Lightweight Word Embeddings (Word2Vec) and CPU BERT Demos

Francis Bond

What is Word2Vec?

- **Goal:** learn **word embeddings** (dense vectors) from raw text.
- **Distributional idea:** words that occur in similar contexts have similar vectors.
- Two classic training objectives:
  - **CBOW:** predict the target word from surrounding context words.
  - **Skip-gram:** predict surrounding context words from the target word.
- Result: geometry in vector space supports similarity, clustering, and some analogies.

How Word2Vec is trained (intuition)

- Slide a window over a corpus; generate (context, target) pairs.
- Train a small neural model to make good predictions of words from contexts (or vice versa).
- Use tricks for efficiency:
  - **Negative sampling** (common): discriminate true pairs from random noise pairs.
  - **Subsampling** frequent words to reduce dominance of *the, of, and, ...*
- The trained weights become word vectors; similarity is often cosine similarity.

Practical note: for classroom laptops, **loading a small pre-trained model** is usually easier than training on a large corpus.

Gensim: CPU-friendly embeddings

- **Gensim** provides efficient implementations of Word2Vec and tools for using pre-trained vectors (KeyedVectors).
- Typical workflow:
  1. install (`pip install gensim`)
  2. load a small pre-trained model (or train a tiny one for a demo)
  3. query similarity / nearest neighbors / analogies
- Keep it lightweight: use a **small model** (tens of MB) and CPU.

```
pip install gensim
```

Gensim: loading vectors + similarity

- You can load vectors saved in Gensim format, or in word2vec format.
- Once loaded, you can ask for nearest neighbors and cosine similarity.

```
from gensim.models import KeyedVectors

# Example (Gensim native format)
# kv = KeyedVectors.load("vectors.kv", mmap="r")

# Example (word2vec text format)
# kv = KeyedVectors.load_word2vec_format("vectors.txt", binary=False)

# Then:
# kv.most_similar("university", topn=10)
# kv.similarity("cat", "dog")
```

Tip: For large vectors, `mmap="r"` keeps RAM usage down.

Compositional / analogy arithmetic (classic demo)

- The famous analogy style query:

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

- In Gensim, use `most_similar` with positive/negative sets.

```
# "king" - "man" + "woman"  -> "queen" (often)
kv.most_similar(positive=["king", "woman"],
                negative=["man"],
                topn=5)
```

Youll often get queen near the top (depending on model + vocabulary).

Hyponymy / is-a vector trick (works... sometimes)

- People sometimes try a relation vector like:

$$\vec{is\_a} \approx \vec{animal} - \vec{dog}$$

  then apply it to a new hyponym: $\vec{queen} + \vec{is\_a}$.

- This is **not a principled guarantee**: hierarchies are not consistently linear in Word2Vec space.

```
# Build a crude "is-a" direction from one example pair
is_a = kv["animal"] - kv["dog"]

# Apply it to another specific term
kv.most_similar(positive=["queen", is_a], topn=10)
```

Sometimes youll see broader categories (`monarch`, `royalty`, `person`...), sometimes you wont. Thats the teaching point.

Why antonyms are hard for Word2Vec (and friends)

- Word2Vec learns from **shared contexts**.
- Antonyms often occur in **very similar contexts**:
  - *hot coffee* / *cold coffee*
  - *high temperature* / *low temperature*
- So embeddings can place antonyms **close together** even though meanings oppose.
- Consequence: nearest neighbors is **not** the same as synonyms.

Useful classroom exercise: compare nearest neighbors for a word and ask students to label each neighbor as synonym / related / antonym / topical.

From static embeddings to contextual models

- Word2Vec gives **one vector per word type**.
- But many words are polysemous:
  - *bank* (river vs. finance)
  - *chestnut* (tree/nut vs. old chestnut = stale joke/idea)
- Contextual models (BERT-family) produce **token embeddings**: the vector for *bank* depends on the sentence.

CPU BERT in class: small models + simple pipelines

- Hugging Face `transformers` runs fine on CPU for small demos.
- Use **distilled** models to keep it manageable:
  - `distilbert-base-uncased` (English)
  - multilingual options exist too, but are often heavier

```
pip install transformers torch --index-url https://download.pytorch.org/wh
```

```
from transformers import pipeline
fill = pipeline("fill-mask", model="distilbert-base-uncased")
```

On CPU its slower than GPU, but fine for short sentences.
The first time we run it, it downloads the model

```
~/.cache/huggingface/
 hub/
     models--distilbert-base-uncased/
```

BERT wow moment: context sensitivity

- Same word, different context ⇒ different predictions.

```
# Same surface form, different sense-cues in context
s1 = "I sat on the bank of the river and watched the water."
s2 = "I went to the bank to open a new account."

# Fill-mask expects a [MASK] token; we mask a nearby word to probe context
print(fill("I sat on the bank of the river and watched the [MASK].")[:3])
print(fill("I went to the bank to open a new [MASK].")[:3])
```

Students see that the **same surrounding topic** drives very different completions.

Negation: a simple demo (and a warning)

- Negation is famously tricky.
- A quick classroom probe: compare predictions with vs. without *not*.

```
print(fill("A robin is a [MASK].")[:5])
print(fill("A robin is not a [MASK].")[:5])
```

Warning: masked-LM objectives do not reason logically. Sometimes the negated sentence still suggests plausible categories (or even repeats the positive behavior). That unpredictability is itself a useful discussion point.

Metaphor / idiom strength: double-edged ...

- BERT often completes conventional metaphors / idioms well because it has seen them in varied contexts.

```
print(fill("His words were a double-edged [MASK]."))[:5])
```

Often `sword` appears near the top. This contrasts with Word2Vec arithmetic: Word2Vec can do some analogies, but it does not represent compositional phrase meaning the same way.

Bert demo code is in one file

- Put everything in a single script, e.g. `bert_demo.py`

```
$ python bert_demo.py
```