# Natural Language Processing Course

Week 2: Neural Language Models and Word Embeddings

Restructured with 4-Part Format

2024

## Week 2: Neural Language Models - Overview

**Part 1: Introduction & Motivation**

- Interactive word association
- The semantic understanding problem
- Real-world impact and applications
- Historical journey to Word2Vec

**Part 2: Core Concepts**

- Distributional hypothesis
- From discrete to continuous
- Word2Vec architecture
- Implementation deep dive

**Part 3: Challenges & Solutions**

- Training at scale
- Evaluation methodologies
- Fundamental limitations
- Advanced techniques

**Part 4: Applications & Future**

- Hands-on applications
- Modern evolution (BERT, GPT)
- Industry state-of-the-art
- Looking forward

**Goal:** Master how computers learn word meaning through context

## Part 1

## Introduction and Motivation

Why Computers Need to Understand Word Meaning

## Interactive Exercise: Word Association Game

When you see this word, what comes to mind?

**OCEAN**

## Interactive Exercise: Word Association Game

When you see this word, what comes to mind?

**OCEAN**

| | | | |
|---|---|---|---|
| **water** | **sea** | **beach** | **waves** |
| 35% of you | 25% of you | 20% of you | 20% of you |

**You naturally understand semantic relationships!**

**But until 2003, computers saw:**

- ocean = ID 7849
- water = ID 2341
- No connection whatsoever!

## The Semantic Gap: Computers vs Humans

**How Humans See Words:**

- cat $\approx$ kitten (similar animals)
- Paris $\leftrightarrow$ France (location relation)
- running $\sim$ ran (same verb, different tense)
- doctor $\leftrightarrow$ hospital (association)

Rich semantic network with relationships, similarities, and associations

**How Computers Saw Words (Pre-2003):**

- cat = 1247
- kitten = 8923
- Paris = 4567
- France = 2109

Arbitrary IDs with no notion of meaning or relationships

**The Challenge: Bridge this semantic gap!**

## Real System Failures Without Semantic Understanding

**Early Google Search (2000):**

- Search: "car" → Missed: "automobile", "vehicle"
- Search: "running shoes" → Missed: "jogging sneakers"

**Machine Translation Disasters:**

- "The spirit is willing but the flesh is weak"
- → Russian → English:
- "The vodka is good but the meat is rotten"

**Customer Service Chatbots (2005):**

- Customer: "I want to return my purchase"
- Bot: "I don't understand. Did you mean 'buy'?"
- → Couldn't link "return" with "refund", "exchange"

**Economic Impact:** Billions lost due to poor search and translation

## Where Word Embeddings Power Your Life (2024)

**Entertainment:**

- **Spotify:** 256-dim song embeddings
- **Netflix:** Show similarity vectors
- **TikTok:** Video understanding
- **YouTube:** Related videos

**Productivity:**

- **Gmail:** Smart compose (BERT)
- **Grammarly:** Context awareness
- **Notion AI:** Semantic search
- **Slack:** Message threading

**Commerce:**

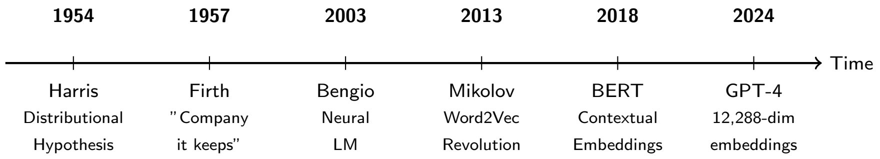
- **Amazon:** Product similarity
- **Google Ads:** Ad matching
- **Airbnb:** Listing embeddings
- **Uber:** Location understanding

**Market Size:**

- Embedding API Market: $2.7B by 2025
- OpenAI Embeddings: 1M+ developers
- Vector Database Market: $4.3B by 2028

**Every AI application today relies on word embeddings!**

## The Journey to Understanding: Timeline



**Key Breakthroughs:**

- **1954-1957:** Theoretical foundation - words defined by context
- **2003:** First neural language model with continuous representations
- **2013:** Word2Vec makes embeddings practical and scalable
- **2018:** Contextualized embeddings (same word, different contexts)
- **2024:** Massive embeddings powering GPT-4, Claude, Gemini

## The 2013 Breakthrough: King - Man + Woman = ?

**The demo that shocked the NLP world:**[1]

$$\text{king - man} + \text{woman} =$$

[1] Mikolov et al. (2013). "Linguistic regularities in continuous space word representations"

## The 2013 Breakthrough: King - Man + Woman = ?

**The demo that shocked the NLP world:**[1]

$$\text{king - man + woman = queen}$$

**Why this was revolutionary:**

- Computer discovered gender relationships automatically
- No one programmed these rules
- Learned purely from reading text
- Worked across many relationship types

**More examples that work:**

- Paris - France + Italy = Rome
- sushi - Japan + Mexico = tacos
- Einstein - scientist + artist = Picasso
- bigger - big + small = smaller
- walking - walk + swim = swimming
- CEO - company + country = president

[1]Mikolov et al. (2013). "Linguistic regularities in continuous space word representations"

## Part 1 Summary: Why This Matters

**Key Insights:**

1. **The Problem:** Computers treating words as meaningless IDs
2. **The Impact:** Billions in losses, poor user experiences
3. **The Solution:** Learn meaning from context (distributional hypothesis)
4. **The Breakthrough:** Word2Vec made it practical (2013)

**What's Next:**

- Part 2: How do we actually create these word vectors?
- Understanding the mathematics and algorithms
- Building Word2Vec from scratch

**Remember:** Every modern AI system (ChatGPT, Claude, Gemini) started here!

## Part 2

## Core Concepts

How Computers Learn Word Meaning from Context

## The Distributional Hypothesis: Foundation

**Core Principle (Firth, 1957):**

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

**Example: What is a "zorb"?**

- The zorb ate the cheese
- I saw a zorb in my garden
- The zorb ran under the couch
- My cat chased the zorb

## The Distributional Hypothesis: Foundation

**Core Principle (Firth, 1957):**

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

**Example: What is a "zorb"?**

- The zorb ate the cheese
- I saw a zorb in my garden
- The zorb ran under the couch
- My cat chased the zorb

**You probably guessed: zorb = mouse (or similar small animal)**

**Mathematical Formulation:**

- Words with similar distributions have similar meanings
- $\text{similarity}(w_1, w_2) \propto P(\text{context}|w_1) \cdot P(\text{context}|w_2)$
- Context defines meaning!

## Interactive: Guess the Word from Context

**Mystery word = [BLANK]. What is it?**

1. The [BLANK] was delicious
2. I ordered [BLANK] with extra cheese
3. The [BLANK] delivery arrived in 30 minutes
4. We shared a large [BLANK] at the party
5. My favorite [BLANK] topping is pepperoni

## Interactive: Guess the Word from Context

**Mystery word = [BLANK]. What is it?**

1. The [BLANK] was delicious
2. I ordered [BLANK] with extra cheese
3. The [BLANK] delivery arrived in 30 minutes
4. We shared a large [BLANK] at the party
5. My favorite [BLANK] topping is pepperoni

**Answer: pizza**

**This is exactly how Word2Vec learns:**

- Sees millions of sentences
- Learns what words appear in similar contexts
- Groups them close together in vector space
- No dictionary needed!

## From Discrete IDs to Continuous Vectors

**One-Hot Encoding (Old Way):**

- Vocabulary size: 50,000 words
- cat = [0,0,1,0,0,...,0] (50K dimensions!)
- dog = [0,0,0,1,0,...,0]

**Problems:**

- No similarity: cat $\cdot$ dog = 0
- Huge vectors: 50K dimensions
- Sparse: 49,999 zeros
- No learning: Fixed representation

**Dense Embeddings (Word2Vec):**

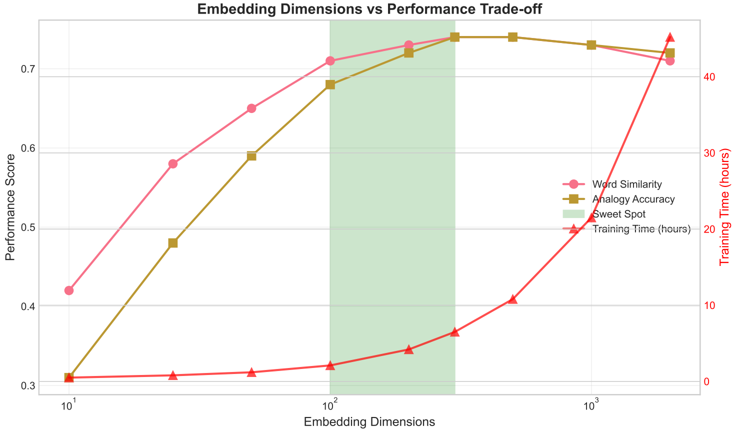
- Typical size: 100-300 dimensions
- cat = [0.2, -0.4, 0.7, ..., 0.1]
- dog = [0.3, -0.3, 0.6, ..., 0.2]

**Benefits:**

- Similarity: cat $\cdot$ dog = 0.89
- Compact: 100-300 dims
- Dense: All values meaningful
- Learnable: Updated during training

**Key: Every dimension captures some semantic property**

## The Goldilocks Zone: Why 100-300 Dimensions?

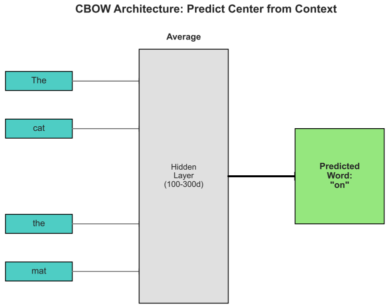


**Empirical Findings:**

- **¡ 50 dims:** Too compressed, loses nuances
- **100-300 dims:** Sweet spot for most tasks
- **¿ 500 dims:** Diminishing returns, overfitting risk

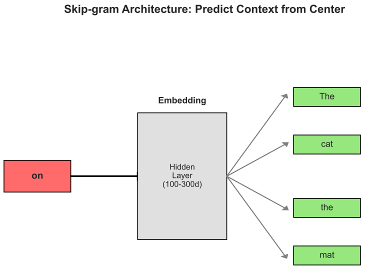
## Word2Vec: Two Architectures

**CBOW (Continuous Bag of Words):**



- Predict center from context
- Input: [the, cat, on, mat]
- Output: sat
- Faster to train
- Better for frequent words

**Skip-gram:**



- Predict context from center
- Input: sat
- Output: [the, cat, on, mat]
- Slower but more accurate
- Better for rare words

## Skip-gram Training: Step by Step

**Sentence:** "The quick brown fox jumps"

**Window size = 2** (look 2 words left and right)

| **Step** | **Center Word** | **Context to Predict** |
|---|---|---|
| 1 | quick | [the, brown] |
| 2 | brown | [the, quick, fox, jumps] |
| 3 | fox | [quick, brown, jumps] |

**Training Process:**

1. Take center word embedding
2. Try to predict context words
3. Measure prediction error
4. Update embeddings to reduce error
5. Repeat millions of times

**Result: Words appearing in similar contexts get similar embeddings**

## Implementing Word2Vec in PyTorch

```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class Word2Vec(nn.Module):
    def __init__(self, vocab_size, embed_dim=100):
        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, neg_samples):
        # Get embeddings
        center_emb = self.center_embeddings(center)
        context_emb = self.context_embeddings(context)
        neg_emb = self.context_embeddings(neg_samples)

        # Positive samples (should be similar)
        pos_score = torch.sum(
            center_emb * context_emb, dim=1
        )
        pos_loss = F.logsigmoid(pos_score)

        # Negative samples (should be different)
        neg_score = torch.bmm(
            neg_emb, center_emb.unsqueeze(2)
        ).squeeze()
        neg_loss = F.logsigmoid(-neg_score).sum(1)

        return -(pos_loss + neg_loss).mean()
```

**Key Components:**

- **Two matrices:** Center and context embeddings
- **Positive samples:** Real context words
- **Negative samples:** Random words (not in context)

**Training Trick:**

- Full softmax over 50K words is expensive
- Solution: Negative sampling
- Only update a few random words
- 5-20 negative samples typical

**Loss Function:**

- Maximize similarity with real context
- Minimize similarity with random words

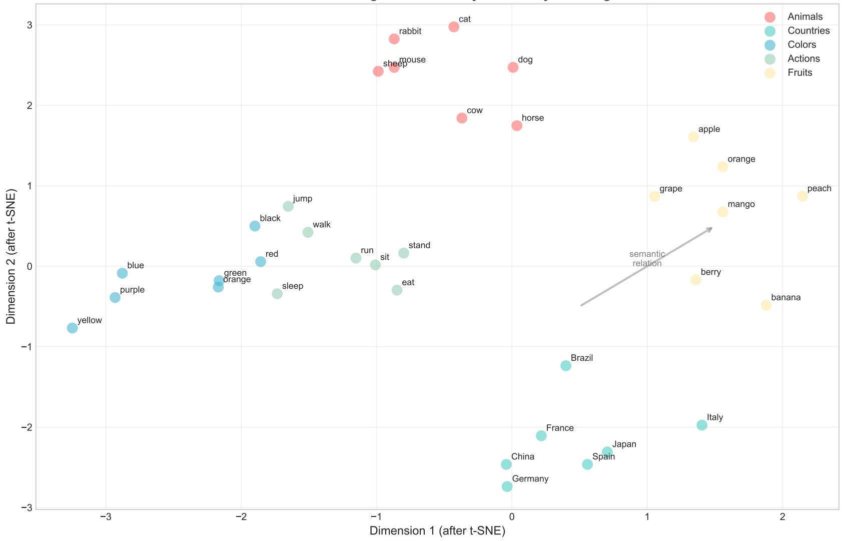
## Training Word2Vec: The Complete Loop

```python
def train_word2vec(corpus, vocab_size, embed_dim=100, epochs=5, window=2):
    model = Word2Vec(vocab_size, embed_dim)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

    for epoch in range(epochs):
        total_loss = 0
        for sentence in corpus:
            # Generate training samples from sentence
            for i, center_word in enumerate(sentence):
                # Get context words within window
                context_words = []
                for j in range(max(0, i-window), min(len(sentence), i+window+1)):
                    if i != j:
                        context_words.append(sentence[j])

                # Get negative samples (5 random words not in context)
                neg_samples = get_negative_samples(vocab_size, 5, avoid=context_words)

                # Forward pass
                loss = model(center_word, context_words, neg_samples)

                # Backward pass
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()

                total_loss += loss.item()

        print(f"Epoch {epoch}: Loss = {total_loss:.4f}")

    return model.center_embeddings.weight.data  # Final embeddings
```

**Result:** After training on millions of sentences, similar words cluster together!

## Visualizing What Word2Vec Learns

Word Embeddings Automatically Cluster by Meaning



## Mathematical Intuition: Why Dot Product = Similarity

**The Skip-gram Objective:**

$$\max \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log P(w_{t+j} | w_t)$$

**Where probability is defined as:**

$$P(w_O | w_I) = \frac{\exp(v_{w_O}^T v_{w_I})}{\sum_{w=1}^{W} \exp(v_w^T v_{w_I})}$$

**Key Insight:**

- Dot product $v_{w_O}^T v_{w_I}$ measures similarity
- Higher dot product $\rightarrow$ higher probability of co-occurrence
- Training maximizes dot product for words that appear together
- Result: Similar words have high dot product (cosine similarity)

Geometry emerges from statistics: Similar contexts $\rightarrow$ Similar vectors

## Part 2 Summary: Core Concepts Mastered

**What We Learned:**

1. **Distributional Hypothesis:** Context defines meaning
2. **Dense Vectors:** 100-300 dimensions capture semantics
3. **Skip-gram Model:** Predict context from center word
4. **Training Process:** Maximize co-occurrence probability
5. **Implementation:** Two embedding matrices + negative sampling

**Key Takeaways:**

- Word meaning emerges from statistical patterns
- No linguistic knowledge required
- Scalable to millions of words
- Foundation for all modern NLP

**Next: Part 3 - Challenges and Solutions**

- How to train on billions of words efficiently?
- How to evaluate embedding quality?
- What are the limitations?

**Part 3**

## Challenges and Solutions

Scaling, Evaluation, and Limitations

## Challenge 1: Computational Complexity

**The Softmax Bottleneck:**
Original formulation requires normalizing over entire vocabulary:

$$P(w_O|w_I) = \frac{\exp(v_{w_O}^T v_{w_I})}{\sum_{w=1}^{W} \exp(v_w^T v_{w_I})}$$

**Problem:**

- Vocabulary size W = 50,000+ words
- Must compute 50,000 dot products per training step
- Billions of training steps needed
- Computationally infeasible!

**Solutions:**

1. **Hierarchical Softmax:**
   - Binary tree of words
   - $O(\log W)$ instead of $O(W)$
   - Path through tree to each word
2. **Negative Sampling:**
   - Only update k random words
   - Typically k = 5-20
   - Dramatic speedup
   - Better performance!

## Solution: Negative Sampling Explained

**Instead of:** Predicting the right word from 50,000 options
**We ask:** Is this word the right context word? (Binary classification)

| Center | Word | Label |
|---|---|---|
| cat | sits (real context) | 1 |
| cat | on (real context) | 1 |
| cat | elephant (random) | 0 |
| cat | democracy (random) | 0 |
| cat | quantum (random) | 0 |

**Sampling Strategy:**

- Sample negative words by frequency: $P(w) \propto f(w)^{3/4}$
- The 3/4 power reduces dominance of very common words
- Gives rare words more chance to be negative samples

**Result: 1000x speedup with better quality embeddings!**

## Challenge 2: How Do We Evaluate Embeddings?

**The Problem:** How do we know if our embeddings are "good"?

**Intrinsic Evaluation:**

- **Word Similarity:**
  - Dataset: WordSim-353
  - Human ratings vs cosine similarity
  - Correlation: 0.6-0.7 typical
- **Word Analogies:**
  - king - man + woman = ?
  - Google analogy dataset
  - Accuracy: 60-75% typical

**Extrinsic Evaluation:**

- **Downstream Tasks:**
  - Sentiment analysis
  - Named entity recognition
  - Machine translation
- **Key Finding:**
  - Good intrinsic $\neq$ Good extrinsic
  - Task-specific fine-tuning helps

**Modern Approach (2024):**

- Skip intrinsic evaluation
- Directly evaluate on downstream tasks
- Use pre-trained embeddings as starting point

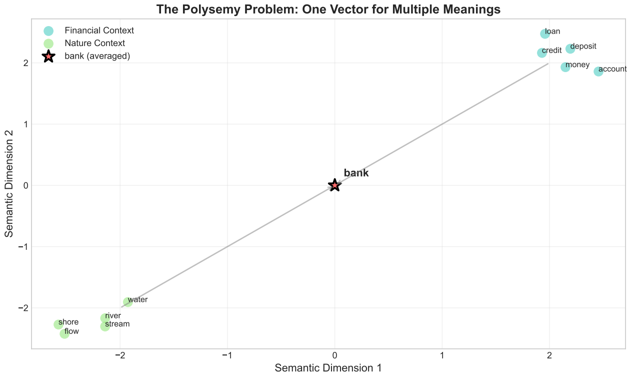
## Challenge 3: The Polysemy Problem

**One Vector Per Word... But Words Have Multiple Meanings!**

**Example: "bank"**

- "I deposited money at the bank" (financial institution)
- "We sat by the river bank" (edge of river)

**Word2Vec gives one vector that averages both meanings:**

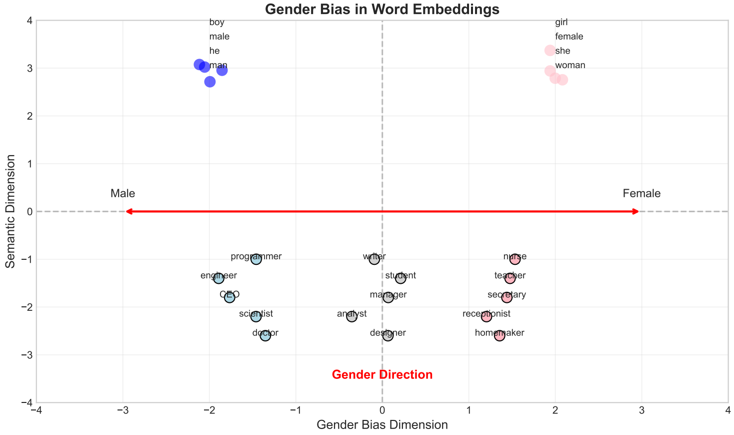


## Challenge 4: Bias in Embeddings

**Embeddings Learn Societal Biases from Text:**

**Problematic Analogies Found:**

- man : computer programmer :: woman : homemaker
- man : doctor :: woman : nurse



## Advanced Techniques: Beyond Basic Word2Vec

**FastText (2016):**

- Uses character n-grams
- "where" = "wh", "whe", "her", "ere", "re"
- Handles unseen words
- Better for morphologically rich languages

**GloVe (2014):**

- Global matrix factorization
- Combines count-based and predictive
- Often better for word analogies

**ELMo (2018):**

- Contextualized embeddings
- Different vector per context
- Bi-directional LSTM
- Solves polysemy problem

**Modern (2024):**

- BERT/GPT embeddings
- Learned during pre-training
- Task-specific fine-tuning
- 768-12,288 dimensions

Word2Vec pioneered the field, but modern methods build on its foundation

## Part 3 Summary: Challenges Addressed

**Challenges We Explored:**

1. **Computational:** Softmax over 50K words $\rightarrow$ Negative sampling
2. **Evaluation:** Intrinsic vs extrinsic metrics
3. **Polysemy:** One vector per word limitation
4. **Bias:** Embeddings reflect societal biases

**Solutions and Evolution:**

- Negative sampling: 1000x speedup
- Task-specific evaluation
- Contextualized embeddings (BERT/GPT)
- Debiasing techniques

**Next: Part 4 - Applications and Future**

- Build real applications with embeddings
- See modern systems in action
- Understand the path forward

**Part 4**

## Applications and Future

From Word2Vec to Modern AI Systems

## Build It: Semantic Search Engine

**Let's build a search engine that understands meaning!**

**Traditional Search:**

- Query: "car"
- Finds: Only documents with "car"
- Misses: "automobile", "vehicle"

**Semantic Search:**

- Query: "car"
- Finds: "car", "automobile", "vehicle", "BMW"
- Understands synonyms and related concepts

**Implementation Steps:**

1. Load pre-trained Word2Vec embeddings
2. Convert documents to vectors (average word embeddings)
3. Convert query to vector
4. Find documents with highest cosine similarity
5. Return ranked results

This is the foundation of Google Search, Elastic Search, and more!

## Semantic Search Implementation

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

class SemanticSearch:
    def __init__(self, word2vec_model):
        self.w2v = word2vec_model
        self.documents = []
        self.doc_vectors = []

    def add_document(self, doc):
        """Add document to search index"""
        self.documents.append(doc)
        # Convert document to vector (average of word vectors)
        words = doc.lower().split()
        vectors = [self.w2v[word] for word in words if word in self.w2v]
        doc_vector = np.mean(vectors, axis=0) if vectors else np.zeros(100)
        self.doc_vectors.append(doc_vector)

    def search(self, query, top_k=5):
        """Find most similar documents to query"""
        # Convert query to vector
        words = query.lower().split()
        vectors = [self.w2v[word] for word in words if word in self.w2v]
        query_vector = np.mean(vectors, axis=0) if vectors else np.zeros(100)

        # Calculate similarities
        similarities = cosine_similarity([query_vector], self.doc_vectors)[0]

        # Return top k results
        top_indices = np.argsort(similarities)[::-1][:top_k]
        return [(self.documents[i], similarities[i]) for i in top_indices]
```

## Real-World Applications (2024)

**Content Recommendation:**

- Netflix: Show embeddings
- Spotify: Song2Vec
- YouTube: Video embeddings
- Amazon: Product2Vec

**Language Understanding:**

- ChatGPT: Token embeddings
- Google Translate: Multilingual embeddings
- Grammarly: Context understanding

**Search and Retrieval:**

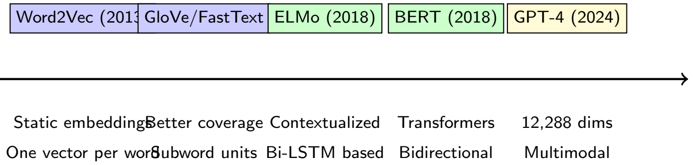
- Google: Semantic search
- Bing: Neural matching
- Enterprise search: Document similarity

**Novel Applications:**

- Code2Vec: Programming embeddings
- Molecule2Vec: Drug discovery
- Graph2Vec: Social networks

**Any data with context can use embedding techniques!**

## Evolution: From Word2Vec to GPT



**Key Progression:**

- Static $\rightarrow$ Contextualized
- Words $\rightarrow$ Subwords $\rightarrow$ Tokens
- 300 dims $\rightarrow$ 12,288 dims
- Single task $\rightarrow$ Multi-task $\rightarrow$ General intelligence

## Future Directions: What's Next?

**Current Research (2024):**

- **Efficient Embeddings:** Maintain quality at 64 dimensions
- **Multimodal:** Text + Image + Audio in same space
- **Dynamic:** Embeddings that update with new information
- **Personalized:** User-specific embedding spaces

**Challenges Being Solved:**

- **Long Context:** Embed entire books (1M+ tokens)
- **Cross-lingual:** Universal embeddings for all languages
- **Interpretability:** Understanding what each dimension means
- **Continual Learning:** Updating without forgetting

**Connection to Next Week:**

- Week 3: RNNs - Processing sequences with embeddings
- Embeddings are the input to all modern NLP models
- Foundation we'll build on for rest of course

## Week 2 Summary: Words Have Meaning!

**Journey We Took:**

1. Started with words as meaningless IDs
2. Learned the distributional hypothesis
3. Built Word2Vec from scratch
4. Tackled challenges (scale, bias, polysemy)
5. Applied to real problems

**Key Takeaways:**

- **Core Insight:** Similar contexts $\rightarrow$ Similar meanings
- **Technical:** Skip-gram + negative sampling = efficient training
- **Practical:** Embeddings power all modern AI
- **Evolution:** Static $\rightarrow$ Contextualized $\rightarrow$ Multimodal

**Your Homework:**

- Build semantic search engine (notebook provided)
- Explore biases in pre-trained embeddings
- Try word arithmetic with different models

**Remember:** Every ChatGPT response starts with embeddings!

## Appendix A: Skip-gram Objective Derivation

**Full Mathematical Formulation:**
Given a sequence of words $w_1, w_2, ..., w_T$, maximize:

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log P(w_{t+j}|w_t)$$

Where:

$$P(w_O|w_I) = \frac{\exp(v'^{T}_{w_O} v_{w_I})}{\sum_{w=1}^{W} \exp(v'^{T}_{w} v_{w_I})}$$

**Gradient with respect to center word:**

$$\frac{\partial \log P(w_O|w_I)}{\partial v_{w_I}} = v'_{w_O} - \sum_{w=1}^{W} P(w|w_I) \cdot v'_w$$

**Interpretation:**

- First term: Move toward actual context word
- Second term: Move away from expected context (all words weighted by probability)
- Result: Embeddings organize by co-occurrence patterns

## Appendix A: Negative Sampling Mathematics

**Original objective (expensive):**

$$\log P(w_O|w_I) = \log \frac{\exp({v'}_{w_O}^T v_{w_I})}{\sum_{w=1}^{W} \exp({v'}_w^T v_{w_I})}$$

**Negative sampling objective (efficient):**

$$\log \sigma({v'}_{w_O}^T v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)}[\log \sigma(-{v'}_{w_i}^T v_{w_I})]$$

Where:

- $\sigma(x) = \frac{1}{1+e^{-x}}$ (sigmoid function)
- $k =$ number of negative samples (typically 5-20)
- $P_n(w) \propto f(w)^{3/4}$ (noise distribution)

**Why it works:**

- Transforms problem to binary classification
- Distinguishing real from noise is sufficient
- Dramatically reduces computation: O(k) instead of O(W)

## Concrete Example: Computing P(cat—context)

**Given:** "The <u>cat</u> sat on the mat" with window=1

**Simplified vocabulary:** {the, cat, sat, on, mat} = {0, 1, 2, 3, 4}

**Embeddings (2D for visualization):**

| Word | $v$ (input) | $v'$ (output) |
|---|---|---|
| the | [0.5, 0.3] | [0.4, 0.6] |
| cat | [0.8, 0.2] | [0.7, 0.4] |
| sat | [0.3, 0.7] | [0.5, 0.8] |
| on | [0.4, 0.5] | [0.3, 0.6] |
| mat | [0.6, 0.4] | [0.8, 0.3] |

**Calculate P(sat—cat):**

$$v_{cat}^T \cdot v'_{sat} = [0.8, 0.2] \cdot [0.5, 0.8] = 0.4 + 0.16 = 0.56 \quad (1)$$

$$\exp(0.56) = 1.75 \quad (2)$$

$$Z = \sum_w \exp(v_{cat}^T \cdot v'_w) = 1.73 + 1.75 + 1.61 + 1.87 + 1.94 = 8.9 \quad (3)$$

$$P(sat|cat) = \frac{1.75}{8.9} = 0.197 \approx 19.7\% \quad (4)$$

## Concrete Example: Gradient Update

**One gradient step for center word "cat":**

**Initial:** $v_{cat} = [0.8, 0.2]$, learning rate $\alpha = 0.1$

**Gradient components:**

- Positive: Pull toward "sat" (actual context)
- Negative: Push away from expected distribution

**Calculation:**

$$\begin{aligned}
\nabla &= v'_{sat} - \sum_w P(w|cat) \cdot v'_w && (5) \\
&= [0.5, 0.8] - (0.194 \cdot [0.4, 0.6] + 0.197 \cdot [0.5, 0.8] + ...) && (6) \\
&= [0.5, 0.8] - [0.48, 0.62] && (7) \\
&= [0.02, 0.18] && (8)
\end{aligned}$$

**Update:**

$$v_{cat}^{new} = v_{cat} + \alpha \cdot \nabla = [0.8, 0.2] + 0.1 \cdot [0.02, 0.18] = [0.802, 0.218]$$

**Result: "cat" moves slightly closer to "sat" in embedding space!**

## Concrete Example: Negative Sampling Calculation

**Training instance:** Center="cat", Positive="sat", Negatives={the, mat}

**Loss calculation step-by-step:**

**1. Positive pair (cat, sat):**

$$\text{score}_{pos} = v_{cat}^T \cdot v'_{sat} = 0.56 \tag{9}$$

$$\text{loss}_{pos} = -\log \sigma(0.56) = -\log(0.636) = 0.452 \tag{10}$$

**2. Negative pairs:**

$$\text{score}_{neg1} = v_{cat}^T \cdot v'_{the} = 0.52 \tag{11}$$

$$\text{loss}_{neg1} = -\log \sigma(-0.52) = -\log(0.373) = 0.987 \tag{12}$$

$$\text{score}_{neg2} = v_{cat}^T \cdot v'_{mat} = 0.70 \tag{13}$$

$$\text{loss}_{neg2} = -\log \sigma(-0.70) = -\log(0.332) = 1.103 \tag{14}$$

**3. Total loss:**

$$\text{Loss} = \text{loss}_{pos} + \text{loss}_{neg1} + \text{loss}_{neg2} = 0.452 + 0.987 + 1.103 = 2.542$$

**Much faster: Only 3 calculations instead of 5,000!**

## Concrete Example: Cosine Similarity Step-by-Step

**Question:** How similar are "king" and "queen"?

**Given embeddings (3D for simplicity):**

- $v_{king} = [0.6, 0.8, 0.2]$
- $v_{queen} = [0.5, 0.7, 0.4]$

**Step 1: Calculate dot product**

$$v_{king} \cdot v_{queen} = (0.6 \times 0.5) + (0.8 \times 0.7) + (0.2 \times 0.4) = 0.3 + 0.56 + 0.08 = 0.94$$

**Step 2: Calculate magnitudes**

$$||v_{king}|| = \sqrt{0.6^2 + 0.8^2 + 0.2^2} = \sqrt{0.36 + 0.64 + 0.04} = 1.02 \tag{15}$$

$$||v_{queen}|| = \sqrt{0.5^2 + 0.7^2 + 0.4^2} = \sqrt{0.25 + 0.49 + 0.16} = 0.95 \tag{16}$$

**Step 3: Compute cosine similarity**

$$\cos(\theta) = \frac{v_{king} \cdot v_{queen}}{||v_{king}|| \times ||v_{queen}||} = \frac{0.94}{1.02 \times 0.95} = \frac{0.94}{0.969} = 0.97$$

**Interpretation: 0.97 = Very similar! (1.0 = identical, 0 = orthogonal)**

## Appendix B: Hands-On Notebook Exercises

**Available in: week02_word2vec_notebook.ipynb**

**Exercise 1: Train Word2Vec from Scratch**

- Load text corpus (Wikipedia sample)
- Implement skip-gram with negative sampling
- Visualize training progress
- Save trained embeddings

**Exercise 2: Word Arithmetic Playground**

- Load pre-trained embeddings
- Implement analogy solver
- Test on Google analogy dataset
- Create your own analogies

**Exercise 3: Bias Detection**

- Measure gender bias in embeddings
- Visualize bias directions
- Implement simple debiasing
- Compare before/after

**Exercise 4: Build Applications**

- Semantic search engine
- Document clustering
- Simple chatbot with semantic understanding

## Appendix C: Domain-Specific Embeddings

**Medical Embeddings (BioWordVec):**

- Trained on PubMed + MIMIC-III
- Captures: drug-disease relationships
- Application: Clinical decision support

**Legal Embeddings (Law2Vec):**

- Trained on case law + statutes
- Captures: Legal concept similarity
- Application: Legal document search

**Financial Embeddings (FinBERT):**

- Trained on financial news + reports
- Captures: Market sentiment
- Application: Trading signals

**Code Embeddings (CodeBERT):**

- Trained on GitHub repositories
- Captures: Programming patterns
- Application: Code search, bug detection

**Lesson: Domain-specific training dramatically improves performance**

## References and Resources

**Essential Papers:**

- Mikolov et al. (2013). "Efficient estimation of word representations in vector space"
- Mikolov et al. (2013). "Distributed representations of words and phrases"
- Pennington et al. (2014). "GloVe: Global vectors for word representation"
- Peters et al. (2018). "Deep contextualized word representations" (ELMo)

**Implementations:**

- Gensim: `https://radimrehurek.com/gensim/`
- FastText: `https://fasttext.cc/`
- Hugging Face: `https://huggingface.co/`

**Datasets:**

- Google Analogy Test Set
- WordSim-353
- SimLex-999

**Next Week:** Recurrent Neural Networks - Processing sequences with embeddings