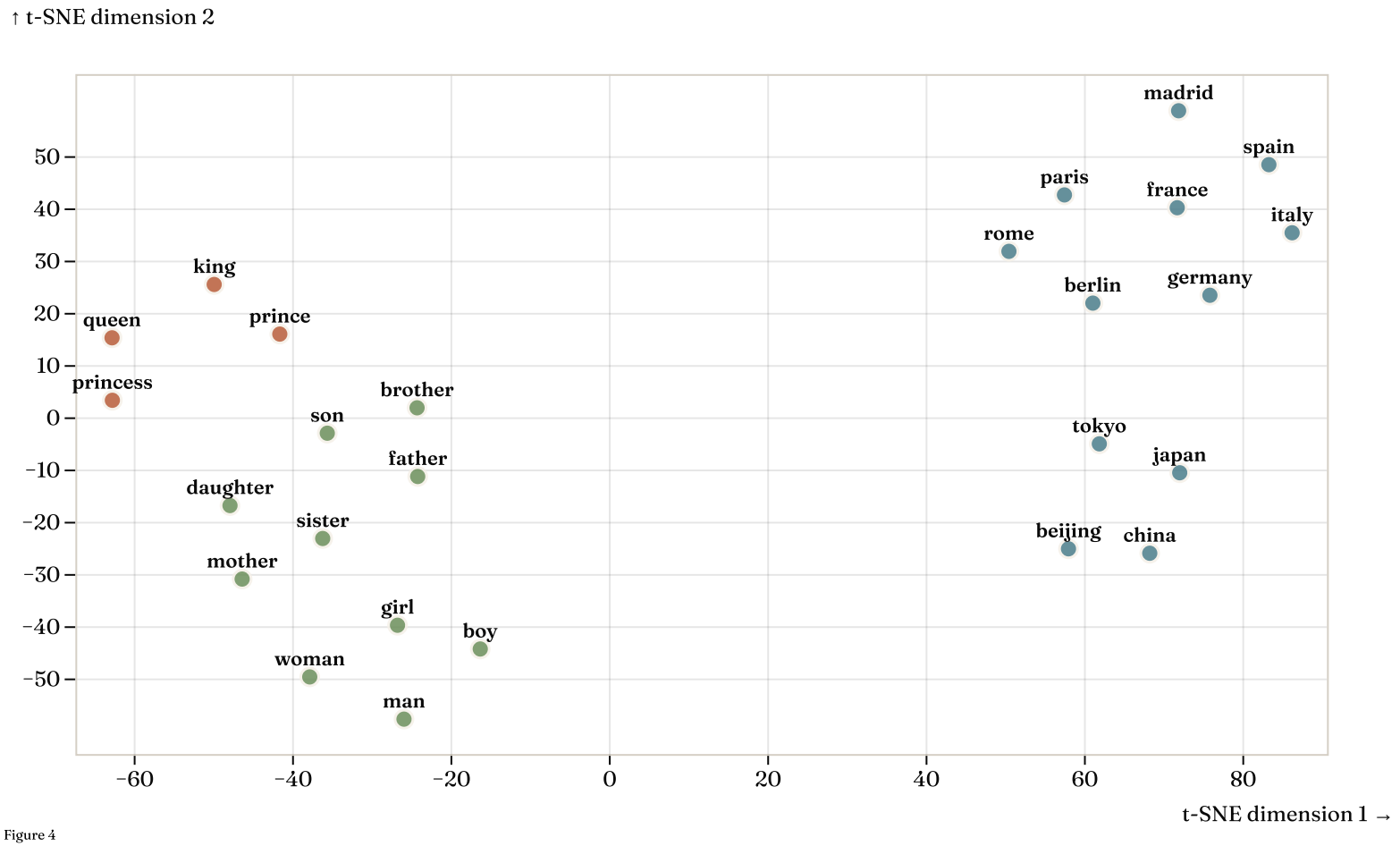


Figure 4

Korean Words in 2D

Economy Politics Society Relations

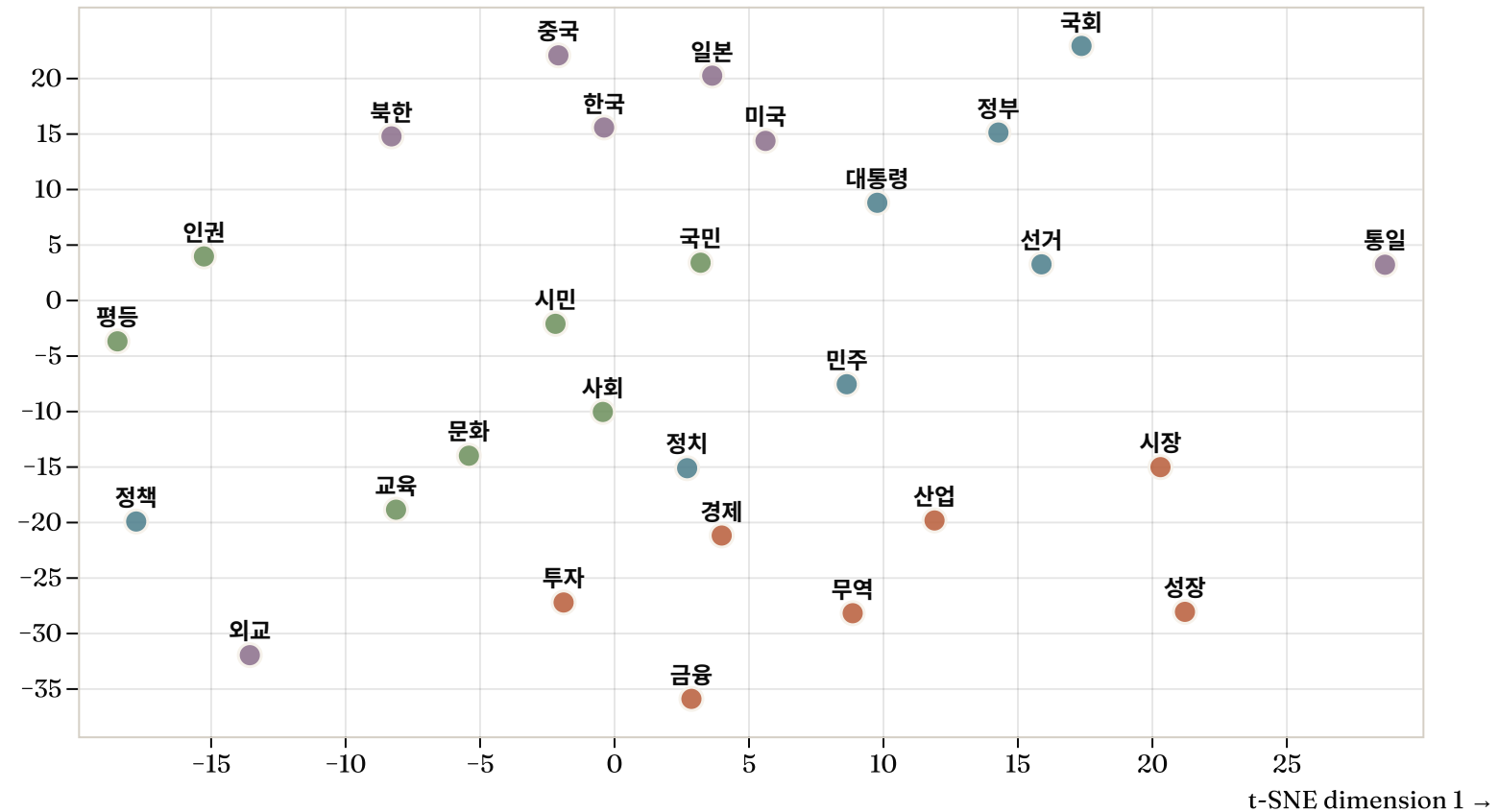


Figure 5

From Word2Vec to BERT

The Limitation of Word2Vec

Word2Vec gives every word **one fixed vector**, no matter the context.

But many words have **multiple meanings**.

English: "bank"

- "I went to the **bank** to deposit money" → financial institution
- "We walked along the river **bank**" → edge of a river
- Word2Vec gives "bank" the **same vector** in both sentences!

The Same Problem in Korean

Korean: “배”

- “**배**가 고프다” → stomach
- “**배**를 타다” → boat
- “**배**를 먹다” → pear

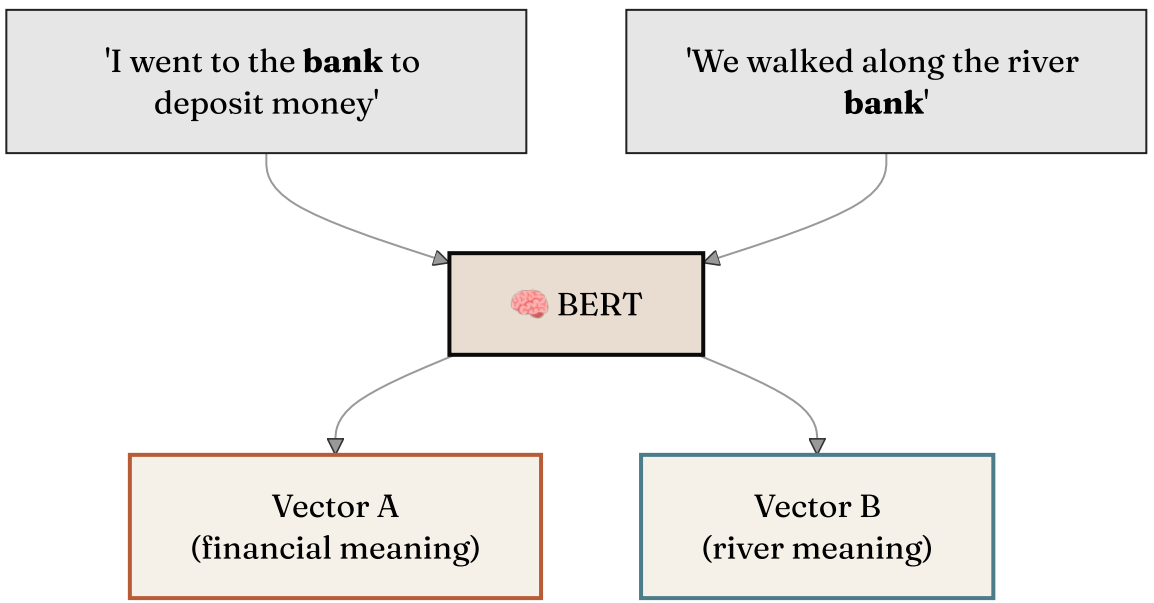
Three completely different meanings — but Word2Vec produces **one** vector for “배.”

We need a model that reads the **whole sentence** before deciding what a word means.

BERT: Contextual Embeddings (2018)

BERT reads the **entire sentence** first, then assigns each word a vector.

The same word gets a **different vector** depending on its context.



BERT in Action

▶ Show code

```
bert = json.load(open(f"{CACHE}/bert_bank.json"))
print('BERT embeddings for "bank" in 3 sentences:\n')
print(f'  💰 "deposit money"  vs  🏞️ "river bank":    {bert["financial_vs_river"]:.3f}')
print(f'  💰 "deposit money"  vs  💰 "approve mortgage": {bert["financial_vs_financial"]:.3f}')
print(f'  🏞️ "river bank"    vs  💰 "approve mortgage": {bert["river_vs_financial"]:.3f}')
```

```
BERT embeddings for "bank" in 3 sentences:

  💰 "deposit money"  vs  🏞️ "river bank":    0.545
  💰 "deposit money"  vs  💰 "approve mortgage": 0.820
  🏞️ "river bank"    vs  💰 "approve mortgage": 0.581
```

Another Example: “apple”

▶ Show code

```
apple = json.load(open(f"{CACHE}/bert_apple.json"))
print('BERT embeddings for "apple" in 3 sentences:\n')
print(f'  🍎 "ate a delicious apple"  vs  🍎 "picked a ripe apple":  {apple["fruit_vs_fruit"]:.3f}')
print(f'  🍎 "ate a delicious apple"  vs  📱 "Apple released iPhone": {apple["fruit_vs_company"]:.3f}')
print(f'  📱 "Apple released iPhone"  vs  🍎 "picked a ripe apple":  {apple["company_vs_fruit"]:.3f}')
```

```
BERT embeddings for "apple" in 3 sentences:

  🍎 "ate a delicious apple"  vs  🍎 "picked a ripe apple":  0.855
  🍎 "ate a delicious apple"  vs  📱 "Apple released iPhone": 0.489
  📱 "Apple released iPhone"  vs  🍎 "picked a ripe apple":  0.451
```

Seeing It: English BERT in t-SNE

The same word “bank” and “apple” — each used in 3 sentences per meaning. BERT puts same-meaning uses **together**.

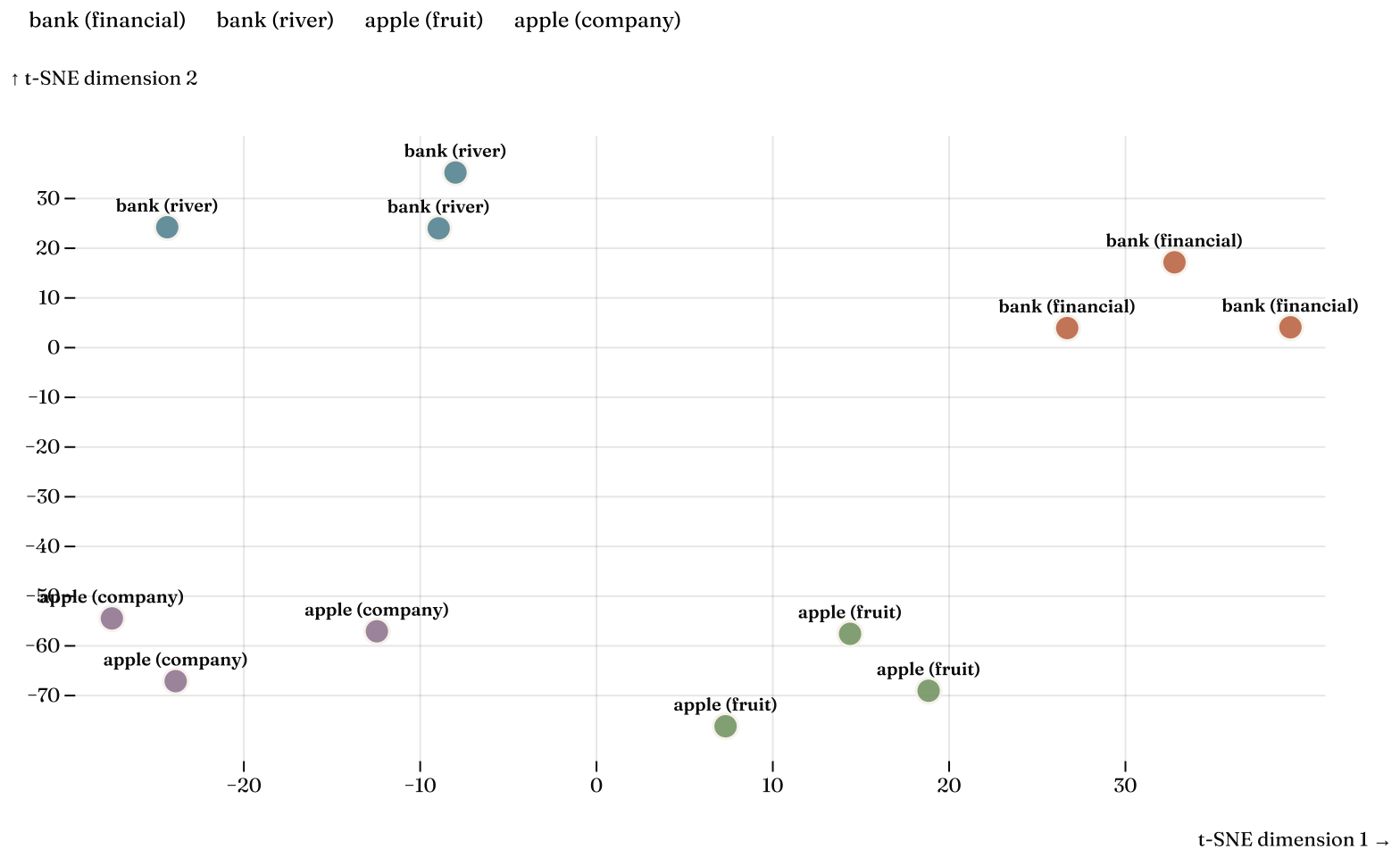


Figure 6

Seeing It: Korean BERT in t-SNE

Same word “배” (3 meanings) and “눈” (2 meanings) — Korean BERT (KLUE) clusters them by meaning.

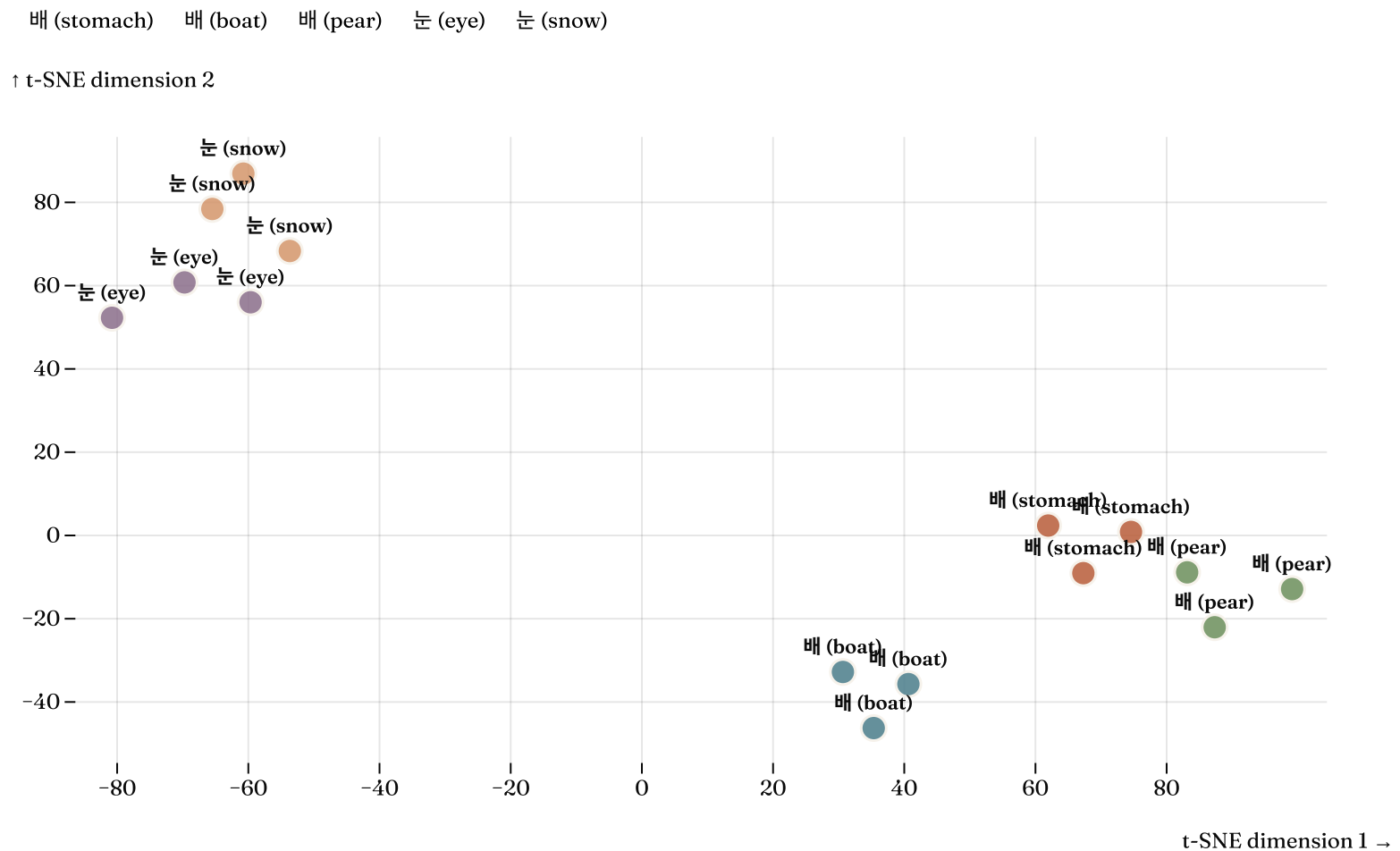


Figure 7

Word2Vec vs. BERT

	Word2Vec (2013)	**BERT (2018)**
Vector type	One fixed vector per word	Different vector per context
Handles ambiguity?	No	Yes
Trained how	Predict nearby words	Predict masked words in sentences
Size	Small and fast	Large, more compute
Best for	Word relationships, analogies	Classification, search, NLU

Three Levels of Embedding

The Big Idea

Everything we have seen so far is about representing **meaning as numbers**.

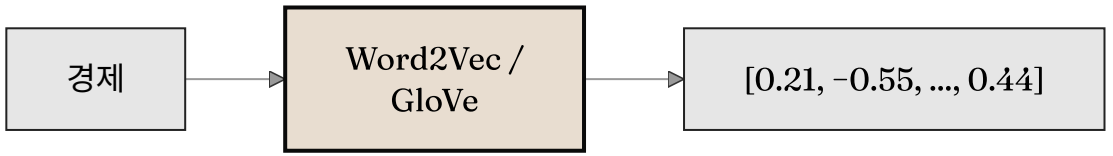
But meaning exists at different levels:

Level	**What gets a vector?**	**Example**
Word	A single word	“경제” → [0.21, −0.55, ...]
Sentence	A full sentence	“경제가 성장했다” → [0.14, 0.33, ...]
Document	An entire text	A whole speech → [0.08, −0.12, ...]

Each level builds on the one before it.

Word Embeddings

One vector per **word**. This is what Word2Vec and GloVe produce.



“경제” always maps to the same numbers, regardless of sentence.

Sentence Embeddings

One vector per **sentence**. BERT produces these.

The same word gets a **different vector** in different sentences.

Document Embeddings

One vector per **document** (a speech, an article, a whole text).

Two common approaches:

Average word vectors

Look up each word's embedding, take the **mean**.

Simple. Works with Word2Vec.

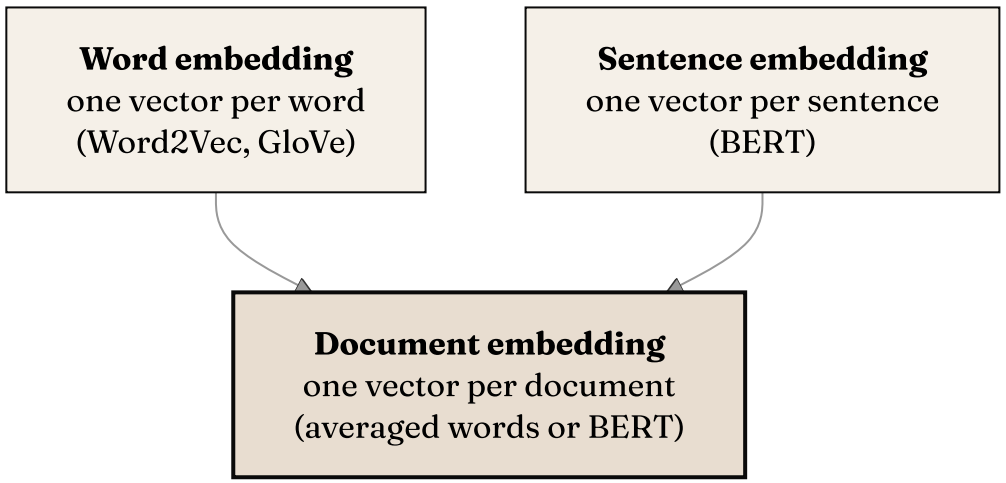
BERT [CLS] token

Feed the text into BERT, take the special **[CLS]** vector.

More powerful. Captures sentence structure.

Both produce a single vector that represents the whole document.

How They Connect



Word and sentence embeddings are the **building blocks.**

Document embeddings are what we use for analysis.

Putting It All Together

What Changes with Embeddings?

	BoW / TF-IDF	**Word Embeddings**
Represents	Word frequency	Word meaning
Vector size	Vocabulary size (thousands)	Fixed (100–768)
Sparse or dense?	Sparse (mostly zeros)	Dense (every value matters)
Synonyms	Unrelated columns	Similar vectors
Levels	Only documents	Words, sentences, documents
Learned from	Your corpus only	Huge external corpus

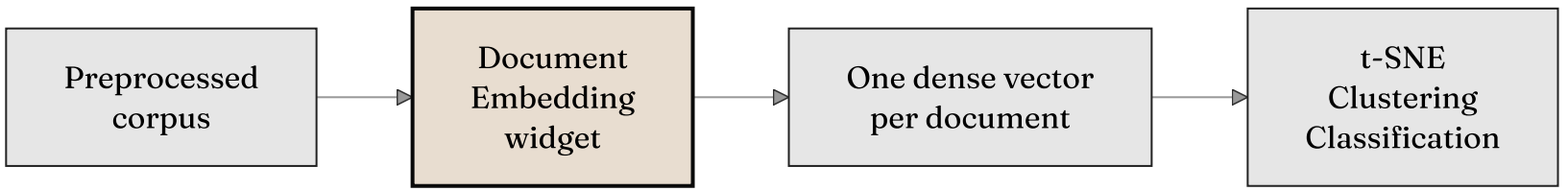
What Embeddings Give Us

1. **Synonyms** — 좋다 and 훌륭하다 are close in vector space
2. **Analogies** — vector arithmetic reveals relationships
3. **Context** — the same word gets different vectors in different sentences (BERT)
4. **Transfer learning** — knowledge from billions of words helps your small corpus

> **Why this matters for our corpus**
>
> Our 749 presidential speeches is a small dataset. Embedding models were trained on **billions** of words. We bring all that knowledge to our analysis for free.

Embeddings in Orange

How does this work in practice? The **Document Embedding** widget does it for you.

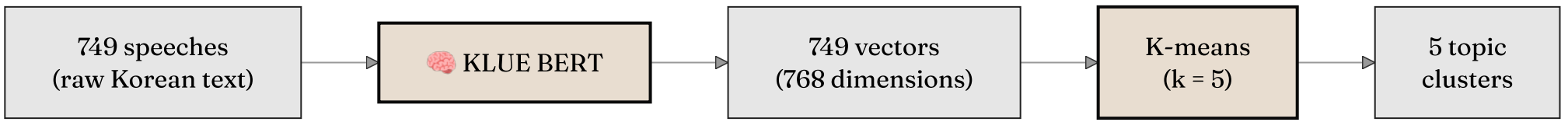


The widget takes each document, runs it through a pre-trained model, and produces a single vector you can use for all downstream analysis.

Applying It: Presidential Speeches

What We Did

We embedded all **749 presidential speeches** using Korean BERT (KLUE/bert-base).



Then we asked: **do speeches cluster by president, or by topic?**

Five Topics Emerged

BERT found **thematic** clusters — not presidential ones:

Topic	Distinctive keywords	n
COVID & Governance	확진자, 방역, 백신, 반부패	146
Press & Q&A	질문, 보니까, 굉장히, 아니고	86
Memorial & National	졸업생, 겨레의, 동지, 민주	198
Events & Congratulatory	창립, 영상메시지, 창간	190
Diplomacy	아세안, 국왕, 말레이시아, 인도네시아	129

These are topics of the **office**, not of any individual president.

Speeches in t-SNE Space

Each dot is one speech, colored by the topic cluster BERT discovered:

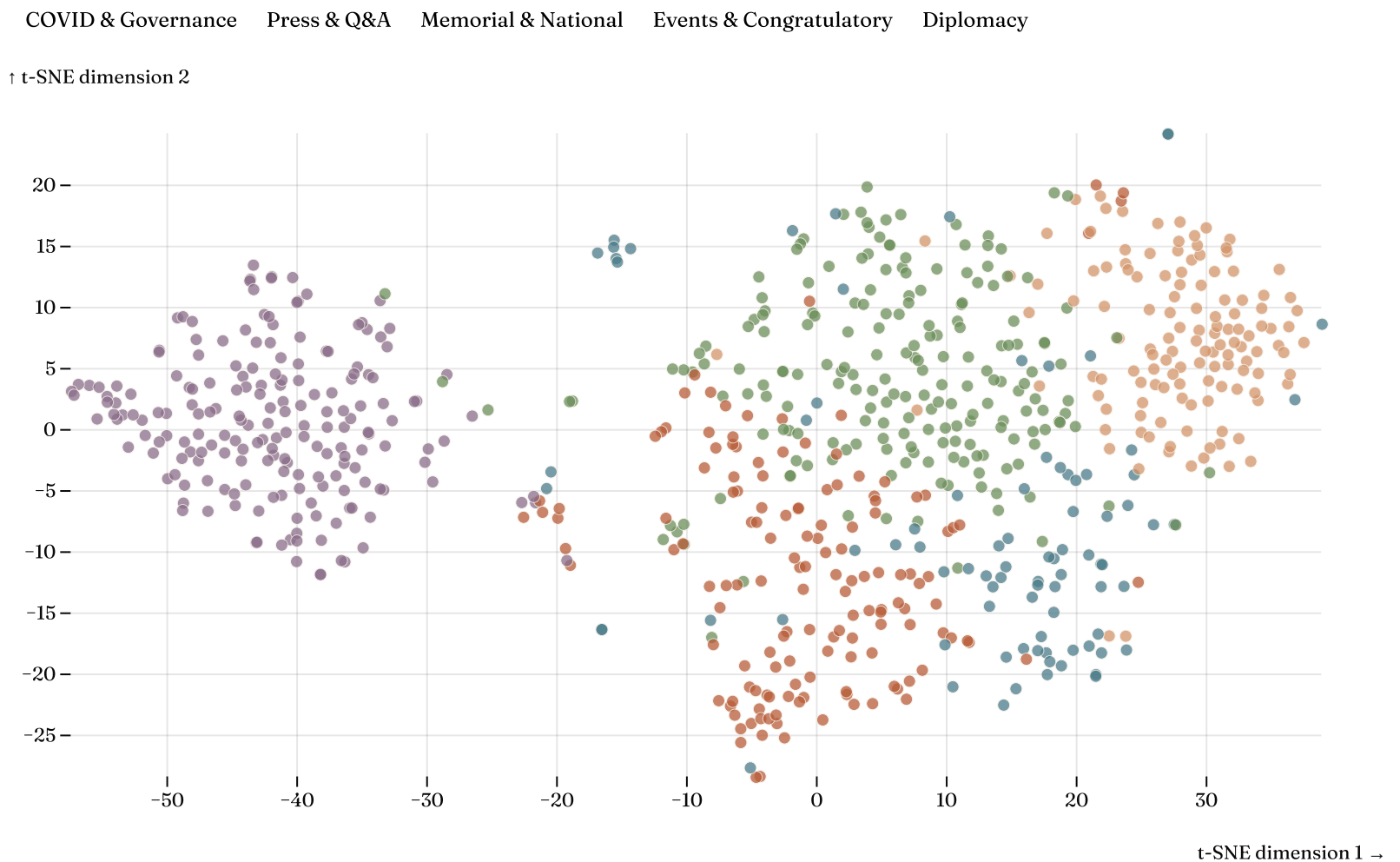


Figure 8

Who Talks About What?

Each president's speeches broken down by topic — the **mix** varies, but every president covers most topics:

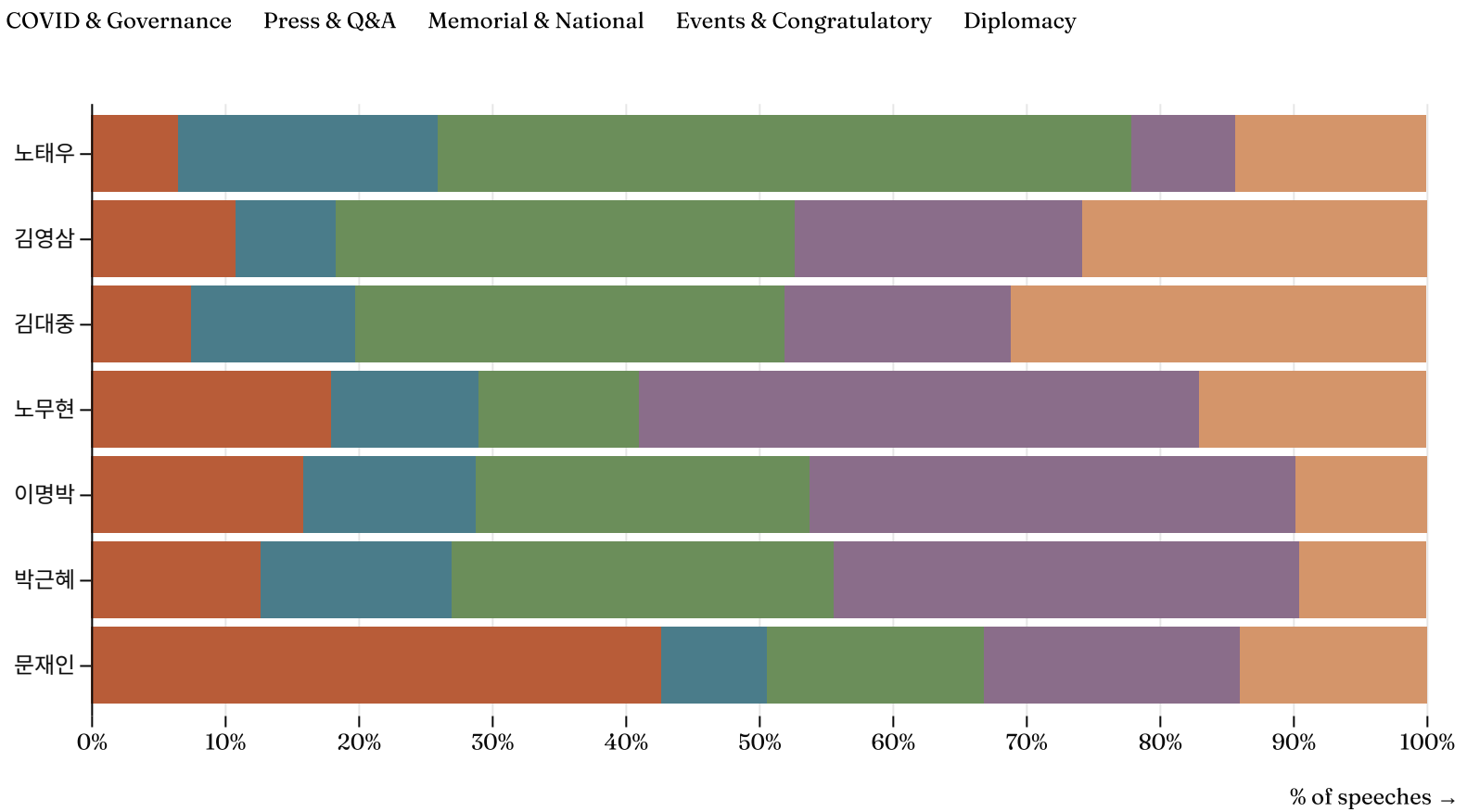


Figure 9

What the Data Tells Us

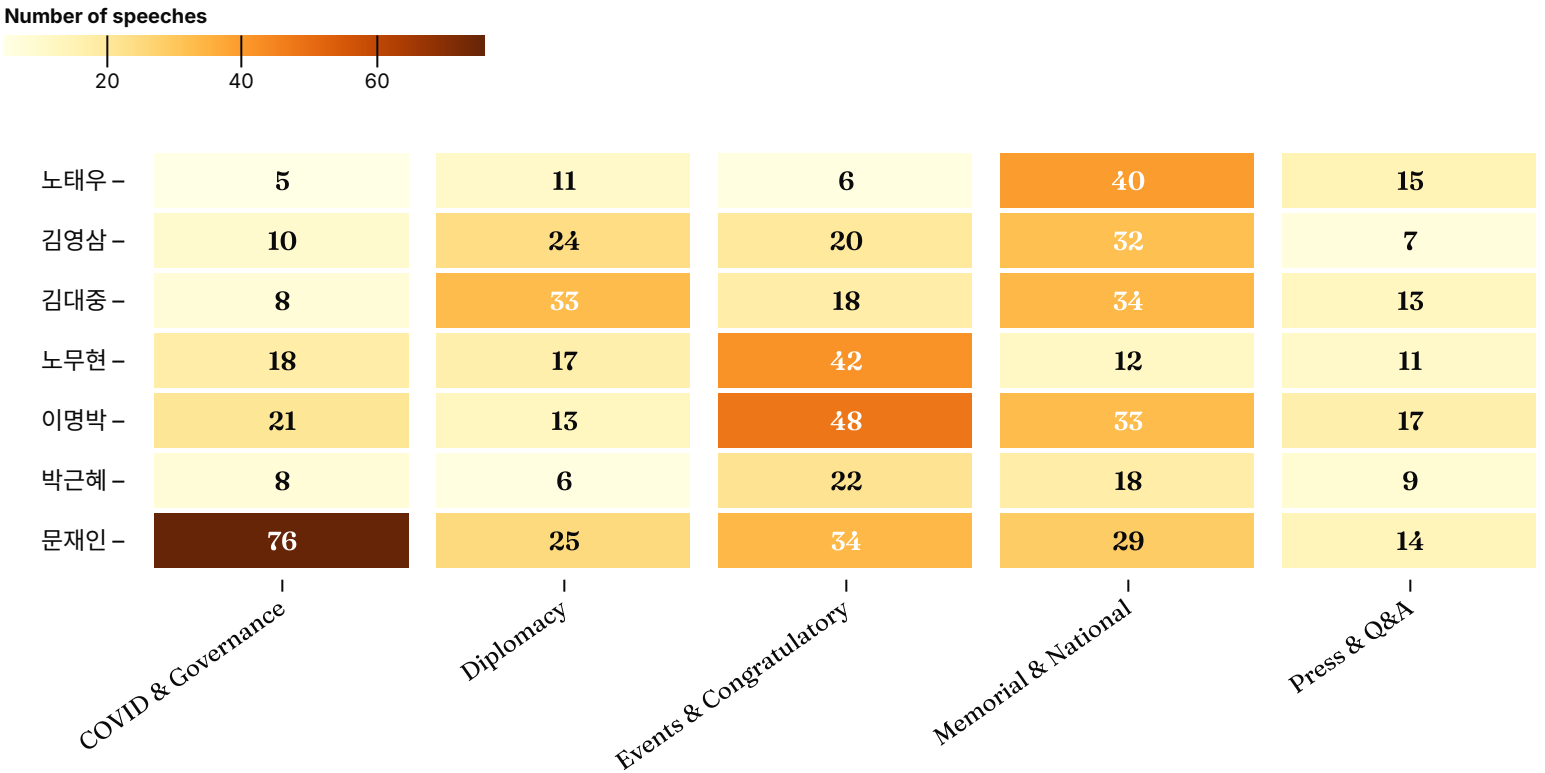


Figure 10

Reading the Results

Key patterns:

- **문재인**: 42% of speeches in the COVID/governance cluster — his presidency overlapped with the pandemic
- **노태우**: Over half in memorial/national identity — early democratic era, focus on national unity
- **김대중 & 김영삼**: Strongest diplomacy presence — active foreign policy periods

> **This is what embeddings give you**
>
> Embeddings cluster speeches by **meaning**, not just vocabulary. The model understands that a COVID press conference and a COVID policy meeting are about the same thing — even when they use different words.

Concept Review

Concept	What it means
Word embedding	A dense vector representing a word's meaning
Distributional hypothesis	Words in similar contexts → similar meanings
Word2Vec	Learns embeddings by predicting context words
Dense vs. sparse	Embeddings are dense; BoW is sparse
Word arithmetic	king − man + woman ≈ queen
t-SNE	Projects high-dimensional vectors to 2D for visualization
BERT	Contextual — same word, different vector per sentence
Document embedding	Average word embeddings → one vector per document

Break

We will resume in 10 minutes.

Hands-On: Embeddings in Orange

Activity Overview

Work in pairs. We will walk through this together.

Part 1: Build an embedding pipeline in Orange

Part 2: Cluster document embeddings with K-Means and Hierarchical Clustering

What you need

- Orange Data Mining open on your laptop
- Presidential speeches CSV (from the Data page)
- Preprocessing script for your OS (from the Data page)

Part 1: Building the Pipeline

Extend your existing workflow:

0. **File** → **Corpus** → **Python Script** (preprocessing) — same as before
1. **Preprocess Text:** tokenize by whitespace, load stopword list
2. Connect to **Document Embedding** widget
    - Select a pre-trained model
3. Connect to **t-SNE** widget to visualize

Part 1: What to Look For

In the t-SNE visualization:

- Color the points by **president**
- Do speeches by the same president cluster together?
- Do you see any thematic groupings?

> **Check your work**
>
> If you see clear clusters, the embeddings are capturing real patterns in the speeches.

Part 2: Clustering Document Embeddings

From Document Embedding, build **two clustering paths**:

Path A: K-Means

Document Embedding → K-Means (k = 5) → t-SNE

Color by cluster assignment.

Path B: Hierarchical

Document Embedding → Distances → Hierarchical Clustering

Inspect the dendrogram.

Part 2: Discussion

Compare the two clustering methods:

- Do K-Means and Hierarchical Clustering find the same groups?
- Color t-SNE by **president**, then by **cluster** — which structure is stronger?
- Look at the dendrogram — at what level do meaningful groups emerge?

> **Think about it**
>
> K-Means forces you to choose **k** in advance. Hierarchical clustering shows structure at every level. Which is more useful for exploratory analysis?

Assignment & Looking Ahead

This week:

- Complete the Orange workflow from today
- DataCamp assignment (see Brightspace)

Next week: Classification I

- We use embeddings as **features** to predict categories
- Can we predict which president gave a speech from its embedding?