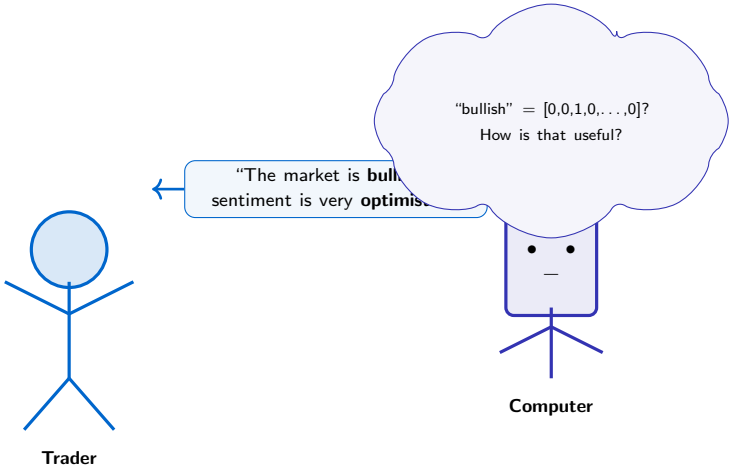
Word Embeddings

Teaching Machines to Understand Language

Methods & Algorithms

MSc Data Science – Spring 2026

How Would You Teach a Computer What "Bullish" Means?



Words are just strings – how do we make them mathematical?

Learning Objectives

After this lecture, you will be able to:

1. **Derive** the Skip-gram objective and explain negative sampling
2. **Analyze** embedding spaces using cosine similarity and analogy tasks
3. **Evaluate** Word2Vec vs GloVe vs contextual embeddings
4. **Apply** embeddings to financial sentiment analysis

LOs at Bloom's levels 4–5: analyze, evaluate, apply, derive.

What Is a Word Embedding?

A **word embedding** maps words to dense vectors where **similar words have similar vectors**.

- Each word $\rightarrow$ real-valued vector in $\mathbb{R}^d$
- Typical d: 50–300 dimensions
- Famous analogy: $\texttt{king} - \texttt{man} + \texttt{woman} \approx \texttt{queen}$

Key Terms

- **Embedding**: learned vector representation
- **Dense vector**: most entries non-zero
- **Cosine similarity**: measures vector closeness

Embeddings turn words into numbers that capture meaning.

Why Not Just Use One-Hot Encoding?

One-Hot Encoding

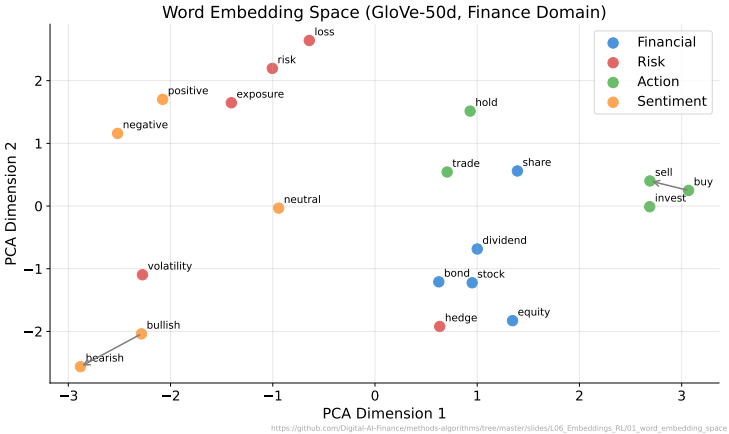
- Dimension = vocabulary size ($V \sim 100$K)
- Every word equally distant from every other
- "bullish" and "optimistic" are orthogonal

Dense Embeddings

- Dimension = 50–300 (compact)
- Similar words are nearby in vector space
- "bullish" $\approx$ "optimistic" by cosine similarity

One-hot: sparse and meaningless distances. Embeddings: dense and semantic.

What Does the Embedding Space Look Like?



Words with related meanings cluster together in embedding space.

The Distributional Hypothesis

Core Idea (Firth, 1957)

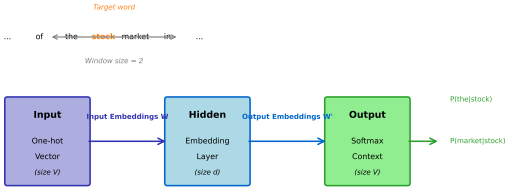
"You shall know a word by the company it keeps."

- Words appearing in **similar contexts** have **similar meanings**
- "The stock _____ sharply today" $\rightarrow$ rose, fell, rallied, plunged
- Context windows capture co-occurrence patterns

The distributional hypothesis is the foundation of all embedding methods.

Skip-gram: Predicting Context from a Word

- Given a **center word**, predict surrounding context words
- Window size c controls how far to look
- The hidden layer weights **become** the embeddings



Skip-gram Architecture: Predict Context from Target

Skip-gram learns embeddings by predicting which words appear nearby.

The Skip-gram Objective

Maximize the average log-probability of context words:

$$\max_{\theta} \frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-c \le j \le c \\ j \neq 0}} \log p(w_{t+j} \mid w_t)$$

where the softmax over the full vocabulary defines:

$$p(w_O \mid w_I) = \frac{\exp\left(\mathbf{v}'_{w_O} \cdot \mathbf{v}_{w_I}\right)}{\sum_{w=1}^{V} \exp\left(\mathbf{v}'_{w} \cdot \mathbf{v}_{w_I}\right)}$$

- T = number of words in the corpus, c = context window
- Denominator sums over **entire vocabulary** – prohibitively expensive!

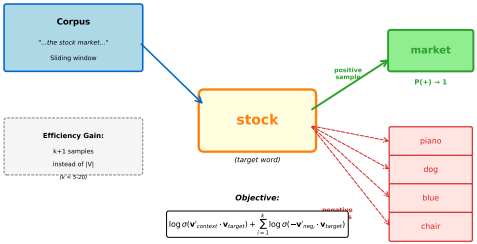
The softmax normalization costs $O(V)$ per update – we need a shortcut.

Negative Sampling: Making Training Feasible

Instead of softmax over V words, sample k **negative** examples:

$$\log \sigma(\mathbf{v}'_{w_O} \cdot \mathbf{v}_{w_I}) + \sum_{i=1}^{k} \mathbb{E}\left[\log \sigma(-\mathbf{v}'_{w_i} \cdot \mathbf{v}_{w_I})\right]$$

- Positive pair: real context word
- Negative pairs: k random words
- Typical k: 5–20



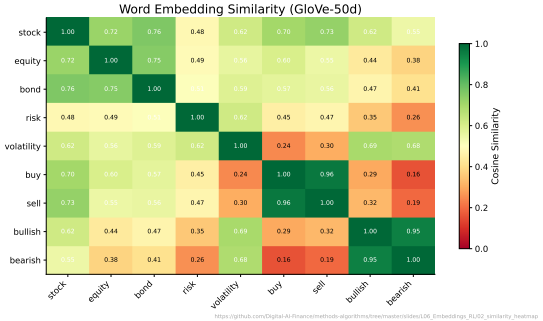
Negative Sampling: Binary Classification Instead of Full Softmax

Negative sampling reduces cost from $O(V)$ to $O(k)$ per update.

Measuring Similarity: Cosine Distance

$$\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}$$

- $+1$: identical direction (synonyms)
- 0: orthogonal (unrelated)
- -1: opposite (antonyms, sometimes)



Cosine similarity is the standard metric for comparing word embeddings.

Word Analogies: The Magic of Vector Arithmetic

Worked Example

$\texttt{king} - \texttt{man} + \texttt{woman} \approx \texttt{queen}$

Step-by-step (simplified 3D vectors):

- $\mathbf{v}_{\text{king}} = [0.9, 0.7, 0.1]$, $\mathbf{v}_{\text{man}} = [0.8, 0.1, 0.1]$, $\mathbf{v}_{\text{woman}} = [0.8, 0.1, 0.9]$
- $\mathbf{v}_{\text{king}} - \mathbf{v}_{\text{man}} + \mathbf{v}_{\text{woman}} = [0.9, 0.7, 0.9]$
- Nearest neighbor: $\texttt{queen} = [0.9, 0.7, 0.85]$ ✓

Finance analogy: $\texttt{stock} - \texttt{equity} + \texttt{debt} \approx \texttt{bond}$

Vector arithmetic captures relational structure: gender, asset class, and more.

GloVe: Global Vectors from Co-occurrence

GloVe minimizes a weighted least-squares objective on the **log co-occurrence matrix**:

$$\sum_{i,j} f(X_{ij}) \left(\mathbf{w}_i \cdot \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

- X_{ij}: how often word i appears near word j
- f: weighting function (caps frequent pairs)

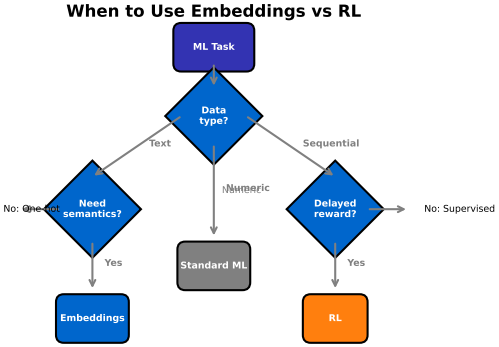
Skip-gram vs GloVe

	Skip-gram	**GloVe**
Type	Predictive	Count-based
Input	Local windows	Global matrix
Speed	Online	Batch

GloVe combines the efficiency of matrix factorization with the quality of prediction-based methods.

When to Use Which Approach?

- **Small corpus**: use pre-trained GloVe or Word2Vec
- **Domain-specific**: fine-tune on your data
- **Context matters**: use BERT / contextual models



Embeddings: Text, categorical -> dense vectors (Word2Vec, BERT)

RL: Sequential decisions with delayed rewards (trading, games)

https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L06_Embeddings_RL/07_decision_flowchart

The right embedding method depends on corpus size, domain, and task requirements.

Financial Sentiment Analysis with Embeddings

Approach

- Train Word2Vec on financial news corpus
- Embedding space reveals sentiment clusters
- Use clusters as features for a classifier

Observed Clusters

- Bullish: rally, surge, upgrade, outperform
- Bearish: crash, plunge, downgrade, miss
- Neutral: report, announce, file, disclose

Word2Vec on domain text captures financial sentiment without manual labeling.

From Words to Documents: Averaging Embeddings

Step-by-step:

1. Look up each word's embedding: $\mathbf{v}_{w_1}, \mathbf{v}_{w_2}, \ldots, \mathbf{v}_{w_n}$
2. Compute the document vector: $\mathbf{d} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{v}_{w_i}$
3. *Optional*: weight by TF-IDF to emphasize important terms

Example: Average embedding of an earnings call transcript $\rightarrow$ single vector $\rightarrow$ sentiment score via logistic regression.

Simple averaging is a strong baseline – often competitive with more complex methods.

Contextual Embeddings: BERT and Beyond

	Static (Word2Vec, GloVe)	**Contextual (BERT, GPT)**
Same word	Same vector always	Different vector per context
"bank"	One vector for all uses	"river bank" $\neq$ "investment bank"
Training	Unsupervised, fast	Pre-trained, fine-tuned
Size	50–300 dims	768–1024 dims
Best for	Simple tasks, speed	Disambiguation, QA, NER

Contextual embeddings solve polysemy but require more compute.

Embeddings as Preprocessing for ML

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import TruncatedSVD
from sklearn.linear_model import LogisticRegression

pipe = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=5000)),
    ('svd',   TruncatedSVD(n_components=50)),
    ('clf',   LogisticRegression())
])
pipe.fit(X_train, y_train)
print(f"Accuracy: {pipe.score(X_test, y_test):.3f}")
```

TF-IDF + SVD is a simple, effective embedding pipeline for text classification.

Named Entity Recognition in Finance

Embedding-Based NER

- Identify company names, tickers, monetary amounts
- Embeddings learn that "AAPL" and "Apple" are related
- Context resolves "Apple" (company vs fruit)

Example Entities

- ORG: Goldman Sachs, JPMorgan
- TICKER: AAPL, TSLA, MSFT
- MONEY: $2.5B, EUR 140M

Pre-trained embeddings + fine-tuning deliver strong NER on financial text.

When to Use Embeddings – and When Not To

Pros

- Capture semantic similarity
- Transfer learning from large corpora
- Compact representation (50–300 dims)

Cons

- Need large corpus to train well
- Not interpretable (black-box vectors)
- Static models ignore context

Use pre-trained embeddings when your corpus is small; fine-tune when domain matters.

Embeddings vs Bag-of-Words vs TF-IDF

	Bag-of-Words	**TF-IDF**	**Embeddings**
Sparsity	Very sparse	Sparse	Dense
Semantics	None	Term importance	Full similarity
Dimensions	V (large)	V (large)	50–300
Training data	None needed	None needed	Large corpus
Interpretable	Yes	Yes	No

Each representation makes a different trade-off between simplicity and expressiveness.

Hands-on: Explore Financial Word Vectors

```
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity

# Hand-crafted finance embeddings (3D for illustration)
words = ['stock','bond','equity','debt','rally','crash']
vecs = np.array([[0.9,0.2,0.1],[0.1,0.9,0.2],
                 [0.85,0.15,0.1],[0.15,0.85,0.25],
                 [0.7,0.1,0.8],[0.7,0.1,-0.8]])

sim = cosine_similarity(vecs)
print("Similarity: stock-equity =", f"{sim[0,2]:.3f}")
print("Similarity: stock-crash  =", f"{sim[0,5]:.3f}")
```

Full notebook: `L06_embeddings.ipynb`

Try the notebook to explore embedding spaces interactively.

Key Takeaways

Concepts

- Distributional hypothesis drives all embeddings
- Skip-gram + negative sampling = efficient training
- Cosine similarity measures semantic closeness
- GloVe combines local and global statistics

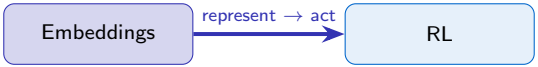
Practice

- Pre-trained > train-your-own (usually)
- Check analogies to validate quality
- Use BERT when context matters
- TF-IDF + SVD is a strong baseline

Embeddings are the bridge between raw text and machine learning.

What's Next: Reinforcement Learning

- Embeddings teach machines to **represent** language
- Reinforcement learning teaches machines to **act**
- Next: how agents learn optimal strategies from rewards



From understanding language to making decisions under uncertainty.

Closing Thought

XKCD #1838 by Randall Munroe (CC BY-NC 2.5)