

Open in app ↗

Sign up

Sign In



Search Medium



Published in Towards Data Science

You have **1** free member-only story left this month. [Sign up for Medium and get an extra one](#)



Saketh Kotamraju

Follow

Jun 23, 2022 · 7 min read · ✨ · 🎧 Listen



Save



An Intuitive Explanation of Sentence-BERT

An intuitive explanation of sentence embeddings using the Siamese BERT (SBERT) network and how to code it




70



1





In this article, I am going to explain everything you need to know about the underlying mechanics behind the Sentence-BERT model. I will also detail how its unique architecture is used for specific, unique tasks like semantic similarity search, and how you can code this.

Nowadays, many of the most widely used transformers for a plethora of NLP tasks, like question answering, language modeling, and summarization are word-level transformers. For example, transformers like BERT, RoBERTa, and the original transformer solve many different tasks by computing word-level embeddings, and they were also trained to perform tasks like masked language modeling. However, for a task like semantic search, which requires a strong sentence-level understanding, using just word-level transformers becomes computationally unfeasible.

Semantic search is a task that involves finding the sentences that are similar to a target/given sentence in meaning. In a paragraph of 10 sentences, for example, a semantic search model would return the top k sentence pairs that are the closest in meaning with each other. Using transformers like BERT would require that both sentences are fed into the network, and when we compare a large number of sentences (several hundred/thousand) the computations required makes BERT an unfeasible choice (it would take several hours of training).

This paper aims to overcome this challenge through Sentence-BERT (SBERT): a modification of the standard pretrained BERT network that uses siamese and triplet networks to create sentence embeddings for each sentence that can then be compared using a cosine-similarity, making semantic search for a large number of sentences feasible (only requiring a few seconds of training time).

What does BERT do?

BERT solves semantic search in a pairwise fashion. It uses a cross-encoder: 2 sentences are passed to BERT and a similarity score is computed. However, when the number of sentences being compared exceeds hundreds/thousands of sentences, this would result in a total of $(n)(n-1)/2$ computations being done (we are essentially comparing every sentence with every other sentence; i.e, a brute force search). On a modern V100

GPU, this would require several hours of training time (65 hours to be specific). Here is a diagram representing the cross-encoder network of BERT:

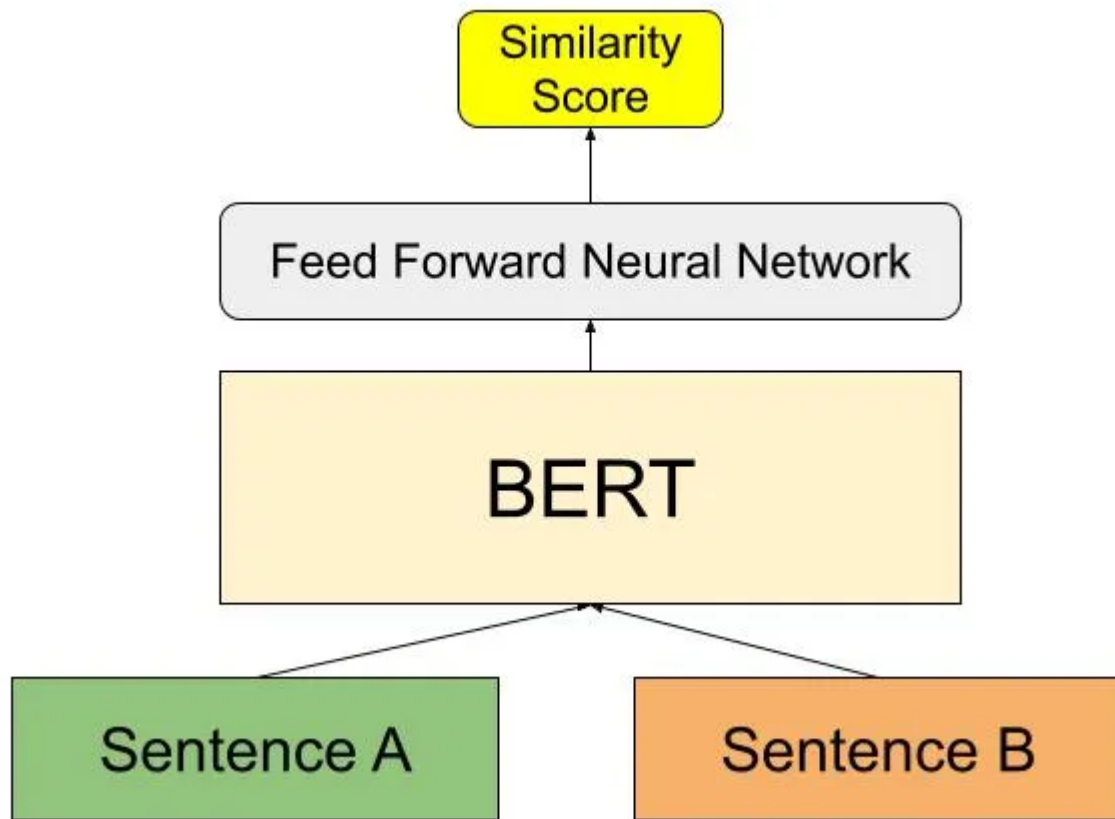


Image by Author

The BERT cross-encoder consists of a standard BERT model that takes in as input the two sentences, A and B, separated by a [SEP] token. On top of the BERT is a feedforward layer that outputs a similarity score.

To overcome this problem, researchers had tried to use BERT to create sentence embeddings. The most common way was to input individual sentences to BERT — and remember that BERT computes word-level embeddings, so each word in the sentence would have its own embedding. After the sentences were inputted to BERT, because of BERT's word-level embeddings, the most common way to generate a sentence embedding was by averaging all the word-level embeddings or by using the output of the first token (i.e. the [CLS] token). However, this method often resulted in bad sentence embeddings, often averaging worse than averaged GLoVe embeddings.

What are Siamese Networks?

Siamese neural networks are special networks that contain two (or more) identical subnetworks/models, where the (usually) two models share/have the same parameters/weights. Parameter updating is mirrored across both sub-models. Siamese networks are most commonly used to compute similarity scores of the inputs, so they have many applications.

One reason why Siamese networks are so powerful and versatile is that they are trained primarily by computing similarities between two things (often images, but in our case, sentences). Because of this, adding new classes (for classification tasks) or comparing two things (two types of images, for example) where one item has a classification not present during the training becomes easy to do. Here is a diagram representing the architecture of siamese networks. The model uses the siamese networks/shared weights/2 CNN's to compute encodings for both images, and then uses a similarity function and an activation function to compute the similarity between both encodings.



So how exactly do you train Siamese Networks?

You may be thinking that, if siamese networks are also trained in a pairwise fashion, shouldn't they also be very computationally inefficient?

This is incorrect. While Siamese networks are trained in a pairwise fashion, they are not trained with every possible pair combinations in the group of items like BERT did. Essentially, the SBERT network uses a concept called triplet loss to train its siamese

architecture. For the sake of explaining triplet loss, I will use the example of comparing/finding similar images. Firstly, the network randomly chooses an anchor image. After this, it finds one positive pair (another image belonging to the same class) and one negative pair (another image belonging to a different class). Then, the network calculates the similarity score between the 2 images for each pair using a similarity score (this varies based on what function you are using). Using these 2 similarity scores, the network will then calculate a loss score, and then update its weights/run backpropagation based on this loss. All the images in the dataset are assigned in/to these triplet groups, so the network is able to be trained on every image in the dataset, without having to run a computationally inefficient brute force/be trained on every combination/pair of images.

The authors in the paper for SBERT use something similar to this, but not the exact mechanism. After all, unlike image classification, in semantic search, sentences do not have a class that they can be associated with. The authors use a unique technique involving finding and minimizing the euclidian distance between sentences and using this metric to find what sentence pairs would be considered positive and what pairs would be considered negative. You can read more about it in their paper [here](#).

What does SBERT do and how does it work?

If you look at the original cross-encoder architecture of BERT, SBERT is similar to this but removes the final classification head. Unlike BERT, SBERT uses a siamese architecture (as I explained above), where it contains 2 BERT architectures that are essentially identical and share the same weights, and SBERT processes 2 sentences as pairs during training.

Let's say that we feed sentence A to BERT A and sentence B to BERT B in SBERT. Each BERT outputs pooled sentence embeddings. While the original research paper tried several pooling methods, they found mean-pooling was the best approach. Pooling is a technique for generalizing features in a network, and in this case, mean pooling works by averaging groups of features in the BERT.

After the pooling is done, we now have 2 embeddings: 1 for sentence A and 1 for sentence B. When the model is training, SBERT concatenates the 2 embeddings which will then run through a softmax classifier and be trained using a softmax-loss function. At inference — or when the model actually begins predicting — the two embeddings

are then compared using a cosine similarity function, which will output a similarity score for the two sentences. Here is a diagram for SBERT when it is fine-tuned and at inference.

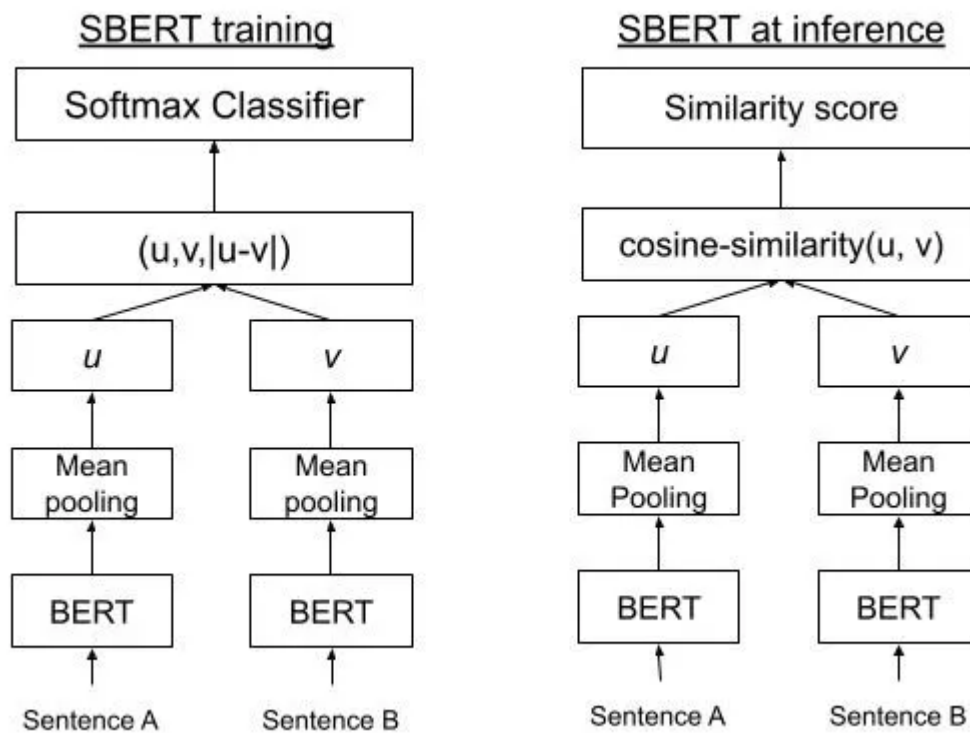


Image by Author

How can you use SBERT?

SBERT has its own Python library that you can find [here](#). Using it is as simple as using a model from the hugging face transformer library. For example, to compute sentence embeddings, you can do this:

```
from sentence_transformers import SentenceTransformer
model = SentenceTransformer('paraphrase-MiniLM-L6-v2')

sentences = ['The quick brown fox jumps over the lazy dog',
             'Dogs are a popular household pet around the world']

embeddings = model.encode(sentences)
for embedding in embeddings:
    print(embedding)
```

Here, I used the 'paraphrase-MiniLM-L6-v2' pretrained model, but there are many other pretrained models you can use that you can find [here](#).

Using SBERT to compute sentence similarity is also easy. Here is how you can compute/find the highest similarity scores between sentences in a list of sentences.

```
from sentence_transformers import SentenceTransformer, util
model = SentenceTransformer('all-MiniLM-L6-v2')

sentences = ['The cat sits outside',
             'A man is playing guitar',
             'I love pasta',
             'The new movie is awesome',
             'The cat plays in the garden',
             'A woman watches TV',
             'The new movie is so great',
             'Do you like pizza?']

#encode the sentences
embeddings = model.encode(sentences, convert_to_tensor=True)

#compute the similarity scores
cosine_scores = util.cos_sim(embeddings, embeddings)

#compute/find the highest similarity scores
pairs = []
for i in range(len(cosine_scores)-1):
    for j in range(i+1, len(cosine_scores)):
        pairs.append({'index': [i, j], 'score': cosine_scores[i]
                                                              [j]})

#sort the scores in decreasing order
pairs = sorted(pairs, key=lambda x: x['score'], reverse=True)

for pair in pairs[0:10]:
    i, j = pair['index']
    print("{} \t\t {} \t\t Score: {:.4f}".format(sentences[i],
                                                  sentences[j], pair['score']))
```

These code snippets were taken from the SBERT library website. You can explore more [here](#). The code above outputs:

```
The new movie is awesome The new movie is so great    Score: 0.8939
The cat sits outside    The cat plays in the garden    Score: 0.6788
I love pasta    Do you like pizza?    Score: 0.5096
I love pasta    The new movie is so great    Score: 0.2560
```



```
I love pasta      The new movie is awesome      Score: 0.2440
A man is playing guitar The cat plays in the garden      Score: 0.2105
The new movie is awesome      Do you like pizza?      Score: 0.1969
The new movie is so great      Do you like pizza?      Score: 0.1692
The cat sits outside      A woman watches TV      Score: 0.1310
The cat plays in the garden      Do you like pizza?      Score: 0.0900
```

. . .

I hope you found this content easy to understand. If you think that I need to elaborate further or clarify anything, drop a comment below.

References

Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks:

<https://aclanthology.org/D19-1410.pdf>

SBERT library:

<https://www.sbert.net/#:~:text=SentenceTransformers%20is%20a%20Python%20framework,for%20more%20than%20100%20languages>

NLP

AI

Bert

Transformers

Machine Learning

Sign up for The Variable

By Towards Data Science

Every Thursday, the Variable delivers the very best of Towards Data Science: from hands-on tutorials and cutting-edge research to original features you don't want to miss. [Take a look.](#)

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.



Get this newsletter



[About](#) [Help](#) [Terms](#) [Privacy](#)

Get the Medium app



Download on the
App Store



GET IT ON
Google Play