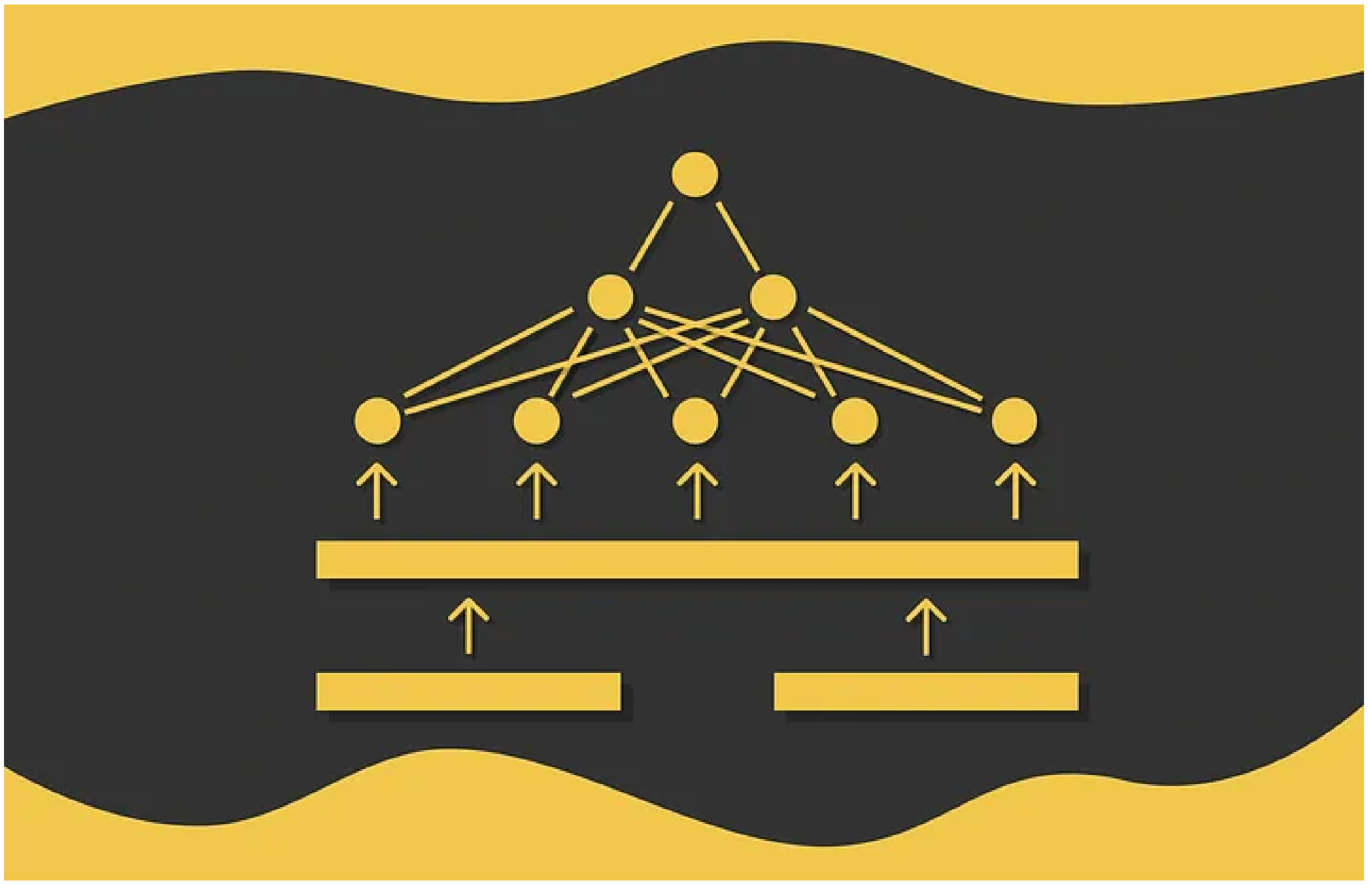
Search Medium Write

Large Language Models: SBERT — Sentence-BERT

Learn how siamese BERT networks accurately transform sentences into embeddings

Vyacheslav Efimov · Following
Published in Towards Data Science · 8 min read · Sep 12

119



Introduction

It is no secret that **transformers** made evolutionary progress in NLP. Based on transformers, many other machine learning models have evolved. One of them is **BERT** which primarily consists of several stacked transformer **encoders**. Apart from being used for a set of different problems like sentiment analysis or question answering, BERT became increasingly popular for constructing word **embeddings** — vectors of numbers representing semantic meanings of words.

Representing words in the form of embeddings gave a huge advantage as machine learning algorithms cannot work with raw texts but can operate on vectors of vectors. This allows comparing different words by their similarity by using a standard metric like Euclidean or cosine distance.

The problem is that, in practice, we often need to construct embeddings not for single words but instead for whole sentences. However, the basic BERT version builds embeddings only on the word level. Due to this, several BERT-like approaches were later developed to solve this problem which will be discussed in this article. By progressively discussing them, we will then reach to the state-of-the-art model called **SBERT**.

> *For getting a deep understanding of how SBERT works under the hood, it is recommended that you are already familiar with BERT. If not, the previous part of this article series explains it in detail.*

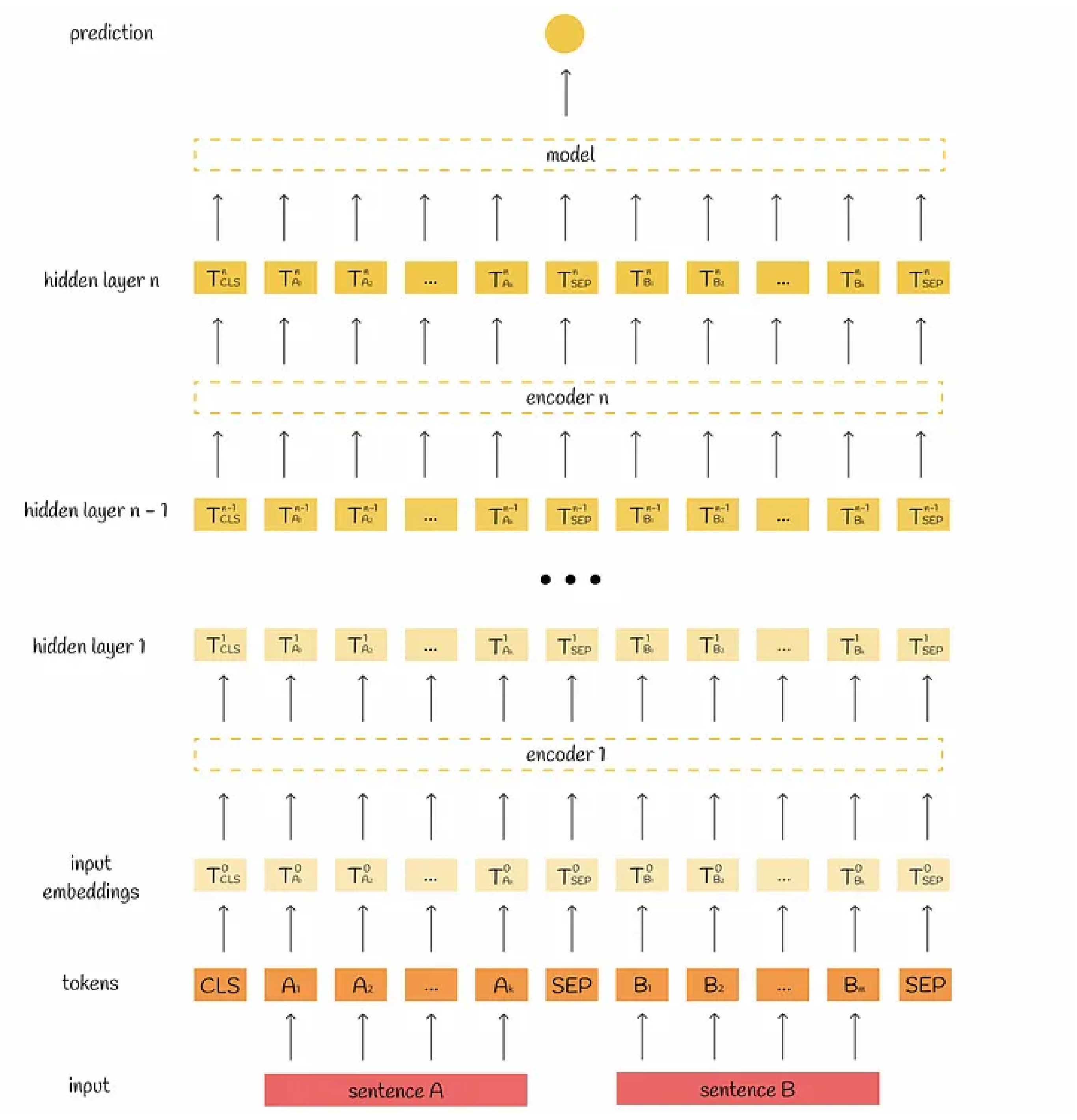
Large Language Models: BERT

Understand how BERT constructs state-of-the-art embeddings

towardsdatascience.com

BERT

First of all, let us remind how BERT processes information. As an input, it takes a *[CLS]* token and two sentences separated by a special *[SEP]* token. Depending on the model configuration, this information is processed 12 or 24 times by multi-head attention blocks. The output is then aggregated and passed to a simple regression model to get the final label.



BERT architecture

For more information on BERT inner workings, you can refer to the previous part of this article series:

Cross-encoder architecture

It is possible to use BERT for calculation of similarity between a pair of documents. Consider the objective of finding the most similar pair of sentences in a large collection. To solve this problem, each possible pair is put inside the BERT model. This leads to quadratic complexity during inference. For instance, dealing with *n = 10 000* sentences requires *n * (n — 1) / 2 = 49 995 000* inference BERT computations which is not really scalable.

Other approaches

Analysing the inefficiency of cross-encoder architecture, it seems logical to precompute embeddings independently for each sentence. After that, we can directly compute the chosen distance metric on all pairs of documents which is much faster than feeding a quadratic number of pairs of sentences to BERT.

Unfortunately, this approach is not possible with BERT: the core problem of BERT is that every time two sentences are passed and processed simultaneously making it difficult to get embeddings that would independently represent only a single sentence.

Researchers tried to eliminate this issue by using the output of the *[CLS]* token embedding hoping that it would contain enough information to represent a sentence. However, the *[CLS]* turned out to not be useful at all for this task simply because it was initially pre-trained in BERT for next sentence prediction.

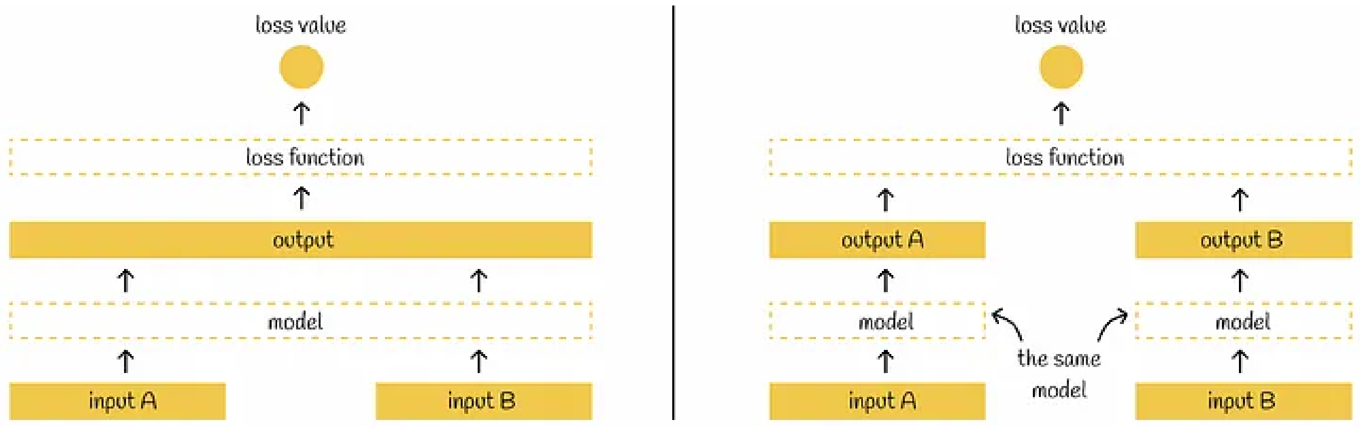
Another approach was to pass a single sentence to BERT and then averaging the output token embeddings. However, the obtained results were even worse than simply averaging GLoVe embeddings.

Deriving independent sentence embeddings is one of the main problems of BERT. To alleviate this aspect, SBERT was developed.

SBERT

SBERT introduces the **Siamese network** concept meaning that each time two sentences are *passed independently* through the same BERT model. Before discussing SBERT architecture, let us refer to a subtle note on Siamese networks:

> *Most of the time in scientific papers, a siamese network architecture is depicted with several models receiving so many inputs. In reality, it can be thought of a single model with the same configuration and weights shared across several parallel inputs. Whenever model weights are updated for a single input, they are equally updated for other inputs as well.*



Non-Siamese (cross-encoder) architecture is shown on left, and the Siamese (bi-encoder) architecture is on the right. The principal difference is that on the left the model accepts both inputs at the same time. On the right, the model accepts both inputs in parallel, so both outputs are not dependent on each other.

Getting back to SBERT, after passing a sentence through BERT, a pooling layer is applied to BERT embeddings to get their lower dimensionality representation: initial 512 768-dimensional vectors are transformed to a single 768-dimensional vector. For the pooling layer, SBERT authors propose choosing a mean-pooling layer as a default one, though they also mention that is possible to use the max-pooling strategy or simply to take the output of the *[CLS]* token instead.

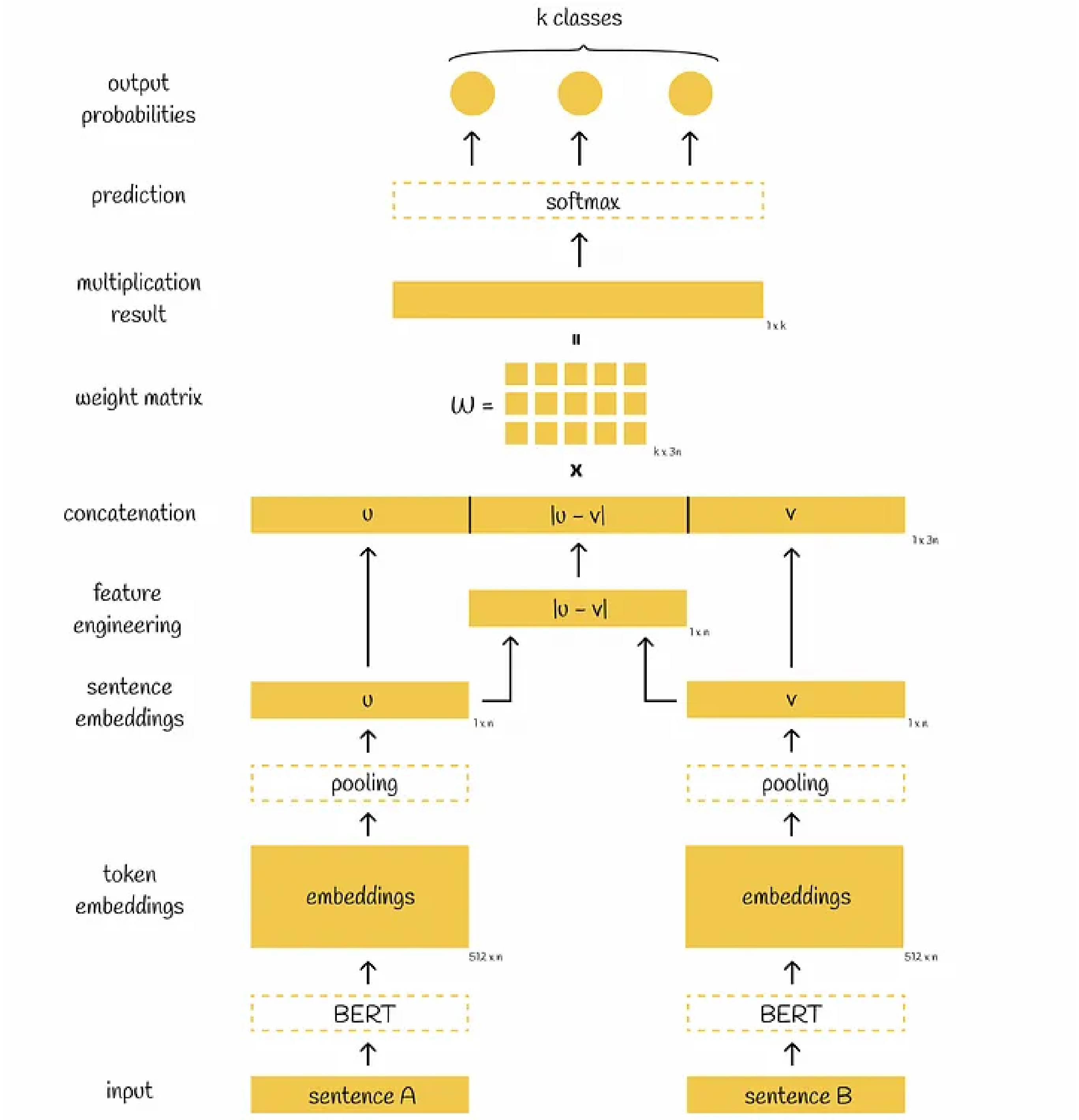
When both sentences are passed through pooling layers, we have two 768-dimensional vectors *u* and *v*. By using these two vectors, authors propose three approaches for optimising different objectives which are going to be discussed below.

Classification objective function

The goal of this problem is to correctly classify a given pair of sentences in one of several classes.

After the generation of embeddings *u* and *v*, the researchers found it useful to generate another vector derived from these two as the element-wise absolute difference |*u*-*v*|. They also tried other feature engineering techniques but this one showed the best results.

Finally, three vectors *u*, *v* and |*u*-*v*| are concatenated, multiplied by a trainable weight matrix *W* and the multiplication result is fed into the softmax classifier which outputs normalised probabilities of sentences corresponding to different classes. The cross-entropy loss function is used to update the weights of the model.



SBERT architecture for classification objective. Parameter n stands for the dimensionality of embeddings (768 by default for BERT base) while k designates the number of labels.

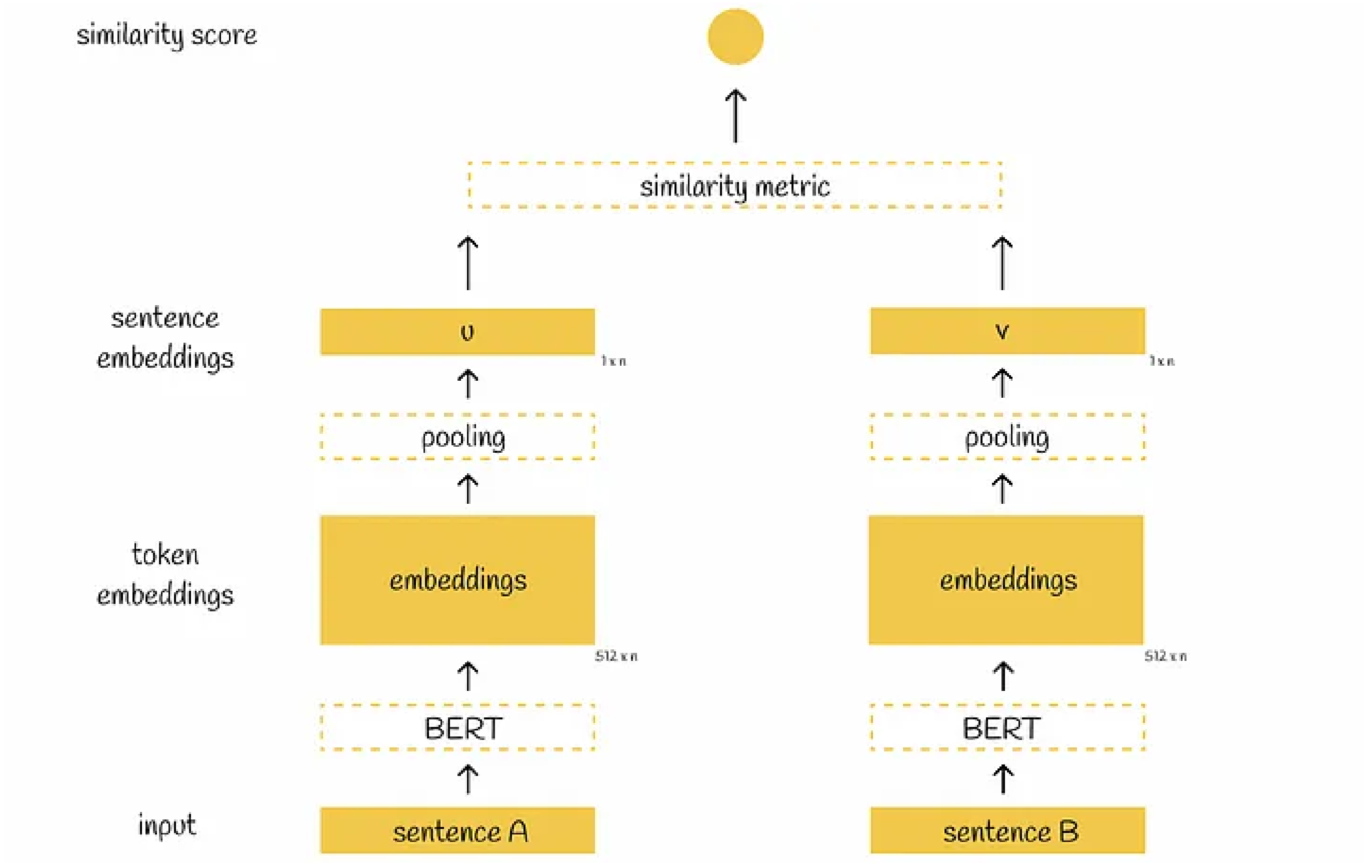
One of the most popular existing problems used to be solved with this objective is **NLI** (Natural Language Inference) where for a given pair of sentences A and B which define hypothesis and premise it is necessary to predict whether the hypothesis is true (*entailment*), false (*contradiction*) or undetermined (*neutral*) given the premise. For this problem, the inference process is the same as for the training.

As stated in the paper, the SBERT model is originally trained on two datasets SNLI and MultiNLI which contain a million sentence pairs with corresponding labels *entailment*, *contradiction* or *neutral*. After that, the paper researchers mention details about SBERT tuning parameters:

> "We fine-tune SBERT with a 3-way softmax-classifier objective function for one epoch. We used a batch-size of 16, Adam optimizer with learning rate 2e−5, and a linear learning rate warm-up over 10% of the training data. Our default pooling strategy is mean."

Regression objective function

In this formulation, after getting vectors u and v, the similarity score between them is directly computed by a chosen similarity metric. The predicted similarity score is compared with the true value and the model is updated by using the MSE loss function. By default, authors choose cosine similarity as the similarity metric.



SBERT architecture for regression objective. Parameter n stands for the dimensionality of embeddings (768 by default for BERT base).

During inference, this architecture can be used in one of two ways:

- By a given sentence pair, it is possible to calculate the similarity score. The inference workflow is absolutely the same as for the training.
- For a given sentence, it is possible to extract its sentence embedding (right after applying the pooling layer) for some later use. This is particularly useful when we are given a large collection of sentences with the objective to calculate pairwise similarity scores between them. By running each sentence through BERT only once, we extract all the necessary sentence embeddings. After that, we can directly calculate the chosen similarity metric between all the vectors (without doubt, it still requires a quadratic number of comparisons but at the same time we avoid quadratic inference computations with BERT as it was before).

Triplet objective function

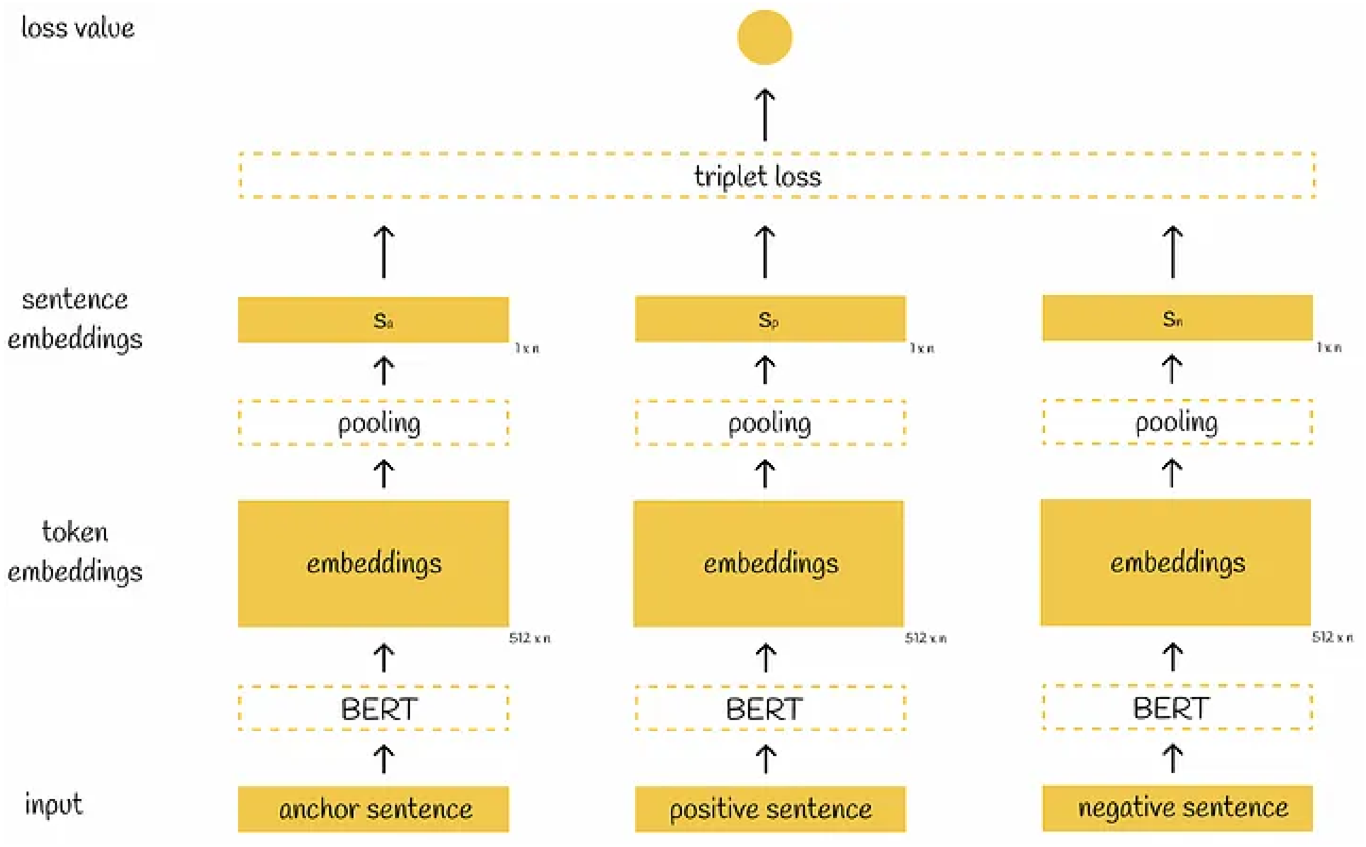
The triplet objective introduces a triplet loss which is calculated on three sentences usually named *anchor, positive* and *negative*. It is assumed that *anchor* and *positive* sentences are very close to each other while *anchor* and *negative* are very different. During the training process, the model evaluates how closer the pair *(anchor, positive)* is, compared to the pair *(anchor, negative)*. Mathematically, the following loss function is minimised:

$$\max(\| s_a - s_p \| - \| s_a - s_n \| + \varepsilon, 0)$$

The triplet loss function from the original paper. Variables s_a, s_p, s_n represent anchor, positive and negative embeddings respectively. Symbol $\|s\|$ is the norm of the vector s. Parameter ε is called margin.

Margin ε ensures that a *positive* sentence is closer to the anchor at least by ε than the *negative* sentence to the *anchor*. Otherwise, the loss becomes greater than 0. By default, in this formula, the authors choose the Euclidean distance as the vector norm and the parameter ε is set to 1.

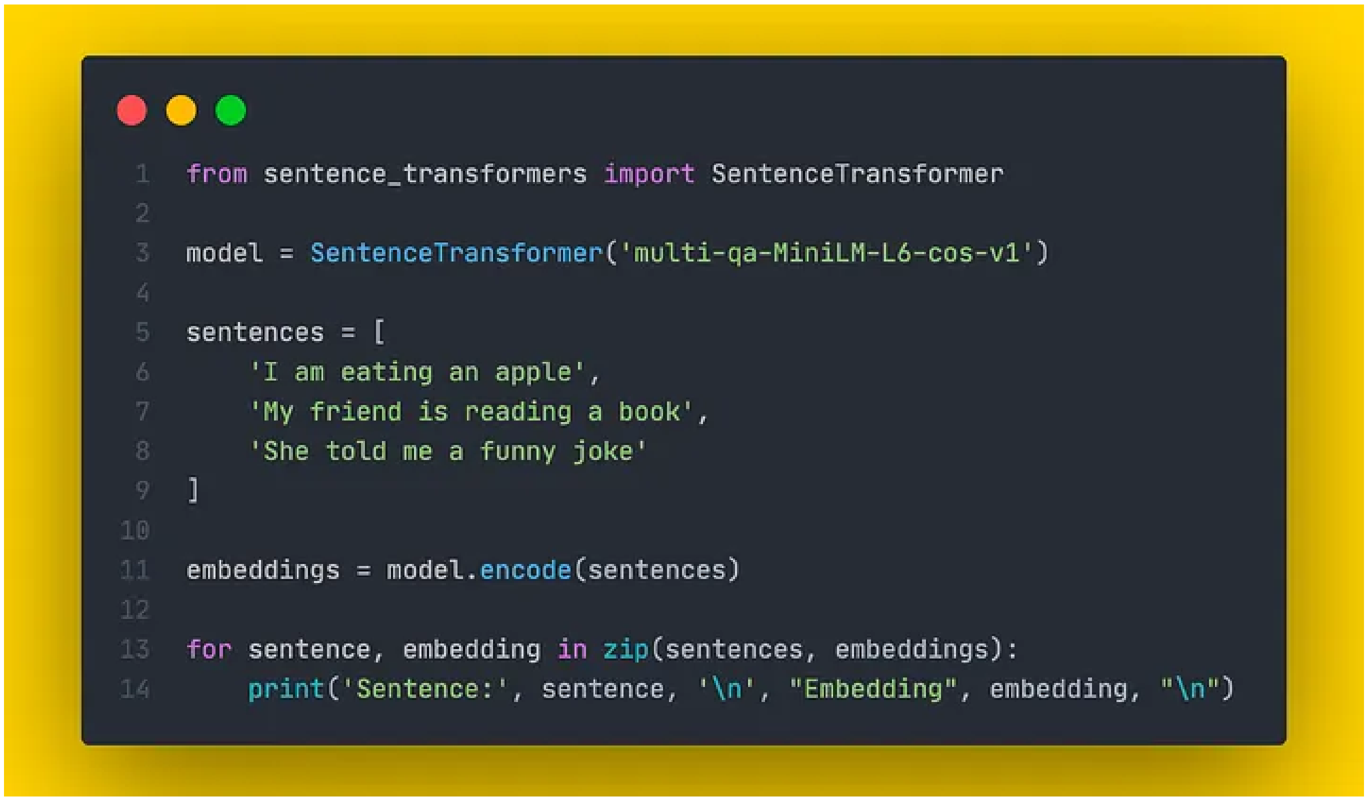
The triplet SBERT architecture differs from the previous two in a way that the model now accepts in parallel three input sentences (instead of two).



SBERT architecture for regression objective. Parameter n stands for the dimensionality of embeddings (768 by default for BERT base).

Code

SentenceTransformers is a state-of-the-art Python library for building sentence embeddings. It contains several pretrained models for different tasks. Building embeddings with SentenceTransformers is simple and an example is shown in the code snippet below.


```python
from sentence_transformers import SentenceTransformer

model = SentenceTransformer('multi-qa-MiniLM-L6-cos-v1')

sentences = [
    'I am eating an apple',
    'My friend is reading a book',
    'She told me a funny joke'
]

embeddings = model.encode(sentences)

for sentence, embedding in zip(sentences, embeddings):
    print('Sentence:', sentence, '\n', "Embedding", embedding, "\n")
```

Constructing embeddings with SentenceTransformers

Constructed embeddings can be then used for similarity comparison. Every model is trained for a certain task, so it is always important to choose an appropriate similarity metric for comparison by referring to the documentation.

Conclusion

We have walked through one of the advanced NLP models for obtaining sentence embeddings. By reducing a quadratic number of BERT inference executions to linear, SBERT achieves a massive growth in speed while maintaining high accuracy.

To finally understand how significant this difference is, it is enough to refer to the example described in the paper where researchers tried to find the most similar pair among $n = 10000$ sentences. On a modern V100 GPU, this procedure took about 65 hours with BERT and only 5 seconds with SBERT! This example demonstrates that SBERT is a huge advancement in NLP.

Resources

- Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks
- SentenceTransformers Documentation | SBERT.net
- Natural Language Inference | Papers with code

All images unless otherwise noted are by the author

Machine Learning NLP Bert Large Language Models Sbert

More from the list: "NLP"

Curated by Himanshu Birla

Jon Gi... in Towards Data ...

Characteristics of Word Embeddings

· 11 min read · Sep 4, 2021

Jon Gi... in Towards Data ...

The Word2vec Hyperparameters

· 6 min read · Sep 3, 2021

Jon Gi... in

The Word2ve

· 15 min rea

View list

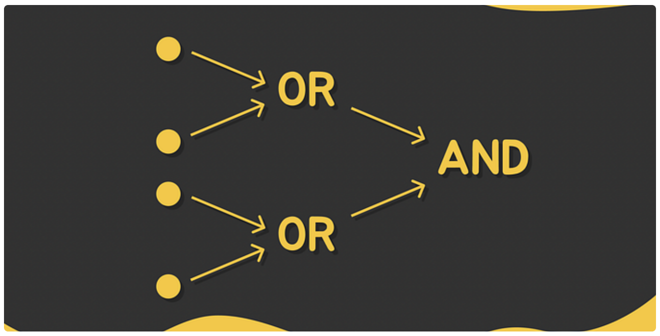
Written by Vyacheslav Efimov

Following

470 Followers · Writer for Towards Data Science

BSc in Software Engineering. Passionate machine learning engineer. Writer at Towards Data Science.

More from Vyacheslav Efimov and Towards Data Science



Vyacheslav Efimov in Towards Data Science

Similarity Search, Part 7: LSH Compositions

Dive into combinations of LSH functions to guarantee a more reliable search

11 min read · Jul 24

43

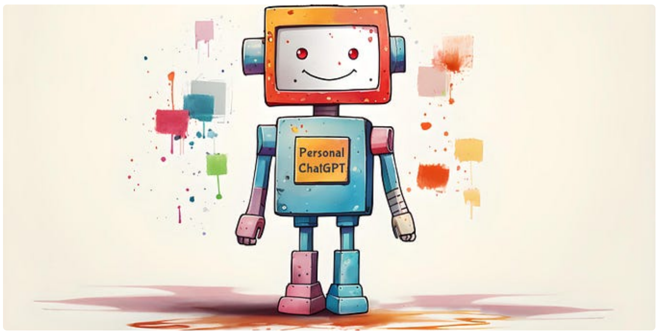
Antonis Makropoulos in Towards Data Science

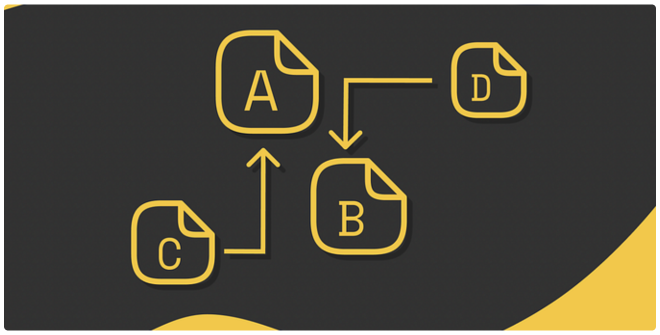
How to Build a Multi-GPU System for Deep Learning in 2023

This story provides a guide on how to build a multi-GPU system for deep learning and...

10 min read · Sep 17

549 11





Robert A. Gonsalves in Towards Data Science

Your Own Personal ChatGPT

How you can fine-tune OpenAI's GPT-3.5 Turbo model to perform new tasks using you...

· 15 min read · Sep 8

595 7

Vyacheslav Efimov in Towards Data Science

Visualised Explanation of PageRank

Discover how Google search engine ranks documents based on their link structure

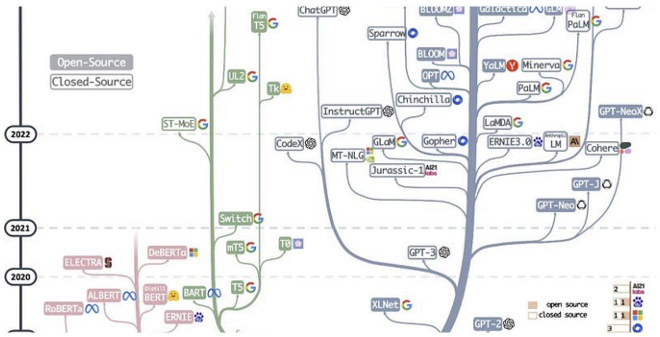
14 min read · Aug 10

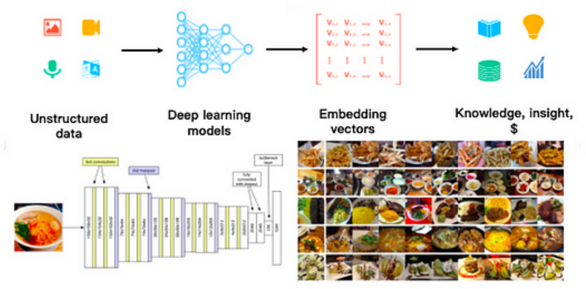
40 1

See all from Vyacheslav Efimov

See all from Towards Data Science

Recommended from Medium





Haifeng Li

A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

15 min read · Sep 14

372

Jayita Bhattacharyya in GoPenAI

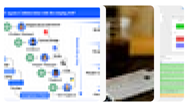
Primer on Vector Databases and Retrieval-Augmented Generation...

Vector Databases Generation (RAG) Langchain Pinecone HuggingFace Large...

9 min read · Aug 16

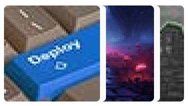
228 1

Lists



Natural Language Processing

669 stories · 283 saves



Predictive Modeling w/ Python

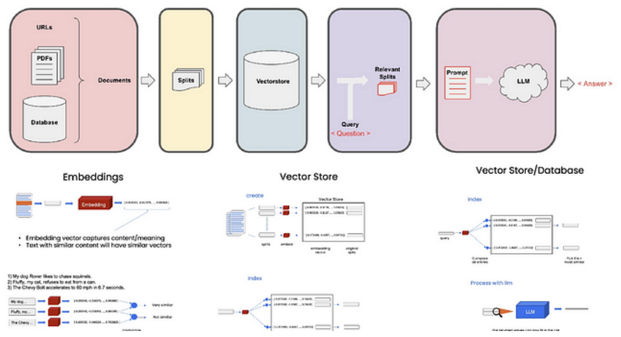
20 stories · 452 saves

Practical Guides to Machine Learning

10 stories · 519 saves

The New Chatbots: ChatGPT, Bard, and Beyond

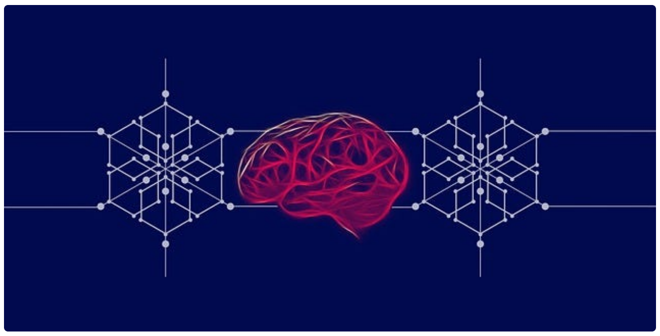
13 stories · 133 saves



TeeTracker

Chat with your PDF (Streamlit Demo)

Conversation with specific files



ai geek (wishesh)

Best Practices for Deploying Large Language Models (LLMs) in...

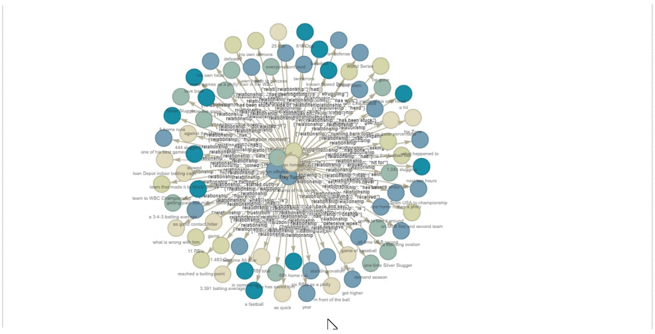
Large Language Models (LLMs) have revolutionized the field of natural language...

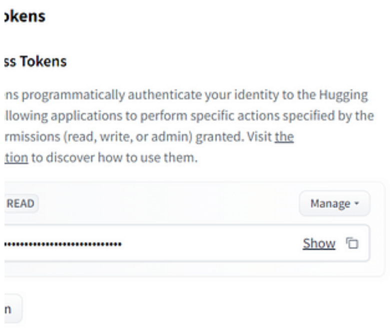
4 min read · Sep 15

56

10 min read · Jun 26

100 1




ɔkens

ss Tokens

ns programmatically authenticate your identity to the Hugging
llowing applications to perform specific actions specified by the
rmissions (read, write, or admin) granted. Visit the
tion to discover how to use them.

READ Manage

Show

Wenqi Glantz in Better Programming

7 Query Strategies for Navigating Knowledge Graphs With…

Exploring NebulaGraph RAG Pipeline with the Philadelphia Phillies

· 17 min read · 4 days ago

501 4

Ankit

Generating Summaries for Large Documents with Llama2 using…

Introduction

11 min read · Aug 28

103 3

See more recommendations