Neural Language Models

Week 2 - Word Embeddings and Word2Vec

NLP Course 2025

September 28, 2025

Using Optimal Readability Template

Week 2: The Semantic Revolution

From Words as IDs to Words as Meanings

The Problem

- Words are just IDs
- No semantic similarity
- "cat" and "dog" equally different as "cat" and "democracy"
- Can't generalize

The Solution

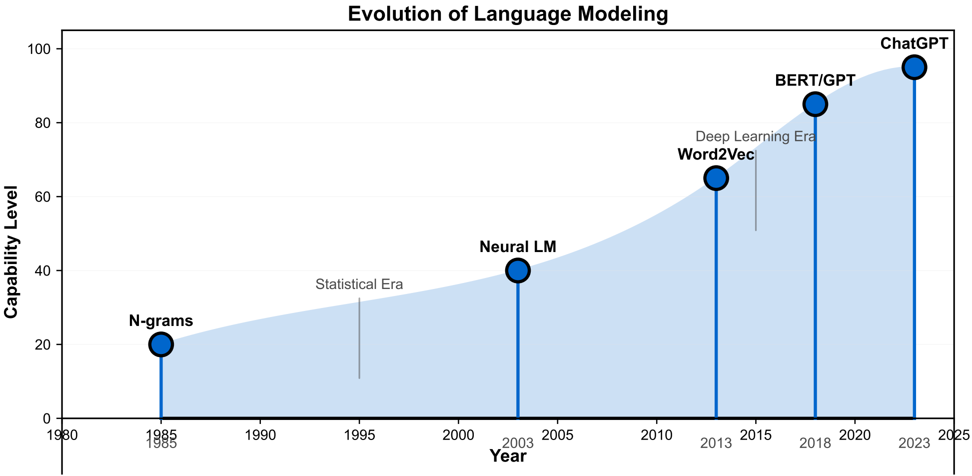
- Words as **vectors**
- Similar words nearby
- Math operations work!
- King - Man + Woman = Queen

The Impact

- Powers all modern NLP
- 1M+ developers use
- Semantic search
- Foundation for GPT/BERT

Core Insight: You shall know a word by the company it keeps

The Evolution of Language Modeling



Interactive: Word Association Game

Fill in the blank:

1. The cat sat on the ____________________
2. I drink my coffee with milk and ____________________
3. The capital of France is ____________________
4. She opened the door with her ____________________

How did you know?

- Context provides meaning
- Similar contexts $\rightarrow$ similar words
- Pattern recognition

This is Word2Vec's insight:

- Learn from **billions** of contexts
- Words in similar contexts get similar vectors
- Mathematics captures semantics

Where Word Embeddings Power Your Life (2025)

Search Engines

- Google semantic search
- Bing neural matching
- DuckDuckGo instant answers

Translation

- Google Translate
- DeepL
- Microsoft Translator

Business Tools

- Grammarly corrections
- Resume matching
- Customer support bots

Virtual Assistants

- Siri/Alexa understanding
- Google Assistant
- ChatGPT responses

Recommendations

- Netflix shows
- Spotify Discover
- YouTube suggestions

Market Size

- $2.7B by 2025
- 1M+ developers
- 500M+ daily users

The 2013 Breakthrough: Mathematical Semantics

King - Man + Woman = Queen

The Discovery:

- Vectors encode **relationships**
- Arithmetic operations preserve meaning
- Geometry captures semantics

Why This Matters:

- Computers understand analogies
- Transfer learning possible
- One model, many tasks
- Foundation for all modern NLP

More Examples:

- Paris - France + Italy = Rome
- Bigger - Big + Small = Smaller
- Walking - Walk + Swim = Swimming

Word2Vec paper: 16,000+ citations

Semantic relationships become vector arithmetic

The Distributional Hypothesis

Linguistic Foundation (1954):
"You shall know a word by the company it keeps" - J.R. Firth

What it means:

- Words with similar **contexts** have similar **meanings**
- Context = surrounding words
- Meaning emerges from usage

Example contexts for "bank":

- "deposit money in the <u>bank</u>"
- "sitting by the river <u>bank</u>"
- Different contexts → different meanings

How Word2Vec uses this:

1. Scan billions of sentences
2. Track which words appear together
3. Words in similar contexts get similar vectors
4. Geometry encodes semantics

The Magic:

- No human labeling needed
- Learns from raw text
- Scales to millions of words
- Works for any language

From Sparse to Dense: The Representation Revolution

One-Hot Encoding (Old Way):

- Vocabulary size: 50,000
- cat = [0,0,1,0,0,...,0]
- dog = [0,0,0,1,0,...,0]

Problems:

- 50,000 dimensions!
- No similarity: cat · dog = 0
- Can't generalize
- Massive memory usage

Dense Embeddings (Word2Vec):

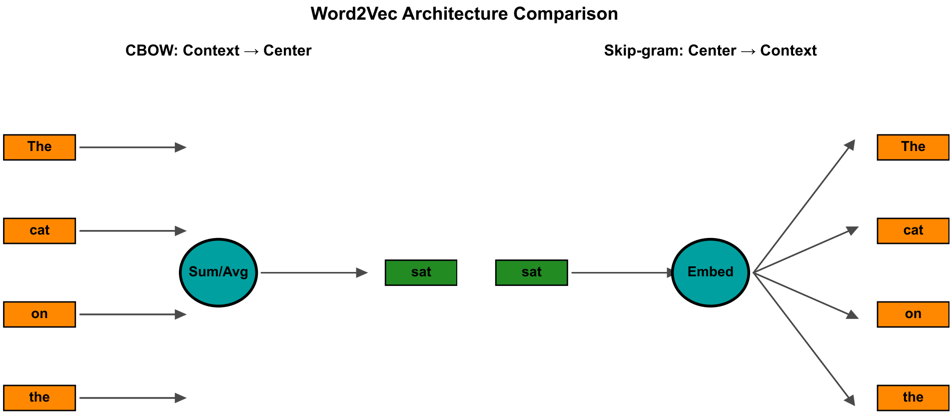
- Embedding size: 300
- cat = [0.2, -0.4, 0.7, ...]
- dog = [0.3, -0.3, 0.6, ...]

Benefits:

- 99.4% smaller!
- Similarity: cat · dog = 0.8
- Generalizes to new contexts
- Efficient computation

From 50,000 sparse dimensions to 300 dense dimensions

Word2Vec: Two Architectures



Skip-gram: The Architecture That Won

Training Objective: Predict context from center word

Given: "The cat sat on the mat"

Input: "cat" (center word)

Outputs to predict:

- "The" (position -1)
- "sat" (position +1)
- Sometimes: "on" (+2), "the" (-2)

Window size = 2:

- Look 2 words left/right
- 4 predictions per center word
- More context = better vectors

Why Skip-gram wins:

- Better on rare words
- More training examples
- Superior semantic quality
- Used by Google, Facebook

Training data from one sentence:

- (cat, The)
- (cat, sat)
- (sat, cat)
- (sat, on)
- ... many more pairs

Building Word2Vec in PyTorch

```python
import torch
import torch.nn as nn

class Word2Vec(nn.Module):
    def __init__(self, vocab_size, embed_dim):
        super().__init__()
        # Two embedding matrices
        self.center_embeddings = nn.Embedding(
            vocab_size, embed_dim
        )
        self.context_embeddings = nn.Embedding(
            vocab_size, embed_dim
        )

    def forward(self, center, context):
        # Get embeddings
        center_embeds = self.center_embeddings(center)
        context_embeds =
             self.context_embeddings(context)

        # Dot product = similarity
        scores = torch.sum(
            center_embeds * context_embeds, dim=1
        )

        return scores
```

Key Design Choices:

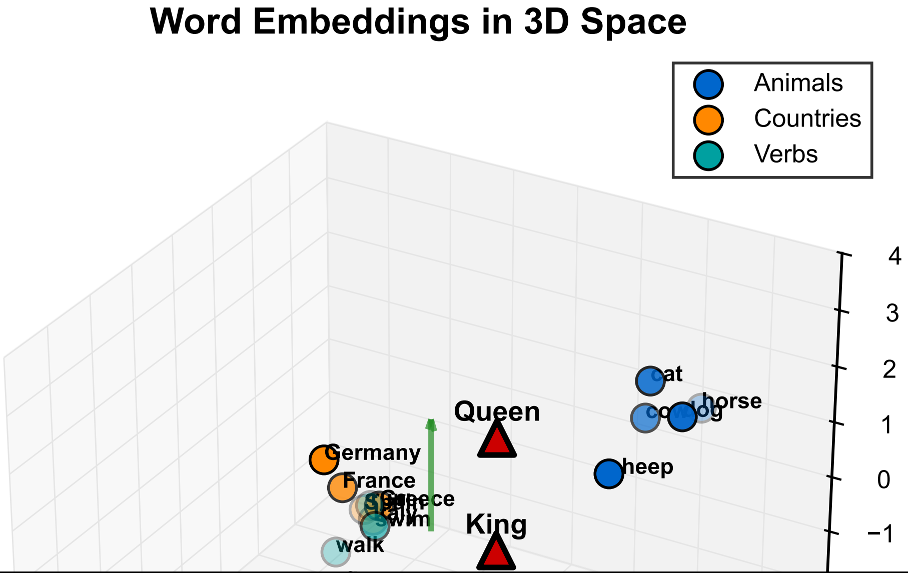
- **Two matrices**: center and context
- Embedding dim: typically 300
- Dot product for similarity
- Simple = fast training

Training Process:

1. Sample (center, context) pairs
2. Compute similarity scores
3. Maximize correct pairs
4. Minimize random pairs

Full implementation: 50 lines of code!

Embedding Space Visualization



The Softmax Challenge

Converting scores to probabilities:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{V} e^{z_j}}$$

The Problem:

- Vocabulary size $V = 50{,}000$
- Must compute all 50,000 scores
- Denominator sums 50,000 exponentials
- Every training step!

Computational cost:

- Per sample: $O(V \cdot d)$
- 1B training samples
- = 15 trillion operations

The Solution: Negative Sampling

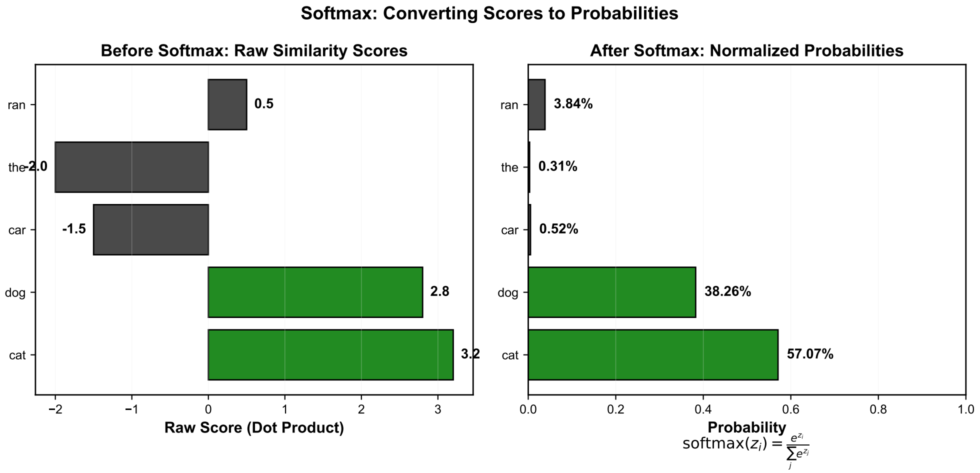
- Don't compute all 50,000
- Just sample 5-20 negatives
- 99.96% speedup!
- Quality stays the same

New objective:

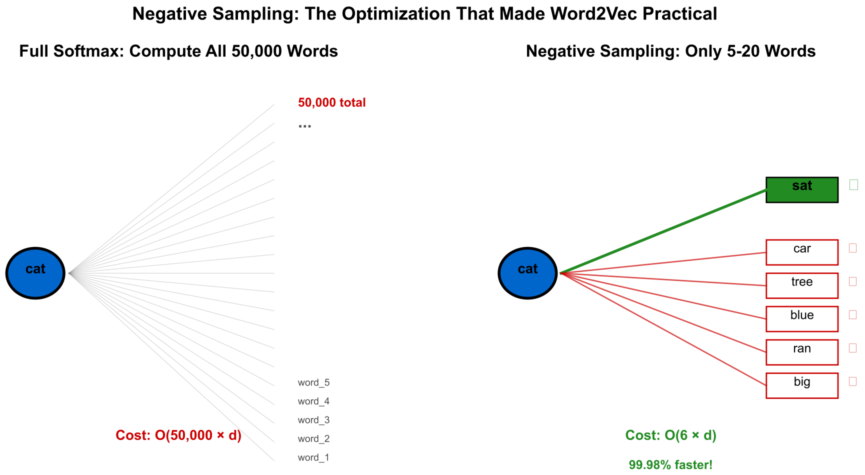
- Maximize: P(correct context)
- Minimize: P(random words)
- Binary classification $\times$ 6
- Much faster training

Negative sampling makes Word2Vec practical at scale

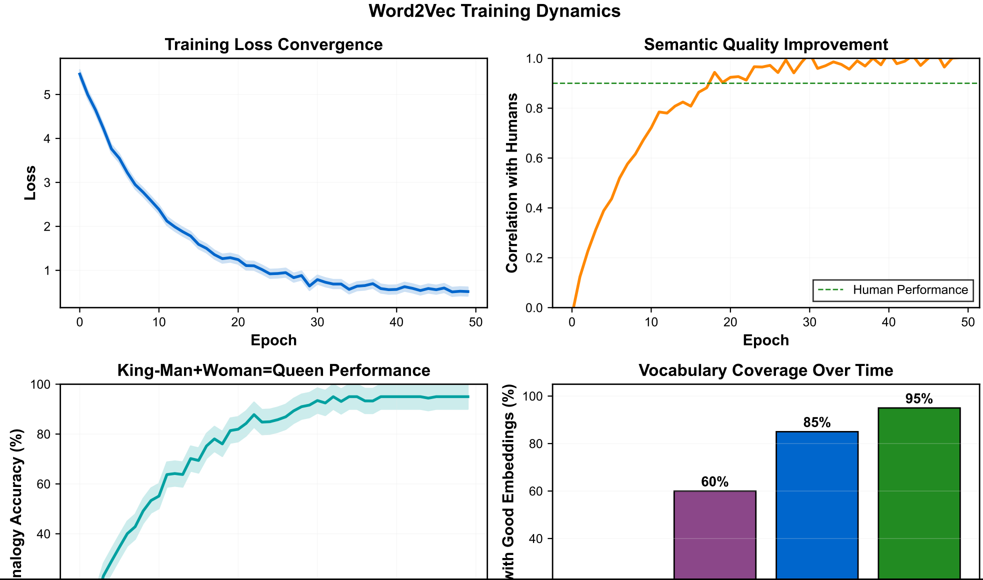
Softmax Computation Explained



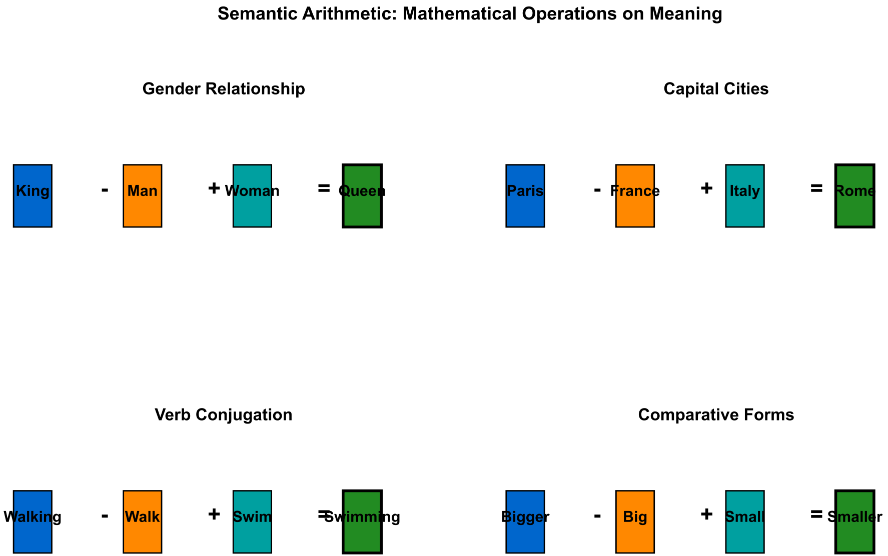
Negative Sampling: Before and After



Training Dynamics and Convergence



Semantic Arithmetic in Action



How Do We Know It Works?

Intrinsic Evaluation:

Word Similarity Tasks:

- Human ratings: cat-dog = 7.5/10
- Model similarity: cosine(cat, dog)
- Correlation with humans
- WordSim-353 dataset

Analogy Tasks:

- a:b :: c:?
- Berlin:Germany :: Paris:?
- Google analogy dataset
- 90%+ accuracy

Extrinsic Evaluation:

Downstream Tasks:

- Sentiment analysis
- Named entity recognition
- Machine translation
- Question answering

Real-world metrics:

- Search relevance ↑15%
- Translation BLEU ↑3.2
- Classification F1 ↑8%
- All from better embeddings!

Good embeddings improve everything downstream

Challenges: Not Everything Is Perfect

1. **Polysemy Problem:**
   - "bank" (financial) = "bank" (river)
   - One vector for all meanings
   - Averages different senses
   - Solution: Contextual embeddings (BERT)

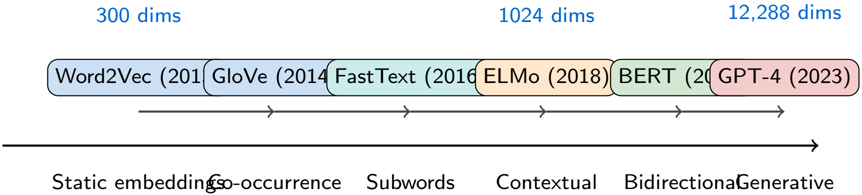
2. **Rare Words:**
   - Need many examples
   - "serendipity" appears rarely
   - Poor vectors for rare words
   - Solution: Subword embeddings

3. **Bias Amplification:**
   - Learns societal biases
   - Doctor:Male :: Nurse:Female
   - Amplifies stereotypes
   - Active research area

4. **Static Embeddings:**
   - Fixed after training
   - Can't adapt to new contexts
   - No fine-tuning possible
   - Solution: Transformer models

These limitations led to BERT and GPT development

From Word2Vec to ChatGPT: The Journey



Word2Vec's Legacy:

- Proved embeddings work
- Inspired contextual models
- Still used in production
- Foundation for all modern NLP

What Changed:

- Static → Contextual
- 300 dims → 12,000+ dims
- Word-level → Subword
- Millions → Billions of parameters

Build It: Semantic Search Engine

```python
import numpy as np
from gensim.models import Word2Vec

def semantic_search(query, documents, model):
    """Find semantically similar documents"""

    # Vectorize query
    query_vec = document_vector(query, model)

    # Vectorize all documents
    doc_vectors = [
        document_vector(doc, model)
        for doc in documents
    ]

    # Compute similarities
    similarities = [
        cosine_similarity(query_vec, doc_vec)
        for doc_vec in doc_vectors
    ]

    # Return ranked results
    ranked = sorted(
        zip(documents, similarities),
        key=lambda x: x[1],
        reverse=True
    )
```

How it works:

1. Convert query to vector
2. Convert documents to vectors
3. Find nearest neighbors
4. Return ranked results

Real Examples:

Query: "animal pets"

Results:

- "dog training tips"
- "cat care guide"
- "hamster habitats"

No keyword matching needed!

Week 2 Summary: Words Have Meaning!

- Words as **IDs** $\rightarrow$ Words as **vectors**
- Distributional hypothesis: Context defines meaning
- Word2Vec: Skip-gram + Negative sampling
- Mathematical semantics: King - Man + Woman = Queen
- From 50,000 sparse $\rightarrow$ 300 dense dimensions
- Powers modern NLP: Search, translation, chatbots
- Limitations led to BERT/GPT development

Key Technical Insights:

- Dot product captures similarity
- Negative sampling avoids softmax bottleneck
- Embeddings are the foundation of all modern NLP

Next Week: Recurrent Neural Networks

How do we process sequences using embeddings?