Neural Language Models and Word Embeddings

Week 2: From Discrete Symbols to Continuous Representations

NLP Course 2025

Professional Template Edition

September 29, 2025

Week 2: Comprehensive Overview

Learning Journey: From one-hot encodings to dense semantic vectors. Master Word2Vec, understand negative sampling, explore semantic arithmetic, and build the foundation for modern NLP.

Learning Objectives

By the end of this lecture, you will:

Understand

- Why one-hot encodings fail
- Distributional hypothesis principle
- Word2Vec architectures (CBOW vs Skip-gram)
- Negative sampling optimization
- Semantic vector arithmetic

Be Able To

- Train word embeddings
- Evaluate embedding quality
- Visualize semantic spaces
- Apply embeddings to NLP tasks
- Debug common issues

Core Message: Words are not just symbols - they carry meaning in their vector geometry

Prerequisites: Basic linear algebra, probability theory, neural network fundamentals

Part 1: Motivation & Problem Setting

Why do we need word embeddings?

Interactive Exercise: Word Association

Complete these sentences - What word naturally comes next?

1. The cat sat on the ______________

Interactive Exercise: Word Association

Complete these sentences - What word naturally comes next?

1. The cat sat on the ______________

   mat, floor, chair

2. Coffee with milk and ______________

Interactive Exercise: Word Association

Complete these sentences - What word naturally comes next?

1. The cat sat on the ____________

   mat, floor, chair

2. Coffee with milk and ____________

   sugar, cream

3. The capital of France is ____________

Interactive Exercise: Word Association

Complete these sentences - What word naturally comes next?

1. The cat sat on the ___________
   mat, floor, chair
2. Coffee with milk and ___________
   sugar, cream
3. The capital of France is ___________
   Paris
4. She was happy but also ___________

Interactive Exercise: Word Association

Complete these sentences - What word naturally comes next?

1. The cat sat on the ____________
   mat, floor, chair
2. Coffee with milk and ____________
   sugar, cream
3. The capital of France is ____________
   Paris
4. She was happy but also ____________
   sad, anxious, tired
5. King is to queen as man is to ____________

Interactive Exercise: Word Association

Complete these sentences - What word naturally comes next?

1. The cat sat on the ____________
   mat, floor, chair
2. Coffee with milk and ____________
   sugar, cream
3. The capital of France is ____________
   Paris
4. She was happy but also ____________
   sad, anxious, tired
5. King is to queen as man is to ____________
   woman
6. Python is a programming ____________

Interactive Exercise: Word Association

Complete these sentences - What word naturally comes next?

1. The cat sat on the ______________
   mat, floor, chair
2. Coffee with milk and ______________
   sugar, cream
3. The capital of France is ______________
   Paris
4. She was happy but also ______________
   sad, anxious, tired
5. King is to queen as man is to ______________
   woman
6. Python is a programming ______________
   language

Interactive Exercise: Word Association

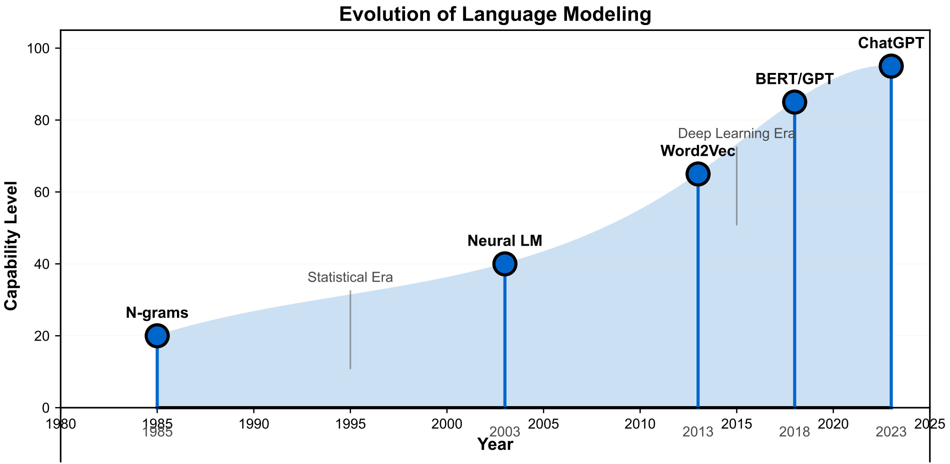
Complete these sentences - What word naturally comes next?

1. The cat sat on the ____________
   mat, floor, chair
2. Coffee with milk and ____________
   sugar, cream
3. The capital of France is ____________
   Paris
4. She was happy but also ____________
   sad, anxious, tired
5. King is to queen as man is to ____________
   woman
6. Python is a programming ____________
   language

Key Insight: You use semantic understanding to predict words!

Humans naturally understand word relationships - how can we teach this to machines?

Historical Context: The Evolution Journey



1980s-2000s

- N-gram models

2003-2013

- Neural LMs

2013-2017

- Word2Vec era

2017-Present

- Transformers

The Fundamental Problem: One-Hot Encoding

Traditional Approach

- Vocabulary size: 50,000 words
- Each word = unique ID
- One-hot vectors:

cat =	[0,0,1,0,0,...,0]
dog =	[0,0,0,1,0,...,0]
car =	[0,0,0,0,1,...,0]
democracy =	[0,0,0,0,0,...,1]

Critical Problems:

- No similarity information
- distance(cat, dog) = distance(cat, car)
- 50,000-dimensional vectors!
- Can't generalize patterns

Mathematical Issues

Cosine similarity between any two words:

$$\cos(\vec{w}_i, \vec{w}_j) = \frac{\vec{w}_i \cdot \vec{w}_j}{|\vec{w}_i| \cdot |\vec{w}_j|} = 0$$

All words are orthogonal!

Storage nightmare:

- 1M vocabulary = 1M dimensions
- Neural network input: 1M $\times$ hidden size
- Impossible to scale

One-hot encoding treats all words as equally different - destroying semantic information

Why This Problem Matters

Example Task: Sentiment Analysis

Training data:

- "This movie is great" → Positive
- "This film is excellent" → ?

With one-hot encoding:

- "great" and "excellent" unrelated
- Can't transfer knowledge
- Need examples for every word

With embeddings:

- Similar words → similar vectors
- Knowledge transfers automatically
- Generalizes to unseen combinations

Real-World Impact

Search Engines:

- Query: "cheap cars"
- Finds: "affordable vehicles"

Translation:

- Unseen word combinations
- Cross-lingual transfer

Recommendations:

- Similar content discovery
- User preference modeling

Chatbots:

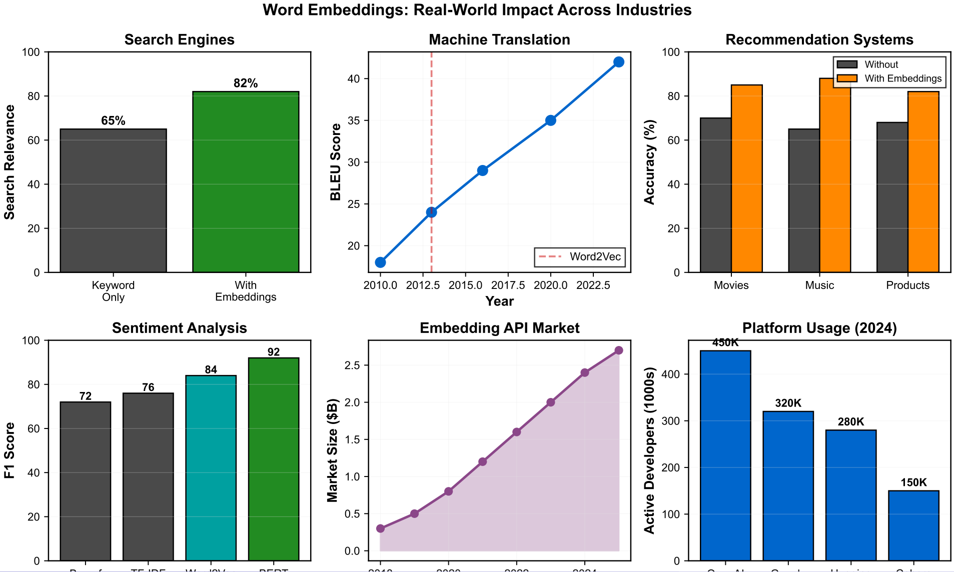
- Intent understanding
- Paraphrase detection

Dense embeddings enable knowledge transfer - the foundation of modern NLP success

The Solution: Dense Distributed Representations

../figures/sparse_to_dense.pdf

Real-World Applications Today



Part 2: Core Theory & Architecture

How Word2Vec learns meaning from context

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

- J.R. Firth (1957)

Core Principle: Words appearing in similar contexts have similar meanings

Example Sentences:

- The cat sat on the mat
- The dog sat on the floor
- A cat chased the mouse
- A dog chased the ball
- The cat is sleeping
- The dog is sleeping

Shared Contexts:

- Both follow "The" and "A"
- Both precede "sat", "chased", "is"
- Similar syntactic positions

Context Windows:

Window size = 2:

The	cat	mllavender3sat	on	the
←context		target	context→	

Mathematical View:

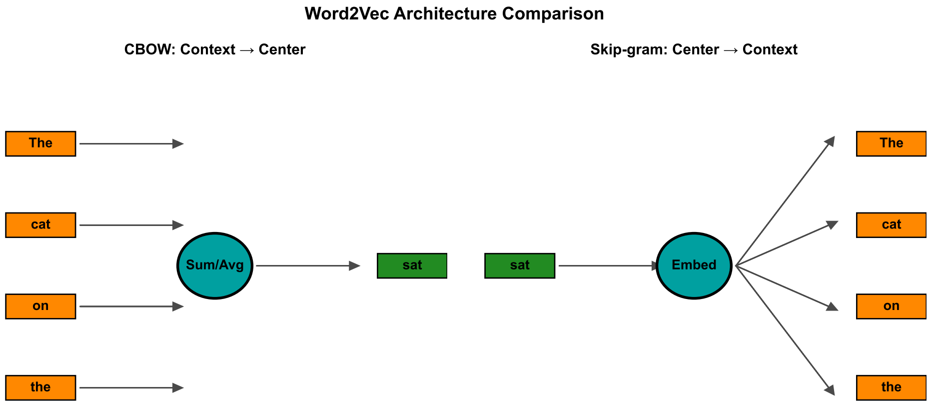
$$\text{context}(\text{cat}) = \{\textit{the}, \textit{sat}, \textit{on}, \textit{chased}, \textit{is}, \ldots\}$$

$$\text{context}(\text{dog}) = \{\textit{the}, \textit{sat}, \textit{on}, \textit{chased}, \textit{is}, \ldots\}$$

$$\Downarrow$$

$$\text{similar contexts} \Rightarrow \text{similar vectors}$$

Word2Vec: Two Complementary Architectures



CBOW (Continuous Bag-of-Words)

- Predict center from context

Skip-gram

- Predict context from center

Skip-gram: Mathematical Foundation

Objective: Maximize probability of context words given center word

Formal Objective Function:

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

Where:

- T = Total words in corpus
- c = Context window size
- w_t = Center word at position t
- w_{t+j} = Context word at offset j

Probability using Softmax:

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

Two Embedding Matrices:

- $W_{in} \in \mathbb{R}^{|V| \times d}$ (input embeddings)
- $W_{out} \in \mathbb{R}^{|V| \times d}$ (output embeddings)

Computational Problem:

- Denominator sums over entire vocabulary
- 50,000 words = 50,000 exponentials!
- Too expensive per training step

The softmax normalization is beautiful mathematically but computationally prohibitive

Negative Sampling: Making Training Feasible

Key Idea: Convert to binary classification problem

Instead of Full Softmax:

- Classify real vs fake pairs
- Positive: (center, actual context)
- Negative: (center, random word)
- Much faster computation

Example:

- Center word: "sat"
- Positive: (sat, cat) $\rightarrow$ 1
- Negative samples:
  - (sat, democracy) $\rightarrow$ 0
  - (sat, quantum) $\rightarrow$ 0
  - (sat, purple) $\rightarrow$ 0

New Objective Function:

$$\log \sigma(v_{w_O}^T \cdot v_{w_I}) +$$

$$\sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)}[\log \sigma(-v_{w_i}^T \cdot v_{w_I})]$$

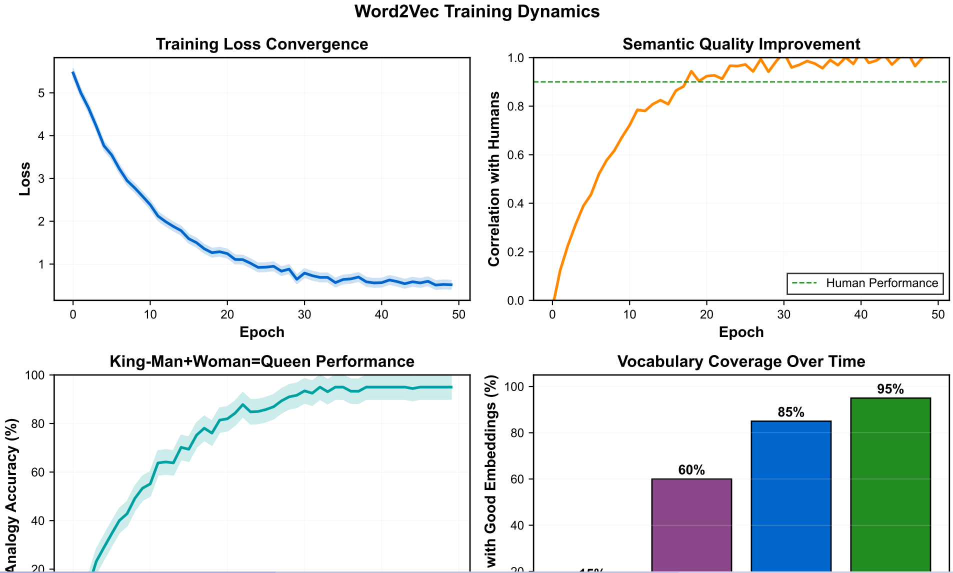
Where:

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

Computational Speedup: From O(—V—) to O(k) where k ¡¡ —V—

Negative sampling reduces 50,000 computations to just 5-20 - enabling practical training

Training Dynamics: How Embeddings Evolve



Implementation: Critical Details

Preprocessing:

- Lowercase text
- Remove punctuation (optional)
- Build vocabulary
- Subsampling threshold: 10^{-3}

Initialization:

```
# Random init [-0.5, 0.5] / dim
W_in = (np.random.rand(V, d) - 0.5) / d
W_out = np.zeros((V, d))
```

Hyperparameters:

- Embedding dim: 100-300
- Window size: 5-10
- Negative samples: 5-20
- Min word frequency: 5

Optimization Tricks:

1. Hierarchical Softmax:

- Binary tree over vocabulary
- O(log —V—) instead of O(—V—)
- Good for large vocabularies

2. Subsampling Frequent Words:

- "the", "a", "is" less informative
- Probability of keeping: $P(w_i) = \sqrt{\frac{t}{f(w_i)}}$
- Speeds training, improves quality

3. Dynamic Window:

- Actual window: random(1, window_size)
- Gives more weight to closer words

These implementation details make the difference between mediocre and excellent embeddings

Checkpoint Quiz: Understanding Check

Test your understanding so far:

Questions:

Q1: Why do we need dense embeddings?

Q2: What's the key insight of distributional hypothesis?

Q3: Skip-gram vs CBOW - which to use?

Q4: Why negative sampling?

Checkpoint Quiz: Understanding Check

Test your understanding so far:

Questions:

Q1: Why do we need dense embeddings?

Q2: What's the key insight of distributional hypothesis?

Q3: Skip-gram vs CBOW - which to use?

Q4: Why negative sampling?

Answers:

A1: One-hot is sparse, no similarity info, can't generalize

A2: Words with similar contexts have similar meanings

A3: Skip-gram for quality, CBOW for speed/frequent words

A4: Avoid expensive softmax over entire vocabulary

Checkpoint Quiz: Understanding Check

Test your understanding so far:

Questions:

Q1: Why do we need dense embeddings?

Q2: What's the key insight of distributional hypothesis?

Q3: Skip-gram vs CBOW - which to use?

Q4: Why negative sampling?

Answers:

A1: One-hot is sparse, no similarity info, can't generalize

A2: Words with similar contexts have similar meanings

A3: Skip-gram for quality, CBOW for speed/frequent words

A4: Avoid expensive softmax over entire vocabulary

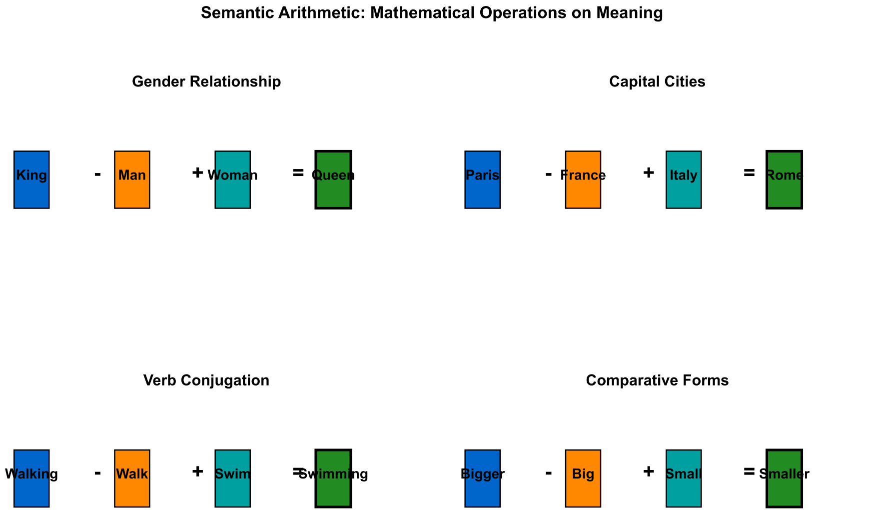
If you understand these, you grasp the core of Word2Vec!

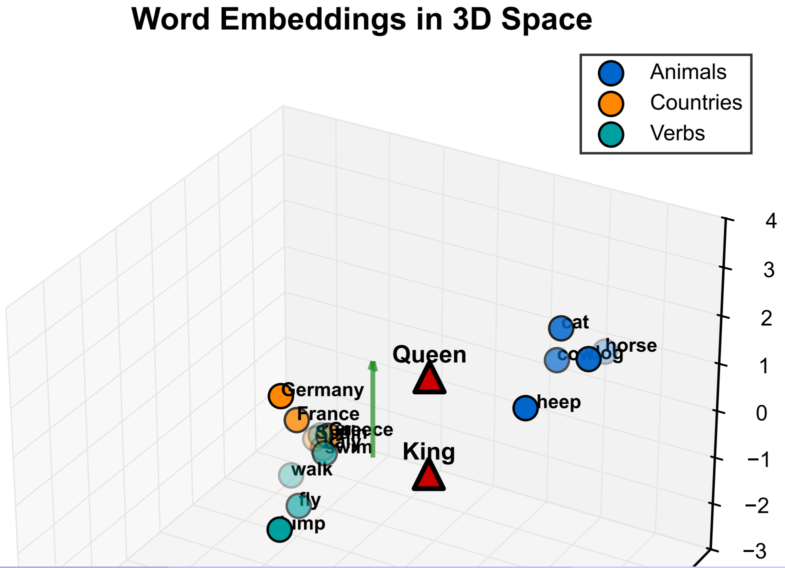
These four concepts form the foundation of all modern word embedding methods

Part 3: Properties & Evaluation

The surprising geometry of meaning

The Magic: Semantic Vector Arithmetic




Word Embeddings in 3D Space
Animals
Countries
Verbs
Queen
King
Germany
France
horse
walk
fly
4
3
2
1
0
−1
−2
−3

Exploring Neighborhoods: Semantic Similarity

Finding nearest neighbors reveals semantic understanding:

Query: "king"

1. queen (0.72)
2. prince (0.70)
3. monarch (0.68)
4. emperor (0.65)
5. royal (0.64)

Query: "beautiful"

1. gorgeous (0.83)
2. lovely (0.81)
3. stunning (0.78)
4. pretty (0.76)
5. wonderful (0.72)

Query: "computer"

1. laptop (0.78)
2. PC (0.75)
3. software (0.71)
4. desktop (0.70)
5. machine (0.68)

Query: "Microsoft"

1. Google (0.75)
2. Apple (0.73)
3. IBM (0.71)
4. Amazon (0.70)
5. Intel (0.68)

Similarity = Cosine Distance in Embedding Space

Evaluation: How Good Are Your Embeddings?

1. Intrinsic Evaluation

Word Similarity:

- WordSim-353
- SimLex-999
- MEN dataset

Analogies:

- Google analogy test
- BATS dataset
- Semantic/syntactic

Metrics:

- Spearman correlation
- Accuracy@1, @5

2. Extrinsic Evaluation

Downstream Tasks:

- Sentiment analysis
- Named entity recognition
- POS tagging

Improvements:

- +3-5% accuracy typical
- Faster convergence
- Better generalization

Transfer Learning:

- Cross-domain
- Cross-lingual

3. Visualization

Techniques:

- t-SNE projection
- PCA analysis
- UMAP

Quality Checks:

- Cluster coherence
- Outlier detection
- Coverage analysis

Tools:

- TensorBoard
- Embedding Projector
- Custom notebooks

Good embeddings: 0.6+ correlation on similarity, 80%+ on analogies, clear visual clusters

Analogy Categories: Testing Semantic Understanding

Google Analogy Test Set: 19,544 questions across categories

Semantic (8,869 questions)

Capital-Country (506):

- Athens : Greece :: Tokyo : Japan
- Paris : France :: Rome : Italy

Currency (866):

- USA : dollar :: Japan : yen
- Europe : euro :: UK : pound

Family (506):

- brother : sister :: father : mother
- uncle : aunt :: nephew : niece

Syntactic (10,675 questions)

Plural (1,332):

- cat : cats :: dog : dogs
- child : children :: person : people

Past Tense (1,560):

- walk : walked :: run : ran
- see : saw :: go : went

Comparative (1,332):

- good : better :: bad : worse
- big : bigger :: small : smaller

Typical Performance: Word2Vec on 1B word corpus

- Semantic analogies: 75-85% accuracy
- Syntactic analogies: 70-80% accuracy

Limitations and Challenges

Fundamental Limitations:

1. Single Vector per Word

- "bank" (river) = "bank" (financial)
- No polysemy handling
- Context-independent

2. Out-of-Vocabulary

- Can't handle new words
- Misspellings break
- Rare words poorly represented

3. Bias in Data

- "doctor" closer to "he"
- "nurse" closer to "she"
- Amplifies stereotypes

Solutions & Evolution:

FastText (2016):

- Character n-grams
- Handles OOV words
- Better for morphology

ELMo (2018):

- Context-dependent
- Different vectors per use
- Bidirectional LSTM

BERT (2018):

- Transformer-based
- Deeply contextual
- Masked language modeling

Word2Vec's limitations directly inspired the transformer revolution

Part 4: Practical Implementation

From theory to practice

Implementation: Training Word2Vec

```python
import gensim.downloader as api
from gensim.models import Word2Vec

# Load corpus (or use your own)
corpus = api.load('text8')  # Wikipedia sample

# Train Word2Vec model
model = Word2Vec(
    sentences=corpus,
    vector_size=300,        # Embedding dimension
    window=5,               # Context window
    min_count=5,            # Ignore rare words
    workers=4,              # Parallel training
    sg=1,                   # 1=skip-gram, 0=CBOW
    negative=15,            # Negative samples
    epochs=5,               # Training iterations
    seed=42                # Reproducibility
)

# Save model
model.save("word2vec.model")

# Get vocabulary size
print(f"Vocabulary: {len(model.wv)} words")
print(f"Embedding shape: {model.wv.vectors.shape}")
```

Gensim makes Word2Vec incredibly accessible - production-quality embeddings in minutes

Using Pre-trained Embeddings

```python
# Load pre-trained model
model = Word2Vec.load("word2vec.model")

# 1. Get word vector
king_vector = model.wv['king']
print(f"Vector for 'king': {king_vector[:5]}...")

# 2. Find similar words
similar = model.wv.most_similar('computer', topn=5)
for word, score in similar:
    print(f"{word}: {score:.3f}")

# 3. Word analogies
result = model.wv.most_similar(
    positive=['king', 'woman'],
    negative=['man'],
    topn=3
)
print(f"king - man + woman = {result[0][0]}")

# 4. Similarity scores
sim = model.wv.similarity('cat', 'dog')
print(f"Similarity(cat, dog) = {sim:.3f}")

# 5. Odd one out
odd = model.wv.doesnt_match(['breakfast', 'lunch', 'dinner', 'car'])
print(f"Odd one out: {odd}")
```

Pre-trained embeddings can be used immediately for various NLP tasks

Advanced Training: Tips from Experience

Data Preparation:

- **Corpus Size:** Minimum 1M words
- **Preprocessing:**
  - Lowercase consistently
  - Keep punctuation (context!)
  - Handle numbers: "123" → "NUM"
- **Sentence Boundaries:** Respect them
- **Domain-Specific:** Add your data

Hyperparameter Tuning:

- **Dimension:** 100 (small), 300 (standard)
- **Window:** 5 (syntax), 10 (semantics)
- **Min Count:** 5-10 typical
- **Negative:** 5-20 (more is better)

Training Strategy:

- **Epochs:** 5-10 for large corpus
- **Learning Rate:** Start 0.025, decay
- **Subsampling:** $t = 10^{-5}$ typical
- **Threads:** Use multiple cores

Quality Checks:

- Monitor loss curve
- Check known analogies
- Visualize samples with t-SNE
- Test on downstream task early
- Compare against baselines

Pro Tip: Start with pre-trained, fine-tune on your domain

Good embeddings require careful attention to both data and hyperparameters

Debugging: Common Issues and Solutions

Problem 1: Poor Analogies

- Symptom: Random results
- Causes:
  - Too small corpus
  - Too few epochs
  - Window too small
- Fix: More data, longer training

Problem 2: Memory Issues

- Symptom: Out of memory
- Causes:
  - Vocabulary too large
  - Dimension too high
- Fix: Increase min_count, reduce dim

Problem 3: Slow Training

- Symptom: Takes days
- Causes:
  - Single thread
  - No negative sampling
  - Full softmax
- Fix: Use workers, negative sampling

Problem 4: Domain Mismatch

- Symptom: Works poorly on your data
- Causes:
  - Pre-trained on different domain
  - Technical jargon missing
- Fix: Fine-tune or train from scratch

Golden Rule: Always validate on your specific use case

Most embedding issues stem from data quality or hyperparameter mismatches

Integration: Embeddings in Neural Networks

```python
import torch
import torch.nn as nn
import numpy as np

class SentimentClassifier(nn.Module):
    def __init__(self, word2vec_model, num_classes=2):
        super().__init__()

        # Load pre-trained embeddings
        weights = word2vec_model.wv.vectors
        vocab_size, embed_dim = weights.shape

        # Embedding layer (frozen initially)
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.embedding.weight = nn.Parameter(torch.FloatTensor(weights))
        self.embedding.weight.requires_grad = False  # Freeze

        # Classification layers
        self.lstm = nn.LSTM(embed_dim, 128, batch_first=True)
        self.dropout = nn.Dropout(0.3)
        self.fc = nn.Linear(128, num_classes)

    def forward(self, x):
        embeds = self.embedding(x)  # Use pre-trained
        lstm_out, _ = self.lstm(embeds)
        pooled = lstm_out[:, -1, :]  # Last hidden state
        return self.fc(self.dropout(pooled))
```

Pre-trained embeddings provide excellent initialization for downstream tasks

Real-World Applications Gallery

../figures/applications_gallery.pdf

Historical Impact: The Word2Vec Revolution

Timeline of Impact:

2013: Word2Vec Published

- Mikolov et al. papers
- Open-source release
- Immediate adoption

2014-2015: Rapid Adoption

- GloVe competition (Stanford)
- Industry integration
- 10,000+ citations

2016-2017: Refinements

- FastText (Facebook)
- Multilingual embeddings
- Domain-specific models

2018: Contextual Era

- ELMo introduces context
- BERT revolution begins
- GPT demonstrates scale

2019-Present: Foundation

- Still used in production
- Initialization for transformers
- 50,000+ citations
- Taught in every NLP course

Final Quiz: Mastery Check

Can you answer these advanced questions?

1. Why does Word2Vec use two embedding matrices (input and output)?

Final Quiz: Mastery Check

Can you answer these advanced questions?

1. Why does Word2Vec use two embedding matrices (input and output)?
   $\rightarrow$ Asymmetric relationship: predicting context from center

2. How does negative sampling approximate the softmax?

Final Quiz: Mastery Check

Can you answer these advanced questions?

1. Why does Word2Vec use two embedding matrices (input and output)?
   $\rightarrow$ Asymmetric relationship: predicting context from center
2. How does negative sampling approximate the softmax?
   $\rightarrow$ Binary classification: real pairs vs random pairs
3. Why does "king - man + woman = queen" work mathematically?

Final Quiz: Mastery Check

Can you answer these advanced questions?

1. Why does Word2Vec use two embedding matrices (input and output)?
   $\rightarrow$ Asymmetric relationship: predicting context from center
2. How does negative sampling approximate the softmax?
   $\rightarrow$ Binary classification: real pairs vs random pairs
3. Why does "king - man + woman = queen" work mathematically?
   $\rightarrow$ Relationships encoded as consistent vector offsets
4. What's the computational complexity: full softmax vs negative sampling?

Final Quiz: Mastery Check

Can you answer these advanced questions?

1. Why does Word2Vec use two embedding matrices (input and output)?

   $\rightarrow$ Asymmetric relationship: predicting context from center

2. How does negative sampling approximate the softmax?

   $\rightarrow$ Binary classification: real pairs vs random pairs

3. Why does “king - man + woman = queen” work mathematically?

   $\rightarrow$ Relationships encoded as consistent vector offsets

4. What’s the computational complexity: full softmax vs negative sampling?

   $\rightarrow$ O(—V—) vs O(k), where k ¡¡ —V—

5. Why can’t Word2Vec handle “apple” (fruit) vs “Apple” (company)?

Final Quiz: Mastery Check

Can you answer these advanced questions?

1. Why does Word2Vec use two embedding matrices (input and output)?
   → Asymmetric relationship: predicting context from center
2. How does negative sampling approximate the softmax?
   → Binary classification: real pairs vs random pairs
3. Why does "king - man + woman = queen" work mathematically?
   → Relationships encoded as consistent vector offsets
4. What's the computational complexity: full softmax vs negative sampling?
   → O(—V—) vs O(k), where k ¡¡ —V—
5. Why can't Word2Vec handle "apple" (fruit) vs "Apple" (company)?
   → Single vector per word type, no context dependence

If you can answer these, you truly understand Word2Vec's design and limitations

Summary: What We've Learned

Core Concepts

- One-hot $\rightarrow$ Dense embeddings
- Distributional hypothesis
- Skip-gram architecture
- Negative sampling trick
- Semantic arithmetic
- Evaluation methods

Technical Skills

- Training Word2Vec
- Hyperparameter tuning
- Using pre-trained models
- Integration with neural nets
- Debugging embeddings

Key Insights

- Context defines meaning
- Geometry encodes semantics
- Relationships are vectors
- Efficiency enables scale
- Limitations inspire progress

Practical Impact

- Powers search engines
- Enables recommendations
- Foundation for transformers
- $100B+ market impact
- 50,000+ citations

Word2Vec: Simple idea, profound impact, lasting legacy

Lab Session Preview

Hands-On Exercises: week02_word_embeddings_lab.ipynb

Part 1: Training (45 min)

- Load and preprocess corpus
- Train Word2Vec from scratch
- Experiment with hyperparameters
- Compare CBOW vs Skip-gram
- Visualize training progress

Part 2: Exploration (30 min)

- Find nearest neighbors
- Test word analogies
- Visualize with t-SNE
- Explore semantic clusters
- Identify interesting patterns

Part 3: Application (45 min)

- Build similarity search engine
- Create analogy solver
- Simple sentiment classifier
- Document similarity system
- Performance evaluation

Bonus Challenges:

- Train on your own corpus
- Implement negative sampling
- Cross-lingual embeddings
- Bias analysis and debiasing
- Compare with GloVe/FastText

Goal: Hands-on mastery of word embeddings

Resources for Deep Dive

Essential Papers

- Mikolov et al. (2013a): Efficient Estimation
- Mikolov et al. (2013b): Distributed Representations
- Goldberg & Levy (2014): word2vec Explained
- Pennington et al. (2014): GloVe
- Bojanowski et al. (2016): FastText

Implementations

- Gensim (Python)
- TensorFlow/Keras
- PyTorch
- FastText library
- spaCy integration

Tutorials

- TensorFlow tutorials
- PyTorch examples
- Gensim documentation

Pre-trained Models

- Google News (3M words)
- GloVe (6B tokens)
- FastText (157 languages)
- Domain-specific models

Visualization

- TensorBoard Projector
- Embedding Explorer
- Custom t-SNE tools

Continue Learning: `https://github.com/your-course/week02-resources`

These resources will deepen your understanding and practical skills

Looking Ahead

What's Next in Our Journey:

Next Week: RNNs & LSTMs

- Sequential processing
- Hidden state dynamics
- Vanishing gradient problem
- LSTM architecture
- Text generation

Building On Word2Vec:

- Embeddings as RNN input
- Sequence modeling
- Context evolution
- Memory mechanisms

Future Topics:

- Week 4: Seq2Seq & Attention
- Week 5: Transformers
- Week 6: BERT & GPT
- Week 7: Advanced architectures
- Week 8-12: Modern NLP

The Big Picture:

- Word2Vec $\rightarrow$ RNN $\rightarrow$ Transformer
- Static $\rightarrow$ Contextual embeddings
- Understanding $\rightarrow$ Generation

You now understand the foundation of all modern NLP!