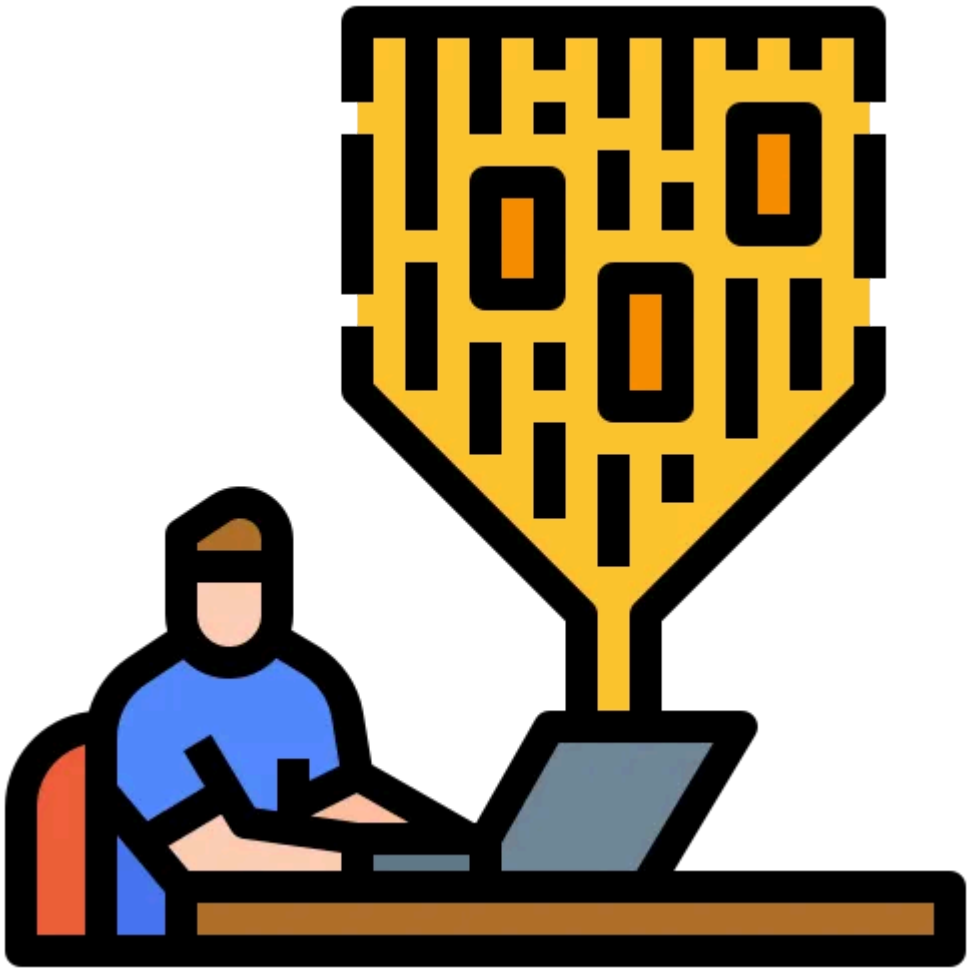
Decoding Sentence Encoders

Mathias Leys · Follow
Published in ML6team
8 min read · Jun 13, 2022



Introduction

The current sentence encoders landscape is quite diverse and can be a bit daunting at first. It is easy to get confused if you don't properly grasp the idea behind

they aim to produce qualitative embeddings for sentence-level tasks. But then again, **regular encoders** also produce high-quality embeddings and are perfectly suited for sentence-level tasks. The architectures behind transformers and their sentence transformer counterparts also seem identical. Then there are also **cross-encoders** which are used for, you guessed it, performing sentence-level tasks.

Call it Lionel because things are getting messy.

Why sentence transformers?

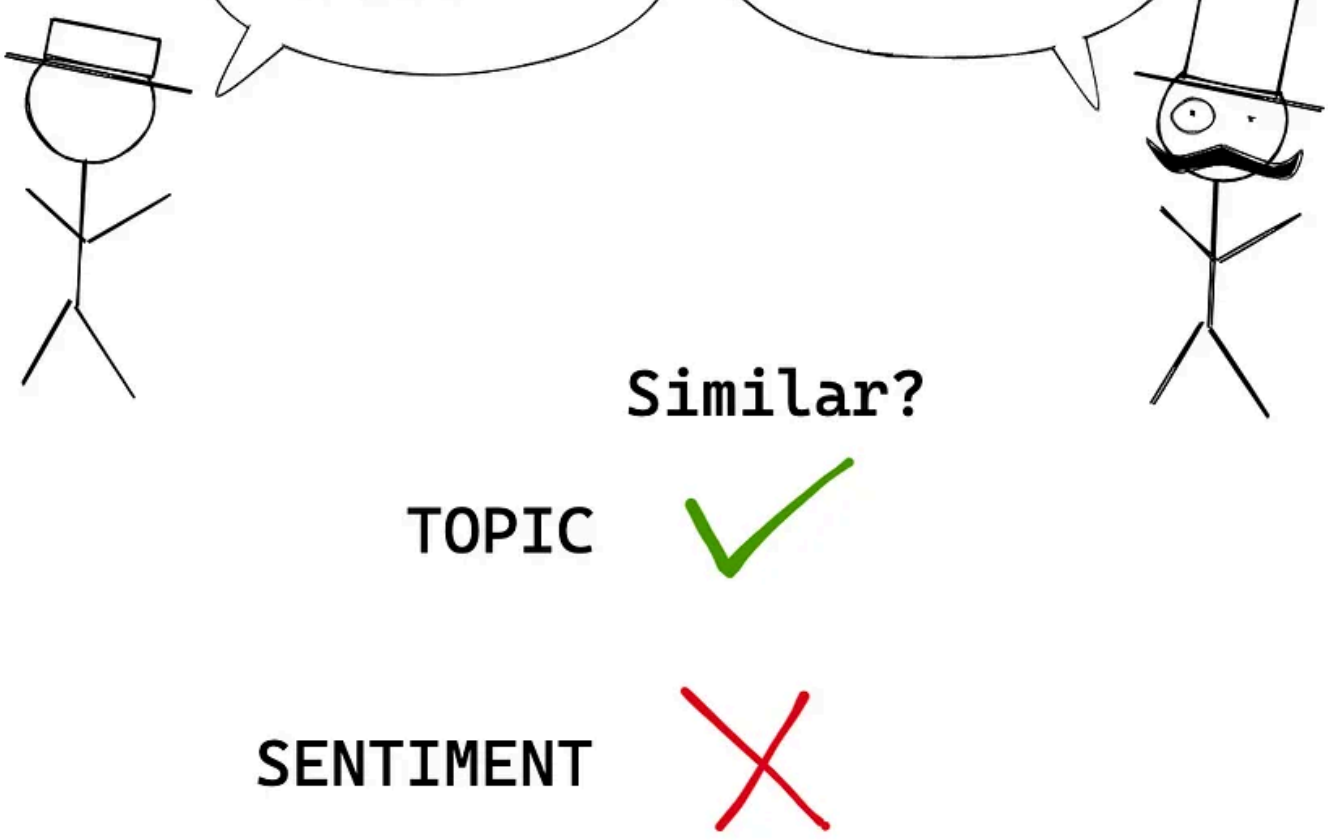
As things were getting messi, it seems only suited that we immediately kick things off (pun intended). We will do so by addressing a common myth:

> "sentence transformers produce better sentence embeddings than regular transformers".

If only it were this simple.
The thing about (sentence) embeddings is that their **quality depends on the use case**. Embeddings may be very suited to a particular use case and not at all suited to another. I will demonstrate this via—as any mathematician would agree — the most rigorous of proof methods: *proof by anecdotal evidence*.

Consider the following statements: *"Nuclear power is dangerous!"* and *"Nuclear power is the future of energy!"*. Politics aside, should these statements be considered similar, i.e. should the sentence embeddings be close together? Well it depends on your perspective. If you are talking about the topic, then definitely yes: both statements are opinions on nuclear power. So in that sense, they are very similar. However, if you are talking about sentiment, then the answer is a resounding no. They are about as dissimilar in terms of sentiment as you can get.

Q.E.D.



All embeddings are equal but some are more equal than others

> ***Intermezzo:*** *This is why large pre-trained language models are trained on very general tasks (such as masked language modeling). The idea is that the embeddings reflect a very broad understanding of language that can later be tuned to a specific use case.*

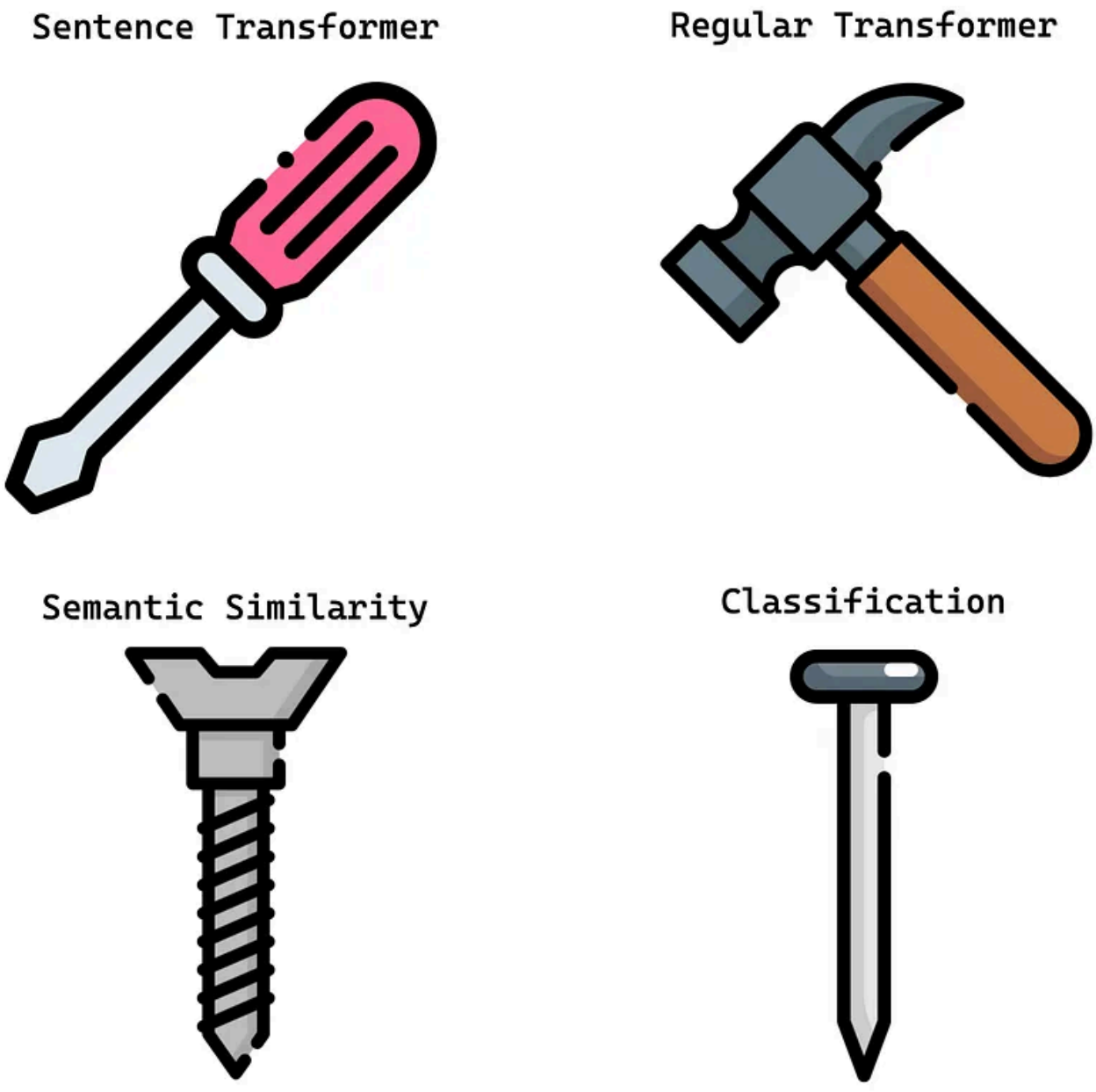
Back to the question at hand though: *"do sentence transformers produce better sentence embeddings?"*. The answer is a resounding, undisputed "**it depends**"!

Regular transformers produce sentence embeddings by performing some pooling operation such as the element-wise arithmetic mean on its token-level embeddings. A good pooling choice for BERT is CLS pooling. BERT has a special `<CLS>` token that is supposed to capture all the sequence information. It gets tuned on next-sentence prediction (NSP) during pre-training.

> 🔔. ***Gotcha alert*** 🔔 *: RoBERTa does not perform NSP and does not have a* `<CLS>` *token. It has a start-of-sentence* `<s>` *token but the embedding is not trained to portray meaningful sentence representations out-of-the-box*

However, for **semantic similarity tasks**, the sentence embeddings aren't great. This is where **sentence transformers** come into play. The training process of sentence transformers is especially designed with semantic similarity in mind. More on the training process later though.

So in conclusion: if you want to perform some sentence classification task, a regular transformer will more than do the trick. However, if you want to do a task such as semantic search or clustering that relies on semantic similarity, you should look towards sentence transformers.



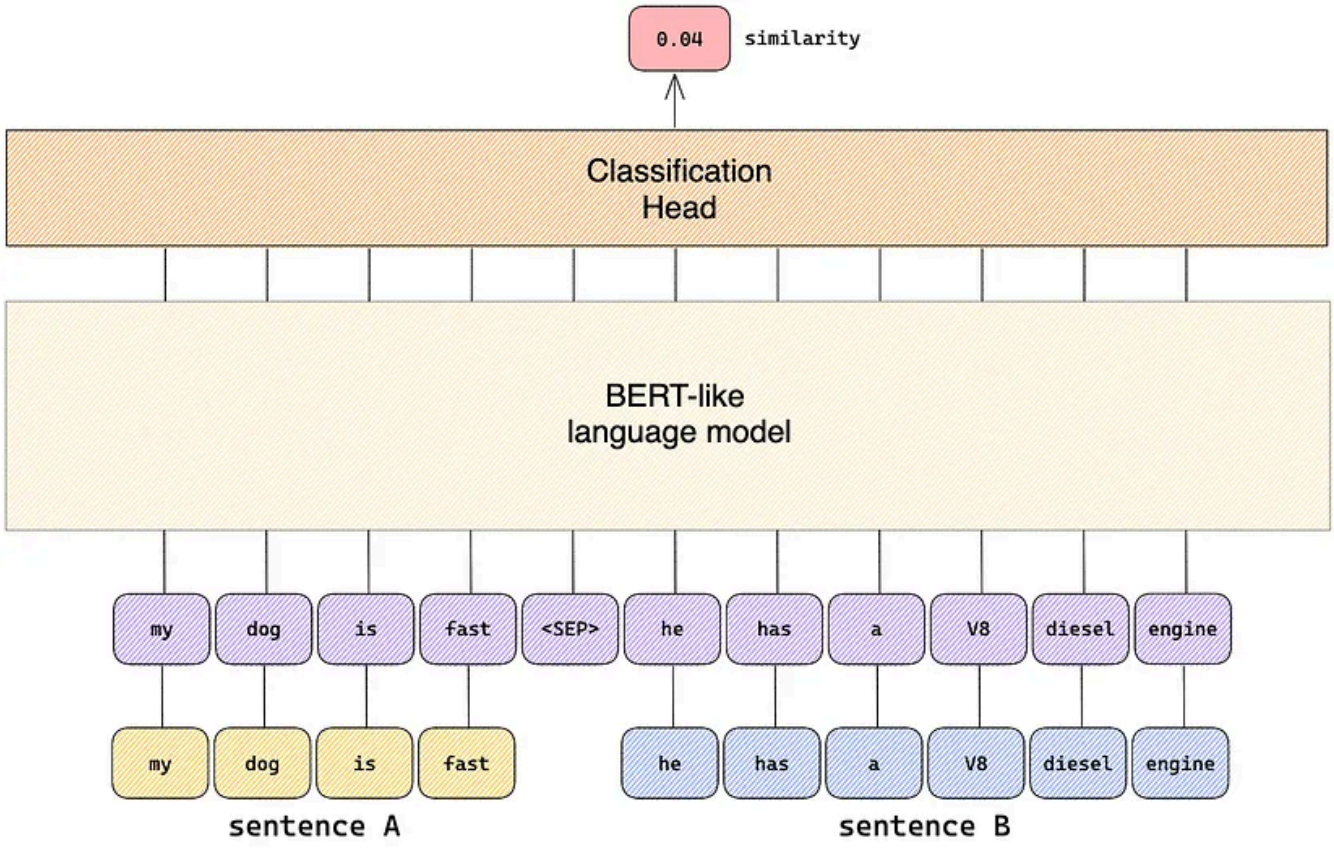
The right tool for the right job, my friends

> "Couldn't you see semantic similarity as a classification problem where you try to classify two sentences as either "relevant" or "not relevant" instead of all this sentence encoder stuff?"

You would be very correct in thinking this. In fact, this is exactly what a **cross-encoder** does. There's no slipping anything past you today.

Essentially what cross-encoders do is concatenate two sentences with a separator `<SEP>` token and feed it into a language model. There is a classification head on top of the language model that is then trained to **predict a target "similarity" value.**

So for example, a cross-encoder would concatenate the sentences *"my dog is fast"* and *"he has a V8 diesel engine"* and predict a low similarity score, even though they both contain words related to speed. This is correct of course. Everyone knows that Snuffles is rocking a V12 TDI under the bonnet. He's very fuel-efficient.



Cross-encoder architecture

X	y
("my dog is fast", "my canine is rapid")	0.95
("my dog is fast", "he has a V8 diesel engine")	0.10
...	...

Cross-encoder sample training data

Whereas cross-encoders tend to perform very well on sentence-level tasks, they do suffer from a major drawback: **cross-encoders do not produce sentence embeddings.**

In the context of information retrieval, this implies that we **cannot pre-compute document embeddings** and efficiently compare these to a query embedding. We are also **not able to index document embeddings** for efficient search.

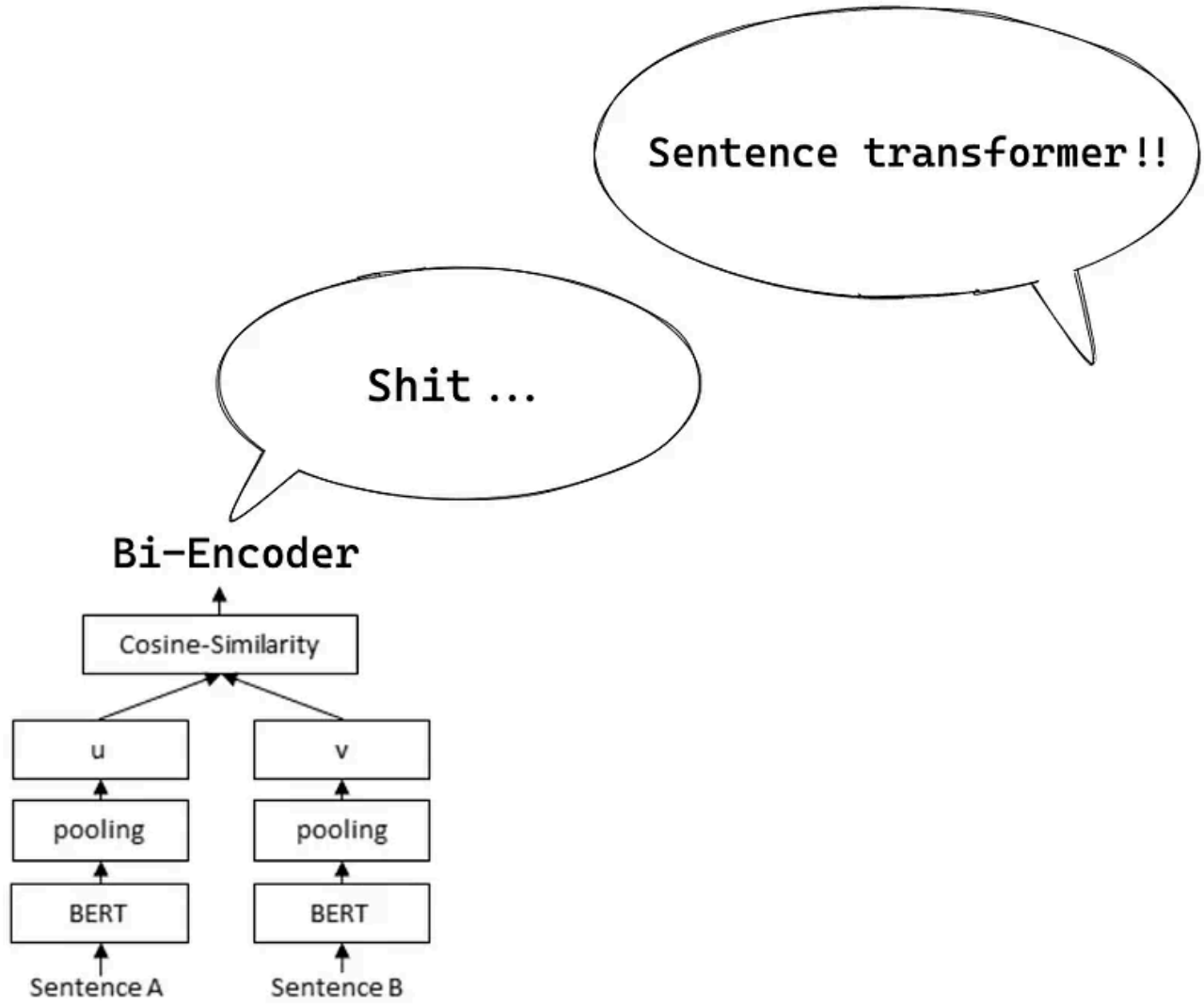
In the context of sentence clustering, this means that we have to pass every possible pair of documents into the cross-encoder and compute the predicted similarity.

I think you can imagine that this **slows down practical applications** to the point of being almost unusable.

Bi-encoders

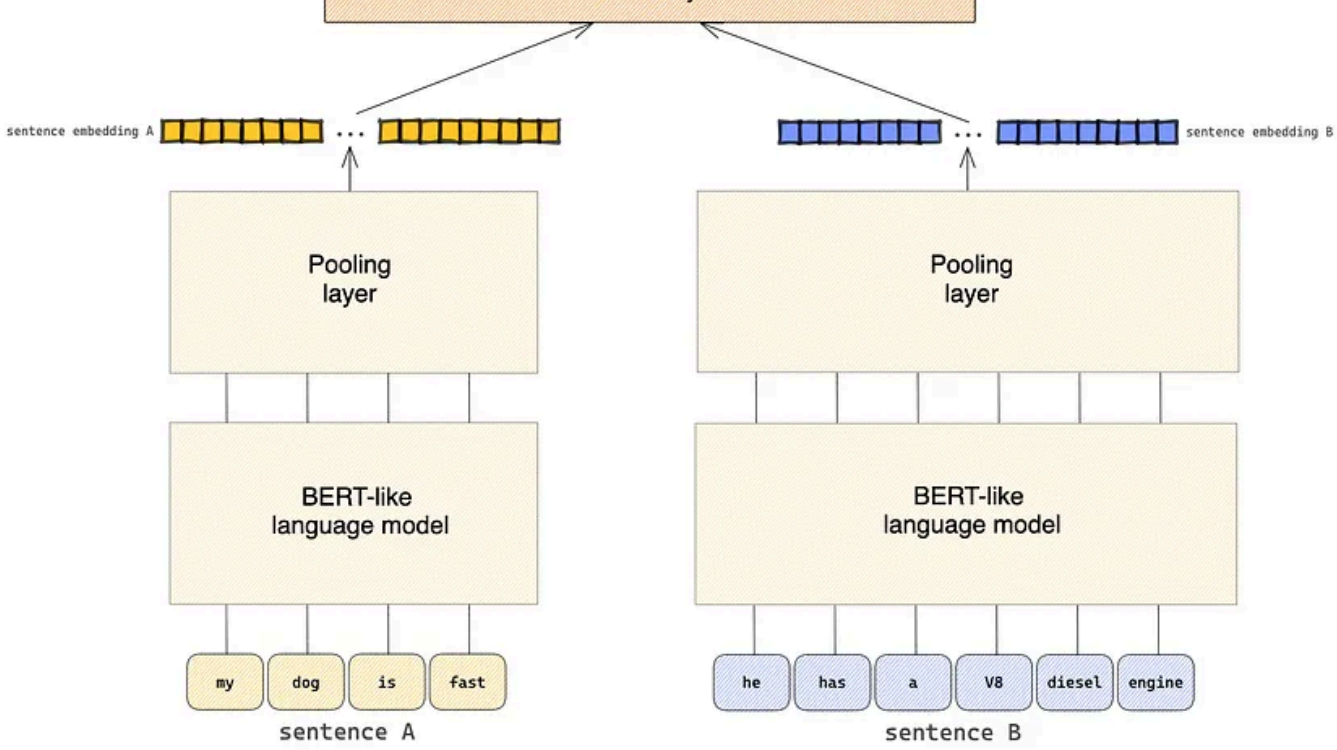
You may have noticed that I have used the terms *"sentence transformer"* and *"bi-encoder"* pretty much interchangeably. There is a good reason for this: they are pretty much interchangeable. Sentence transformer just rather refers to the Python package `sentence-transformers` and the original paper SBERT, whereas bi-encoder refers more to the actual architecture.

by your full name



Bi-encoder, sentence transformer. What's in the name?

The inefficiency of cross-encoders is exactly were **bi-encoders** come into play. Although cross-encoders tend to be more accurate, bi-encoders have the advantage that they actually **produce sentence embeddings** which make them much faster and more practical in real-world settings as they allow for indexing, pre-computing embeddings, etc.



Bi-encoder architecture

As an illustration, clustering just 10.000 sentences with a cross-encoder would require computing similarity scores for about 50 million sentence pairs which would take around 65 hours with a BERT architecture. In that time, you could watch all 8 Harry Potter movies twice, listen to every Beatles song ever made and you would still have a few hours to spare.

In comparison, the same task with a bi-encoder would take around 5 seconds. That is about the length of the *"You're a wizard Harry"* scene alone.

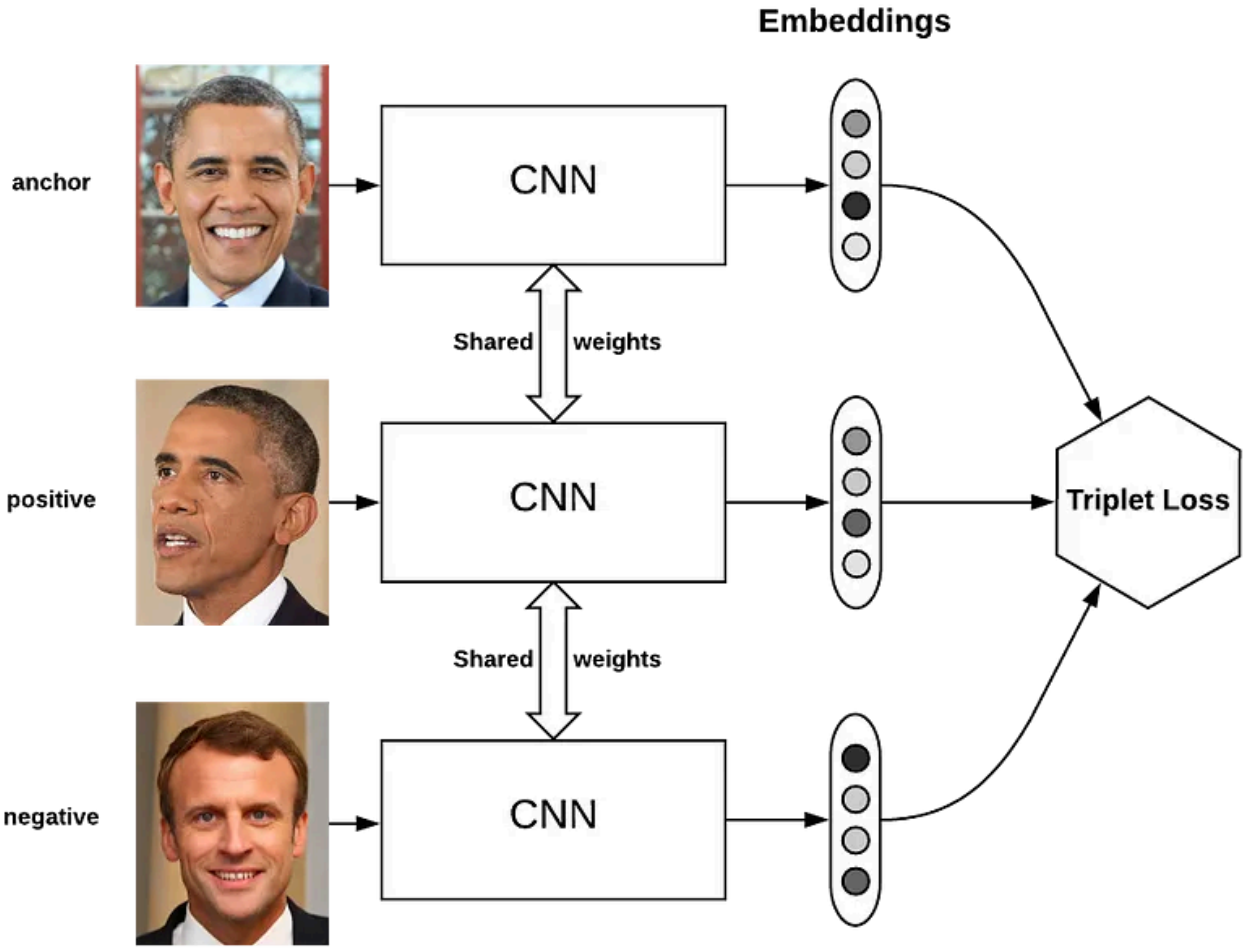
It should be noted, however, that knowledge distillation training procedures where a bi-encoder student model tries to imitate a cross-encoder teacher model seem to be very promising. However, this is really a topic in and of itself so I won't go into it here.

Sentence transformer training

The real power of sentence transformers lies in their training procedure, which is specifically designed for semantic similarity. Sentence transformers are trained as so-called **Siamese networks**.

embedding to the positive data point embedding and **maximize the distance between the anchor data point embedding to the negative data point embedding.**

Siamese training is an idea borrowed from computer vision. Say you want to design a **facial similarity** system. You would structure your data as triplets of an "anchor" image, a "positive" image (i.e., a different image of the same person) and a "negative" image (i.e., an image of a different person). The network is then trained so that the embeddings of the anchor and the positive image are very close and the embeddings of the anchor and the negative image are not.



Siamese networks are also used for facial similarity

Fully analogously, in the context of sentence transformers, the data is structured as triplets of an anchor sentence along with a positive sentence which is semantically similar to the anchor and a negative sentence which is semantically dissimilar to the anchor.

anchor	positive	negative
"my dog is fast" "I am hungry" ...	"my canine is rapid" "I want to eat" ...	"the lights are on" "The store is open" ...

Bi-encoder sample triplet training data

> *This training process forces sentence transformers to produce semantically meaningful embeddings in the sense that* ***sentence embeddings of semantically similar sentences are close together and sentence embeddings of semantically dissimilar sentences are not.***

Conclusion

In the above paragraphs, we went over sentence transformers in great technical detail. However, let's not lose sight of the bigger picture and recap what we discussed.

👉 We discussed **cross-encoders** which approach **semantic similarity as a classification problem.** We concluded that while cross-encoders are often very accurate, they are also often **not very practical** in a real-life setting.

👉 We saw how **bi-encoders** are designed to offer a solution to the inefficiency of cross-encoders.

👉 We rounded off by discussing the **bi-encoder training process** which is what really makes it stand out as their underlying architecture is identical to that of a regular transformer.

Sentence Transformers | Transformers | NLP | Semantic Search | SEO