

Homework 4: Word Embeddings with Word2Vec

Points: 20 | Due: See WebCampus for deadline

Author: Richard Young, Ph.D. | UNLV Lee Business School

Compute: CPU (free tier)

Learning Objectives

1. **Understand** how words become numbers (and why it matters)
 2. **Train** your own Word2Vec model on domain-specific data
 3. **Document** embedding successes AND failures
 4. **Create** business-relevant word analogies
 5. **Compare** your embeddings to pre-trained models
-

Why This Matters for Business

Recommendation Systems: Netflix and Spotify use embeddings to understand that users who like “thriller” probably like “suspense.” These semantic relationships power personalized recommendations worth billions.

Customer Support: Zendesk uses embeddings to route tickets—understanding that “can’t login” and “password not working” mean the same thing, even without shared words.

Competitive Intelligence: Investment firms train embeddings on SEC filings to find companies with similar business models, even when they use different terminology.

Search & Discovery: Google’s search understands that “cheap flights to Vegas” and “affordable airfare Las Vegas” should return similar results—that’s embeddings at work.

Grading

Component	Points	Effort	What We're Looking For
Data Prep	3	*	Text preprocessed for Word2Vec
Model Training	4	*	Word2Vec model trained successfully
Similarities	4	**	Word similarities explored and analyzed
Failures	4	**	At least 1 embedding failure documented
Analogies	3	***	3 business-relevant analogies tested
Comparison	2	*	Compared to pre-trained embeddings
Total	20		

Effort Key: * Straightforward | ** Requires thinking | *** Challenge

The Big Picture

Before embeddings, computers saw words as arbitrary symbols. **Word2Vec changed everything** by learning that similar words should have similar numbers.

The magic: king - man + woman \approx queen

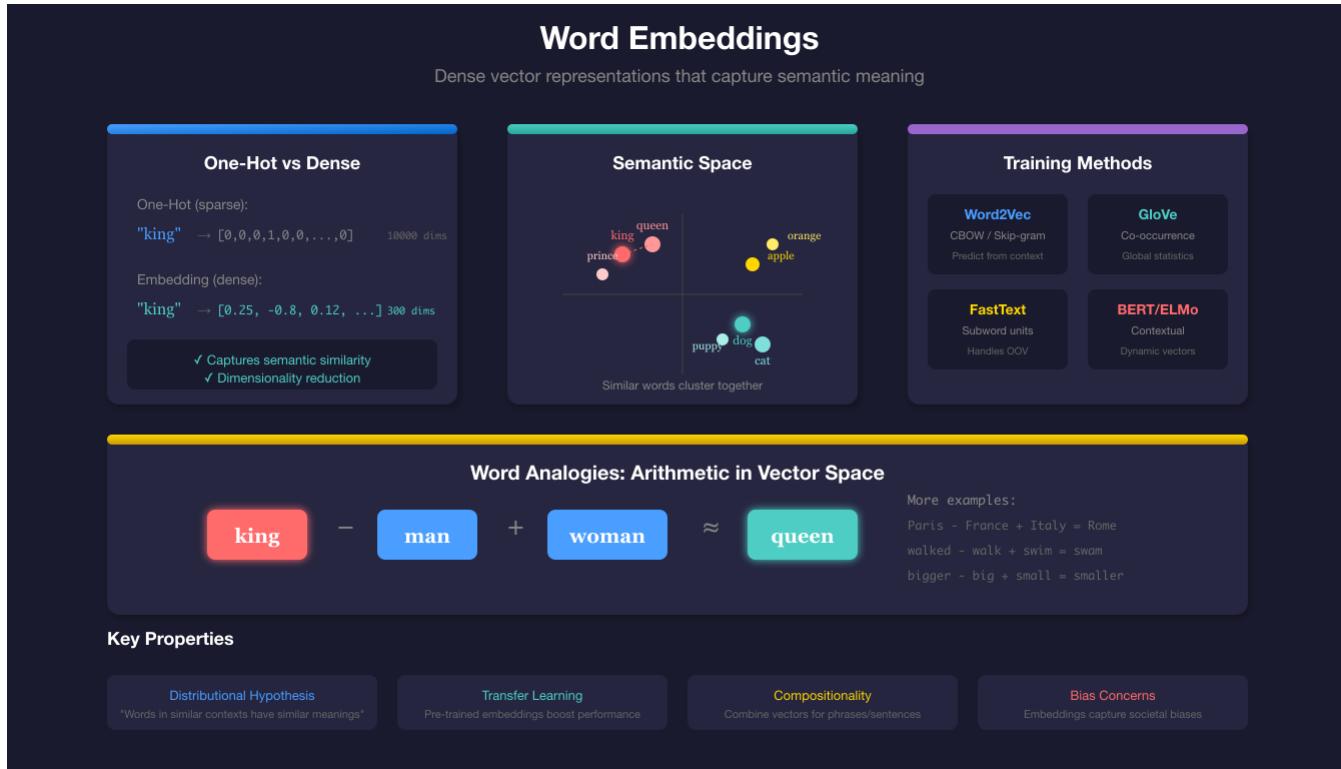


Figure 1: Word Embeddings

Instructions

1. Open MIS769_HW4_Word_EMBEDDINGS.ipynb in Google Colab
2. Load your dataset (need 10,000+ documents for good embeddings)
3. Preprocess text for Word2Vec training
4. Train your model and explore word similarities
5. Hunt for embedding failures (antonyms, rare words, polysemy)
6. Create and test business-relevant analogies
7. Compare your embeddings to pre-trained GloVe

What Your Output Should Look Like

Word Similarities:

- WORD SIMILARITIES
-

```
'good' is similar to:
great          0.842
excellent      0.798
nice           0.756
bad            0.712 ← Antonym problem!
decent          0.694
```

Analogy Results:

bad - better + good = ?
 → worse (0.634) ✓ Makes sense!
 → bad (0.598)
 → terrible (0.521)

Antonym Test:**□ ANTONYM TEST**

good	↔ bad	: 0.712	□	TOO SIMILAR!
love	↔ hate	: 0.689	□	TOO SIMILAR!
best	↔ worst	: 0.758	□	TOO SIMILAR!

Common Mistakes (and How to Avoid Them)

Mistake	Symptom	Fix
Too little data	Poor similarities, random results	Need 10,000+ documents minimum
min_count too high	"Word not in vocabulary" errors	Lower to 5 or 10
min_count too low	Noisy results from rare words	Increase to 10-20
Not lowercasing	"Good" and "good" are different vectors	Lowercase during preprocessing
Short training	Poor quality embeddings	Increase epochs to 15-20
Wrong sg parameter	Skip-gram vs CBOW confusion	sg=1 for skip-gram (usually better)

If you see this error:

KeyError: "word 'xyz' not in vocabulary"

The word appeared fewer than min_count times. Try a more common word or lower min_count.

If similarities seem random: - Your corpus is too small - Try more epochs - Check preprocessing (are you keeping meaningful words?)

Questions to Answer

- **Q1:** Which similarities made sense? Which surprised you?
- **Q2:** Document an embedding failure: what happened and why?

- **Q3:** Which analogies worked? Why do some fail?
 - **Q4:** How do your embeddings differ from pre-trained ones?
-

Submission

Upload to Canvas: Your completed .ipynb notebook with all cells executed

A handwritten signature in black ink, appearing to read "Richard Young".

Richard Young, Ph.D.

```
# Get word vector
vector = model.wv["good"] # → array of 100 floats

# Check vocabulary
"word" in model.wv # → True/False
len(model.wv) # → vocabulary size
```

Key Parameters Explained: | Parameter | What it does | Recommendation | |-----|-----|-----|
---| vector_size | Dimensions of embedding | 100-300 | | window | Words to consider as context |
5-10 | | min_count | Minimum word frequency | 5-20 | | sg | Algorithm choice | 1 (skip-gram) for small
data | | epochs | Training passes | 10-20 |

Going Deeper (Optional Challenges)

(Challenges A, B, C continue on next page)