Natural Language Processing Course

Week 2: Neural Language Models and Word Embeddings

Joerg R. Osterrieder
www.joergosterrieder.com

Week 2

Neural Language Models

Teaching Computers the Meaning of Words

The Problem: Computers Don't Know Word Meanings

Last week's limitation:
N-grams treat "cat" and "dog" as completely unrelated as "cat" and "democracy"

But humans know:

- cat and kitten are similar
- Paris and France are related
- running and ran are the same verb

The breakthrough question (2003):[1]

What if we could teach computers that similar words have similar meanings?

Impact: This idea revolutionized NLP and powers every modern AI system

[1] Bengio et al. (2003). "A neural probabilistic language model", JMLR

Where You Use Word Embeddings Every Day

Search Engines (2024):

- Search "car" → also finds "automobile"
- Google uses 3072-dim embeddings[2]
- Semantic search, not just keywords

Translation:

- Knows "love" in English = "amour" in French
- Handles words never seen before
- Meta's system: 20M+ misspellings[3]

Recommendations:

- Netflix: similar movies
- Spotify: related songs
- Amazon: 1536-dim embeddings[4]

Your Phone:

- Autocorrect knows "teh" → "the"
- Voice assistants understand context
- Smart reply suggestions

[1] Google Gemini Embeddings (2024)
[2] Meta MOE: Misspelling Oblivious Embeddings (2024)
[3] Amazon Titan Embeddings (2024)

Week 2: What You'll Master

By the end of this week, you will:

- **Understand** why "king - man + woman = queen" actually works
- **Build** your own Word2Vec from scratch
- **Create** visualizations showing word relationships
- **Implement** a system that finds similar words
- **Know** why 100-300 dimensions is the sweet spot[5]

Core Insight: Words are defined by the company they keep

[5] Empirical studies show diminishing returns beyond 300 dimensions

The Magic of Word Arithmetic

The famous example that shocked the NLP world (2013):[6]

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

The computer answers: queen!

How is this possible?

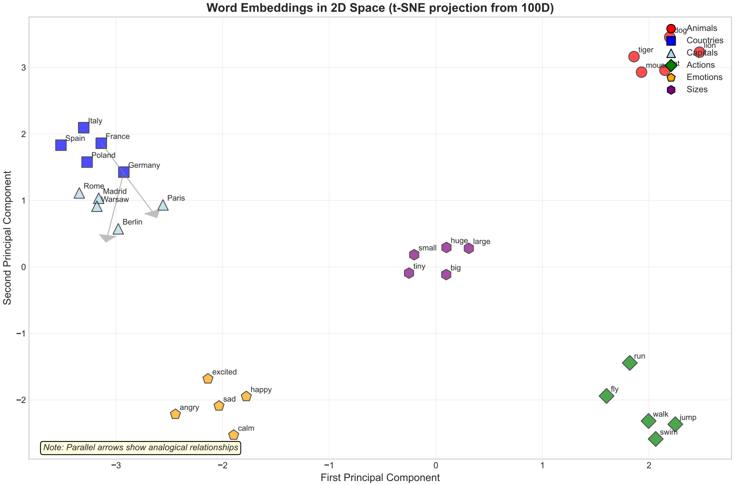
- Words represented as vectors in space
- Similar meanings = nearby vectors
- Relationships = consistent directions

More examples that work:

- Paris - France + Italy = Rome
- bigger - big + small = smaller
- walking - walk + swim = swimming

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

Intuition: Words as Points in Space



Key observations:

- Animals cluster together
- Countries form another cluster

The Distributional Hypothesis

Core Principle (Harris, 1954; Firth, 1957):

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

Example contexts for "cat":

- The cat sat on the mat
- I fed my cat this morning
- The cat chased the mouse

Similar contexts for "dog":

- The dog sat on the mat
- I fed my dog this morning
- The dog chased the ball

Insight: Similar contexts → similar meanings → similar vectors

From IDs to Vectors: The Key Innovation

N-gram approach (discrete):

- cat = ID 1247
- dog = ID 3891
- No notion of similarity!

Neural approach (continuous):[7]

- cat = [0.2, -0.4, 0.7, ..., 0.1] (100 numbers)
- dog = [0.3, -0.3, 0.6, ..., 0.2] (100 numbers)
- Similarity = cosine distance = 0.95

Why 100-300 dimensions?[8]

- Too few (¡ 50): Can't capture nuances
- Just right (100-300): Best performance/efficiency
- Too many (¿ 500): Diminishing returns, overfitting

[1] Bengio et al. (2003) introduced distributed representations
[2] Empirical studies across multiple tasks and corpora

Word2Vec: The Algorithm That Changed NLP

Skip-gram Model (Mikolov et al., 2013):[9]
"Predict context words from center word"

Training example:
Sentence: "The quick brown fox jumps"

Center	→	**Context**
brown	→	the
brown	→	quick
brown	→	fox
brown	→	jumps

The genius insight:

- Words that appear in similar contexts get similar vectors
- No linguistic knowledge needed!
- Just predict surrounding words

[9] Mikolov et al. (2013). "Efficient estimation of word representations in vector space", ICLR

Building Word2Vec: The Core Algorithm

```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):
        """Initialize with typical 100-dim embeddings"""
        super().__init__()
        self.in_embed = nn.Embedding(vocab_size, embed_dim)
        self.out_embed = nn.Embedding(vocab_size, embed_dim)

    def forward(self, center, context, neg_samples):
        """Skip-gram with negative sampling"""
        center_embeds = self.in_embed(center)
        context_embeds = self.out_embed(context)
        neg_embeds = self.out_embed(neg_samples)

        pos_score = torch.sum(center_embeds * context_embeds, dim=1)
        pos_score = F.logsigmoid(pos_score)

        neg_score = torch.bmm(neg_embeds, center_embeds.unsqueeze(2))
        neg_score = F.logsigmoid(-neg_score).sum(1)

        return -(pos_score + neg_score).mean()
```

Key Design Choices:

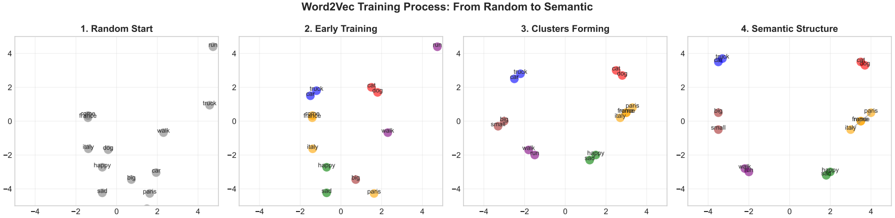
- Two embedding matrices: input and output
- Negative sampling for efficiency[1]
- Log-sigmoid for numerical stability
- 100 dimensions: common heuristic

Training Speed:

- 12–40 hours on 100GB text[2]
- 18.7× faster with GPU
- Processes millions of words/sec

[a] Typically 5–20 negative samples
[b] Based on Wikipedia corpus

How Word2Vec Learns: Visual Intuition

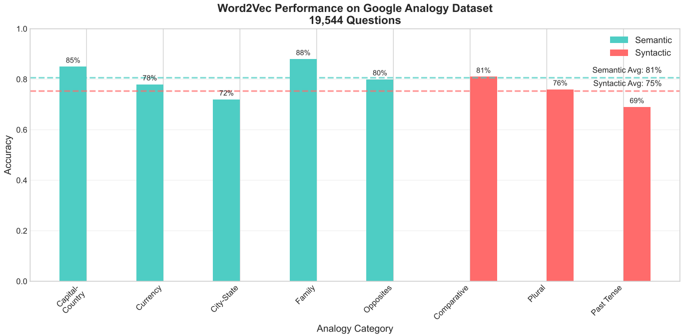


Training progression:

1. **Random start:** Words scattered randomly
2. **Early training:** Frequent words move first
3. **Middle stage:** Clusters begin forming
4. **Convergence:** Semantic structure emerges

Key insight: The algorithm discovers semantic relationships purely from co-occurrence!

Evaluating Word Embeddings: Analogy Task



Key Insights

- Google analogy dataset: 19,544 questions[10]
- Semantic: capital-country, gender, family
- Syntactic: plurals, tense, comparatives
- Skip-gram achieves 55-75% accuracy

Mikolov et al. (2013). Dataset publicly available

Real Impact: Before and After Word Embeddings

Before (2012):

- One-hot encoding: 50K dimensions
- No similarity between words
- Huge, sparse matrices
- Poor generalization

Concrete improvements:

- Sentiment analysis: 5-10% accuracy gain
- Named entity recognition: 3-5% F1 improvement
- Machine translation: Enables zero-shot translation
- Information retrieval: Semantic search possible

After (2013+):

- Dense vectors: 100-300 dims
- Semantic similarity captured
- 100x smaller models
- Handles unseen words better

Word embeddings are the foundation of all modern NLP systems

Limitations: One Vector Per Word?

The polysemy problem:
"I went to the **bank** to deposit money"
"I sat by the river **bank** and fished"

Word2Vec gives "bank" one vector - averaging both meanings!

Other limitations:

- No handling of word order: "dog bites man" = "man bites dog"
- Fixed vocabulary: Can't handle new words
- Context-independent: Same vector regardless of sentence
- Struggles with rare words (need 5-10 occurrences minimum)

Next week preview:

RNNs will process words in sequence, maintaining context and order

Week 2 Exercise: Build a Semantic Search Engine

Your Mission: Create a system that finds similar words/documents

Dataset: Wikipedia articles (first 10,000)

Tasks:

1. Train Word2Vec on Wikipedia text
2. Implement similarity search:
   - Input: "computer"
   - Output: ["laptop", "PC", "processor", "software"...]
3. Build document search:
   - Average word vectors $\rightarrow$ document vector
   - Find similar articles
4. Evaluate on analogy task

Bonus Challenges:

- Visualize embeddings with t-SNE
- Compare 50, 100, 300 dimensions
- Implement both CBOW and Skip-gram
- Try negative sampling vs hierarchical softmax

You'll discover: How Google understands "car" = "automobile"!

Key Takeaways: Words Have Meaning!

What we learned:

- Words can be represented as dense vectors (typically 100-300 dims)
- Similar words have similar vectors (measured by cosine similarity)
- Relationships are consistent directions in vector space
- Simple algorithm (predict context) learns complex semantics

Revolutionary impact:

Word embeddings reduced NLP model sizes by 100x while improving accuracy

But remember the limitation:
One word = one vector (no context sensitivity)

Next week: RNNs
Learn how to process sequences and maintain context!

References and Further Reading

Foundational Papers:

- Bengio et al. (2003). "A neural probabilistic language model", JMLR
- Mikolov et al. (2013). "Efficient estimation of word representations", ICLR
- Mikolov et al. (2013). "Distributed representations of words", NIPS
- Pennington et al. (2014). "GloVe: Global vectors for word representation", EMNLP

Recommended Reading:

- Original word analogy dataset and evaluation code
- Jay Alammar's illustrated Word2Vec (visual guide)
- CS224N Stanford lecture notes on word embeddings

Practical Resources:

- Gensim library for training Word2Vec
- Pre-trained embeddings: Google News (3M words, 300d)
- FastText for handling out-of-vocabulary words