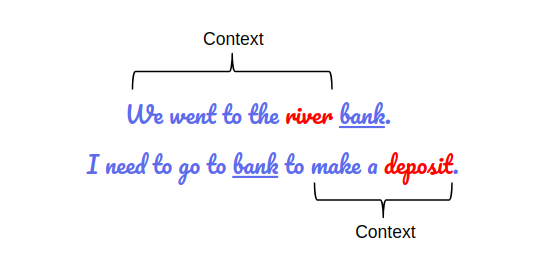
**BERT**

BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. It is designed to pre-train deep bidirectional representations from an unlabeled text by jointly conditioning on both left and right contexts. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks.

* It’s easy to get that BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Each word here has a meaning. For now, the key takeaway from this line is – **BERT is based on the Transformer architecture.**
* It is pre-trained on a large corpus of unlabeled text including the entire Wikipedia (that’s 2,500 million words!) and Book Corpus (800 million words).
* BERT is a **“deeply bidirectional”** model. Bidirectional means that BERT learns information from both the left and the right sides of a token’s context during the training phase.

The bidirectionality of a model is important for truly understanding the meaning of a language. Let’s see an example to illustrate this. There are two sentences in this example and both of them involve the word “bank”:



*BERT captures both the left and right context*

If we try to predict the nature of the word “bank” by only taking either the left or the right context, then we will be making an error in at least one of the two given examples.

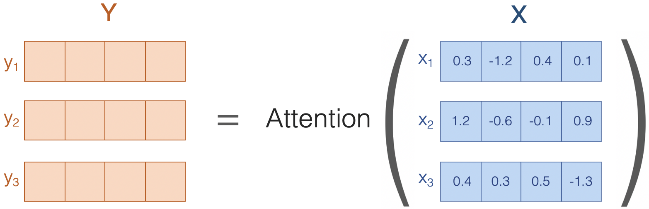
One way to deal with this is to consider both the left and the right context before making a prediction. That’s exactly what BERT does!

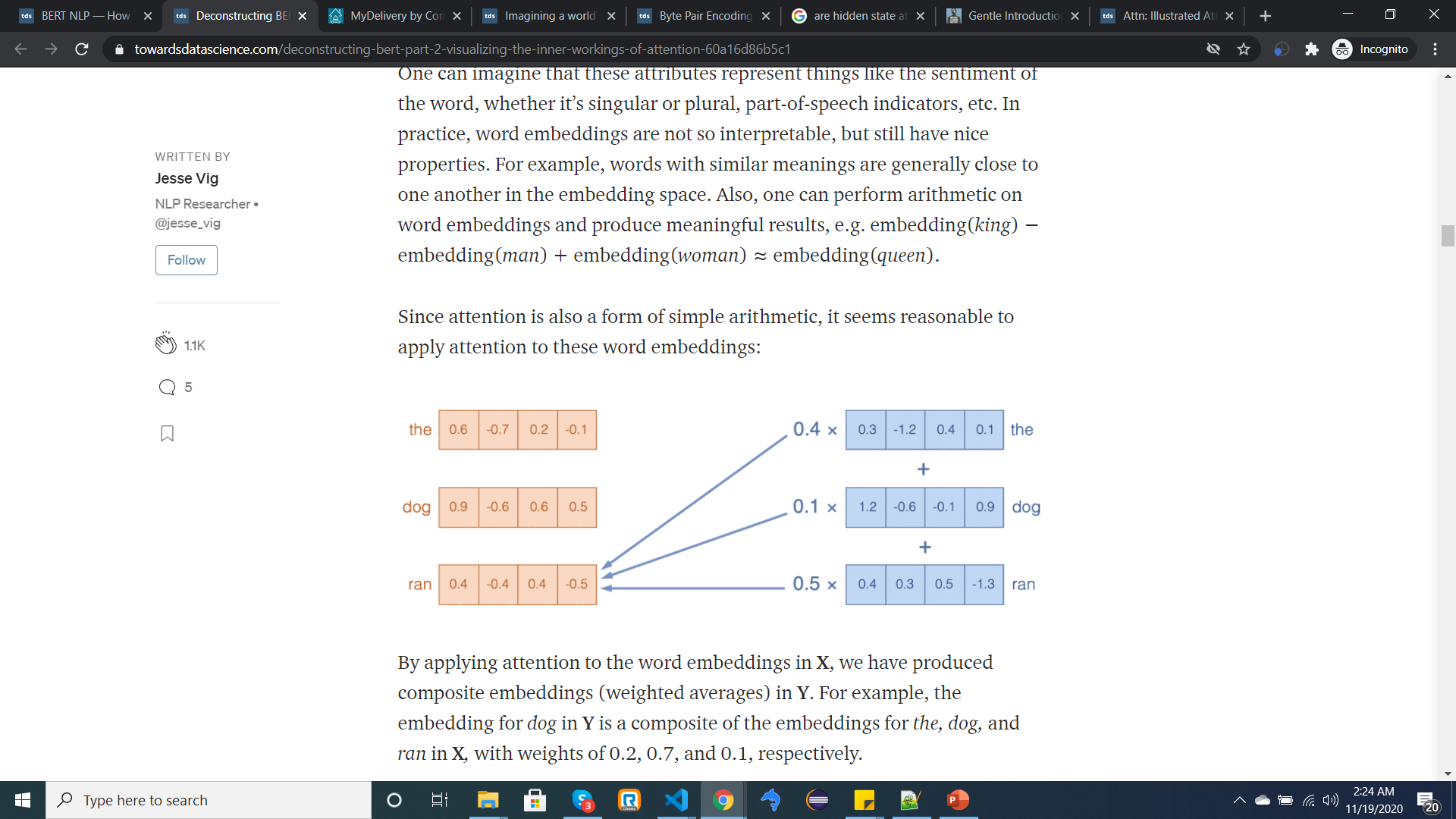
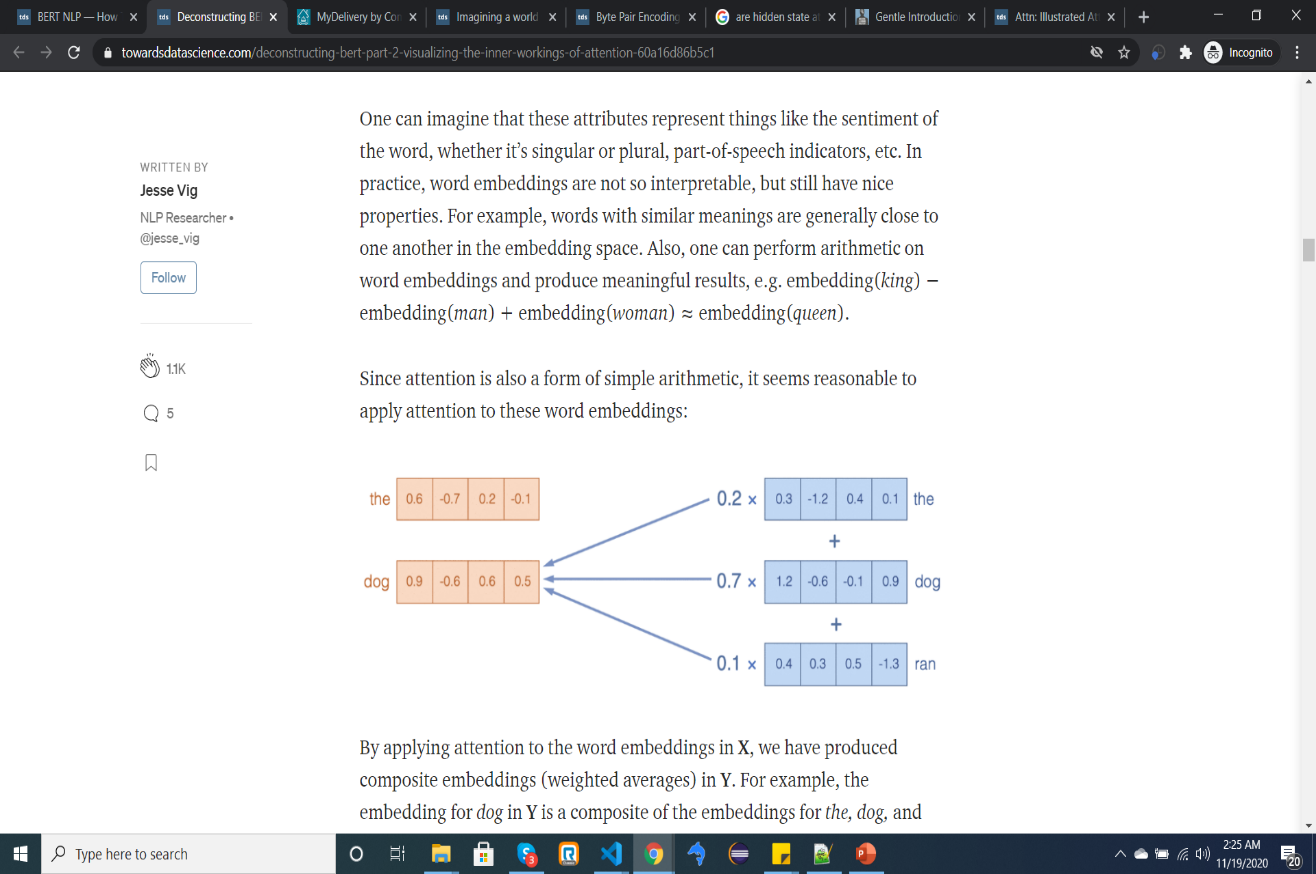
For understanding BERT we will have to understand Transformer**.**

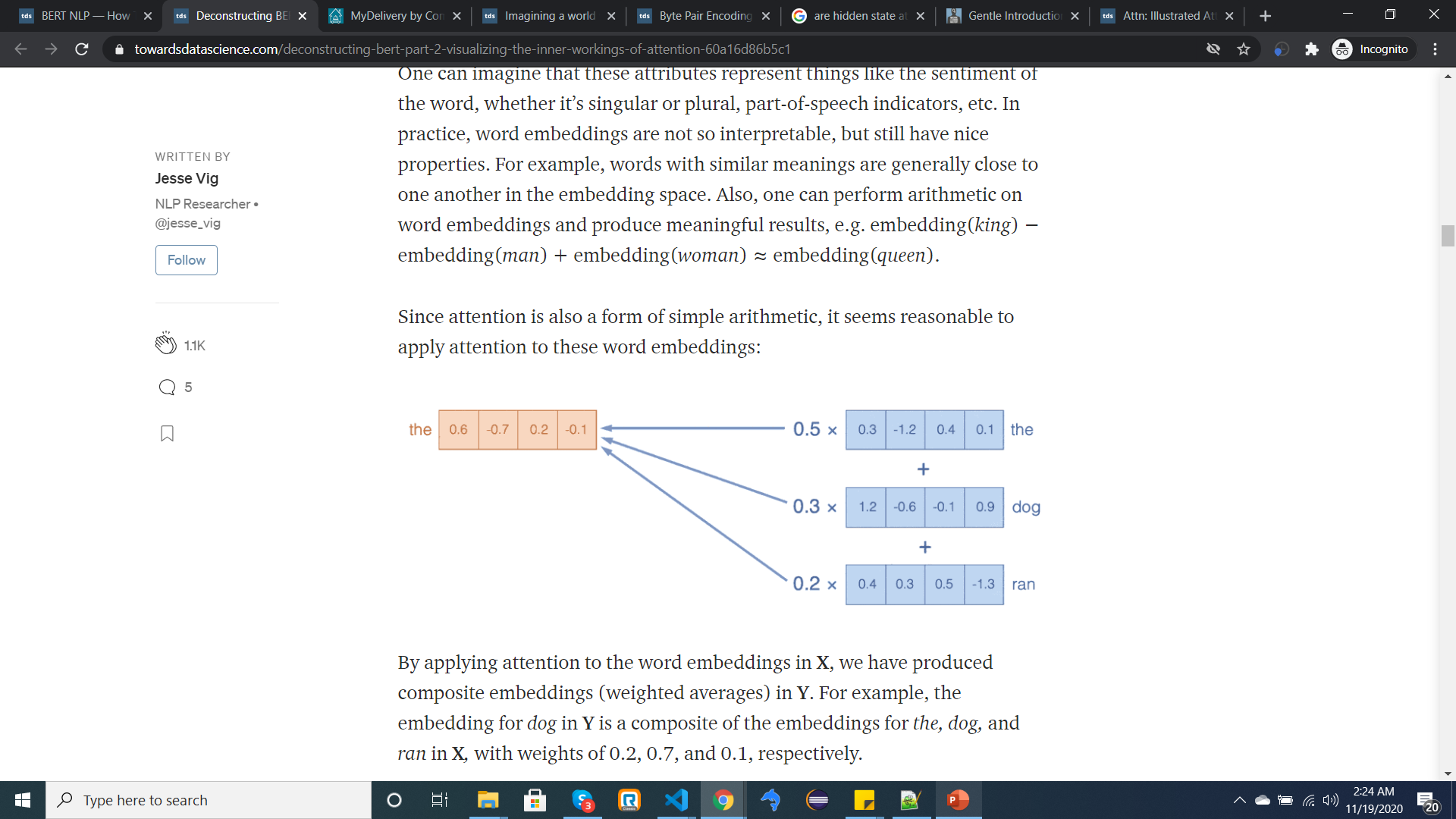
**Transformers**

These have been the latest development to handle the sequential data by implementing the below mechanism

* **Non-sequential**: Sentences are processed as a whole rather than word by word. As We can see that the architecture of the Transformer is such a way that we can send the inputs simultaneously so It does not suffer from the problem we face in LSTM called long-term dependency. **Transformers**do not rely on past hidden states to capture dependencies with previous words, they process a sentence as a whole, a reason why there is no risk to lose (or 'forget') past information.
* **Attention Mechanism**:

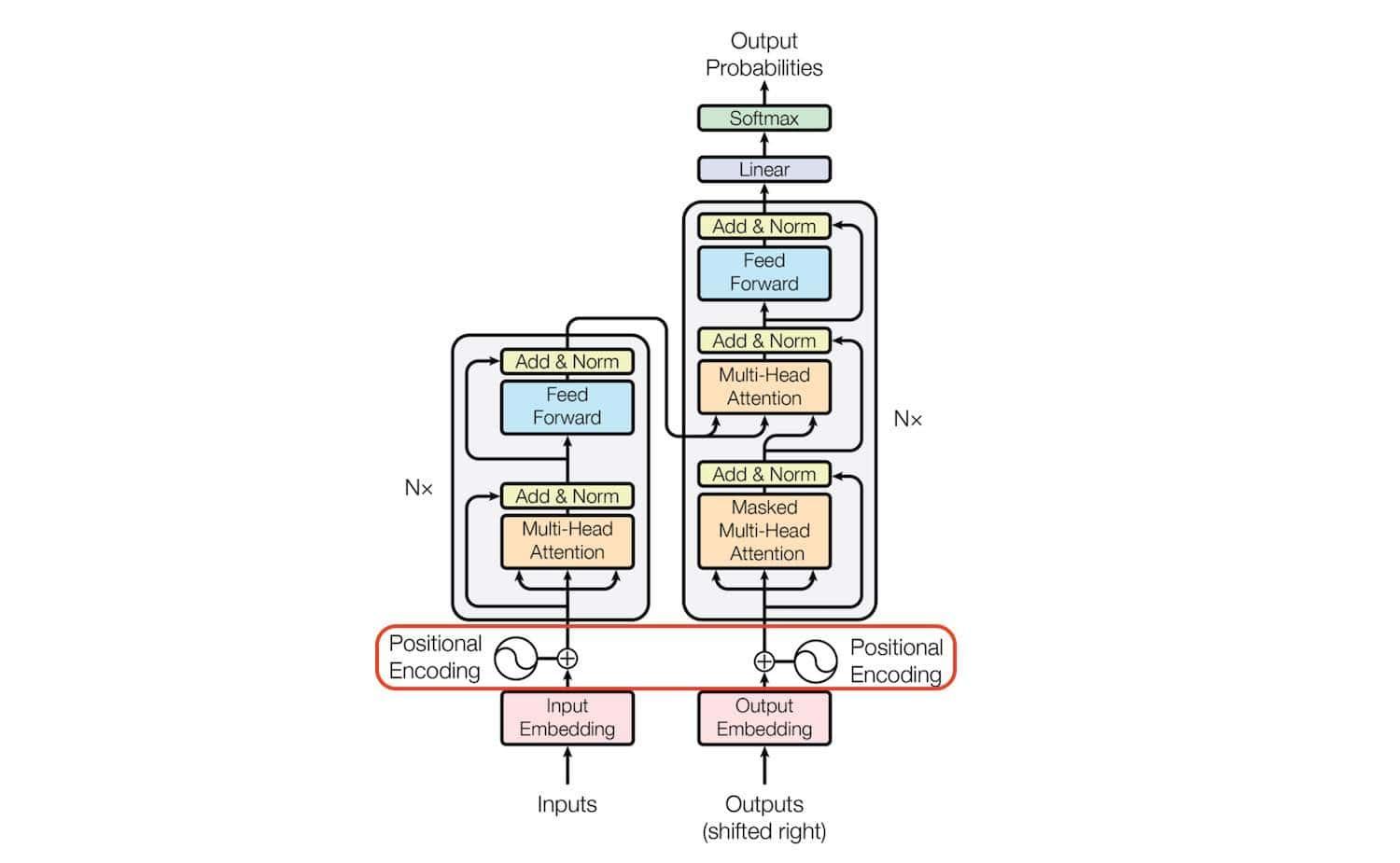
1. Attention is a way for a model to assign weight to input features based on their importance to some task. When deciding whether an image contains a dog or cat, for example, a model might pay more attentionto — i.e. place more weight on — the furry parts of the image as opposed to the lamp or window in the background.
2. *The dog from down the street ran up to me and \_\_\_\_*” may want to pay more attention to the word *dog*than *street*
3. Suppose you have some sequence **X**, where each element is a vector (referred to as a *value*). In the following example, **X** consists of 3 vectors, each of length 4
4. Attention is simply a function that takes **X**as input and returns another sequence **Y** of the same length, composed of vectors of the same length of those in **X**:

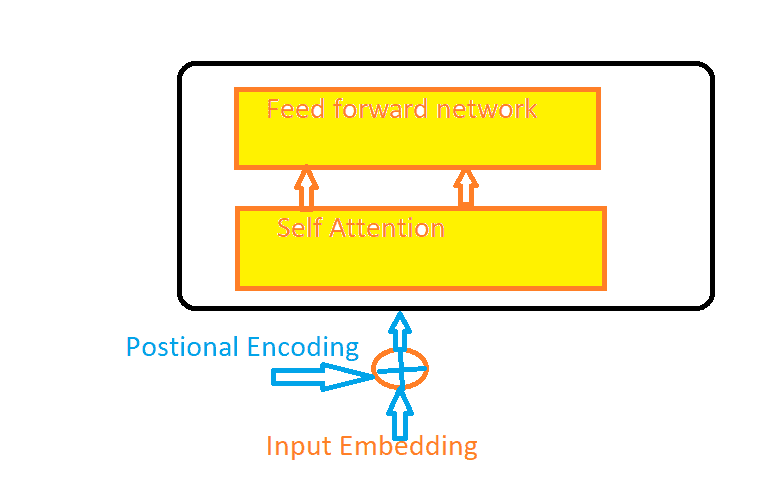
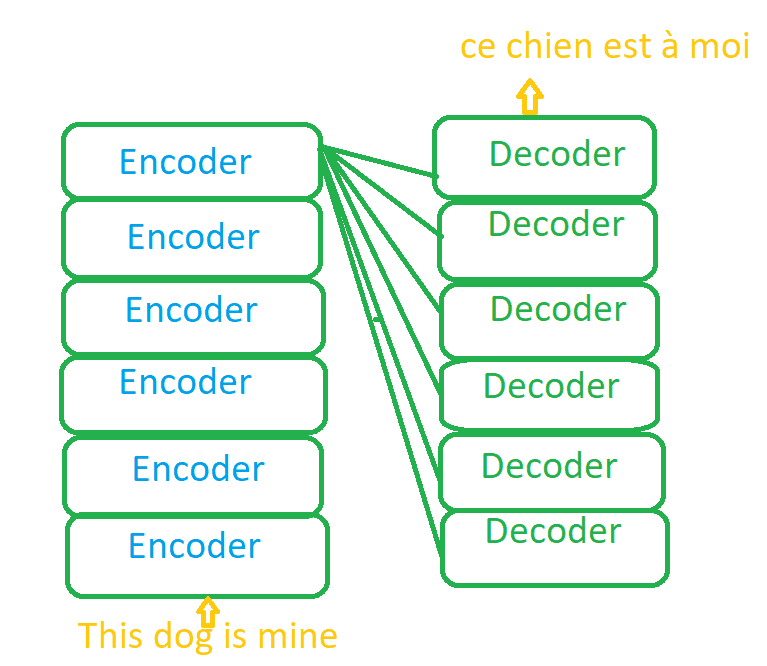




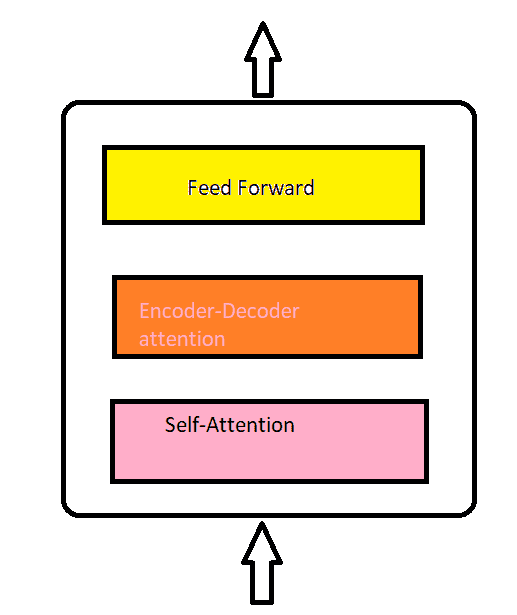
Attention=Weighted Average!

* **Positional embeddings**: Position and order of words are the essential parts of any language. They define the grammar and thus the actual semantics of a sentence. The transformer model itself doesn’t have any sense of position/order for each word. Consequently, there’s still the need for a way to incorporate the order of the words into our model. so the positional encoding is a way to add the piece of information about the position of each word.



The transformer architecture consists of the stack of six **Encoders** and **Decoders.**

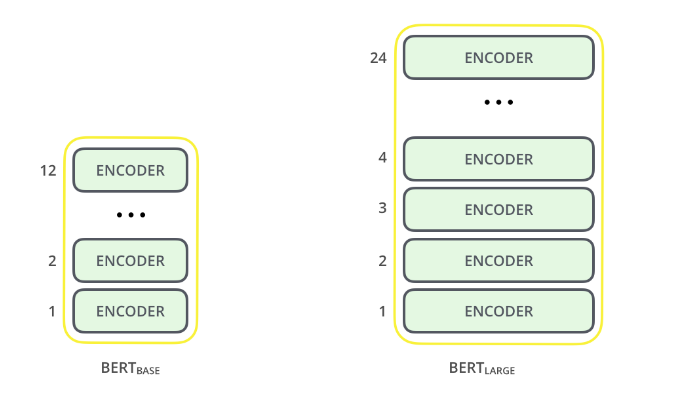
Each encoder consists of a Self Attention layer and a Feed forward network. The purpose of Encoder is to understand the What is English? and What is the context?

The Purpose of the decoder is Map One language with another language. Like How English is mapped in French.

**ARCHITECTURE**

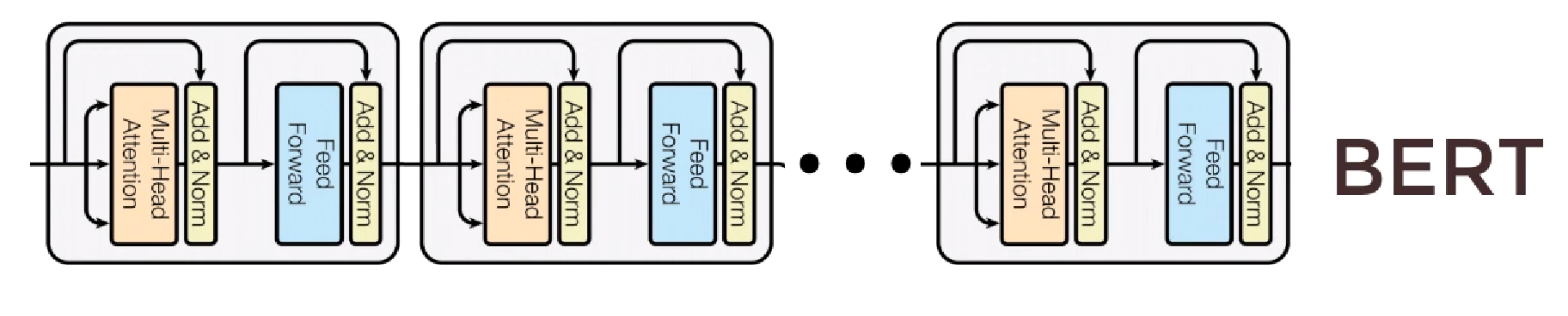
The BERT architecture builds on top of the Transformer. We currently have two variants available:

* BERT Base: 12 layers (transformer blocks), 12 attention heads, and 110 million parameters
* BERT Large: 24 layers (transformer blocks), 16 attention heads, and, 340 million parameters.



https://www.inblog.in/uploads/5e756c16d6c941ab243b3d4cc56b574b.png https://www.inblog.in/uploads/5e756c16d6c941ab243b3d4cc56b574b.png

**Why BERT if we have Transformer?**



* The Transformer has a language translation lock which means it can only do Machine Translation problems.
* However, BERT can solve multiple types of problems! Which include:
  + Machine Translation
  + Question Answering
  + Sentiment Analysis
  + Text Summarization
  + Word embedding

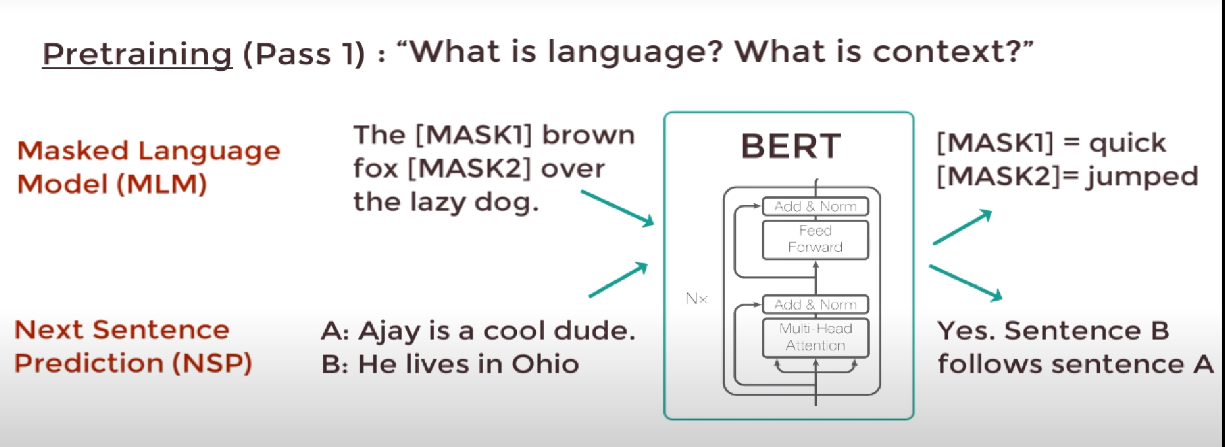
**BERT Training**

The training of BERT is done in two phases:

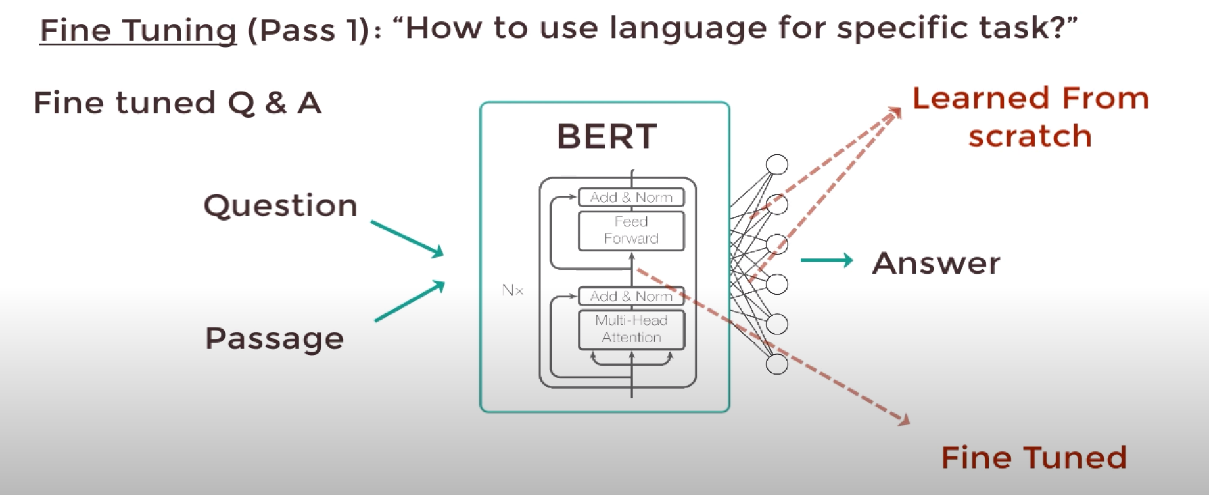
1. Pre-train BERT to understand language – Model understands what is language and context
2. Fine-tune BERT to learn specific task – Model knows the language but now learns to solve a problem.

**A high-level explanation of the two phases:**

1. **Pretraining phase – Understanding of language**

* To understand the language BERT trains on two unsupervised tasks simultaneously:
  + Masked Language Model (MLM) - BERT takes in a sentence with random words filled with “masks”. The goal is to output this mask correctly.
  + Next Sentence Prediction (NSP) - BERT takes in two sentences and it determines if the second sentence actually follows the first. This is a binary classification task. This helps BERT understand the context between different sentences themselves.
* Using MLM and NSP BERT get a good understanding of language!

1. **Fine-Tuning – Using language for a specific task**

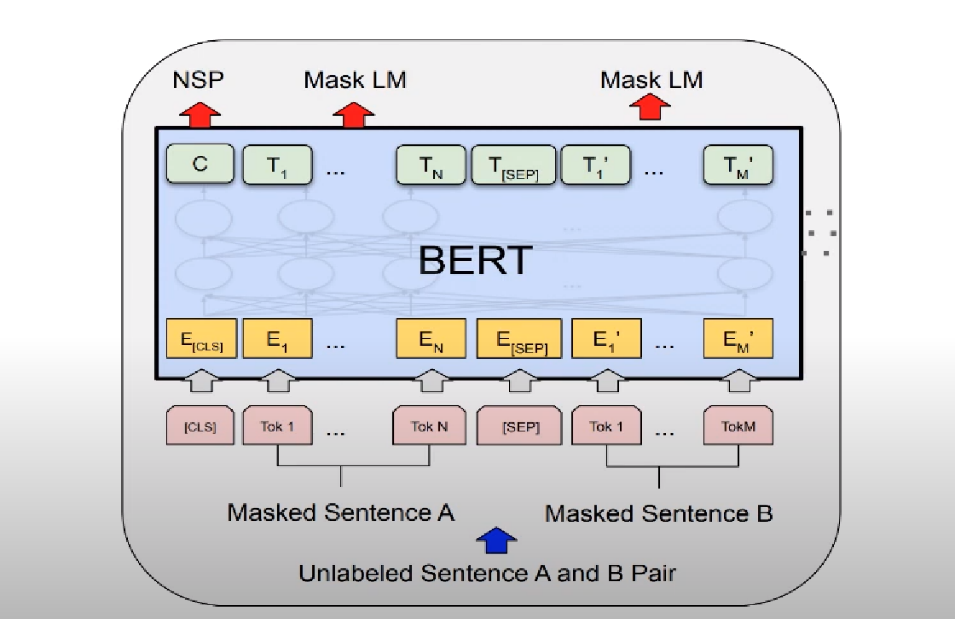


Let’s say we have a question answering a task from a given passage.

* We just need to replace the fully connected output layer with a fresh new set of output layers that can output the answer to the question we want. Then we can perform supervised training using a question answering dataset.
* The model won’t take long to train as only the output parameters are learned from scratch. The rest of the model parameters are fine tuned.
* We can do this with any NLP problem! Not just a Q/A task.

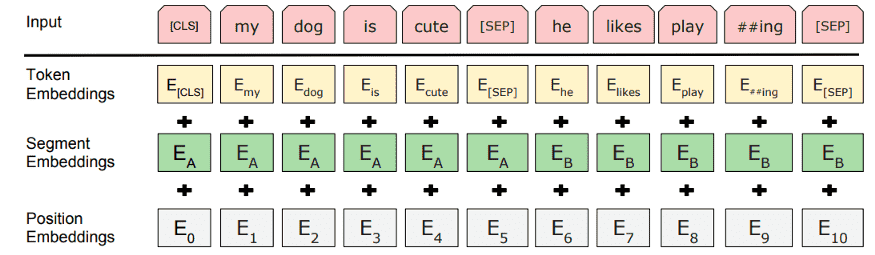
**Low-level explanation of the two phases:**

1. **Pre-training phase – Understanding of language**



* Masked Language Modelling and Next Sentence Prediction are trained simultaneously.
* C outputs either 1 or 0 depending on if sentence B follows sentence A or not.
* T1...TN is word vectors that correspond to the output.
* The masked sentence is converted to embeddings using pre-trained embeddings. Let’s see how the embeddings are done:

**Inputs to the BERT**



BERT use three embeddings to compute the input representations.

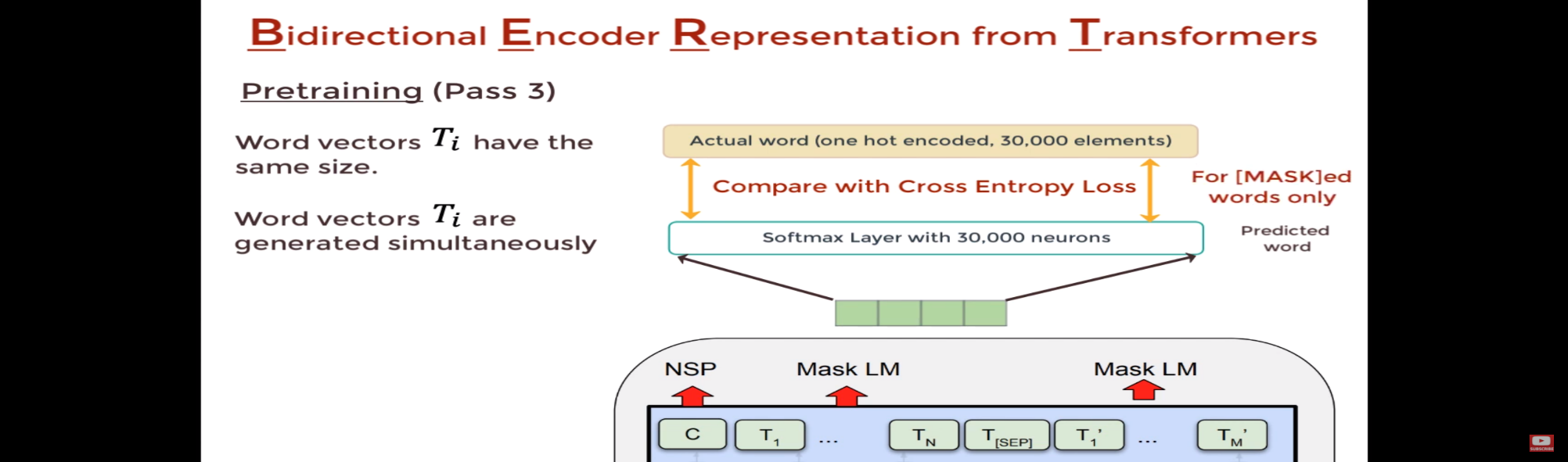
* + **Token Embedding:** Generally it is called Word embedding. The main paper uses “WordPieces'' embeddings that have a vocabulary of 30K words. The vectors also encode the semantic meaning of the words.
  + **Segment embeddings:**Sentence number that has been encoded into a vector.
  + **Position Embedding:** Position of the words within that sentence.

Adding these three embeddings together we get an embedding vector that we use as input to BERT.

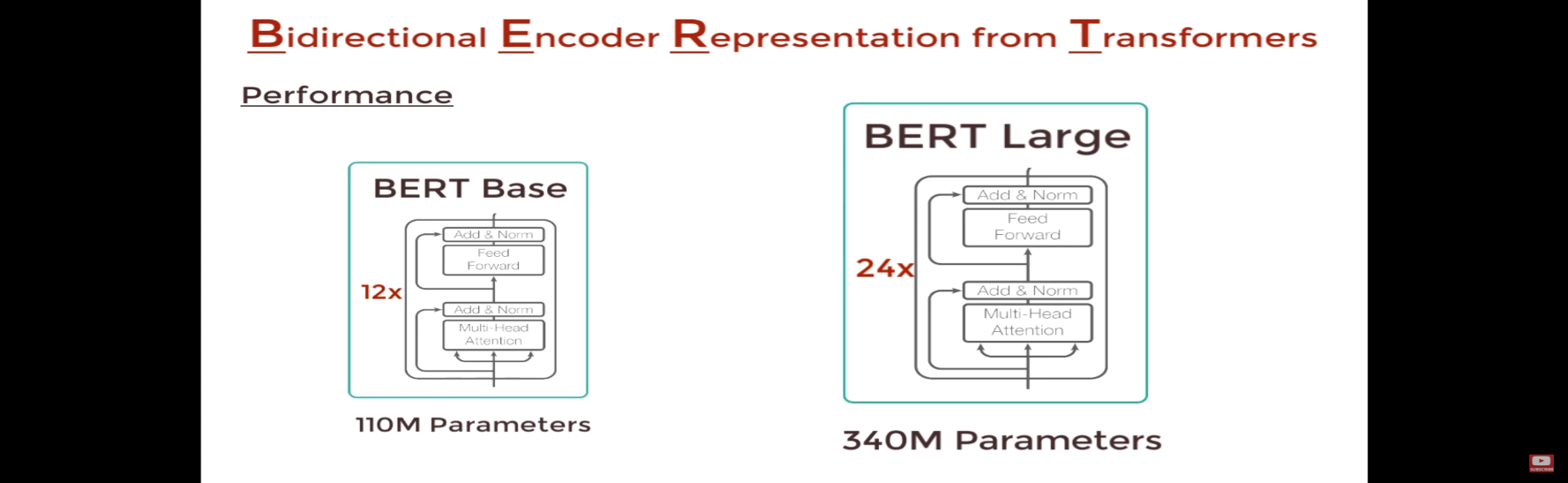
Segment + Position embeddings are required for temporal ordering since all these vectors are fed simultaneously into BERT. Language models need these ordering preserved.

1. **Fine-Tuning – Using language for a specific task**

* **Training the BERT**



* + Take each output word vector and pass it to fully connected layered output with neurons = tokens of the vocabulary. In that output layer, we would apply Softmax activation which will convert the word vector to a distribution.
  + We compare the actual distribution with the output distribution and train the network using cross-entropy loss.
  + Note that the loss only considers the predictions of the masked words and it ignores all the other words that are output by the network. Therefore, more focus is given to predicting the masked values.
* Our model performance will also depend on how big our BERT is. If we are using BERT large, we will get way better performance as it has way more parameters than default BERT:



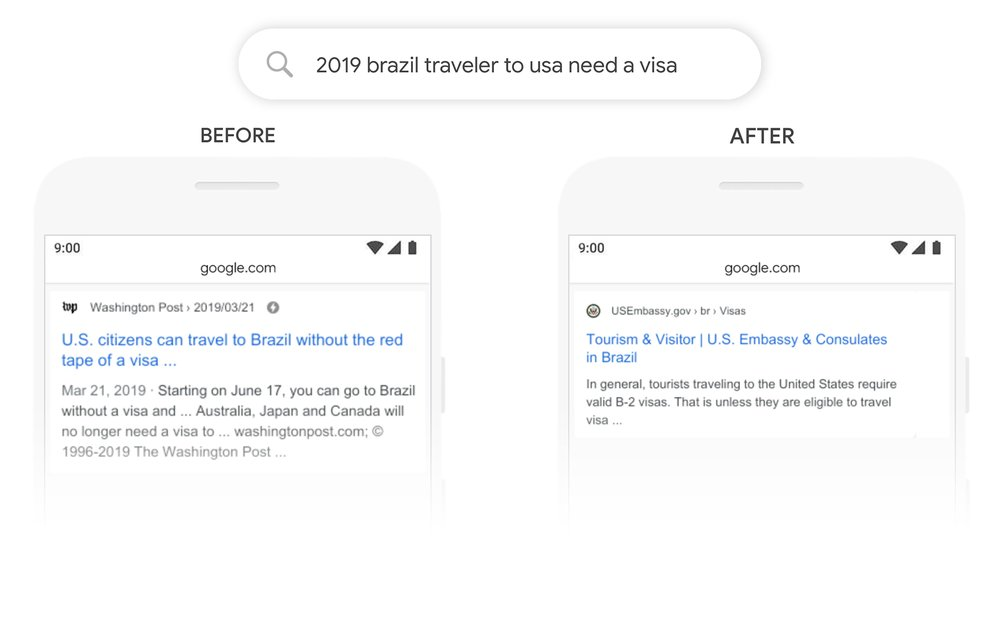
**List of all the pre-trained models HuggingFace library provides us :**

| bert-base-uncased | 12-layer, 768-hidden, 12-heads, 110M parameters.  Trained on lower-cased English text. |
| --- | --- |
| bert-large-uncased | 24-layer, 1024-hidden, 16-heads, 336M parameters.  Trained on lower-cased English text. |
| bert-base-cased | 12-layer, 768-hidden, 12-heads, 109M parameters.  Trained on cased English text. |
| bert-large-cased | 24-layer, 1024-hidden, 16-heads, 335M parameters.  Trained on cased English text. |
| bert-base-multilingual-uncased | (Original, not recommended) 12-layer, 768-hidden, 12-heads, 168M parameters.  Trained on lower-cased text in the top 102 languages with the largest Wikipedias  (see [details](https://github.com/google-research/bert/blob/master/multilingual.md)). |
| bert-base-multilingual-cased | (New, **recommended**) 12-layer, 768-hidden, 12-heads, 179M parameters.  Trained on cased text in the top 104 languages with the largest Wikipedias  (see [details](https://github.com/google-research/bert/blob/master/multilingual.md)). |
| bert-base-chinese | 12-layer, 768-hidden, 12-heads, 103M parameters.  Trained on cased Chinese Simplified and Traditional text. |
| bert-base-german-cased | 12-layer, 768-hidden, 12-heads, 110M parameters.  Trained on cased German text by Deepset.ai  (see [details on deepset.ai website](https://deepset.ai/german-bert)). |
| bert-large-uncased-whole-word-masking | 24-layer, 1024-hidden, 16-heads, 336M parameters.  Trained on lower-cased English text using Whole-Word-Masking  (see [details](https://github.com/google-research/bert/#bert)). |
| bert-large-cased-whole-word-masking | 24-layer, 1024-hidden, 16-heads, 335M parameters.  Trained on cased English text using Whole-Word-Masking  (see [details](https://github.com/google-research/bert/#bert)). |
| bert-large-uncased-whole-word-masking-fine-tuneed-squad | 24-layer, 1024-hidden, 16-heads, 336M parameters.  The bert-large-uncased-whole-word-masking model is fine-tuned on SQuAD  (see details of fine-tuning in the [example section](https://github.com/huggingface/transformers/tree/master/examples)). |
| bert-large-cased-whole-word-masking-finetuned-squad | 24-layer, 1024-hidden, 16-heads, 335M parameters  The bert-large-cased-whole-word-masking model fine-tuned on SQuAD  (see [details of fine-tuning in the example section](https://huggingface.co/transformers/examples.html)) |
| bert-base-cased-finetuned-mrpc | 12-layer, 768-hidden, 12-heads, 110M parameters.  The bert-base-cased model fine-tuned on MRPC  (see [details of fine-tuning in the example section](https://huggingface.co/transformers/examples.html)) |
| bert-base-german-dbmdz-cased | 12-layer, 768-hidden, 12-heads, 110M parameters.  Trained on cased German text by DBMDZ  (see [details on dbmdz repository](https://github.com/dbmdz/german-bert)). |
| bert-base-german-dbmdz-uncased | 12-layer, 768-hidden, 12-heads, 110M parameters.  Trained on uncased German text by DBMDZ  (see [details on dbmdz repository](https://github.com/dbmdz/german-bert)). |
| cl-tohoku/bert-base-japanese | 12-layer, 768-hidden, 12-heads, 111M parameters.  Trained on Japanese text. Text is tokenized with MeCab and WordPiece and this requires some extra dependencies,  [fugashi](https://github.com/polm/fugashi) which is a wrapper around [MeCab](https://taku910.github.io/mecab/).  Use pip install transformers["ja"] (or pip install -e .["ja"] if you install from source) to install them.  (see [details on cl-tohoku repository](https://github.com/cl-tohoku/bert-japanese)). |
| cl-tohoku/bert-base-japanese-whole-word-masking | 12-layer, 768-hidden, 12-heads, 111M parameters.  Trained on Japanese text. Text is tokenized with MeCab and WordPiece and this requires some extra dependencies,  [fugashi](https://github.com/polm/fugashi) which is a wrapper around [MeCab](https://taku910.github.io/mecab/).  Use pip install transformers["ja"] (or pip install -e .["ja"] if you install from source) to install them.  (see [details on cl-tohoku repository](https://github.com/cl-tohoku/bert-japanese)). |
| cl-tohoku/bert-base-japanese-char | 12-layer, 768-hidden, 12-heads, 90M parameters.  Trained on Japanese text. Text is tokenized into characters.  (see [details on cl-tohoku repository](https://github.com/cl-tohoku/bert-japanese)). |
| cl-tohoku/bert-base-japanese-char-whole-word-masking | 12-layer, 768-hidden, 12-heads, 90M parameters.  Trained on Japanese text using Whole-Word-Masking. Text is tokenized into characters.  (see [details on cl-tohoku repository](https://github.com/cl-tohoku/bert-japanese)). |
| TurkuNLP/bert-base-finnish-cased-v1 | 12-layer, 768-hidden, 12-heads, 125M parameters.  Trained on cased Finnish text.  (see [details on turkunlp.org](http://turkunlp.org/FinBERT/)). |
| TurkuNLP/bert-base-finnish-uncased-v1 | 12-layer, 768-hidden, 12-heads, 110M parameters.  Trained on uncased Finnish text.  (see [details on turkunlp.org](http://turkunlp.org/FinBERT/)). |
| wietsedv/bert-base-dutch-cased | 12-layer, 768-hidden, 12-heads, 110M parameters.  Trained on cased Dutch text.  (see [details on wietsedv repository](https://github.com/wietsedv/bertje/)). |

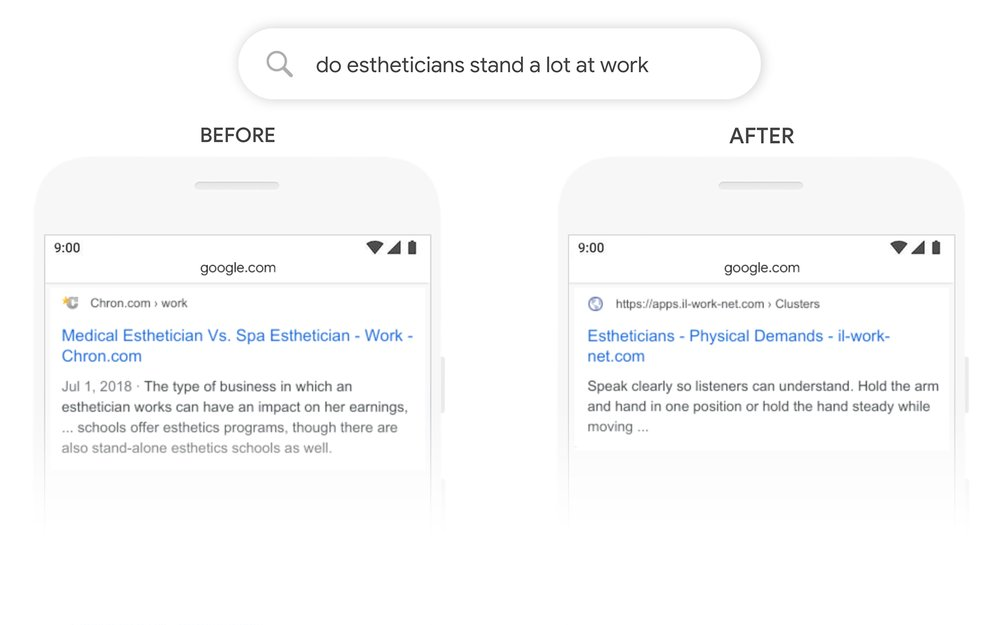
* There are thousands more community-made models too which can be used!
* Here are examples of pre-trained models in action on HuggingFace’s official website: <https://huggingface.co/transformers/task_sum3mary.html>

**How Google used BERT to improve their search!**

* Reference article: <https://blog.google/products/search/search-language-understanding-bert>
* At Google applying BERT models to both rankings and featured snippets in search, they are able to do a much better job helping to find useful information.
* They realized particularly for longer, more conversational queries, or searches where prepositions like “for” and “to” matter a lot to the meaning, a search was able to understand the context of the words in the query entered.
* Here’s a search for “2019 brazil travelers to the USA need a visa.” The word “to” and its relationship to the other words in the query are particularly important to understanding the meaning. It’s about a Brazilian traveling to the U.S. and not the other way around.



* Here is another example where BERT has helped Google grasp the subtle nuances of language that computers don’t quite understand the way humans do.



# **Conclusion**

BERT is undoubtedly a breakthrough in the use of Machine Learning for Natural Language Processing. The fact that it’s approachable and allows fast fine-tuning will likely allow a wide range of practical applications. A lot of AI companies have already started to implement BERT in their NLP tasks.

**References:**

1. <https://arxiv.org/abs/1810.04805>

**2**[.https://inblog.in/A-gentle-introduction-to-BERT-Model-JfGFFXb97v](about:blank)

**3**.<https://huggingface.co/transformers/pretrained_models.html>

**4.**<https://jalammar.github.io/illustrated-bert/>

**5.**<https://www.youtube.com/watch?v=xI0HHN5XKDo&feature=youtu.be>

**6.** [https://blog.google/products/search/search-language-understanding-bert](about:blank)