Lesson 39: Word Embeddings

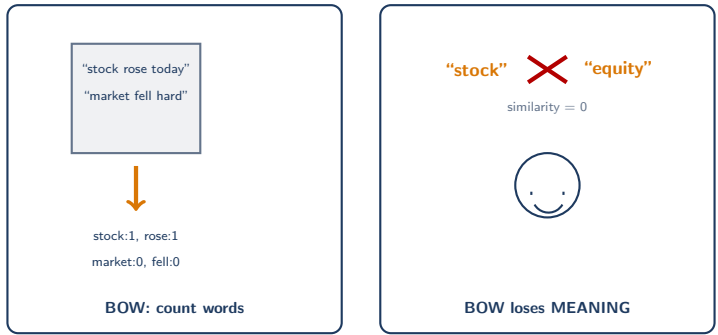
Data Science with Python – BSc Course

Data Science Program

BSc Course

45 Minutes

Previously: Turning Text into Numbers



L38 gave us numbers, but: "stock" and "equity" are treated as completely unrelated. Can we do better?

BOW and TF-IDF count words but ignore that synonyms share meaning

Learning Objectives

The Problem: Bag of Words treats every word as independent – "king" and "queen" are as different as "king" and "banana". How do we capture the MEANING of words numerically?

After this lesson, you will be able to:

1. **Explain** why dense embeddings beat sparse word counts (Remember)
2. **Compare** Word2Vec architectures: Skip-gram vs CBOW (Understand)
3. **Use** pre-trained Word2Vec and GloVe via gensim (Apply)
4. **Evaluate** word similarity and analogies in financial context (Analyze)

Finance Application: Semantic search in financial documents, similar company detection

king − man + woman = ?

The most famous equation in NLP:

$$\vec{\text{king}} - \vec{\text{man}} + \vec{\text{woman}} \approx \vec{\text{queen}}$$

What does this mean?

- Words are represented as vectors (lists of numbers)
- Vector arithmetic captures semantic relationships
- The "royalty" direction is preserved when you swap gender

Finance version:

$$\vec{\text{Apple}} - \vec{\text{technology}} + \vec{\text{banking}} \approx \vec{\text{JPMorgan}}$$

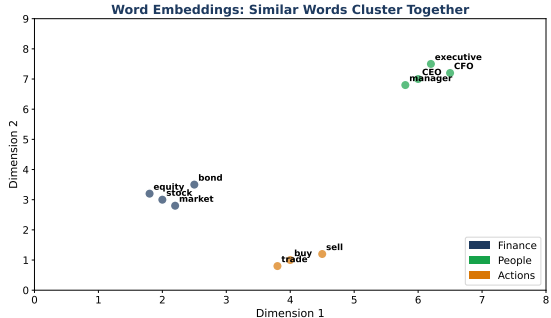
How can simple vectors capture such rich meaning?

This slide's equation launched a revolution in how we represent language

What IS a Word Embedding?

GPS Coordinates for Words:

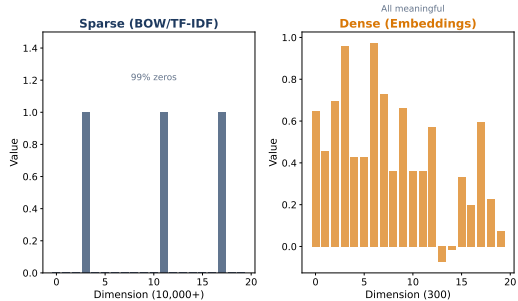
- GPS gives every location a coordinate (latitude, longitude)
- Embeddings give every word a coordinate in "meaning space"
- Similar words = nearby coordinates



Embedding = a dense vector that encodes a word's meaning as coordinates

Sparse vs Dense: Why Embeddings Win

BOW/TF-IDF: 50,000D, mostly zeros, no synonym similarity
Embeddings: 100–300D, every value meaningful, similar words = similar vectors

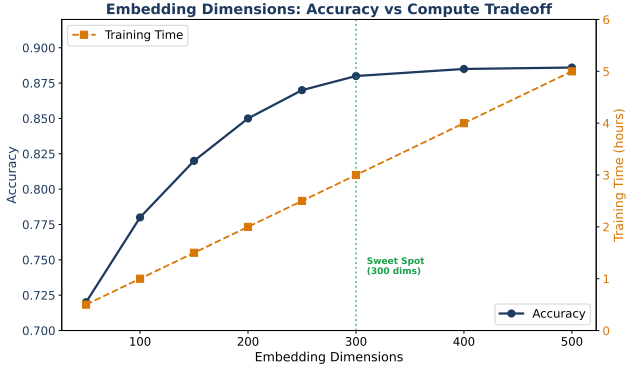


Dense beats sparse: smaller, faster, and captures synonymy

Embedding Dimensions: How Many?

Trade-off between capacity and efficiency:

- Too few (50D): cannot capture nuanced meanings
- Sweet spot (200–300D): good for most tasks
- Too many (1000D+): overfitting, diminishing returns

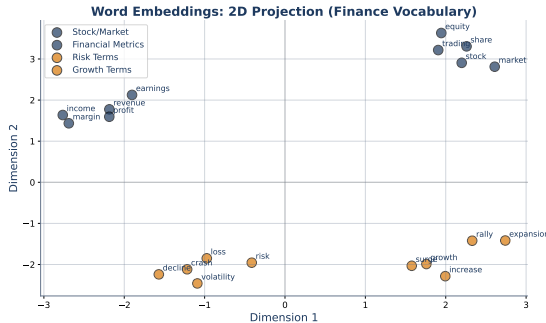


300 dimensions is the standard choice for Word2Vec and GloVe

2D Embedding Visualization

Word Clusters in Semantic Space

- Similar words cluster together (financial terms, emotions, etc.)
- Semantic relationships preserved as geometric relationships
- t-SNE reduces 300D vectors to 2D for visualization

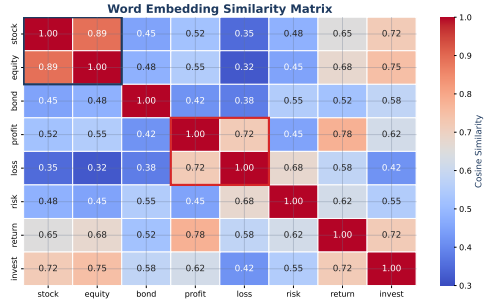


t-SNE projection reveals semantic structure in embedding space

Word Similarity Matrix

Measuring Semantic Relatedness

- Cosine similarity between word vectors quantifies relatedness
- "stock" and "equity" = high similarity; "stock" and "banana" = low
- Used for: synonym detection, search, recommendation



Cosine similarity: $\cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{||\vec{a}|| \; ||\vec{b}||}$. Range $[-1, 1]$.

Word2Vec: You Know a Word by Its Neighbors

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

- Words appearing in similar contexts get similar vectors
- Word2Vec learns by reading millions of sentences

Example – what word fits?

- "The _____ rose 5% after earnings" $\rightarrow$ stock, share, price
- "The _____ fell sharply on Monday" $\rightarrow$ market, index, dollar

Result: 100–300 dimensional vectors for every word in vocabulary

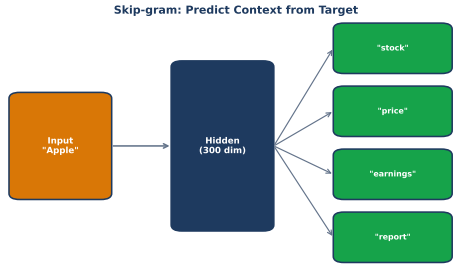
---

Word2Vec: learn word meaning from context. "Tell me your neighbors, I'll tell you who you are."

Skip-gram: Predict Context from Center

Given center word, predict surrounding words

- Input: “stock” → Output: “The”, “rose”, “5%”
- Better for rare words and small datasets
- Most popular Word2Vec variant

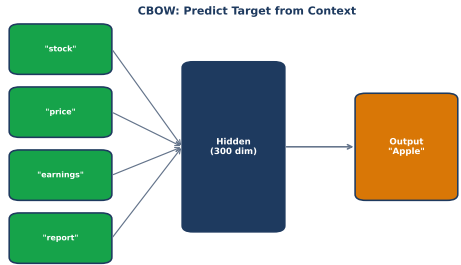


Skip-gram: “Given this word, what words appear nearby?”

CBOW: Predict Center from Context

Given surrounding words, predict center word

- Input: “The”, “rose”, “5%” → Output: “stock”
- Faster to train than Skip-gram
- Better for frequent words and large datasets

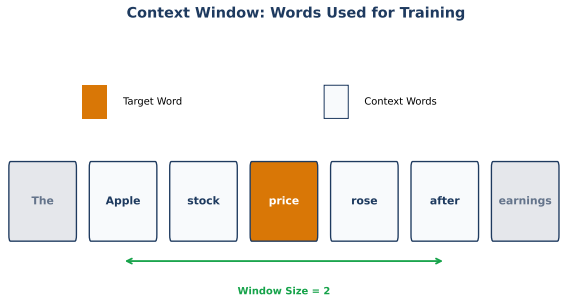


CBOW: “Given context, which word is missing?” – fill-in-the-blank at scale

Context Window Size

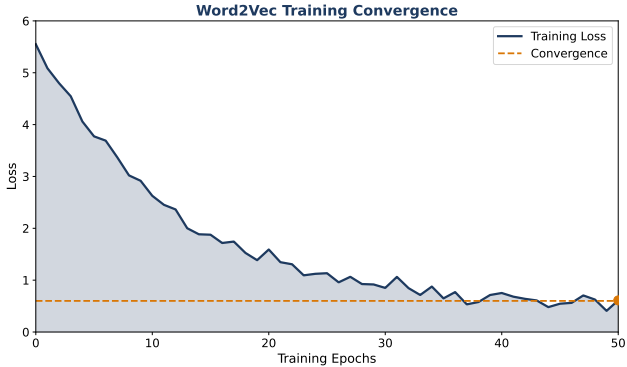
How many neighbors to consider?

- Small window (2–3): captures syntactic patterns (verb/noun relationships)
- Large window (5–10): captures semantic/topical patterns



Window size controls what kind of similarity the embeddings capture

Training Loss Convergence



Loss decreases as the model learns better word representations

Checkpoint: Test Your Understanding

Q1: Why are word embeddings better than Bag of Words for measuring word similarity?

Q2: What is the difference between Skip-gram and CBOW?

Q3: If "stock" and "equity" have cosine similarity 0.85, what does that tell you?

Answers: Q1: Dense vectors capture synonymy; BOW treats all words as orthogonal. Q2: Skip-gram predicts context from word; CBOW predicts word from context. Q3: Very similar meaning.

Pre-trained Embeddings: Don't Train From Scratch

Three major pre-trained models:

- **Word2Vec:** Google News, 3M words, 300D
- **GloVe:** Wikipedia + web text, 400K words, 300D
- **FastText:** Handles out-of-vocabulary words via subwords

Why use pre-trained?

- Trained on billions of words (you don't have that much data)
- Capture general language knowledge out of the box
- Fine-tune on your domain if needed

Loading with gensim:

```
import gensim.downloader as api
model = api.load('word2vec-google-news-300')
```

Pre-trained embeddings: free, powerful, ready to use. Always start here.

gensim: Essential Operations

Get a word vector (300 numbers):

```
vec = model['stock'] # shape:  (300,)
```

Find nearest neighbors:

```
model.most_similar('stock', topn=5)
```

→ [('stocks', 0.79), ('equities', 0.72), ('shares', 0.70), ...]

Measure similarity:

```
model.similarity('stock', 'equity') → 0.68
model.similarity('stock', 'banana') → 0.04
```

Word analogies:

```
model.most_similar(positive=['king', 'woman'],
                   negative=['man'], topn=1)
```

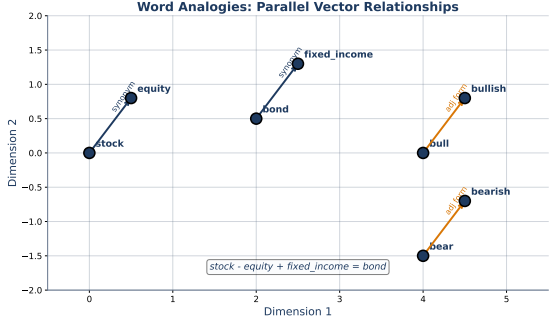
→ [('queen', 0.71)]

Four operations: lookup, nearest neighbors, similarity, analogies

Word Analogies: Vector Arithmetic

Relationships encoded as vector offsets:

- $\vec{\text{king}} - \vec{\text{man}} + \vec{\text{woman}} \approx \vec{\text{queen}}$
- The "gender" direction is a consistent vector offset
- Works for countries/capitals, tenses, comparatives

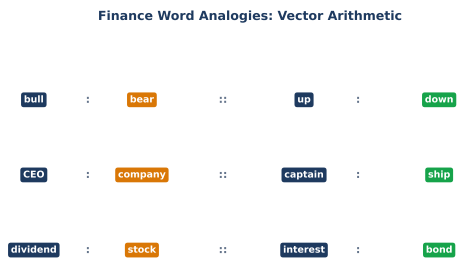


Analogy = parallel vector offset. Same direction, different starting point.

Finance: Sector Analogies

Financial word analogies reveal industry structure:

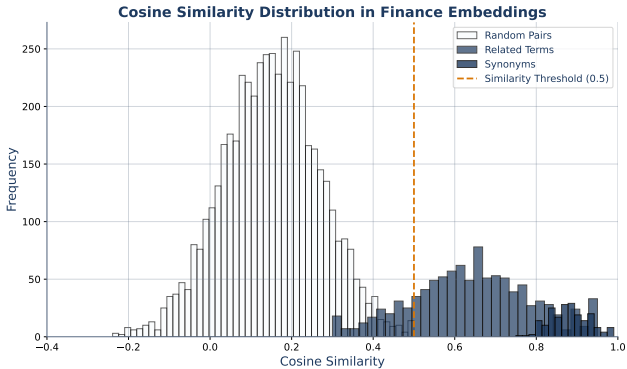
- "Apple" − "technology" + "banking" ≈ "JPMorgan"
- "CEO" − "company" + "country" ≈ "president"
- "bull" − "optimism" + "pessimism" ≈ "bear"



Embeddings learn financial structure without being told about sectors

Most Word Pairs Have Low Similarity

- Random word pairs: cosine similarity near 0 (unrelated)
- Synonyms and related terms: similarity 0.5–0.9
- This distribution validates that embeddings capture real meaning

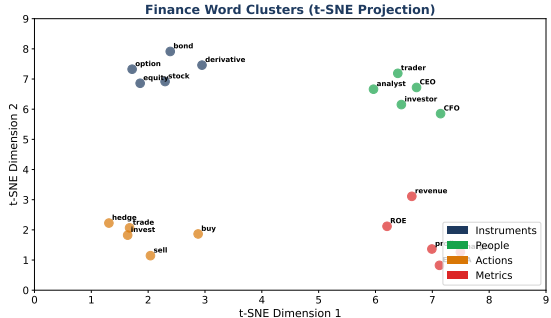


Most word pairs have low similarity; synonyms cluster high

Finance: Sector Clusters in Embedding Space

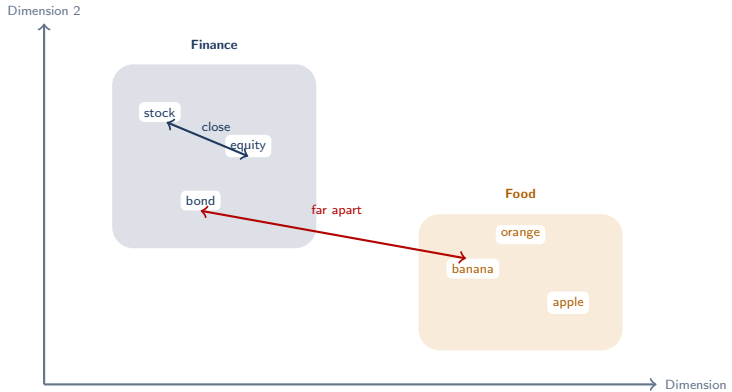
Sector-related words cluster automatically:

- Banking terms group together (loan, mortgage, deposit)
- Tech terms group together (software, algorithm, cloud)
- Embeddings discover industry structure from raw text



No labels needed – embeddings discover sector structure from co-occurrence

The Geometry of Meaning



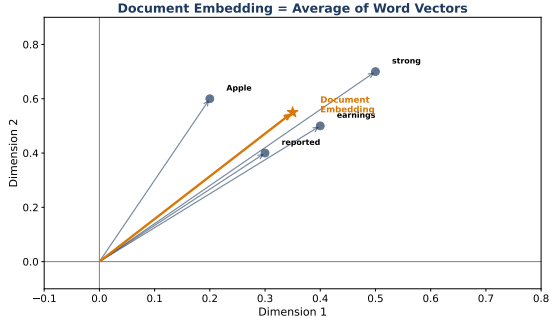
Key insight: Distance in vector space ≈ difference in meaning. Related words cluster; unrelated words are far apart.

Embedding space organizes words by meaning, not alphabetically

From Word Vectors to Document Vectors

Three steps to embed a document:

1. **Look up** each word's pre-trained vector (300D)
2. **Average** all word vectors: $\vec{d} = \frac{1}{n} \sum_{i=1}^{n} \vec{w}_i$
3. **Use as features** in any classifier (logistic regression, SVM, etc.)

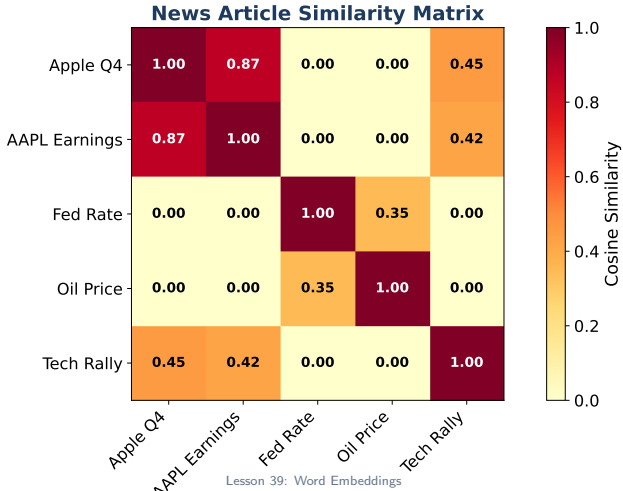


Average word vectors = simple but effective document representation

News Article Similarity

Finding related financial news with embeddings:

- Convert each headline to a document vector (average word vectors)
- Compute cosine similarity between all pairs
- Cluster similar news for automated topic detection



Document Embedding Pipeline

Complete code pattern:

Step 1 – Embed documents:

```
def doc_vector(text, model):
  words = text.lower().split()
  vecs = [model[w] for w in words if w in model]
  return np.mean(vecs, axis=0) if vecs else np.zeros(300)
```

Step 2 – Build feature matrix:

```
X = np.array([doc_vector(doc, model) for doc in docs])
```

Step 3 – Classify:

```
X_train, X_test, y_train, y_test = train_test_split(X, y)
clf = LogisticRegression().fit(X_train, y_train)
print(clf.score(X_test, y_test))
```

Advantage over TF-IDF: fixed 300D regardless of vocabulary size, captures synonymy

Hands-On Exercise (25 min)

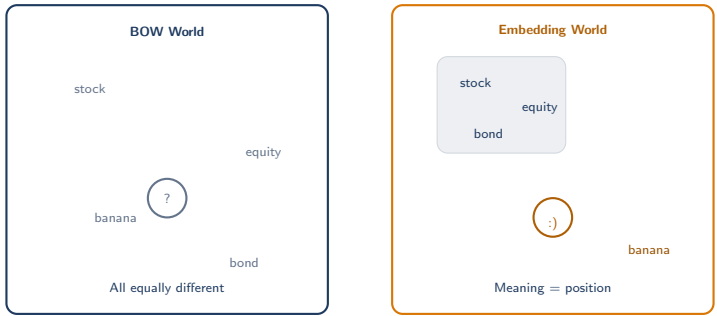
Task: Explore Financial Word Embeddings

1. Load pre-trained Word2Vec with gensim
2. Find 10 most similar words to "stock", "risk", "profit"
3. Test financial analogies: "bull" $-$ "positive" $+$ "negative" $=$?
4. Create document vectors for 20 financial news headlines
5. Cluster documents using K-Means on their vectors

Deliverable: Similarity results + 2D t-SNE visualization of document clusters.

Extension: Compare Word2Vec vs GloVe for financial terminology

Words as Coordinates in Meaning-Space



The leap: From "words as IDs" to "words as coordinates in meaning-space."

Embeddings turned NLP from pattern matching into geometry

Key Takeaways

What you should remember:

1. **Embeddings = dense vectors** that encode semantic meaning (300D)
2. **Word2Vec** learns from context (Skip-gram or CBOW)
3. **Cosine similarity** measures how related two words are
4. **Pre-trained models** (Word2Vec, GloVe, FastText) are free and powerful

Connection to Module 8:

- L37 (Preprocessing): clean text before embedding
- L38 (BOW/TF-IDF): sparse baseline; embeddings are the dense upgrade
- L40 (Sentiment): embeddings power better sentiment classifiers

---

Memory: Embedding = word as numbers. Similar words = similar vectors. Use pre-trained.

Preview: Reading Between the Lines

We can now represent words as vectors with meaning.

Next question: What **opinion** does the text express?

Coming in L40 – Sentiment Analysis:

- VADER: rule-based sentiment scoring (fast, no training)
- FinBERT: finance-specific deep learning sentiment
- From text sentiment to trading signals

Module 8 progression:

Clean Text (L37) $\rightarrow$ Count Words (L38) $\rightarrow$ **Meaning** (L39) $\rightarrow$ Opinion (L40)

Next lesson: L40 Sentiment Analysis – when Elon tweets, stocks move