Sentence-BERT
Used in
Calculating Semantic Text Similarity

To Rank Job Applicants

By
Mariam Ahmed Amin

Table of Contents

Abstract

The proposed problem is to find a matching score between the applicant's qualifications and the job's requirements.
The basic idea that the matching is based on, is to get the matching between an atom sentence in the applicant's qualifications to another atom sentence in the job's requirements.
This matching score between the two sentences will be multiplied later to some factors according to the category of these sentences.
So the proposed task we have can be considered to find the semantic similarity between two sentences.

Introduction

We have more than one option to implement our proposed task (Semantic Text Similarity) as it's a core Natural Language Processing task.

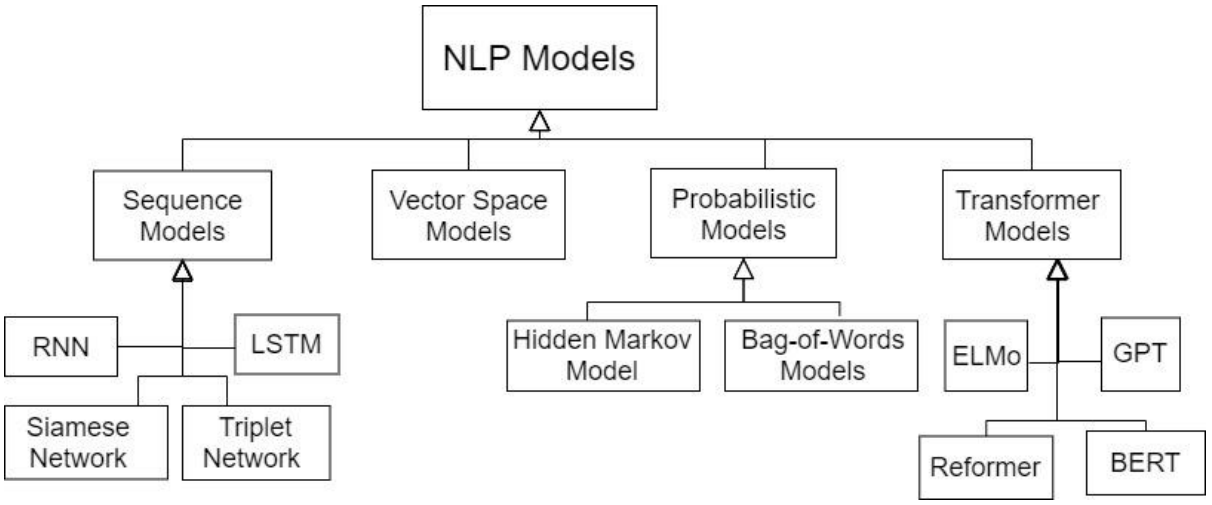


Image source: created by me

Because we process sentences, we prefer to perform **sequence modeling**.

It is popular that sequence models (RNN, LSTM.. etc) are used to process sentences because it does sequence modeling and it processes the words of the sentences sequentially and considers the dependencies not just as if they are many separated words. This is to consider the whole context of the sentence. But unfortunately, sequence models take a long time to train, require much memory and do not allow for parallelization

Therefore, attention mechanisms have become important for sequence modeling in many tasks, as it allows modeling of dependencies without caring about the length of the sequence as it allows for parallelization so the memory and the time utilized during training are reduced. This brings us to prefer *Transformer* models because they basically depend on attention.

Background

Recurrent neural networks (RNN), long short-term memory(LSTM) and gated recurrent unit (GRU) neural networks, have been considered the state of the art in sequence modeling such as language modeling and machine translation.
But unfortunately, the sequential nature of these sequence models does not allow for parallelization within training examples, which becomes a serious problem at longer sequence lengths, as the constraints of the memory limit batching across examples.

And if you process a sentence sequentially, you need to keep track of long-term dependencies in the sentence, then it is not possible to use parallel computing.

Adding to the parallelization problem, there is two other popular issues with RNNs are:

- Loss of information: It is harder to keep track of the specific details of the words as moving forward.
- Vanishing Gradient: when you back-propagate, the gradients can become really small so your model will not learn properly.

Although LSTM eliminated vanishing and exploding gradients that the RNN suffered from, it still takes a long time to train, requires much memory and does not allow for parallelization. In contrast, transformers are based on attention.

Like sequence models, transformers are designed to model sequential input data. However, unlike sequence models, transformers do not require that the sequential data be processed sequentially or in order as it uses parallelization. Using parallelization reduces training time and memory complexity.

Additionally, the gradient steps that need to be taken from the last output to the first input in a transformer is one. For RNNs, the number of steps increases with longer sequences. Transformers don't suffer from the vanishing gradients problem that happens in long sequences.

Sequence modeling	
Challenges with RNNs	Transformer networks
• Long term dependencies • Gradient vanishing and explosion • Much training steps • No parallelization	• Facilitates long term dependencies • No gradient vanishing and explosion • Fewer training steps • No recurrence and that facilitates parallel computations

Table source: Lecture

The move towards transformers led to the development of pretrained systems such as BERT (Bidirectional Encoder Representations from transformers) and GPT (Generative Pre-trained transformer), which have been trained with large language datasets, such as Wikipedia Corpus and Common Crawl(unlabeled data and this is really a turning point), and can be then fine-tuned to specific tasks.

So that will bring us to discuss the transformer models in details.

Discussion

Transformer

Before starting, I strongly encourage you to watch this lecture.

The transformer is a deep learning model adopting the mechanism of attention.
The attention in the transformer is a layer of calculations that let your model focus on the most important tokens of the sequence in each step.
The importance of the tokens is measured by *the ability to define the context* of the sequence. So it's concluded that the transformer selects the tokens that most define the context at each step.

Scaled dot-product attention units are the building blocks of the transformer.
When a sentence is passed into it, attention weights are calculated between every token simultaneously.
The similar tokens have higher dot products and non-similar tokens will have lower dot products.
Then the attention unit produces embeddings for every token in context that contain information about the token along with a weighted combination of other relevant tokens each weighted by its attention weight.

Transformer Architecture:

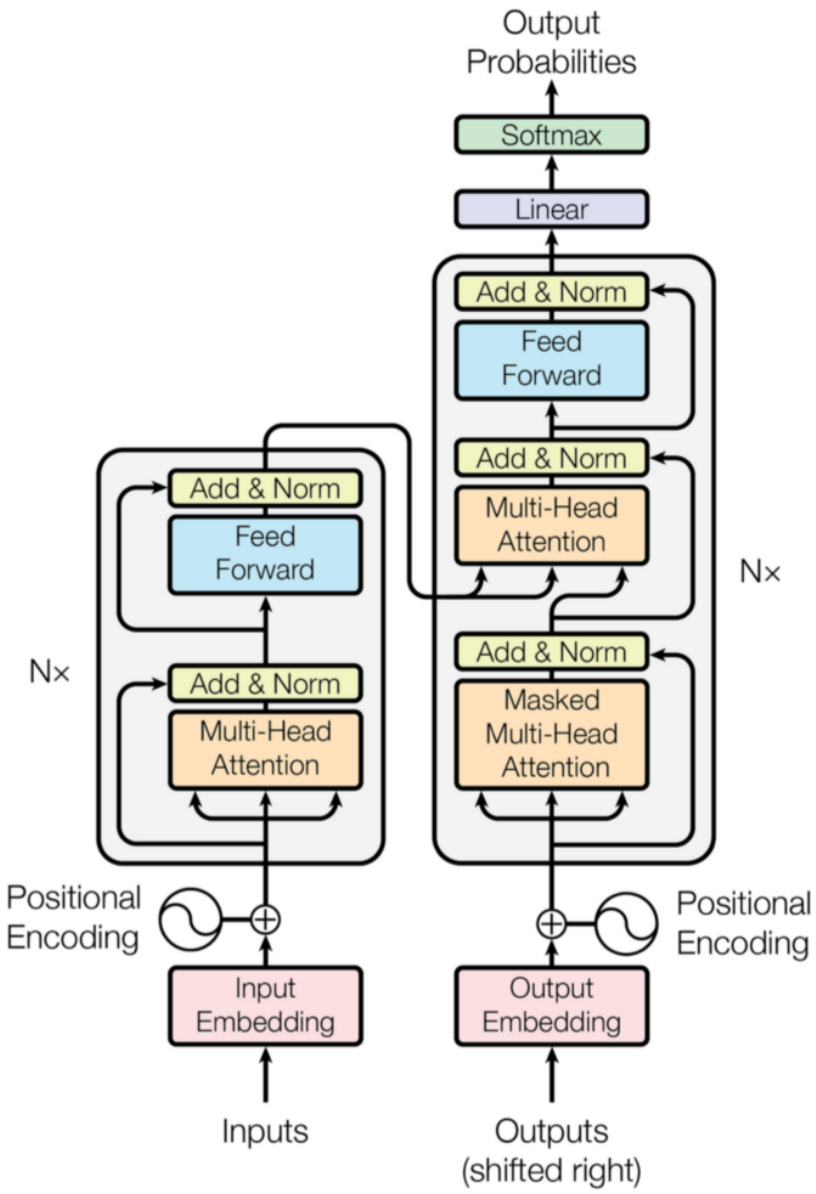


Image source: Paper:Attention Is All You Need

Encoder

The encoder consists of a stack of 6 identical layers. Each layer has two sub-layers. The first is a multi-head self-attention, and the second is a fully connected feed-forward network. We apply a residual connection around each of the two sub-layers, then perform layer normalization. To facilitate these residual connections, all sub-layers in the model and the embedding layers, produce outputs with dimensions = 512.

Decoder

The decoder also consists of a stack of 6 identical layers. In addition to the two sub-layers in each encoder layer, the decoder adds a third sub-layer, which performs multi-head attention over the output of the encoder stack. As done in the encoder, we apply residual connections around each of the sub-layers, followed by layer normalization. The self-attention sub-layer in the decoder stack is modified to ensure that the predictions for position i can depend only on the known outputs at positions less than i.

Multi-Head Attention

Imagine that you are translating English into French. You can consider the word embeddings in the English language the keys and values. The queries will then be the French equivalent. You can then calculate the dot product between the query and the key to obtain the similarity between the English and French sentences. As similar vectors have higher dot products and non-similar vectors will have lower dot products. The intuition here is to identify the corresponding words in the queries that are similar to the keys. This would allow your model to focus on the right tokens when translating each word. We then run a softmax to limitize the output to range [0,1].

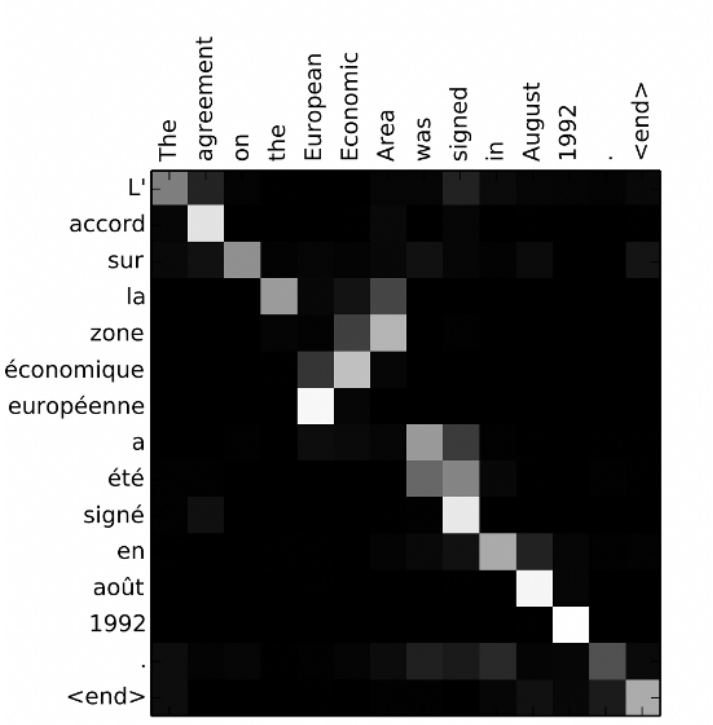
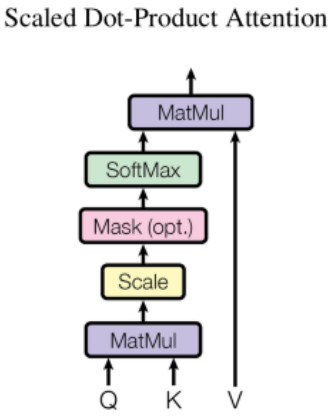


Image source: Link

Scaled dot product attention



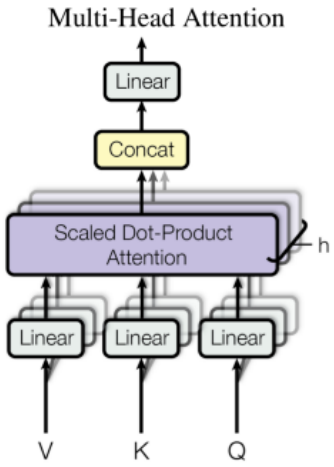


Image source: Paper:Attention Is All You Need

To illustrate more what is attention, let's discuss self-attention.

Self-attention, is an **attention** mechanism relating different positions of a single sequence in order to compute a representation of it.

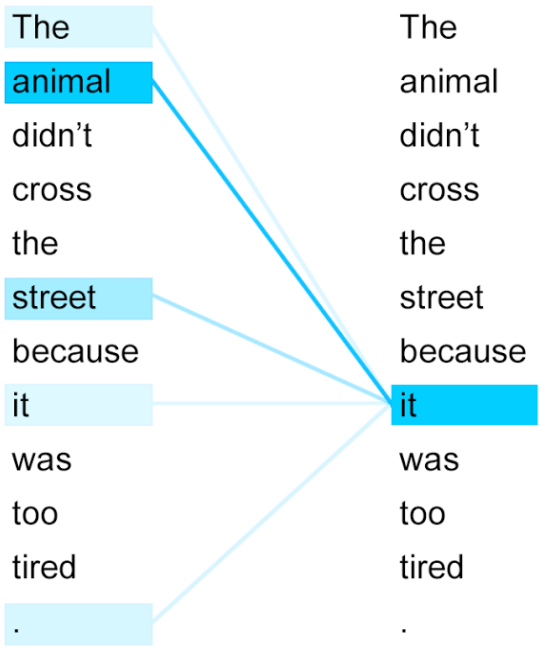


image source Link

The attention formula:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Image source: Link

So when we apply self-attention:

Assume Q is the query matrix (one word in the sequence), K are all the keys (all the words in the sequence) and V (initially set equals Q)

To simplify the above formula, say that the *Attention* is the matmul between *V(values)* and the attention-weights *a*, where the attention-weights “a” are calculated by the queries and keys.

The attention weights are defined by:

$$a = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

Image source: Link

This means that the weights ***a*** represent how each word of the sequence is related by all the other words in the sequence.
The SoftMax function is applied to the attention weights *a* to have an output between 0 and 1 while having their sum = 1.

The attention mechanism is repeated many times with linear projections of Q, K and V. This allows the system to learn from many representations of Q, K and V. These representations are done by multiplying Q, K and V by weight matrices which are learned during the training.

After the multi-attention heads, we have a pointwise feed-forward layer. This feed-forward network has identical parameters for each position, which can be described as a separate, identical linear transformation of each token from the given sequence. Then linear projection then Softmax function are applied to the output of the decoder.

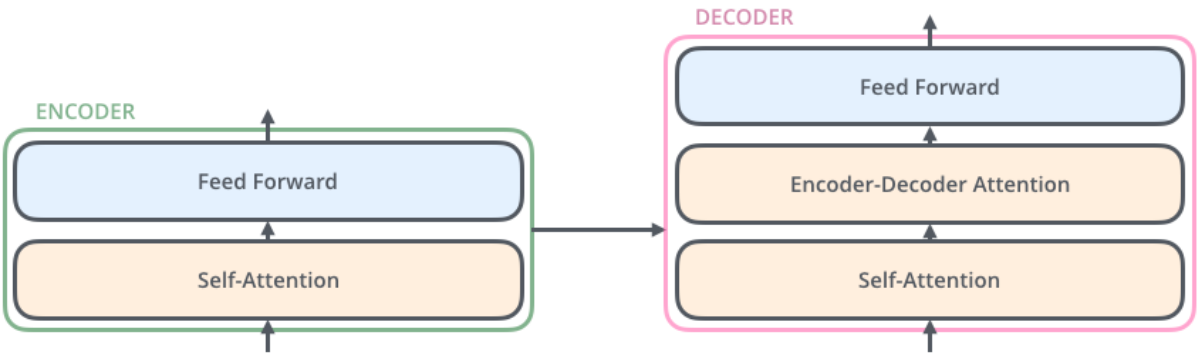


Image source:[Transformer architecture, self-attention | Kaggle]

Now, we can discuss BERT which is a framework using transformers.

BERT (Bidirectional Encoder Representations from transformers)

BERT is a transformer-based machine learning technique developed by Google.

It is an evolution of self-attention and transformer architecture. BERT uses encoder-only transformers. *It's deeply bidirectional*, as it uses both left and right contexts in all layers.

It is a method of training a general-purpose "language understanding" model on a corpus of unlabelled text including the entire Wikipedia(that's 2,500 million words) and Book Corpus (800 million words) and then fine tune that model to be used in some NLP tasks. This is similar to transfer learning.
BERT outperforms previous approaches because it is the first unsupervised, deeply bidirectional system for pre-training NLP.

Why does this matter?
Pre-trained representations can be context-free or contextual, and contextual representations can be unidirectional or bidirectional. Context-free models such

as GloVe or word2vec generate a single word embedding representation for each word in the vocabulary. For example, the word “ball” would have the same context-free representation in “kick the ball” and “go to the ball”.

Contextual models instead generate a representation of each word that is based on the other words in the sentence. For example, in the sentence “kick the ball hard” a unidirectional contextual model would represent “ball” based on “kick the” but not “hard.”
However, BERT represents “ball” using both its previous and next context, which makes it ***bidirectional***.

BERT is pre-trained on two NLP tasks:

- Masked Language Modeling
- Next Sentence Prediction

BERT Pretraining

Masked Language Modeling
Given a sequence where some of its tokens are masked. And we want to predict the masked tokens. Masking allows the model to be trained bidirectionally.

Choose 15% of the tokens at random: mask them with a probability of 80%, replace them with a random token with a probability of 10%, or keep as is with a probability of 10%.

The model then tries to predict the original value of the masked words based on the context provided by the other non-masked tokens in the sequence. It is similar to CBOW architecture in Word2Vec that predicts a word given a context.

Next Sentence Prediction

Given two sentences, the model predicts whether the second one can logically follow the first one.

This task is used for capturing relationships between sentences ***since language modelling doesn't do this***.

In the BERT training process, the model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the following sentence to the first sentence in the original document or not.

During training, the second sentence is the subsequent of the first sentence in the 50% of the data, while in the other 50% a random sentence from the corpus is chosen as the second sentence.

The assumption is that the random sentence can not be considered as a subsequent to the first sentence.

The input sentences are processed in the following way before entering the model to help the model separate them:

- Token [CLS] is inserted at the beginning of the first sentence and token [SEP] is inserted at the end of both two sentences.
- A *segment embedding* is added to each token to indicate whether it is in sentence A or B.
- To let the model detect the order of the token in the sequence, a positional embedding is added to each token to indicate its position in the sequence.

Bert Evaluation

To evaluate BERT performance, we compared it to other state-of-the-art NLP systems. BERT achieved all of its results with almost no task-specific changes to the NN architecture.

On SQuAD v1.1, BERT achieves 93.2% F1 score , surpassing the score of the human-level model (91.2%).

SQuAD1.1 Leaderboard

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar et al. '16)	82.304	91.221
1 Oct 05, 2018	BERT (ensemble) *Google AI Language* https://arxiv.org/abs/1810.04805	**87.433**	**93.160**
2 Sep 09, 2018	nlnet (ensemble) *Microsoft Research Asia*	85.356	91.202
3 Jul 11, 2018	QANet (ensemble) *Google Brain & CMU*	84.454	90.490

Image source Link

To use BERT(Fine-tuning)

BERT can be used for a wide variety of tasks, while only adding a layer.
In the fine-tuning, most hyper-parameters stay the same as in the training.

After much searching, I chose S-BERT to calculate the semantic similarity.

S-BERT is a modification of the pretrained BERT network that is fine tuned using siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity to express the semantic similarity.

This reduced the cost of finding the most similar pair from 65 hours with BERT / RoBERTa to about only 5 seconds with SBERT, while maintaining the accuracy.

What are siamese and triplet networks?

- **Siamese Networks:** learns what makes two inputs the same.
  So it can detect that “How old are you?” is the same as “What is your age?” and can also detect that “ Where are you from?” is not equal to “Where are you going?”.
- **Triplet networks:** learns whether two inputs are the same, neutral or opposites.

So adding siamese and triplet networks is a fine tuning step to make the sentence embedding more contextualized, representative and have ais more semantically meaningful.

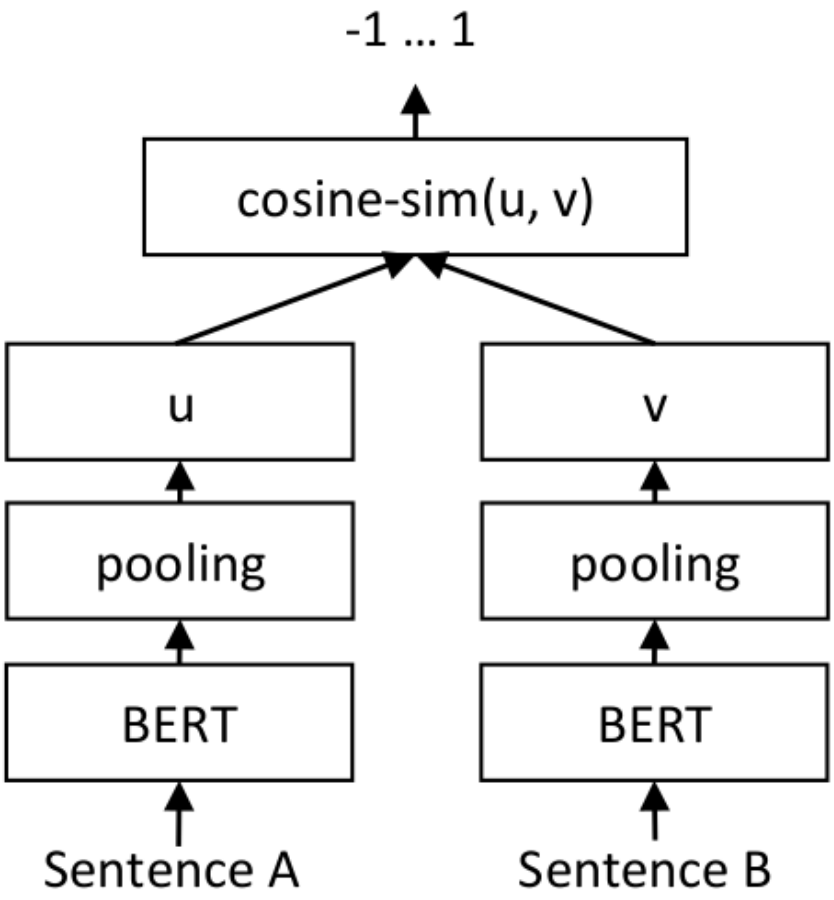


Image source: Paper Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks

In the semantic Similarity task, for each sentence pair, we pass sentence A and sentence B the network which yields the embeddings *u* und *v*.

The similarity between the two sentences is computed by applying cosine similarity function on the two embeddings u and v and the result is compared to the provided gold similarity score.

This allows our network to be fine-tuned and let the model generate semantically meaningful outputs.

Conclusions

SentenceTransformers is a Python framework for state-of-the-art multimedia embeddings(sentence, text and image).

The initial work is described in paper ***Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks***.

SentenceTransformers can be considered as BERT Transformer fine tuned with siamese and triplet networks. This framework can be used to compute representative embeddings for sentences. These embeddings can then be paired to the cosine-similarity function to find the semantic similarity between them (similarity in the meaning).

References

Natural Language Processing | Coursera
CS480/680 Lecture 19: Attention and Transformer Networks - YouTube
How LSTM networks solve the problem of vanishing gradients | by Nir Arbel | DataDrivenInvestor
Understanding of LSTM Networks - GeeksforGeeks
Paper: Attention Is All You Need
Paper: Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks
Transformer (machine learning model) - Wikipedia
Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)
What is BERT | BERT For Text Classification (analyticsvidhya.com)
What is a Transformer?. An Introduction to Transformers and… | by Maxime | Inside Machine learning | Medium
google-research/bert: TensorFlow code and pre-trained models for BERT (github.com)
Google AI Blog: Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing (googleblog.com)
BERT Explained: State of the art language model for NLP | by Rani Horev | Towards Data Science
BERT (Language Model) (devopedia.org)
Semantic Textual Similarity — Sentence-Transformers documentation (sbert.net)
Transformer architecture, self-attention | Kaggle