Embeddings: From One-Hot to RAG

The Complete Journey — All Concepts, All Formulas

Prof. Dr. Jörg Osterrieder

Methods and Algorithms — MSc Data Science

Spring 2026

When All You Have Is Linear Algebra...

"You pour the data into this big pile of linear algebra, then collect the answers on the other side."

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

What Will You Learn Today?

- **Analyze** why one-hot encoding fails for NLP tasks
- **Evaluate** trade-offs between static and contextual embeddings
- **Apply** cosine similarity to measure semantic closeness
- **Design** an embedding-based sentiment classifier for financial text
- **Critique** embedding approaches for specific finance use cases

Bloom's levels 4–5: Analyze, Evaluate, Apply, Design, Critique

Roadmap

From one-hot encoding to RAG: the complete embeddings journey

Why Can't Machines Read Text?

- **The problem:** text is categorical, not numeric — models need numbers
- An earnings call transcript has 10,000 words — how do we feed that to a model?
- **Sparse representation:** each word is an isolated symbol with no relationships
- **High dimensionality:** vocabulary sizes of 50,000–100,000 are common
- **No similarity:** "bullish" and "optimistic" look equally different as "bullish" and "penguin"

The fundamental challenge: converting discrete symbols into continuous vectors

What Is One-Hot Encoding?

Motivate: A vocabulary of 10,000 words means 10,000-dimensional sparse vectors.

Visualize:

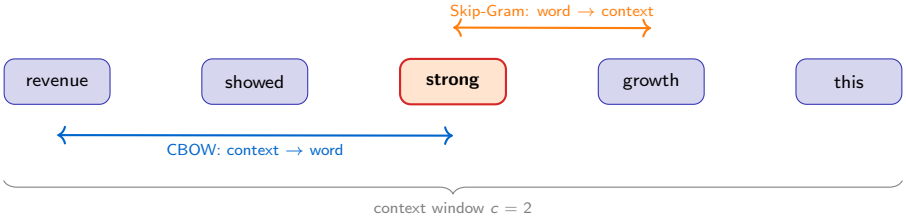
$$\mathbf{e}_{\text{stock}} = [\quad 0,\quad 0,\ \cdots,\ \underset{\text{position } i}{\overset{\downarrow}{1}},\ \cdots,\ 0,\quad 0\quad]$$

Formalize: $\mathbf{e}_i \in \{0,1\}^{|V|}$ where $e_{ij} = 1$ iff $j = i$

One-hot: simple but wasteful. No similarity between any two words.

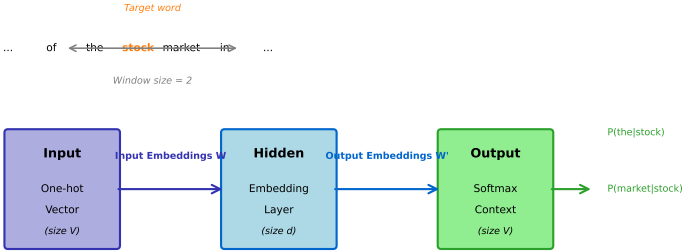
What Does 'Context' Tell Us About Meaning?

- **Distributional hypothesis:** "You shall know a word by the company it keeps" (Firth, 1957)
- Finance: "bullish" appears near "growth", "revenue", "upgrade"



The distributional hypothesis: the foundation of all word embedding methods (Firth, 1957)

How Does the Skip-Gram Network Look?



Skip-gram Architecture: Predict Context from Target

- **Input:** one-hot vector $\mathbf{e}_w \in \mathbb{R}^{|V|}$, **Hidden:** embedding $\mathbf{v}_w = W^\top \mathbf{e}_w \in \mathbb{R}^d$
- **Output:** context matrix W' produces softmax over vocabulary

Skip-Gram: predict context words from the center word. Two embedding matrices: W (input) and W' (output).

What Is the Skip-Gram Objective?

Motivate: We want to maximize the probability of seeing the actual context words.

Finance example: predicting "revenue" appears near "earnings".

Formalize:

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

where $p(w_O \mid w_I) = \dfrac{\exp({\mathbf{v}'_{w_O}}^{\top} \mathbf{v}_{w_I})}{\sum_{w=1}^{|V|} \exp({\mathbf{v}'_{w}}^{\top} \mathbf{v}_{w_I})}$

The softmax denominator sums over ALL words in the vocabulary — this will be a problem.

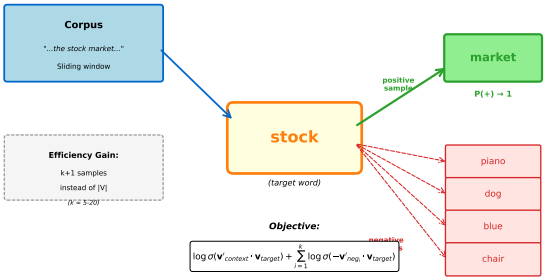
Why Is Full Softmax Intractable?

- $O(|V|)$ **per update** for $V = 100{,}000+$ — every gradient step touches the entire vocabulary
- The denominator $\sum_{w=1}^{|V|} \exp(\mathbf{v}'_w{}^\top \mathbf{v}_{w_I})$ requires computing all dot products
- **Two solutions:** hierarchical softmax (tree structure) and **negative sampling** (binary classification)

Computing the full softmax is the main computational bottleneck of Word2Vec.

How Does Negative Sampling Fix This?

Motivate: Replace V-class softmax with $K+1$ binary classifiers.



Negative Sampling: Binary Classification Instead of Full Softmax

Formalize:

$$J_{\text{NEG}} = \log \sigma({\mathbf{v}'_{w_O}}^{\top} \mathbf{v}_{w_I}) + \sum_{k=1}^{K} \mathbb{E}_{w_k \sim P_n} \left[\log \sigma(-{\mathbf{v}'_{w_k}}^{\top} \mathbf{v}_{w_I}) \right]$$

Push **positive pairs** together, push **negative pairs** apart.

Negative sampling: $O(K)$ **instead of** $O(|V|)$**. Typically** $K = 5$–20 **negatives per positive pair.**

How Do We Sample Negative Words?

1: Initialize W, W' randomly
2: **for** each word w_t in corpus **do**
3: **for** each context word w_c in window **do**
4: **Positive:** update (w_t, w_c) to increase σ
5: **for** $k = 1$ to K **do**
6: Sample $w_n \sim P_n(w)$
7: **Negative:** update (w_t, w_n) to decrease σ
8: **end for**
9: **end for**
10: **end for**

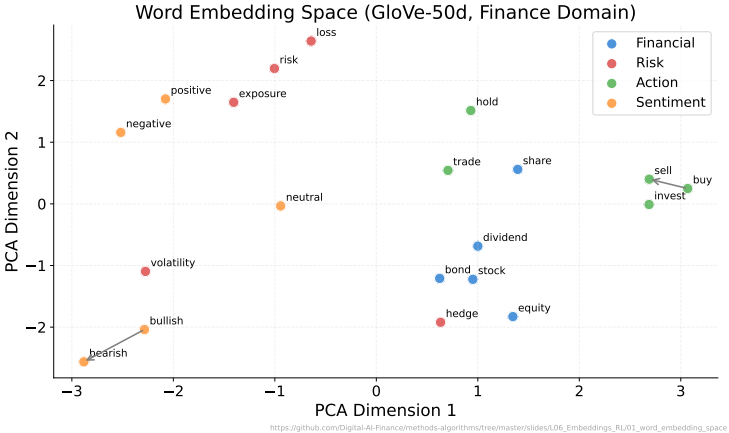
Noise distribution:

$$P_n(w) = \frac{f(w)^{3/4}}{\sum_{w'} f(w')^{3/4}}$$

The $3/4$ exponent **upweights rare words** relative to their raw frequency.

The 3/4 exponent upweights rare words relative to their frequency (Mikolov et al., 2013).

What Does the Embedding Space Look Like?

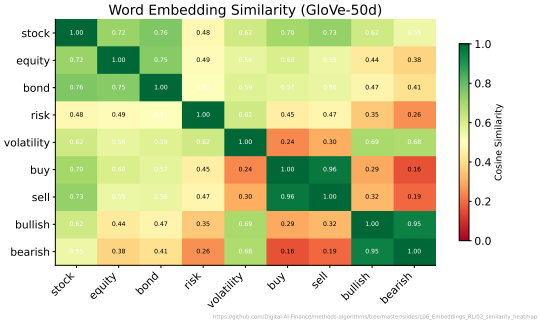


- Similar words cluster together in the learned embedding space
- Finance terms (stock, bond, equity) form a semantic neighborhood

t-SNE projection of 50-dimensional word vectors. Similar words cluster in embedding space.

How Do We Measure Semantic Closeness?

Motivate: Euclidean distance fails when vectors have different magnitudes.



Formalize:

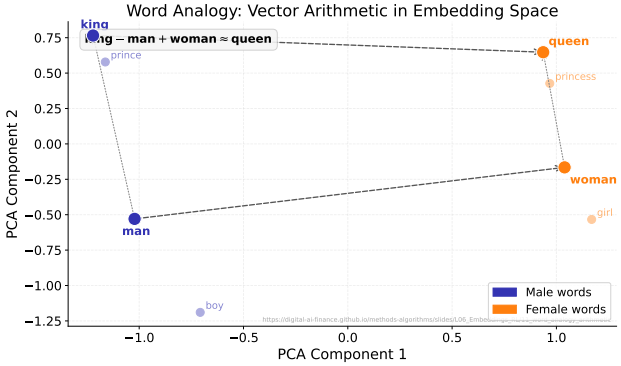
$$\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}$$

Finance examples:

- $\cos(\text{bullish}, \text{positive}) = 0.82$
- $\cos(\text{bullish}, \text{negative}) = -0.31$

Cosine similarity: range $[-1, 1]$. Direction matters, not magnitude.

Can We Do Arithmetic with Words?



Classic:

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

Finance:

$$\mathbf{v}_{\text{stock}} - \mathbf{v}_{\text{equity}} + \mathbf{v}_{\text{debt}} \approx \mathbf{v}_{\text{bond}}$$

Vector arithmetic captures **relational structure** in embedding space.

Vector arithmetic captures relational structure. Success rate: 40–70% (Levy & Goldberg, 2014).

What If We Use Global Statistics Instead?

Motivate: Word2Vec only uses local context windows. **GloVe** uses global co-occurrence statistics.

Formalize:

$$J = \sum_{i,j=1}^{|V|} f(X_{ij}) \left(\mathbf{w}_i^\top \tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2$$

where X_{ij} = co-occurrence count and the weighting function is:
$f(x) = \min\left((x/x_{\max})^{0.75},\ 1 \right)$

- Combines count-based (LSA) and prediction-based (Word2Vec) advantages
- Weighting caps frequent pairs, upweights informative rare pairs

GloVe: combines global co-occurrence statistics with embedding learning (Pennington et al., 2014).

How Do We Handle Unknown Words?

Motivate: The OOV problem — "bullish" unseen if only "bull" was trained.

Example: "bullish" $\rightarrow$ [<bu, bul, ull, lli, lis, ish, sh>]

Formalize:

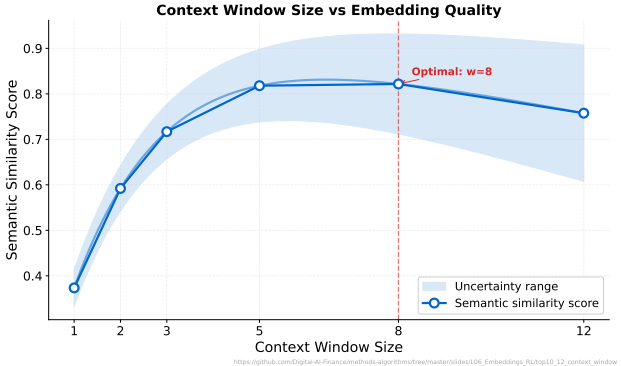
$$\mathbf{v}_w = \sum_{g \in \mathcal{G}(w)} \mathbf{z}_g$$

where $\mathcal{G}(w)$ is the set of character n-grams (typically $n = 3$–6).

- Any new word can be represented via its subword pieces
- Captures morphology: "un-" prefix, "-tion" suffix carry meaning

FastText: character n-grams solve the OOV problem and capture morphology (Bojanowski et al., 2017).

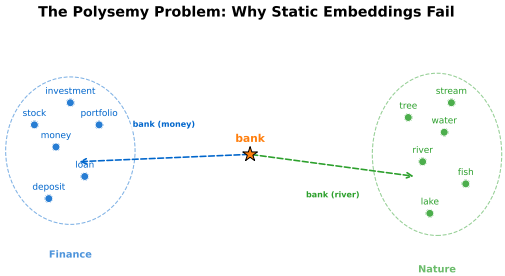
How Do Hyperparameters Affect Embeddings?



- **Small window** ($c = 2$): captures syntactic patterns (POS, grammar)
- **Large window** ($c = 10$): captures semantic/topical similarity
- **Dimensions:** 50–300 typical; diminishing returns beyond 300

Window size and dimensions are the two most important hyperparameters for static embeddings.

What Happens When Words Have Multiple Meanings?



Static: 1 vector for "bank". Contextual (BERT): different vectors based on meaning.

- "Bank" = financial institution OR riverbank
- "Interest" = financial vs. personal
- Static: **one vector per word**

The polysemy problem: static embeddings assign one vector per word regardless of context.

How Do Models Split Text into Tokens?

Example: "unaffordable" $\rightarrow$ [`un`, `##afford`, `##able`]

BPE (GPT family):
merge $= \arg\max_{(a,b)} \text{count}(ab)$
Frequency-based: merge the most common adjacent pair.

WordPiece (BERT family):
$$\text{merge} = \arg\max_{(a,b)} \frac{\text{freq}(ab)}{\text{freq}(a) \cdot \text{freq}(b)}$$
Likelihood-based: merge pairs that most increase corpus likelihood.

- Both reduce vocabulary to 30K–50K tokens while preserving subword meaning
- No more OOV — any word can be split into known subword tokens

BPE: frequency-based merging. WordPiece: likelihood-ratio merging. Both reduce vocabulary while keeping subword meaning.

How Do Transformers Know Word Order?

Motivate: Transformers process all tokens in parallel — they need position information injected.

Formalize:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right)$$

- Each position gets a **unique frequency signature** — no two positions are alike
- Sinusoidal encoding allows the model to attend to relative positions
- Added directly to token embeddings: $\mathbf{x}_i = \mathbf{e}_{\text{token}} + \mathbf{e}_{\text{pos}}$

Sinusoidal position encoding: each position gets a unique frequency signature (Vaswani et al., 2017).

Which Words Should I Pay Attention To?

Motivate: Q/K/V analogy — Query = what I'm looking for, Key = what I offer, Value = what I contain.

The key formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V$$

- $QK^\top$: how much should each token attend to every other token?
- $\sqrt{d_k}$: scaling prevents softmax from saturating for large d_k
- Result: each token is a **weighted sum** of all value vectors

Scaled dot-product attention: the core mechanism of all transformer models (Vaswani et al., 2017).

Why Use Multiple Attention Heads?

Motivate: One head captures syntax, another captures semantics — multiple heads in parallel.

Formalize:

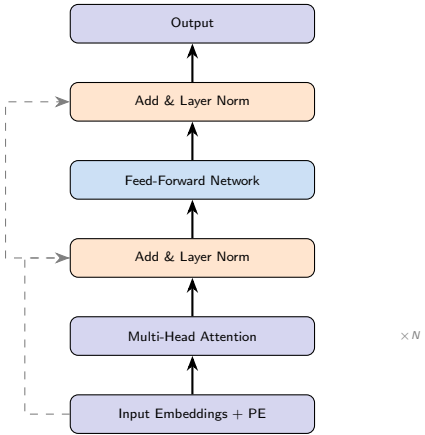
$$\text{MultiHead}(Q, K, V) = \text{Concat}(h_1, \ldots, h_H)\, W^O$$

where $h_i = \text{Attention}(QW_i^Q,\ KW_i^K,\ VW_i^V)$

- Each head projects Q, K, V to dimension $d_k = d_{\text{model}}/H$
- BERT uses $H = 12$ heads; GPT-3 uses $H = 96$ heads
- Concatenated outputs are projected back to d_{model}

Multi-head attention: H **parallel attention functions capture different relationship types.**

What Does a Transformer Block Look Like?



The transformer block: the building unit repeated N times in BERT (12×) and GPT (12–96×).

How Does BERT Read in Both Directions?

Motivate: GPT reads left-to-right only. BERT masks random words and predicts them from **both sides**.

Formalize — Masked Language Modeling:

$$\mathcal{L}_{\text{MLM}} = -\sum_{i \in \mathcal{M}} \log p(x_i \mid \mathbf{x}_{\setminus \mathcal{M}})$$

where $\mathcal{M}$ = set of masked positions (15% of tokens).

- Example: "The [MASK] reported strong earnings" $\rightarrow$ predict "company"
- Bidirectional context: uses both left and right words to predict
- Pre-trained on BookCorpus + Wikipedia (3.3B words)

BERT: Masked Language Modeling + Next Sentence Prediction. Pre-trained on BookCorpus + Wikipedia.

How Do We Fine-Tune for Classification?

Motivate: The `[CLS]` token representation feeds a linear layer for downstream tasks.

Finance: classifying an earnings call as positive/negative.

Formalize:

$$\hat{y}_c = \text{softmax}(\mathbf{W}\,\mathbf{h}_{[\text{CLS}]} + \mathbf{b})_c, \qquad \mathcal{L} = -\sum_{c=1}^{C} y_c \log(\hat{y}_c)$$

- **Pre-train** on large unlabeled corpus (expensive, done once)
- **Fine-tune** on small labeled dataset (cheap, task-specific)
- Only the classification head + top BERT layers are updated

Cross-entropy loss on the [CLS] representation: the standard approach for text classification.

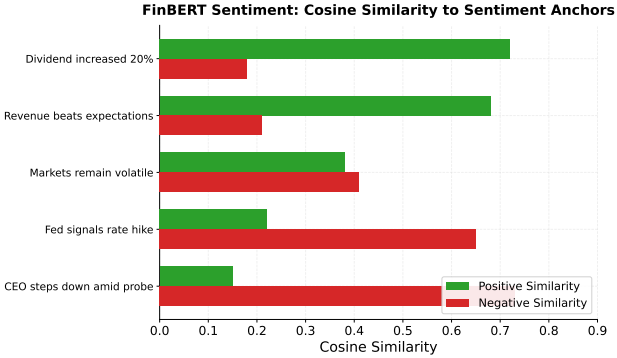
When Should You Use BERT vs GPT?

Property	BERT	GPT
Direction	Bidirectional	Left-to-right
Pre-training	MLM + NSP	Next token prediction
Best for	Classification, NER, QA	Text generation
Finance use	Sentiment, entity extraction	Report generation
Parameters	110M (base) / 340M (large)	117M (1) to 175B (3)

- **BERT for understanding**, **GPT for generation**
- Both use the same transformer architecture — the difference is the training objective
- Modern LLMs (GPT-4, Claude) use decoder-only architecture at massive scale

BERT for understanding, GPT for generation. Both use the same transformer architecture.

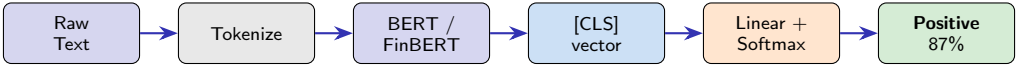
Why Does General BERT Fail on Financial Text?



- "Liability" = negative in general English, **neutral in finance**
- "Volatile" = negative generally, descriptive in markets
- FinBERT: BERT fine-tuned on financial news, earnings calls, analyst reports

FinBERT: BERT fine-tuned on financial text achieves 87% accuracy on sentiment (Araci, 2019).

How Does the Full Sentiment Pipeline Work?

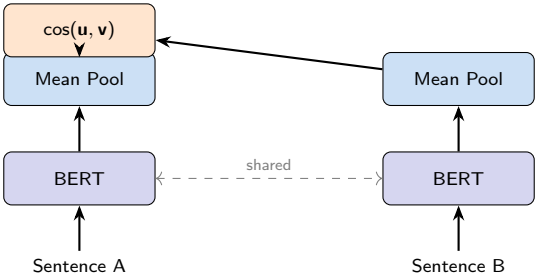


- Finance running example: "Q3 revenue exceeded expectations with 12% growth"
- The `[CLS]` token aggregates the full sentence meaning into one vector
- Linear layer maps 768-dim vector to 3 classes: positive / negative / neutral

End-to-end sentiment classification: text → tokens → BERT → [CLS] → softmax → label.

How Do We Get Sentence-Level Embeddings?

Motivate: BERT gives token-level embeddings, not sentence-level. Sentence-BERT uses a **siamese network**.

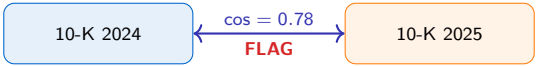


$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(\cos(\mathbf{u}_i, \mathbf{v}_i) - y_i)^2$ where $y_i \in [-1, 1]$

Sentence-BERT: cosine similarity MSE loss for contrastive sentence learning (Reimers & Gurevych, 2019).

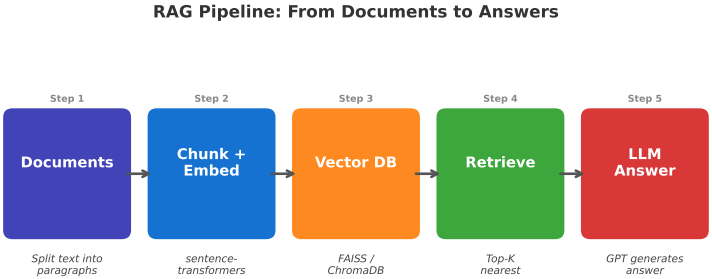
Can We Detect Changes in SEC Filings?

- Compare 10-K filings year-over-year using **document embeddings**
- If $\cos(\mathbf{d}_{2024}, \mathbf{d}_{2025}) < 0.85$, flag for manual review
- Material changes in risk factors, revenue recognition, or legal proceedings are detectable



Document similarity via cosine: a simple but powerful compliance monitoring tool.

How Does RAG Connect Embeddings to LLMs?



- Query → embed → search vector DB → retrieve top-k → augment prompt → generate
- Grounds LLM responses in **factual, up-to-date** documents — reduces hallucination

RAG: Retrieval-Augmented Generation. Combines embedding search with LLM generation.

What Entities Can We Extract from Financial Text?

- **Tickers:** AAPL, MSFT, TSLA — link mentions to structured databases
- **Monetary amounts:** $2.4B revenue, 500M write-down
- **Dates & events:** “Q3 2025 earnings”, “Fed meeting March 19”

BERT + token classification head: predict B-ORG, I-ORG, B-MONEY, etc. per token.

Named Entity Recognition: extract structured data from unstructured financial text.

Can Embeddings Generate Trading Signals?

- News sentiment → daily feature vector → ML model → buy/sell signal
- Aggregate across all articles for a given day and ticker:

$$s_t = \frac{1}{N} \sum_{i=1}^{N} \text{sentiment}(\text{article}_i) \cdot \text{relevance}(\text{article}_i)$$

- **Walk-forward validation** required — no peeking at future data
- Combine with price features, volume, and technical indicators

Sentiment-based trading signals: aggregate news embeddings into daily features for ML models.

Which Embedding Should You Choose?

Method	Best For	Limitation
One-Hot	Tiny vocab, baseline	No similarity
Word2Vec / GloVe	Static, fast, large corpus	No polysemy
FastText	Morphologically rich, OOV	Still static
BERT	Context-dependent, classification	Slow, 512 tokens
Sentence-BERT	Document comparison	Needs fine-tuning
FinBERT	Financial text	Domain-specific only

- Start with the **simplest method that works**
- Upgrade when you have evidence it will improve results
- Always benchmark against a bag-of-words baseline

Start with the simplest method that works. Upgrade when you have evidence it will improve results.

What Are the Key Takeaways?

- **Map** discrete text to continuous vectors with embeddings
- **Capture** distributional semantics with Word2Vec / GloVe
- **Add context** via transformer attention mechanisms
- **Specialize** for finance with FinBERT and domain fine-tuning
- **Connect** embeddings to LLMs via RAG for grounded generation

Five ideas that changed NLP: dense vectors, distributional learning, attention, fine-tuning, retrieval.

Key Terms

- **Embedding** — dense vector representing a discrete symbol
- **One-Hot** — sparse binary vector, one dimension per word
- **Skip-Gram** — predict context words from center word
- **Cosine Similarity** — direction-based similarity in $[-1, 1]$
- **Attention** — weighted sum using query–key–value
- **BERT** — bidirectional transformer, masked LM pre-training
- **Fine-Tuning** — adapt pre-trained model to downstream task
- **RAG** — retrieval-augmented generation for grounded LLM output

Master these 8 terms and you can discuss embeddings with any NLP practitioner.

What Should You Try Next?

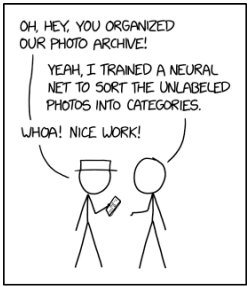
1. Load pre-trained Word2Vec vectors and find the 10 nearest neighbors to “inflation”
2. Compute cosine similarity between “bullish” and “bearish” — is it positive or negative?
3. Use FinBERT to classify 5 financial headlines as positive / negative / neutral

Colab notebook: `https://colab.research.google.com/`

See the L06 Colab notebook for starter code and pre-loaded data.

Hands-on: the best way to learn embeddings is to build with them. See the L06 Colab notebook.

From One-Hot to RAG: Journey Complete



"I trained a neural net to identify the most important part of any image, and it says it's this neural net."

XKCD #2173 by Randall Munroe (CC BY-NC 2.5). From one-hot to RAG: the journey is complete!