Word Embeddings: Word2Vec, GloVe, FastText

Lecture 11: The Neural Revolution in NLP

PSYC 51.07: Models of Language and Communication - Week 3

Winter 2026

Today's Lecture 📋

1. 🚀 **The 2013 Revolution: Word2Vec**
2. 🏗️ **Word2Vec Architectures: CBOW & Skip-gram**
3. 🌍 **GloVe: Global Vectors**
4. 🔤 **FastText: Subword Information**
5. 📊 **Comparison & Evaluation**
6. 💡 **Applications & Best Practices**

Goal: Understand how neural methods learn dense word representations

From Count-Based to Prediction-Based 🎯

Count-Based (LSA, LDA):

- Build co-occurrence matrix
- Apply matrix factorization
- Global statistics
- Linear relationships
- Interpretable factors

Advantages:

- Fast training
- Leverages global statistics
- Mathematically grounded

Word2Vec: The Revolution 🚀

2013: Everything changed

Key Innovation:

- Shallow neural network
- Predict context from word (CBOW)
- Predict word from context (Skip-gram)
- Dense, low-dimensional vectors
- Captures semantic relationships
- Fast training with negative sampling

Impact:

- Sparked deep learning in NLP

Word2Vec: Core Intuition 💡

Words that appear in similar contexts should have similar representations

Example Context Window
"The quick brown **[TARGET]** jumped over the lazy dog"

Context: [The, quick, brown, jumped, over, the, lazy, dog]

Target: fox

Key Idea:

- Train a model to predict target from context (or vice versa)
- The learned **weights** become our word vectors!
- Similar words will have similar weight patterns

We don't actually care about the prediction task—we want the embeddings!

Context Windows: Worked Example 📝

Sentence: "The cat sat on the mat"

Window size = 2 (2 words on each side)

```
1Position 0: "The"  → Context: [cat, sat]
2Position 1: "cat"  → Context: [The, sat, on]
3Position 2: "sat"  → Context: [The, cat, on, the]
4Position 3: "on"   → Context: [cat, sat, the, mat]
5Position 4: "the"  → Context: [sat, on, mat]
6Position 5: "mat"  → Context: [on, the]
```

Skip-gram training pairs (target → context):

```
1(The, cat), (The, sat)
2(cat, The), (cat, sat), (cat, on)
3(sat, The), (sat, cat), (sat, on), (sat, the)
```

CBOW: Continuous Bag of Words 👜

Predict the center word from context words

```
1Context words:
2  "the" ──┐
3  "cat" ──┼──→ Average ──→ "sat"
4  "on"  ──┼──→ Hidden  ──→ (predict)
5  "the" ──┘
```

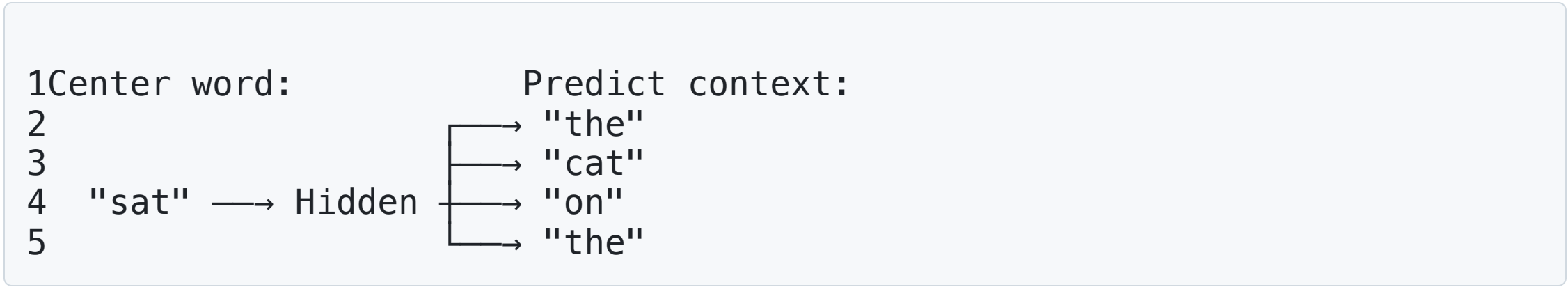
Architecture:

1. Input: One-hot vectors of context words
2. Hidden: Average of input embeddings
3. Output: Softmax over vocabulary

Training:

Skip-gram: The Inverse Approach 🔄

Predict context words from the center word



Architecture:

1. Input: One-hot vector of center word
2. Hidden: Word embedding
3. Output: Softmax predicting each context word

Training:

Negative Sampling: The Speed Trick ⚡

Problem: Softmax over entire vocabulary is too expensive!

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

For 100k vocabulary: Need to compute 100k exponentials per training example!

Solution: Negative Sampling

Instead of predicting across all words, create a binary classification:

- **Positive sample:** Actual context word (label = 1)
- **Negative samples:** K random words (label = 0)

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

Negative Sampling: Concrete Example 🎯

Training pair: ("cat", "sat") - cat is center, sat is context

```
1# Positive sample: Does "sat" appear near "cat"? YES (label=1)
2positive_pair = ("cat", "sat", label=1)
3
4# Negative samples: Random words that DON'T appear near "cat"
5# Sample 5 random words from vocabulary
6negative_pairs = [
7    ("cat", "algorithm", label=0),
8    ("cat", "president", label=0),
9    ("cat", "quantum", label=0),
10    ("cat", "democracy", label=0),
11    ("cat", "software", label=0),
12]
13
14# Train binary classifier: Is this a real context pair?
15# Instead of 100k-way softmax → 6 binary predictions!
```

Word2Vec in Practice 💻

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

# Load pre-trained model (300 dimensions, 3B words)
model = api.load('word2vec-google-news-300')

# Find similar words
similar = model.most_similar('computer', topn=5)
print(similar)
# Output: [('computers', 0.72), ('laptop', 0.69), ('PC', 0.68), ...]

# Word analogies: king - man + woman = ?
result = model.most_similar(
    positive=['king', 'woman'],
    negative=['man'],
    topn=1
)
print(result)
```

Word2Vec in Practice 💻

```
21# Train your own model
22sentences = [
23    ['the', 'cat', 'sat', 'on', 'the', 'mat'],
24    ['the', 'dog', 'ran', 'in', 'the', 'park'],
25    # ... more sentences
26]
27
28custom_model = Word2Vec(
29    sentences,
30    vector_size=100,    # embedding dimension
31    window=5,           # context window
32    min_count=1,        # ignore rare words
33    sg=1,               # 1=skip-gram, 0=CBOW
34    negative=5,         # negative sampling
35    workers=4           # parallel threads
36)
```


continued

GloVe: Global Vectors for Word Representation 🌍

Combining the best of count-based and prediction-based methods

Motivation:

- Word2Vec uses **local** context windows
- LSA uses **global** co-occurrence statistics
- Can we get the best of both?

Key Insight:

Ratios of co-occurrence probabilities encode meaning better than raw probabilities!

Co-occurrence Example:

Probe word	P(word|ice)	P(word|steam)	Ratio
solid	high	low	>> 1

GloVe: The Ratio Intuition 🔢

Why ratios matter more than raw probabilities:

```
1Given words: "ice" and "steam"
2Probe with: "solid", "gas", "water"
3
4P(solid | ice)  = 0.00019    P(solid | steam) = 0.000022
5P(gas | ice)    = 0.000066   P(gas | steam)   = 0.00078
6
7Ratio: P(solid|ice) / P(solid|steam) = 8.9   → "solid" relates to ice
8Ratio: P(gas|ice)   / P(gas|steam)   = 0.085 → "gas" relates to steam
9Ratio: P(water|ice) / P(water|steam) = 1.36  → "water" is neutral
```

The insight: These ratios distinguish relevant context words from irrelevant ones!

GloVe learns vectors where: $\frac{\vec{w}_{\text{ice}}^T \vec{w}_{\text{solid}}}{\vec{w}_{\text{steam}}^T \vec{w}_{\text{solid}}} \approx 8.9$

GloVe: The Objective Function 🎯

Goal: Learn vectors that capture co-occurrence statistics

Objective Function:

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

where:

- X_{ij} = number of times word j appears in context of word i
- $\vec{w}_i$ = word vector for word i
- $\tilde{\vec{w}}_j$ = separate context vector for word j
- $b_i, \tilde{b}_j$ = bias terms
- $f(X_{ij})$ = weighting function

GloVe vs. Word2Vec 🥊

Word2Vec (Skip-gram):

- Local context windows
- Online training
- Predicts context from word
- Stochastic updates
- Negative sampling trick
- Each occurrence matters

Advantages:

- Works well on small corpora
- Can train incrementally

GloVe in Practice 💻

```
1import gensim.downloader as api
2import numpy as np
3
4# Load pre-trained GloVe embeddings
5# Options: glove-wiki-gigaword-50, -100, -200, -300
6# Or: glove-twitter-25, -50, -100, -200
7glove = api.load("glove-wiki-gigaword-100")
8
9# Use just like Word2Vec
10similar = glove.most_similar('computer', topn=5)
11print(similar)
12
13# Analogies
14result = glove.most_similar(
15    positive=['france', 'berlin'],
16    negative=['paris'],
17    topn=1
18)
```

GloVe in Practice 💻

```
21# Vector arithmetic
22king = glove['king']
23man = glove['man']
24woman = glove['woman']
25result_vec = king - man + woman
26
27# Find closest word
28closest = glove.similar_by_vector(result_vec, topn=1)
29print(closest)  # Should be close to 'queen'
30
31# Compute similarity
32similarity = glove.similarity('cat', 'dog')
33print(f"Similarity: {similarity:.3f}")
```

...continued

FastText: Enriching with Subword Information

The Problem with Word2Vec and GloVe:

Out-of-Vocabulary (OOV) Problem

- Word2Vec/GloVe: One vector per word
- New word?
- Misspelling?
- Rare morphological form?

Example:

- Have: "running", "runner"
- Need: "runnable" →
- But these words share morphology! ("run" + suffix)

FastText: Character N-grams 🧩

Key Idea: Break words into character n-grams

Example: "where" (with n=3)

Character trigrams: <wh, whe, her, ere, re>

Plus: <where> (the whole word)

Word vector:

$$\vec{w}_{\text{where}} = \frac{1}{6}\left(\vec{z}_{<\text{wh}} + \vec{z}_{\text{whe}} + \vec{z}_{\text{her}} + \vec{z}_{\text{ere}} + \vec{z}_{\text{re}>} + \vec{z}_{<\text{where}>}\right)$$

Advantages:

- Handles OOV words!
- Captures morphology
- Better for rare words

FastText: Morphology in Action 🔡

How FastText handles related words:

```
1# Word: "unhappiness" broken into character 3-grams:
2# <un, unh, nha, hap, app, ppi, pin, ine, nes, ess, ss>
3
4# Shares n-grams with:
5# "unhappy"   → <un, unh, nha, hap, app, ppy
6# "happiness" → hap, app, ppi, pin, ine, nes, ess, ss>
7# "happy"     → hap, app, ppy
8
9# Therefore: vec("unhappiness") is close to:
10#   - vec("unhappy")     (prefix overlap)
11#   - vec("happiness")   (suffix overlap)
12#   - vec("sadness")     (similar suffix pattern)
```

This is why FastText excels at morphologically rich languages!

FastText in Practice 💻

```
from gensim.models import FastText
import gensim.downloader as api

# Load pre-trained FastText model
# Available in 157 languages!
fasttext_model = api.load('fasttext-wiki-news-subwords-300')

# Works for known words
vec_cat = fasttext_model['cat']

# Also works for unknown words! (OOV)
vec_unknownword = fasttext_model['unknownword']  # Still get a vector!

# Even works for misspellings (somewhat)
vec_misspelling = fasttext_model['computr']  # Close to "computer"

# Train your own FastText model
sentences = [['the', 'cat', 'sat'], ['the', 'dog', 'ran']]
```

FastText in Practice 💻

```
    sentences,
    vector_size=100,
    window=5,
    min_count=1,
    min_n=3,      # minimum n-gram length
    max_n=6,      # maximum n-gram length
    word_ngrams=1 # use word + ngrams
)

# Check similar words
similar = ft_model.wv.most_similar('cat', topn=5)

# Morphological awareness
# 'running', 'runner', 'runnable' will be close
```

...continued

Embedding Methods Comparison 📊

LDA	2003	Probabilistic	✗	Slow	Topic modeling
Word2Vec	2013	Neural (local)	✗	Fast	General NLP
GloVe	2014	Hybrid (global)	✗	Fast	Large corpora
FastText	2017	Neural + ngrams	✓	Fast	Morphology-rich

When to Use What?

- **Word2Vec (Skip-gram):** General purpose, good semantic quality, most popular
- **GloVe:** Large corpus, want deterministic training, good interpretability
- **FastText:** Morphologically rich languages, need OOV handling, many rare words
- **LSA/LDA:** Topic modeling, document similarity, interpretable dimensions

Typical dimensions: 50-300 (100-300 most common)

Evaluating Word Embeddings 📏

Intrinsic Evaluation:

1. Word Similarity

- WordSim-353 dataset
- SimLex-999 dataset
- Spearman correlation with human judgments

2. Word Analogies

- Google Analogy Dataset (19k questions)
- Semantic: *Athens:Greece::Baghdad:Iraq*
- Syntactic: *good:better::bad:worse*
- Accuracy: % correct

Limitations of Static Embeddings ⚠️

The fundamental problem: One vector per word type

Example: Polysemy

1. "I deposited money at the **bank**" (financial institution)
2. "We sat by the river **bank**" (riverside)
3. "The plane will **bank** left" (tilt)

All get the ! This conflates different meanings.

Other Limitations:

- No compositional semantics
- "hot dog" ≠ "hot" + "dog"
- Bias in embeddings
- Limited to training corpus

Bias in Word Embeddings ⚖️

Embeddings learn the biases in their training data

Gender Bias Examples

- $\vec{\text{man}} : \text{computer } \vec{\text{programmer}} :: \text{wo}\vec{\text{man}} : \text{home}\vec{\text{maker}}$
- $\vec{\text{man}} : \text{doc}\vec{\text{tor}} :: \text{wo}\vec{\text{man}} : \text{nu}\vec{\text{rse}}$
- "man" more associated with "career", "woman" with "family"

Other Biases:

- Racial/ethnic stereotypes
- Age bias
- Religious bias
- Socioeconomic bias

Bias Detection: Code Example 🔍

```
import gensim.downloader as api
model = api.load('word2vec-google-news-300')

# Measure gender bias: which words are closer to "man" vs "woman"?
def gender_bias_score(word):
    """Positive = closer to man, Negative = closer to woman"""
    return model.similarity(word, 'man') - model.similarity(word, 'woman')

occupations = ['doctor', 'nurse', 'engineer', 'teacher',
               'programmer', 'secretary', 'scientist', 'receptionist']

for job in occupations:
    score = gender_bias_score(job)
    direction = "→ man" if score > 0 else "→ woman"
    print(f"{job:12}: {score:+.3f} {direction}")

# Output:
# doctor      : +0.089 → man
```

Bias Detection: Code Example 🔍

```
21# teacher     : −0.046 → woman
22# programmer  : +0.091 → man
23# secretary   : −0.107 → woman
```

...continued

These biases reflect stereotypes in the training data (news articles)!

Debiasing Approaches 🔧

How can we reduce bias in embeddings?

1. Post-processing:

- Identify bias subspace
- Neutralize: Remove bias from neutral words
- Equalize: Ensure equal distances

2. Training-time:

- Counterfactual data augmentation
- Adversarial debiasing
- Constrained optimization

3. Data curation:

Practical Tips for Using Embeddings 💡

1. **Start with pre-trained models**
    - Word2Vec: Google News (300d, 3B words)
    - GloVe: Common Crawl (300d, 840B tokens)
    - FastText: Available in 157 languages
2. **Fine-tune if you have domain data**
    - Medical: PubMed, clinical notes
    - Legal: case law, contracts
    - Social media: tweets, posts
    - Can significantly improve performance
3. **Choose dimensions wisely**
    - More dims = more expressiveness, more data needed

Loading Pre-trained Embeddings: Quick Reference 💻

```
import gensim.downloader as api

# Word2Vec (Google News, 3B words, 300d)
w2v = api.load('word2vec-google-news-300')

# GloVe options
glove_50d = api.load('glove-wiki-gigaword-50')      # Small, fast
glove_100d = api.load('glove-wiki-gigaword-100')    # Balanced
glove_300d = api.load('glove-wiki-gigaword-300')    # Best quality
glove_twitter = api.load('glove-twitter-100')       # Social media

# FastText (handles OOV!)
fasttext = api.load('fasttext-wiki-news-subwords-300')

# Basic usage (same for all)
similar = model.most_similar('computer', topn=5)
vector = model['cat']
similarity = model.similarity('dog', 'cat')
```

Real-World Applications 🌟

Search & Information Retrieval:

- Semantic search
- Query expansion
- Document ranking
- Question answering

Text Classification:

- Sentiment analysis
- Spam detection
- Topic classification
- Intent detection

Discussion Question 💬

Why do word analogies work so well?

The famous example:

$$\vec{\text{king}} - \vec{\text{man}} + \vec{\text{woman}} \approx \vec{\text{queen}}$$

Think about:

- What does vector subtraction represent linguistically?
- Why does this capture semantic relationships?
- What are the limitations of this approach?
- Does this mean embeddings "understand" gender?

Deeper Question

Are these embeddings truly capturing *meaning*, or just statistical patterns?

Summary 🎯

What we learned today:

1. **Word2Vec (2013):** Neural revolution in embeddings
    - CBOW: Predict center from context
- Skip-gram: Predict context from center
- Negative sampling for efficiency

2. **GloVe (2014):** Combining count + prediction
    - Global co-occurrence statistics
- Factorization objective
- Ratio of probabilities

3. **FastText (2017):** Subword information
    - Character n-grams

Key References 📚

Foundational Papers:

- Mikolov et al. (2013a). "Efficient Estimation of Word Representations in Vector Space"
- Mikolov et al. (2013b). "Distributed Representations of Words and Phrases and their Compositionality"
- Pennington et al. (2014). "GloVe: Global Vectors for Word Representation"
- Bojanowski et al. (2017). "Enriching Word Vectors with Subword Information"

Bias & Ethics:

- Bolukbasi et al. (2016). "Man is to Computer Programmer as Woman is to Homemaker?"
- Caliskan et al. (2017). "Semantics derived automatically from language corpora contain human-like biases"

Questions? 🙋

Next Lecture:

Contextual Embeddings: ELMo, Universal Sentence Encoder, BERT

One word, multiple meanings!