Embeddings: From Words to Vectors

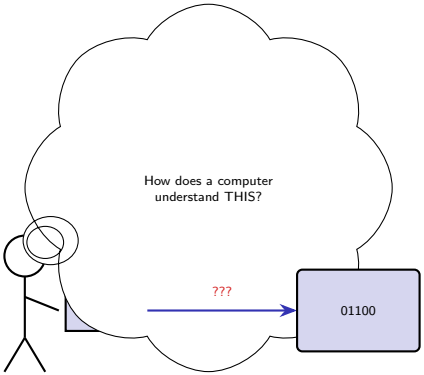
A Complete Guide for Beginners — No Prior Knowledge Required

Prof. Dr. Jörg Osterrieder

Methods and Algorithms — BSc Data Science

Spring 2026

How Does a Computer “Read”?



Computers process numbers, not words. This lecture bridges that gap.

What You Will Learn

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

1. Understand why computers need numbers, not words
2. See how Word2Vec turns words into meaningful vectors
3. Watch word arithmetic actually work: king $-$ man $+$ woman $\approx$ queen
4. Learn how modern models like BERT and FinBERT understand finance
5. Know which embedding tool to use for your specific problem

---

No prior NLP or machine learning knowledge is assumed.

Roadmap

1. The Problem with Words
2. How Word2Vec Works
3. Making It Practical
4. Modern Embeddings
5. Embeddings in Action

Six sections, from basic concepts to real-world applications.

The Problem — Computers Only Understand Numbers

The word "profit" means something to you. To a computer, it's just 6 characters.

How do we give a machine the ability to understand that "profit" and "revenue" are related, but "profit" and "penguin" are not?

This lecture answers that question, one building block at a time.

Think About It

- "profit" ↔ "revenue" ✓ related
- "profit" ↔ "penguin" × unrelated

You know this instantly. A computer sees only character sequences: `p-r-o-f-i-t` vs. `p-e-n-g-u-i-n`.

The gap between human language and machine computation is the central challenge of natural language processing.

The Naive Solution — Give Every Word a Number

- **One-hot encoding**: assign each word a unique binary vector
- With 50,000 words, each vector has 50,000 entries — mostly zeros
- "cat" and "dog" are equally far apart as "cat" and "spaceship"

One-hot says all words are equally different. That's obviously wrong.

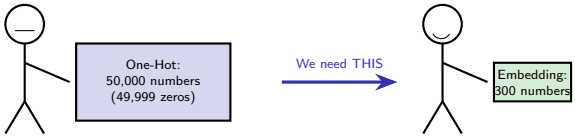
One-Hot Example

```
cat       = [0, 0, 1, 0, ...]
dog       = [0, 1, 0, 0, ...]
spaceship = [1, 0, 0, 0, ...]
```

Every pair is distance $\sqrt{2}$ apart — no similarity captured at all.

One-hot encoding is the simplest representation, but it cannot capture any notion of word similarity.

Why One-Hot Doesn't Work



The jump from sparse one-hot to dense embeddings is the central idea of this lecture.

The Key Insight — Words Are Known by Their Company

If "profit" and "revenue" always appear in similar sentences, they must mean similar things.

- "You shall know a word by the company it keeps" (J.R. Firth, 1957)
- This is the **distributional hypothesis** — meaning comes from context
- **This single insight powers ALL embedding methods**

Evidence

"The company reported strong **profit** growth this quarter."

"The company reported strong **revenue** growth this quarter."

Same neighbors → similar meaning. The context is the clue.

The distributional hypothesis is the theoretical foundation for Word2Vec, GloVe, and all modern embedding approaches.

The Training Game — Predict the Neighbors

- Fill in the blank — your brain uses CONTEXT to guess the missing word
- Word2Vec plays exactly this game, millions of times, with real text
- A network good at predicting neighbors must **understand** meaning

We don't care about the predictions — we care about what the network learned along the way.

The Game

"The stock _____ rose sharply today"

Your guesses:

- **price** ✓
- **market** ✓

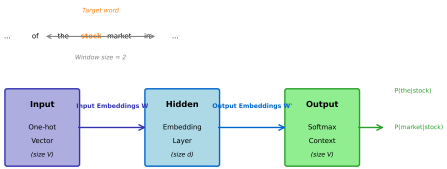
Word2Vec learns: "price" and "market" fit similar blanks → similar vectors.

The key trick: we don't care about the predictions themselves — we care about what the network learned along the way.

The Skip-Gram Idea

- Given one word (input), predict the words around it (output)
- The neural network has ONE hidden layer
- The weights of that hidden layer become the word vectors

The prediction task is just a trick — what we really want are those hidden weights.



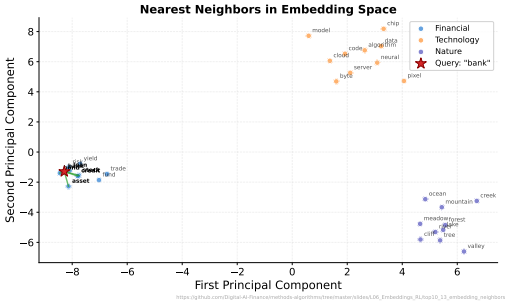
Skip-gram Architecture: Predict Context from Target

Skip-gram is one of two Word2Vec architectures. The other, CBOW, predicts the center word from its neighbors.

What the Network Actually Learns

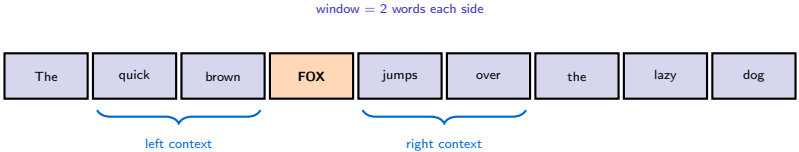
- After training on millions of sentences, each word gets a **dense vector** — 300 numbers capturing its meaning
- Words with similar meaning end up with similar vectors — automatically!
- The network was never TOLD that “profit” and “revenue” are related — it figured it out from context

Training uses **negative sampling** — checking a small random sample instead of all 50,000 words.



Dense vectors are far more compact and meaningful than one-hot vectors. 300 dimensions vs. 50,000.

The Context Window



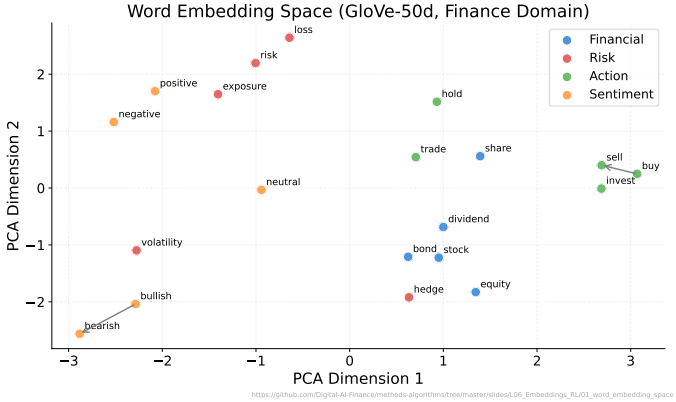
A larger window captures broader meaning; a smaller window captures syntax and grammar.

Typical window sizes range from 2 to 10. Financial text often benefits from larger windows (8–10).

Words as Points in Space

- Each word becomes a point in space
- Financial terms cluster together
- Sentiment words cluster together

The computer learned these groupings entirely from reading text — no human labeled them.

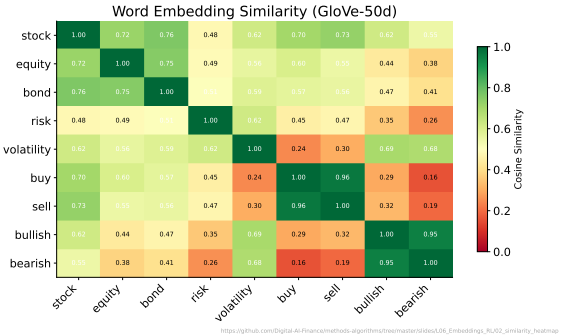


This visualization projects 300 dimensions down to 2D using t-SNE. Clusters are real, distances are approximate.

How Close Are Two Words? — A Worked Example

Cosine similarity measures direction, not length:

- Vectors A = (3, 4) and B = (6, 8) point in the SAME direction: cosine similarity = 1.0
- C = (4, −3) points at 90° to A: cosine similarity = 0
- Values range from −1 (opposite) to +1 (identical)

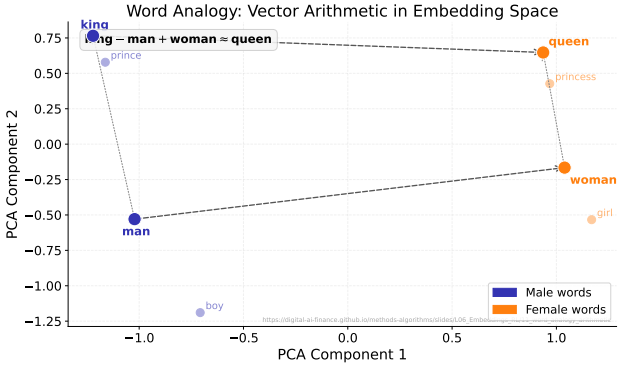


Cosine similarity is the standard measure for comparing word vectors. It ignores vector length and focuses on direction.

The Magic — king − man + woman ≈ queen

- The computer learned that “royalty” and “gender” are separate dimensions!
- The vector from “man” to “king” captures “royalty” — apply that same shift to “woman” and you land near “queen”

Nobody taught the model this. It discovered the relationship from patterns in text.

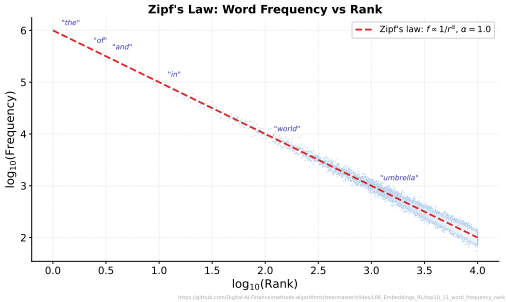


Word analogies were the first dramatic demonstration that embeddings capture deep semantic relationships.

What About Words the Model Has Never Seen?

- **FastText** breaks words into character pieces
- "un-happi-ness" — even if "unhappiness" wasn't in training data, the pieces were
- Finance example: "cryptocurrency" is rare, but "crypto" and "currency" are common

FastText solves the out-of-vocabulary (OOV) problem — it handles words never seen before.



FastText was developed at Facebook AI Research in 2017. It extends Word2Vec with subword information.

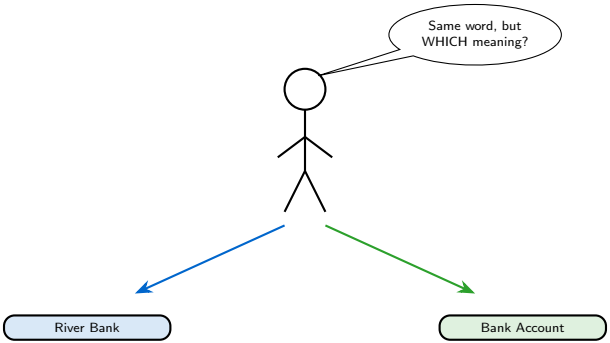
How to Choose Your Settings

Setting	What It Controls	Good Starting Point
Vector size	How many numbers per word	100–300
Window	How far to look for context	5–10 words
Min count	Ignore rare words below this	5–100

Bigger vectors = more detail but slower.
Larger windows = broader meaning.

These are hyperparameters — settings you choose before training. Experimenting is the best way to find what works.

The “Bank” Problem — Same Word, Different Meaning

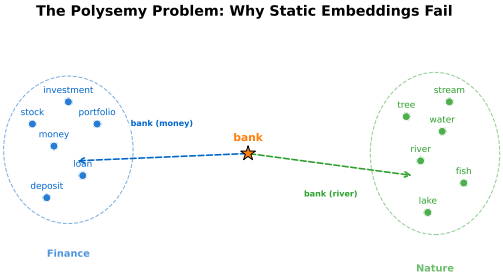


This is called polysemy — one word, multiple meanings. A fundamental limitation of static embeddings.

The Fix — Embeddings That Read Context

- **Static embeddings** (Word2Vec, GloVe): one vector per word, regardless of context
- **Contextual embeddings** (BERT, GPT): different vector depending on the sentence

"River bank" and "bank account" get DIFFERENT vectors. A revolutionary improvement — but it requires much bigger models.



Contextual embeddings were introduced by ELMo (2018) and perfected by BERT (2019). They solved the polysemy problem.

How Do Computers Chop Words into Pieces?

- **Tokenization**: computers split words into pieces before processing
- **BPE** (Byte-Pair Encoding): start with characters, merge frequent pairs
- This is why ChatGPT sometimes splits words oddly

WordPiece uses the same idea but optimizes for language model likelihood.

BPE Example

`unhappiness`

Step 1: split into characters
`u n h a p p i n e s s`

Step 2: merge frequent pairs
`un h app i ness`

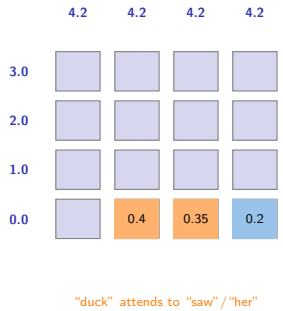
Step 3: final tokens
`[un, happi, ness]`

Tokenization happens before any model sees the text. It determines how words are broken into processable units.

What Is a Transformer? (No Math!)

- A **transformer** reads ALL words at once and figures out which words matter most for each other
- "I saw her duck" — is "duck" a bird or a verb? It checks every word to decide
- Transformers know word ORDER — "not" before "good" changes everything

This is the attention mechanism.

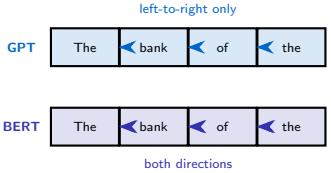


The transformer architecture (Vaswani et al., 2017) is the foundation for GPT, BERT, and virtually all modern NLP models.

BERT — Reading in Both Directions

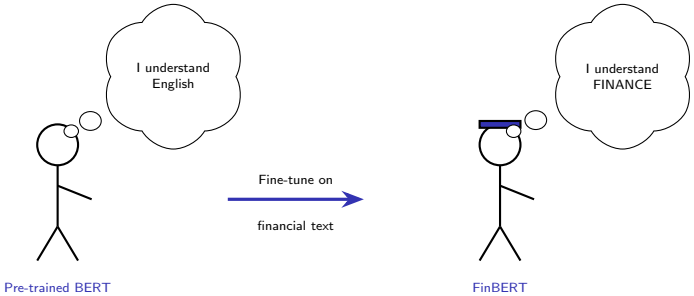
- **BERT** reads a sentence forward AND backward simultaneously
- "The bank of the river" — it sees context on BOTH sides and understands "bank" = riverside

BERT is better for **understanding** text.
GPT is better for **generating** text.



BERT = Bidirectional Encoder Representations from Transformers (Devlin et al., 2019). Pre-trained on Wikipedia and BooksCorpus.

Teaching a General Model Your Specialty

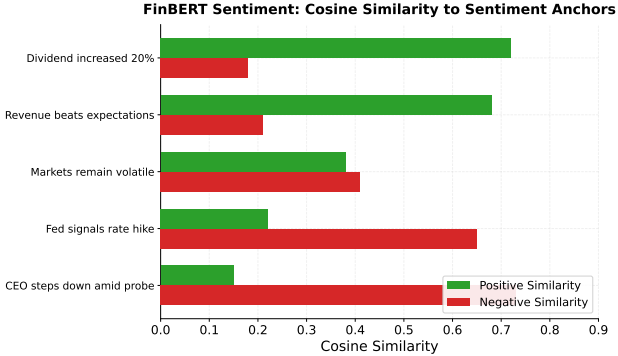


Fine-tuning is like a medical student who knows general science and then specializes in cardiology.

FinBERT — BERT That Speaks Finance

- General BERT doesn't know that "bull market" is positive
- **FinBERT** does — trained on financial news, earnings reports, and analyst notes
- 87% accuracy on financial sentiment (Araci, 2019)

Much better than general-purpose models on financial text.

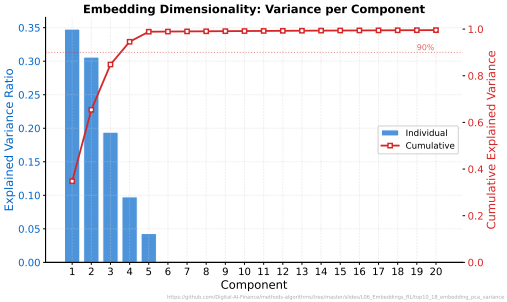


FinBERT classifies financial text as positive, negative, or neutral — critical for automated sentiment analysis.

From Words to Whole Paragraphs

- Word2Vec gives a vector per WORD — but what about entire DOCUMENTS?
- **Sentence embeddings** (Sentence-BERT): one vector per sentence or paragraph
- Application: "Find all news articles SIMILAR to this one"

Sentence embeddings unlock document search, clustering, and duplicate detection.



Sentence-BERT (Reimers & Gurevych, 2019) produces fixed-size vectors for variable-length text.

Spotting Changes in SEC Filings

- Companies file quarterly reports (10-Q). Embed this quarter's filing and last quarter's
- If the vectors are very different $\rightarrow$ something changed
- Banks use this to automatically flag material changes in regulatory filings

Worked example: Embed two paragraphs, compute cosine similarity. Score drops from 0.95 to 0.72 $\rightarrow$ flag for review.

Automated filing comparison saves hundreds of analyst hours per quarter at large financial institutions.

Can Computers Highlight Names and Numbers in Text?

- **Named Entity Recognition** (NER): computers identify and classify names, dates, amounts, organizations in text
- Example: [Apple Inc.]$_{\text{ORG}}$ reported [$89.5 billion]$_{\text{MONEY}}$ in revenue on [January 26]$_{\text{DATE}}$
- Embeddings make this possible — the model learned that numbers following "$" are monetary amounts

One of the most intuitive and practical NLP applications.

NER is used in compliance, contract analysis, and automated report extraction throughout the financial industry.

Sentiment Analysis — End to End

Worked example: "Revenue exceeded expectations" → tokenize → embed → FinBERT → **92% positive**

Sentiment analysis is the most common application of embeddings in finance. It powers news-driven trading signals.

Can Embeddings Generate Trading Signals?

- Sudden NEGATIVE sentiment shifts predict short-term price drops
- But the market may already have priced in publicly available news
- **Embeddings are a tool, not a crystal ball** — combine with other data

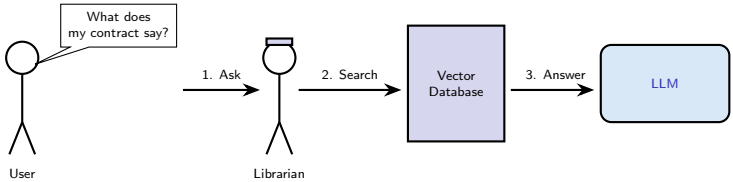
The Pipeline

1. Embed news daily
2. Measure sentiment shift
3. Correlate with returns

Predictive horizon: **1–3 days**. Longer-term signals are weaker.

Academic evidence suggests a 1–3 day predictive horizon for news sentiment. Longer-term signals are weaker.

RAG — How ChatGPT Uses Embeddings



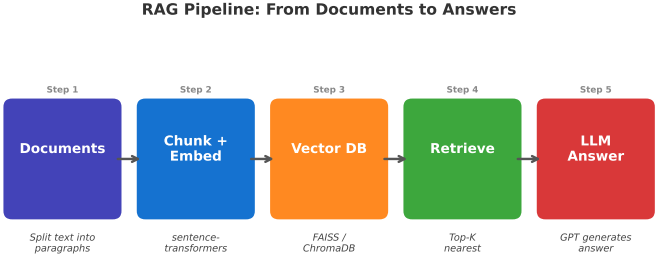
Retrieval-Augmented Generation: find the right documents first, then let the LLM read them and answer.

RAG lets AI answer questions about YOUR documents — not just what it was trained on.

The RAG Pipeline — Documents to Answers

RAG in 5 steps:

1. Chunk your documents
2. Embed each chunk
3. Store in vector database
4. Question arrives → find most similar chunks
5. Send chunks + question to LLM for answer



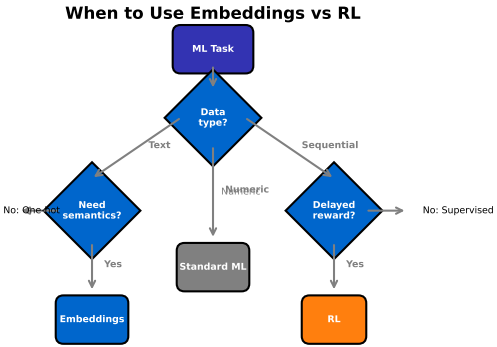
This is how companies build "chat with your data" products.

RAG avoids the need to re-train large models. You keep the same LLM and just feed it relevant context at query time.

Which Embedding Should You Choose?

Quick decision guide:

- Small dataset, English → Word2Vec or GloVe
- Financial text → FinBERT
- Compare documents → Sentence-BERT
- Chat / QA system → RAG with vector database
- Unknown words → FastText



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

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

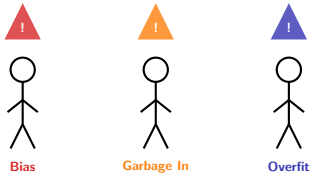
https://github.com/Digital-AI-Finance/methods-algorithms/tree/master/slides/L06_Embeddings_RL/07_decision_flowchart

There is no single best embedding. The right choice depends on your data size, domain, and application.

Three Things That Can Go Wrong

1. **Bias**: Embeddings learn biases from training data — "doctor" closer to "man" than "woman"
2. **Garbage in**: Train on bad text → get bad embeddings. Data quality is everything
3. **Overfitting**: Fine-tune on too little data → model memorizes instead of generalizing

Always audit your embeddings for bias before deploying.



Bias in embeddings is an active research area. Debiasing techniques exist but are not yet foolproof.

The Journey Complete

From confused to confident — you've traveled the full embedding journey.

Key Takeaways + What to Try Next

1. **Embeddings** turn words into vectors where similar words are nearby
2. **Word2Vec** learns from context: predict neighbors $\rightarrow$ capture meaning
3. **Cosine similarity** measures how alike two vectors are (-1 to $+1$)
4. **Contextual models** (BERT) solve the polysemy problem
5. **FinBERT** specializes BERT for financial text (87% sentiment accuracy)
6. **RAG** connects embeddings to LLMs for document Q&A
7. **Choose wisely**: Word2Vec for small data, FinBERT for finance, RAG for chat

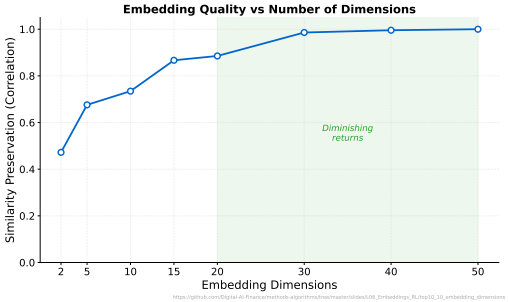
For the mathematical details behind each concept, see **L06f: Embeddings Complete**.

Each takeaway maps to a section of this lecture. Revisit any section to deepen your understanding.

GloVe — The Matrix Alternative

- **GloVe** (Global Vectors): count how often word pairs co-occur in a big matrix, then compress it
- Similar quality to Word2Vec, sometimes better on analogy tasks
- Developed at Stanford (Pennington et al., 2014)

GloVe combines the best of count-based and prediction-based methods.



GloVe pre-trained vectors are freely available at `nlp.stanford.edu/projects/glove/` in sizes from 50 to 300 dimensions.

Key Terms Glossary

- **Embedding** — vector representation of a word
- **One-hot** — sparse binary vector (one 1, rest 0s)
- **Word2Vec** — learns embeddings by predicting context
- **Skip-gram** — Word2Vec variant: predict neighbors from center word
- **Cosine similarity** — measures direction similarity (-1 to $+1$)
- **Context window** — number of neighbor words to consider
- **FastText** — extends Word2Vec with subword pieces
- **BERT** — bidirectional transformer for understanding text
- **FinBERT** — BERT fine-tuned on financial text
- **Tokenization** — splitting text into processable units
- **RAG** — retrieval-augmented generation for document Q&A
- **Fine-tuning** — adapting a pre-trained model to a specific domain

These 12 terms cover the core vocabulary of this lecture. Master them and you can follow any NLP discussion.

References

- Mikolov et al. (2013) — Word2Vec: Efficient Estimation of Word Representations in Vector Space
- Pennington et al. (2014) — GloVe: Global Vectors for Word Representation
- Bojanowski et al. (2017) — Enriching Word Vectors with Subword Information (FastText)
- Vaswani et al. (2017) — Attention Is All You Need (Transformer)
- Devlin et al. (2019) — BERT: Pre-training of Deep Bidirectional Transformers
- Araci (2019) — FinBERT: Financial Sentiment Analysis with Pre-Trained Language Models
- Reimers & Gurevych (2019) — Sentence-BERT: Sentence Embeddings using Siamese Networks

All papers are freely available on arXiv. Start with Mikolov (2013) for foundations, Devlin (2019) for modern approaches.