



★ Member-only story

BERT For Measuring Text Similarity

High-performance semantic similarity with BERT



James Briggs · [Follow](#)

Published in Towards Data Science · 5 min read · May 5, 2021



778



8



Sentence Similarity

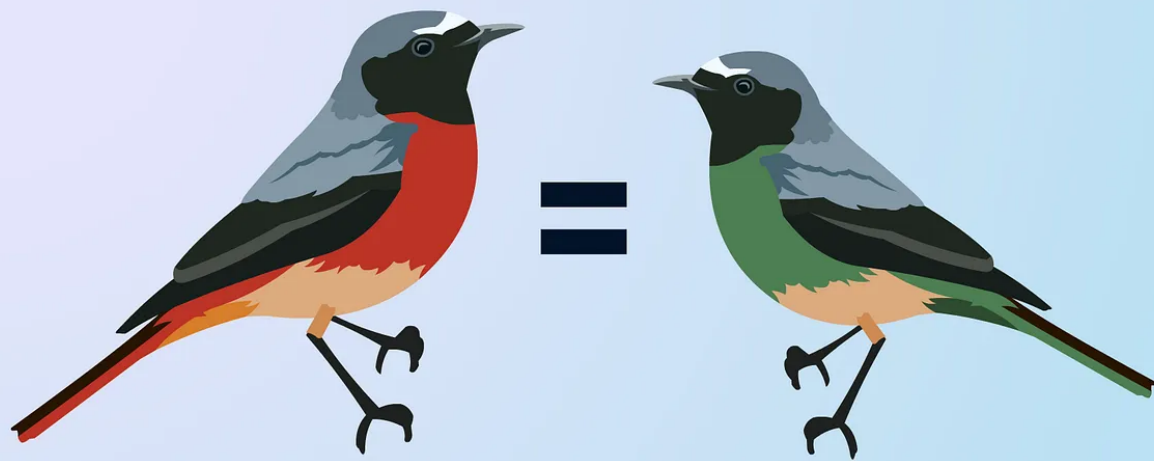


Image by author

All we ever seem to talk about nowadays are BERT this, BERT that. I want to write about something else, but BERT is just *too* good — so this article will be about BERT *and* sequence similarity!

A big part of NLP relies on similarity in highly-dimensional spaces. Typically an NLP solution will take some text, process it to create a big vector/array representing said text — then perform several transformations.

It's highly-dimensional magic.

Sentence similarity is one of the clearest examples of how powerful highly-dimensional magic can be.

The logic is this:

- Take a sentence, convert it into a vector.
- Take many other sentences, and convert them into vectors.
- Find sentences that have the smallest distance (Euclidean) or smallest angle (cosine similarity) between them — more on that [here](#).
- We now have a measure of semantic similarity between sentences — easy!

At a high level, there's not much else to it. But of course, we want to understand what is happening in a little more detail and implement this in Python too! So, let's get started.

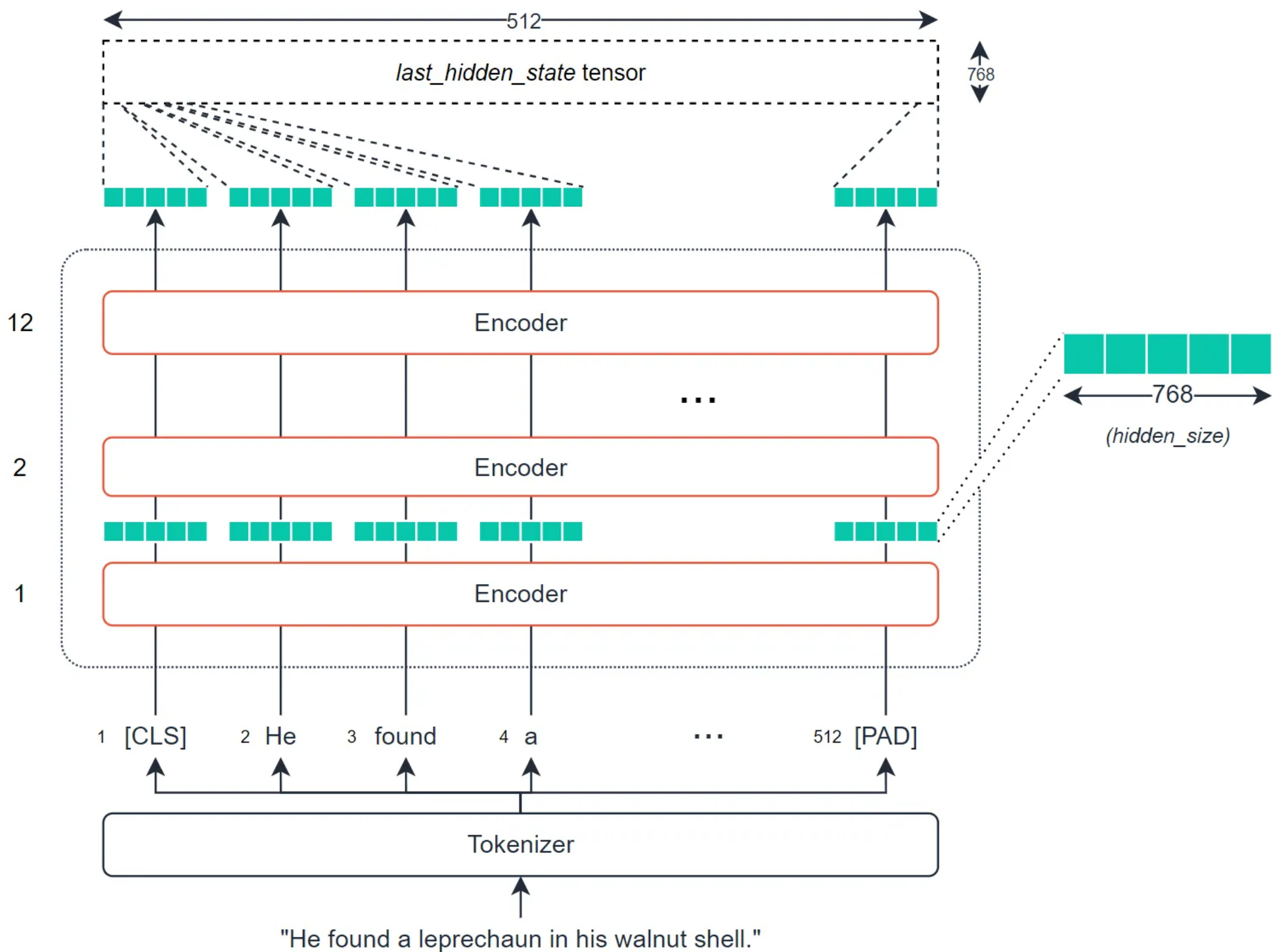
. . .

Why BERT Helps

BERT, as we already mentioned — is the MVP of NLP. And a big part of this is down to BERT's ability to embed the meaning of words into densely packed vectors.

We call them *dense* vectors because every value within the vector has a value and has a reason for being that value — this is in contrast to *sparse* vectors, such as one-hot encoded vectors where the majority of values are 0.

BERT is **great** at creating these dense vectors, and each encoder layer (there are several) outputs a set of dense vectors.



BERT base network — with the hidden layer representations highlighted in green.

For BERT base, this will be a vector containing 768. Those 768 values contain our numerical representation of a single token — which we can use as contextual word embeddings.

Because there is one of these vectors for representing each token (output by each encoder), we are actually looking at a tensor of size 768 by the number of *tokens*.

We can take these tensors — and transform them to create semantic representations of the input sequence. We can then take our similarity metrics and calculate the respective similarity between different sequences.

The simplest and most commonly extracted tensor is the *last_hidden_state* tensor — which is conveniently output by the BERT model.

Of course, this is a pretty large tensor — at 512x768 — and we want a vector to apply our similarity measures to it.

To do this, we need to convert our *last_hidden_states* tensor to a vector of 768 dimensions.

Creating The Vector

For us to convert our *last_hidden_states* tensor into our vector — we use a **mean pooling** operation.

Each of those 512 tokens has a respective 768 values. This pooling operation will take the mean of all token embeddings and compress them into a single 768 vector space — creating a ‘sentence vector’.

At the same time, we can’t just take the mean activation as is. We need to consider null padding tokens (which we should not include).

. . .

In Code

That’s great on the theory and logic behind the process — but how do we apply this in reality?

We’ll outline two approaches — the easy way and the slightly more complex way.

Easy — Sentence-Transformers

The easiest approach for us to implement everything we just covered is through the `sentence-transformers` library — which wraps most of this process into a few lines of code.

First, we install sentence-transformers using `pip install sentence-transformers`. This library uses HuggingFace's *transformers* behind the scenes — so we can actually find *sentence-transformers* models [here](#).

We'll be making use of the `bert-base-nli-mean-tokens` model — which implements the same logic we've discussed so far.

(It also uses 128 input tokens, rather than 512).

Let's create some sentences, initialize our model, and encode the sentences:

Great, we now have four sentence embeddings — each containing 768 values.

Now what we do is take those embeddings and find the cosine similarity between each. So for sentence 0:

| *Three years later, the coffin was still full of Jello.*

We can find the most similar sentence using:

Now, this is the easier — more abstract approach. Seven lines of code to compare our sentences.

Involved — Transformers And PyTorch

Before getting into the second approach, it is worth noting that it does the same thing as the first — but at one level lower.

With this approach, we need to perform our own transformation to the *last_hidden_state* to create the sentence embedding. For this, we perform the mean pooling operation.

Additionally, before the mean pooling operation, we need to create *last_hidden_state*, which we do like so:

After we have produced our dense vectors `embeddings`, we need to perform a *mean pooling* operation to create a single vector encoding (the **sentence embedding**).

To do this mean pooling operation, we will need to multiply each value in our `embeddings` tensor by its respective `attention_mask` value — so that we ignore non-real tokens.

Once we have our dense vectors, we can calculate the cosine similarity between each — which is the same logic we used before:

We return almost the same results — the only difference being that the cosine similarity for index **three** has shifted from 0.5547 to 0.5548 — a minor difference due to rounding.

• • •

That’s all for this introduction to measuring the semantic similarity of sentences using BERT — using both *sentence-transformers* and a lower-level implementation with *PyTorch* and *transformers*.

You can find the full notebooks for both approaches [here](#) and [here](#).

I hope you’ve enjoyed the article. Let me know if you have any questions or suggestions via [Twitter](#) or in the comments below. If you’re interested in more content like this, I post on [YouTube](#) too.

Thanks for reading!

• • •

References

N. Reimers, I. Gurevych, [Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks](#) (2019), Proceedings of the 2019 Conference on Empirical Methods in NLP

 [NLP With Transformers Course](#)

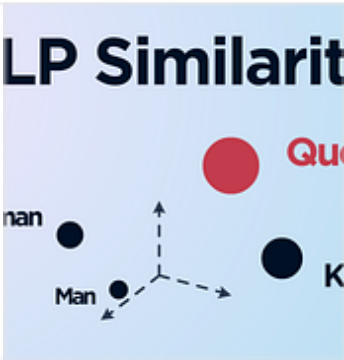
. . .

If you're interested in learning more about NLP similarity metrics (including cosine similarity, which we used here) — check out this article I wrote explaining the most popular metrics:

Similarity Metrics in NLP

Euclidean distance, dot product, and cosine similarity

[towardsdatascience.com](#)

A diagram titled "LP Similarity" showing a 3D coordinate system with axes. It features several points: a red point labeled "Query", a black point labeled "Man", and another black point labeled "K". Dashed lines connect the points to the axes, illustrating the concept of similarity metrics in a vector space.

. . .

**All images are by the author except where stated otherwise*



Written by James Briggs

9.6K Followers · Writer for Towards Data Science

Follow



Freelance ML engineer learning and writing about everything. I post a lot on YT
<https://www.youtube.com/c/jamesbriggs>

More from James Briggs and Towards Data Science

 James Briggs in Towards Data Science


The Right Way to Build an API with Python

All you need to know on API development in Flask

 · 7 min read · Sep 11, 2020

 1.1K  12



 Jacob Marks, Ph.D. in Towards Data Science

How I Turned My Company’s Docs into a Searchable Database with...

And how you can do the same with your docs

15 min read · Apr 25

 1.6K  22



 Barr Moses in Towards Data Science

Zero-ETL, ChatGPT, And The Future of Data Engineering

The post-modern data stack is coming. Are we ready?

9 min read · Apr 4


 1K  21



 James Briggs in Towards Data Science

GPU-Acceleration Comes to PyTorch on M1 Macs

How do the new M1 chips perform with the new PyTorch update?

 · 6 min read · Jun 1, 2022

 291  8



See all from James Briggs

See all from Towards Data Science

Recommended from Medium



Skanda Vivek in Towards Data Science

Fine-Tune Transformer Models For Question Answering On Custom...

A tutorial on fine-tuning the Hugging Face RoBERTa QA Model on custom data and...

★ · 5 min read · Dec 15, 2022



361



2



Andrea D'Agostino in Towards Data Science

How to Train a Word2Vec Model from Scratch with Gensim

In this article we will explore Gensim, a very popular Python library for training text-base...

★ · 9 min read · Feb 7



64





Angel Das in Towards Data Science

Generating Word Embeddings from Text Data using Skip-Gram...

Introduction to embeddings in natural language processing using Artificial Neural...

★ · 13 min read · Nov 10, 2022



249



4



Teemu Maatta in Better Programming

OpenAI's Embedding Model With Vector Database

The updated Embedding model offers State-of-the-Art performance with 4x longer...

★ · 12 min read · Jan 10



240



7



Dr. Mandar Karhade, MD. PhD. in Towards AI

OpenAI Releases Embeddings model: text-embedding-ada-002

It is Powerful, cheaper, and more flexible!

★ · 5 min read · Dec 16, 2022



99



5



James Briggs in Towards Data Science

Advanced Topic Modeling with BERTopic

How do we organize the world's most unorganizable data?

★ · 12 min read · Dec 20, 2022



137



1



See more recommendations