The Art of Pooling Embeddings 🎨

Mathias Leys · Follow
Published in ML6team
8 min read · Jun 20, 2022

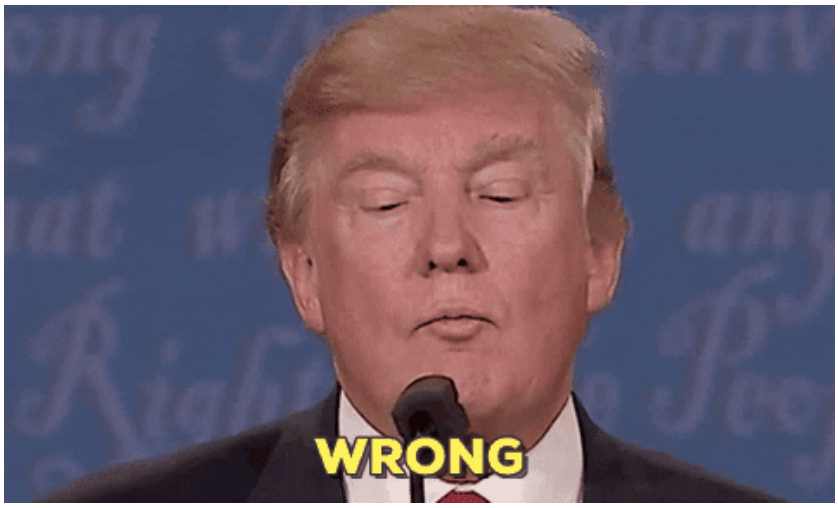
Introduction

There are some questions that everyone has asked themselves at least once in their life: *"what is the meaning of life?"*, *"what happens after you die?"* and of course: *"what is the purpose of sentence embeddings?"*. Or is it just me?

Jokes aside, sentence embeddings can be a bit of an unintuitive concept at first glance.

> *Think about it:*
> *Why would you need a compressed representation of the information in a sequence when you have access to all information present in that sequence?*

In other words, couldn't you just use your token embeddings to do whatever you were going to use your sentence embedding for? You would base your decision on much more and more granular information so it stands to reason that this is the better option? … Right?



If the above paragraph made you second-guess everything you know about NLP, then this blogpost is for you.

In this post, you'll find that **sentence embeddings are very simple in design**. However, you will also find that this **apparent simplicity hides a surprising amount of complexity and nuance**.

What are sentence embeddings?

Let's not get too ahead of ourselves just yet. Before we get to the "Why", let's first discuss the "What" and make sure that we really understand what sentence embeddings even are first.

Sentence embeddings play a completely analogous role to token-level embeddings with the main difference being that, as the name suggests, there is only **one embedding for the entire sequence** rather than one embedding per token.

The process of converting a sequence of embeddings into a sentence embedding is called *"pooling"*. Intuitively, this entails compressing the granular token-level representations into a single fixed-length representation that is supposed to reflect the meaning of the entire sequence.

The fact that sentence embeddings are **compressions** is very important to always keep in the back of your mind. This is, for instance, the reason why you likely wouldn't get sentence embeddings that can grasp the fine details in very long texts. The compression would simply be too lossy to still capture this level of granularity.

Note that transformers will also truncate any input sequence longer than a `max_seq_length` *(the exact value varies from model to model but is often* `512` *).*

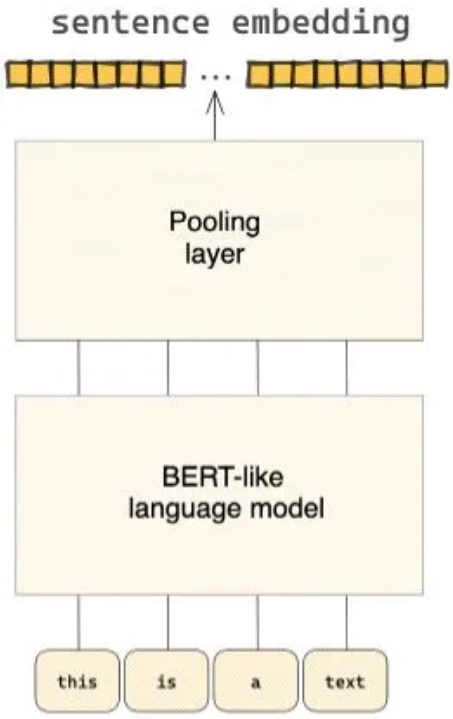
> *Sentence embeddings are inherently* ***compressions*** *of the information in a sequence of text and compressions are inherently* ***lossy****. This implies that sentence embeddings are representations with a* ***lower level of granularity****.*

Consider the following simple example: you have a text that consists of 200 tokens and are using a language model that produces 512-dimensional embeddings. The resulting embeddings will be 200 distinct 512-dimensional arrays. The sentence embedding will, by contrast, represent the information in the sequence in only one single 512-dimensional array.

Pooling methods

Now that we have a solid grasp of how pooling is used in computing sentence embeddings, let's take a more formal look into what goes on behind the scenes.

More formally, pooling entails adding a layer on top of your language model that performs some aggregating function (think `mean`, `max`, etc.).



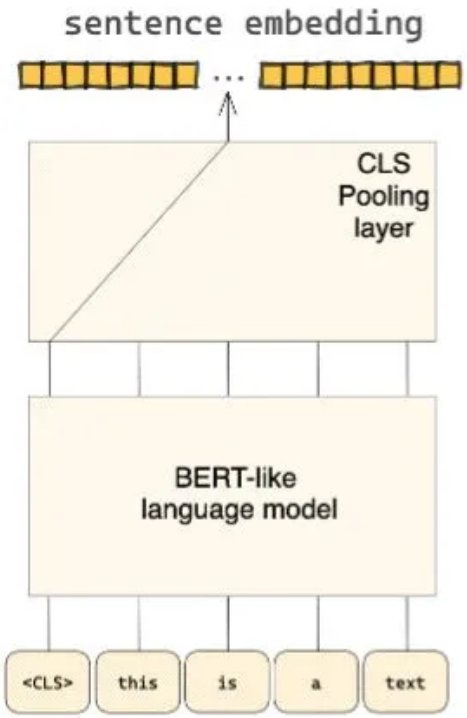
A pooling layer aggregates token-level embeddings into one sentence-level embedding

Theoretically, you could use any aggregating function in your pooling layer. In practice, we notice that two methods are predominantly used:

☝️ CLS pooling

The first commonly used pooling method is CLS pooling. Essentially, this method entails appending a special `<CLS>` token to the start of every sequence. This special token is meant to

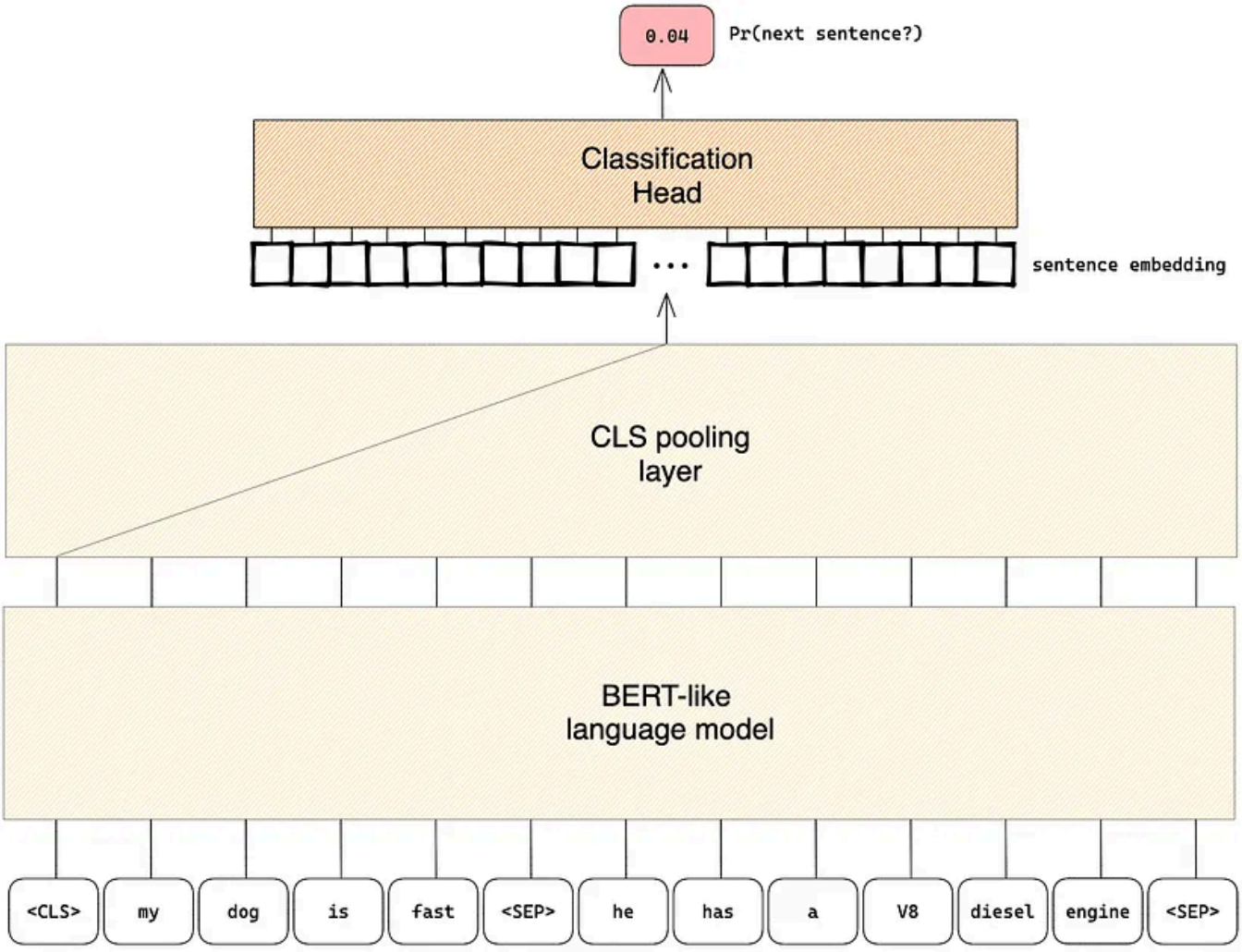
capture the sequence-level information. Therefore, the pooling layer aggregates by simply taking the **CLS token embedding** and using this as the sentence embedding.



CLS pooling aggregates by taking the token embedding of a special CLS token

During the training process, some sentence-level classification task based on this CLS embedding will tune the CLS token representation via backpropagation.
FYI: hence the name CLS (Classification) token.

For instance, in BERT's pre-training process, this task is NSP (Next Sentence Prediction).

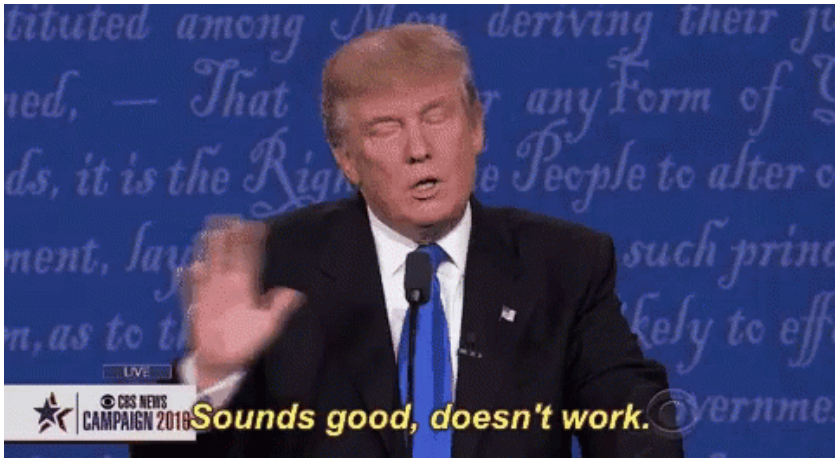


Visual representation of BERT's Next Sentence Prediction (NSP) training task

It is evident that the CLS embedding needs to have been fine-tuned on some (sentence-level) task. However, not every language model does this in its original pre-training procedure.

As discussed, BERT performs NSP in its original training process so a pre-trained BERT produces meaningful CLS representations out-of-the-box.

However, for instance RoBERTa does not use NSP in its pre-training process nor does it perform any other task that tunes the CLS token representation. Therefore, it does **not** produce meaningful CLS representations out-of-the-box.

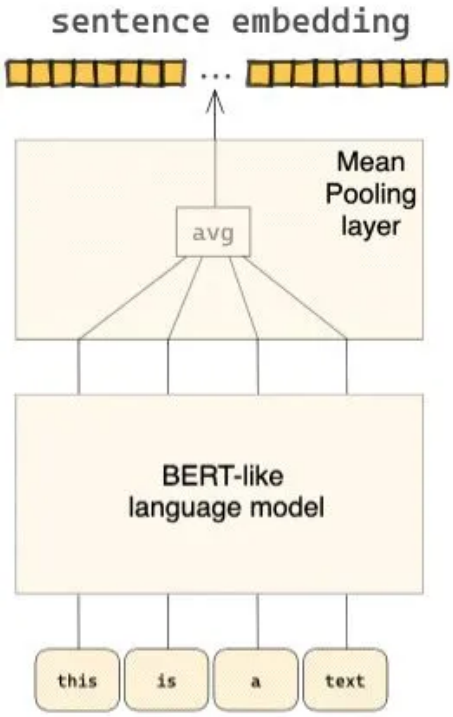


If you want to use CLS pooling on RoBERTa, you will need to fine-tune its CLS representation on some task first.

🔔 Nitpick alert 🔔
If we're being pedantic, RoBERTa doesn't even have a CLS token. It has a start of sequence `<s>` token that you can use for the same purpose.

✌️ Mean pooling
The second commonly used pooling method is mean pooling.

As the name suggests, it aggregates by taking the element-wise arithmetic mean of the token-level embeddings.
Mean pooling should not be confused with nice pooling which aggregates by politely asking the token embeddings.



Mean pooling aggregates by taking the element-wise arithmetic mean

Besides this, max pooling and mean_sqrt_len pooling are also sometimes used but much less frequently.

These pooling methods work completely analogously to mean pooling except for the fact that they aggregate by taking the element-wise max and by taking the element-wise mean divided by the square root of the number of token in the sequence respectively.

There is unfortunately **no clear consensus** on which pooling method to use although HuggingFace uses CLS pooling by default for sequence classification taks, so make of that what you will.

Why use sentence embeddings?

Now that we have a clear view on the main mechanisms behind sentence embeddings, we can finally get back to the question at hand:

> *Why would you need a compressed representation of the information in a sequence when you have access to all information present in that sequence?*

☝ **The whole is greater than the sum of its parts**
Aristotle — a notorious fan of sentence embeddings — once said *"the whole is greater than the sum of its parts"*. This quote applies well to this situation.

Sure, the token embeddings of a sequence represent much **more information** but does the sentence embedding capture the **same information**? Or does it grasp information that relates to the sequence **as a whole** rather than the individual constituents? *Spoiler alert: it's the latter.*

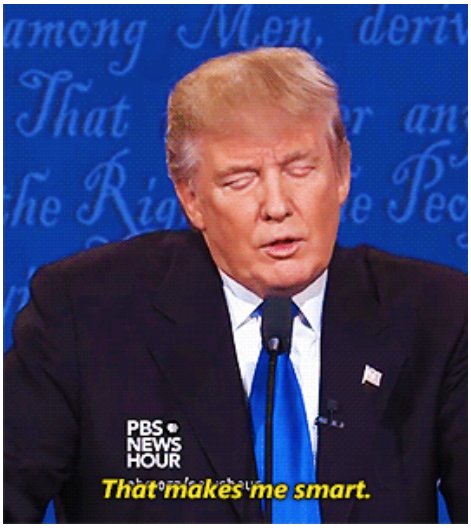
Sentence embeddings are trained for tasks that require knowledge of the meaning of a sequence as a whole rather than the individual tokens.
Some concrete examples of such tasks are:
👉 Sentiment analysis
👉 Semantic similarity (in sentence transformers' pre-training)
👉 NSP (in BERT's pre-training)
👉 ...

Therefore, it makes perfect intuitive sense to opt to use a sentence embedding for these tasks over token embeddings. Empirically, we also notice that sentence-level transfer outperforms word-level transfer on such tasks.

✌ **Sometimes you just need fixed-length representations**
For some tasks, it just makes sense to use **fixed-length** embeddings. Take sequence clustering for instance. How would you go about clustering sentences based on a token embedding group of variable length representing each sentence? Here, it just makes sense to have a single fixed-length representation per sentence.

A note on similarity measures

Congrats, at this point you should be comfortable with the most important concepts related to sentence embeddings!



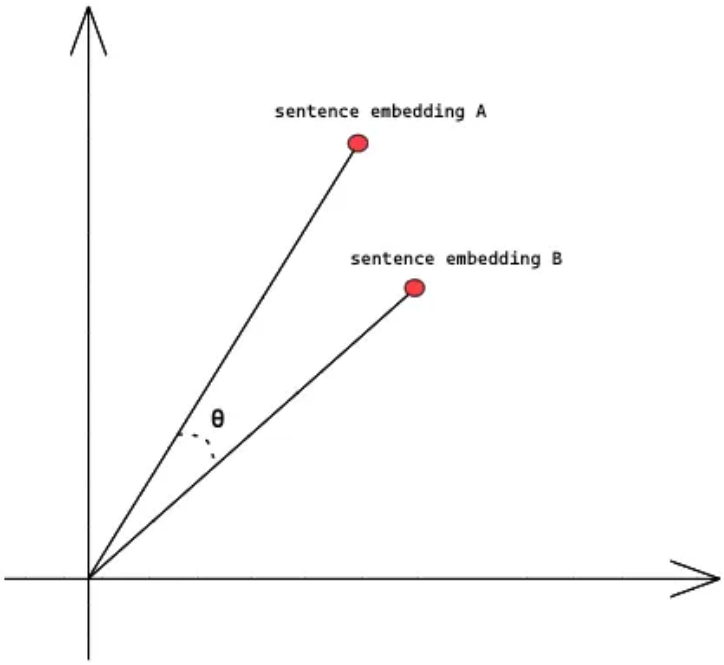
We can round off by quickly taking a look at embedding similarity measures.

Many sentence-level tasks (e.g., semantic search, clustering, etc.) rely on some notion of "closeness" of sentence embeddings or rather a notion of **similarity**.

There is a wide range of possible similarity measures but the most commonly used one when talking about sentence embeddings is **cosine similarity**.

The idea here is that you see the cosine of the angle between the embeddings as a measure of similarity between the embeddings. If the angle between the embeddings is small, the cosine will

be close to 1 and as the angle grows, the cosine of the angle decreases.



cosine similarity for two-dimensional embeddings

Euclidean distance and the **dot product** are also both commonly used when working with sentence embeddings, just not really with sentence transformers as these are specifically designed with cosine similarity in mind. If you are curious about sentence transformers, feel free to check out our blogpost on that topic here.

As mentioned before, there exists a very wide range of similarity measures such as Manhattan distance, Minkowski distance or Chebyshev distance just to name a few. However, unless you really know what you're doing, I would strongly advice you to just stick to using cosine similarity.

Conclusion

In this blogpost we studied the art of pooling.

👉 We found that sentence embeddings are rather **simple by design** but their **interpretation is not immediately clear**.

👉 We discussed in-depth how sentence embeddings are created via different **pooling methods**.

👉 Then, we found that sentence embeddings are meant to represent the **meaning of the sequence as a whole** and that this information is not necessarily fully present in the individual token embeddings.

👉 Finally, we rounded off our discussion by briefly covering **embedding similarity measures** which are commonly needed when using sentence embeddings.