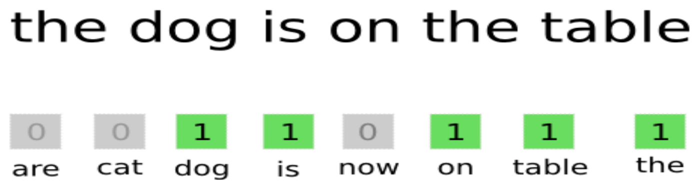
Beyond Simple Word Embeddings / BERT

By Kaveri Kale(PhD)
IIT Bombay

Classic techniques

Bag-of-words

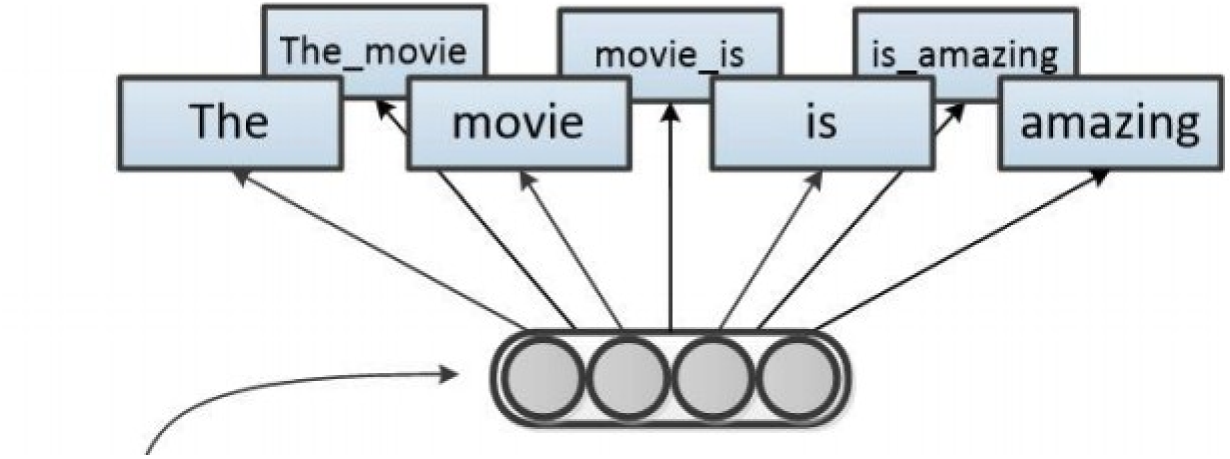
- This is done by deciding on a set of n words that will form the vocabulary supported by the mapping, and assigning each word in the vocabulary a unique index.
- Then, each document is represented by a vector of length n, in which the i-th entry contains the number of occurrences of the word i in the document.



- For example, the sentence “dog eat dog world, baby!” (after cleaning punctuation) might be represented by a 550-length vector v (assuming a vocabulary of 550 words was chosen), which is zero everywhere except the following entries:
- V_{76}=1, as the 76th word of the vocabulary is world.
- V_{200}=2, as the 200th word of the vocabulary is dog.
- V_{322}=1, as the 322nd word of the vocabulary is eat.
- The word babywas not selected for inclusion in the vocabulary, so it induces a value of 1 on no entry of the vector.

Classic techniques

Bag-of-n-grams



Classic techniques

tf-idf weighting

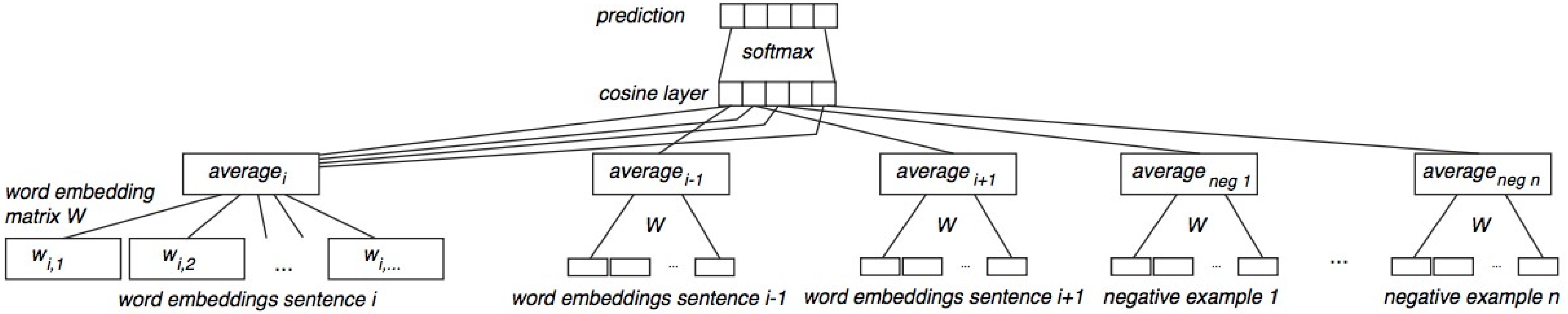
- A final related technique worth mentioning in the context of bag-of-words is term frequency–inverse document frequency, commonly denoted as tf-idf.

$$IDF_i = log\left(\frac{\text{\# of documents in corpus}}{\text{\# of documents in which word i appears in}}\right)$$

Classic techniques

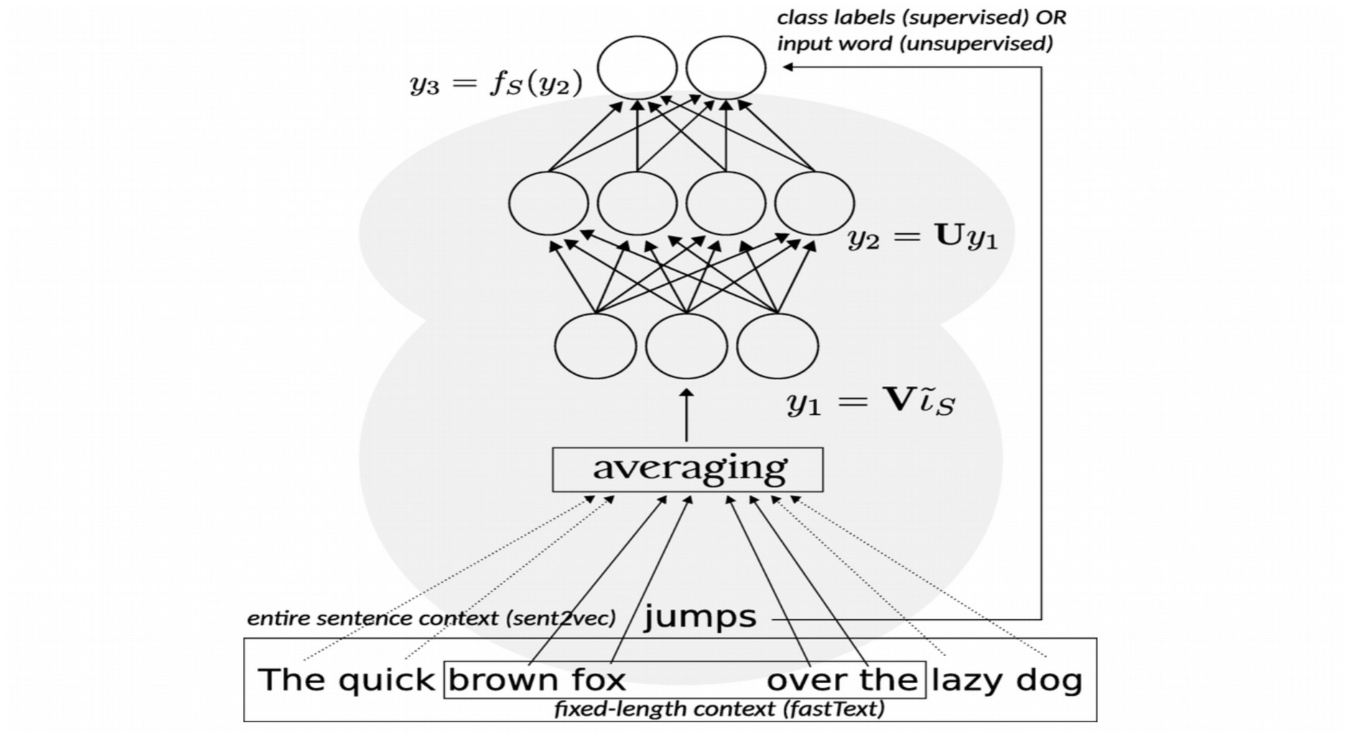
Averaging word embeddings

- There is a very intuitive way to construct document embeddings from meaningful word embeddings: Given a document, perform some vector arithmetics on all the vectors corresponding to the words of the document to summarize them into a single vector in the same embedding space; two such common summarization operators are average and sum.
- Use a fixed (unlearnable) operator for vector summarization — e.g., averaging — and learn word embeddings in a preceding layer, using a learning target that is aimed at producing rich document embeddings; a common example is using a sentence to predict context sentences. Thus the main advantage here is that word embeddings are optimized for averaging into document representations.



Classic techniques

- **Sent2Vec**
  This technique is very much a combination of the two above approaches: The classic CBOW model of word2vec is both extended to include word n-grams and adapted to optimize the word (and n-grams) embeddings for the purpose of averaging them to yield document vectors.

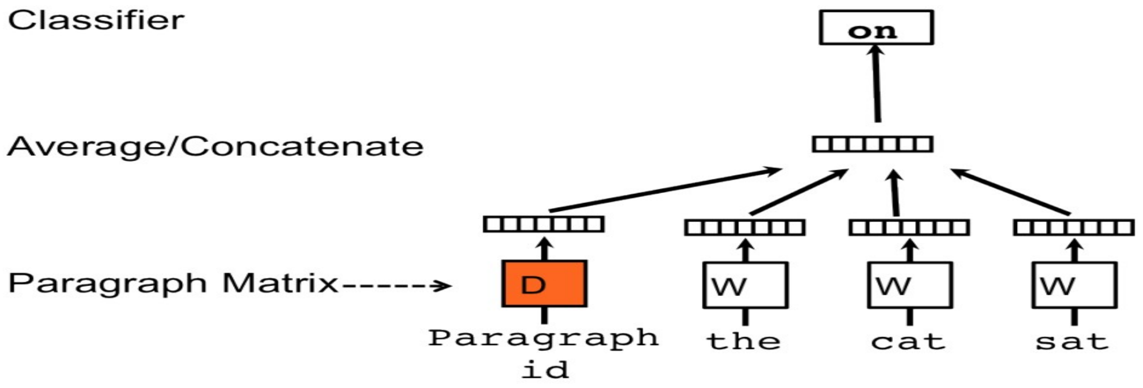

class labels (supervised) OR
input word (unsupervised)
$y_3 = f_S(y_2)$
$y_2 = \mathbf{U} y_1$
$y_1 = \mathbf{V} \tilde{\iota}_S$
averaging
entire sentence context (sent2vec)
jumps
The quick brown fox over the lazy dog
fixed-length context (fastText)

Classic techniques

Paragraph vectors (doc2vec)

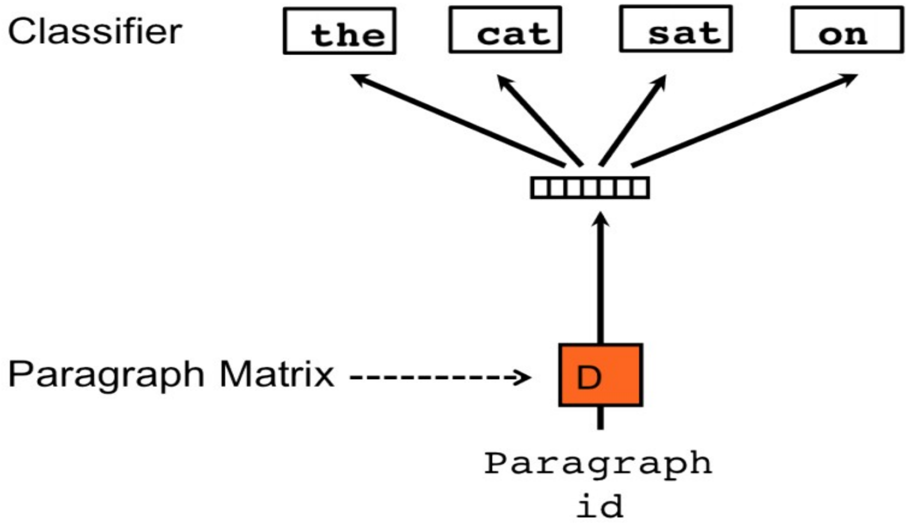
- It is perhaps the first attempt to generalize word2vec to work with word sequences.
- Two variants of the paragraph vectors model: Distributed Memory and Distributed Bag-of-Words.

- **Paragraph Vectors: Distributed Memory (PV-DM)**
- The PV-DM model augments the standard encoder-decoder model by adding a memory vector, aimed at capturing the topic of the paragraph, or context from the input. The training task here is quite similar to that of a continuous bag of words; a single word is to be predicted from its context. In this case, the context words are the preceding words, not the surrounding words, as is the paragraph.



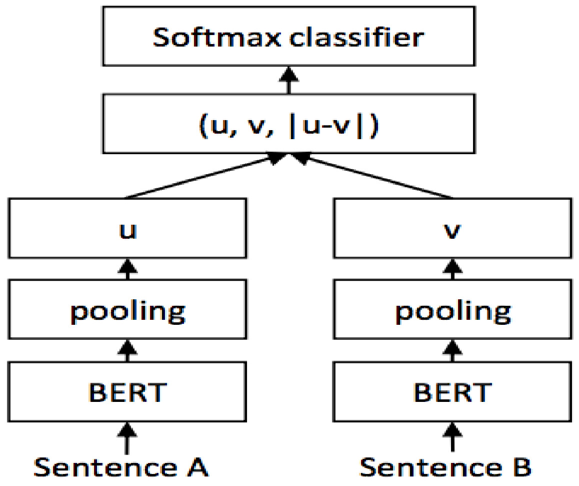
Classic techniques

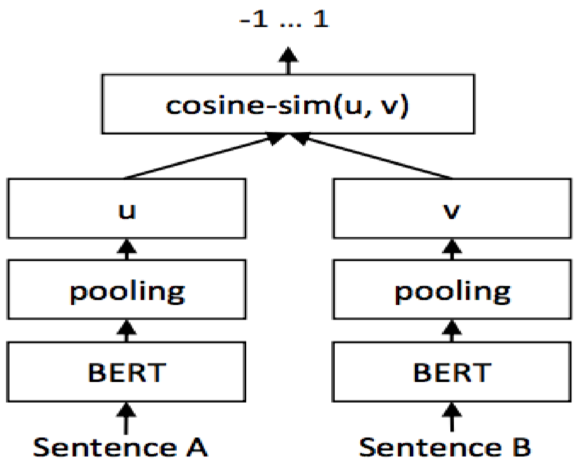
- **Paragraph Vectors: Distributed Bag of Words (PV-DBOW)**
  The second variant of paragraph vectors, despite its name, is perhaps the parallel of word2vec's skip-gram architecture; the classification task is to predict a single context word using only the paragraph vector. At each iteration of stochastic gradient descent, a text window is sampled, then a single random word is sampled from that window, forming the below classification task.



Sentence-BERT (SBERT)

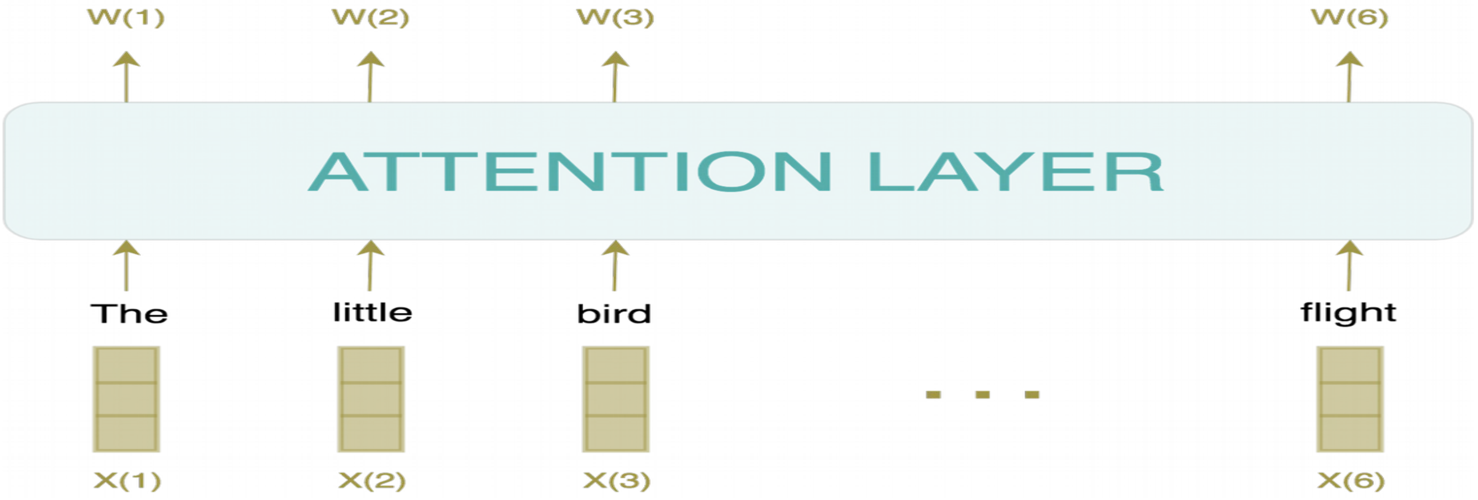
- Sentence-BERT, aims to adapt the BERT architecture to derive semantically meaningful sentence embeddings that can be compared using cosine-similarity.
- BERT Architecture for Classification and Inference



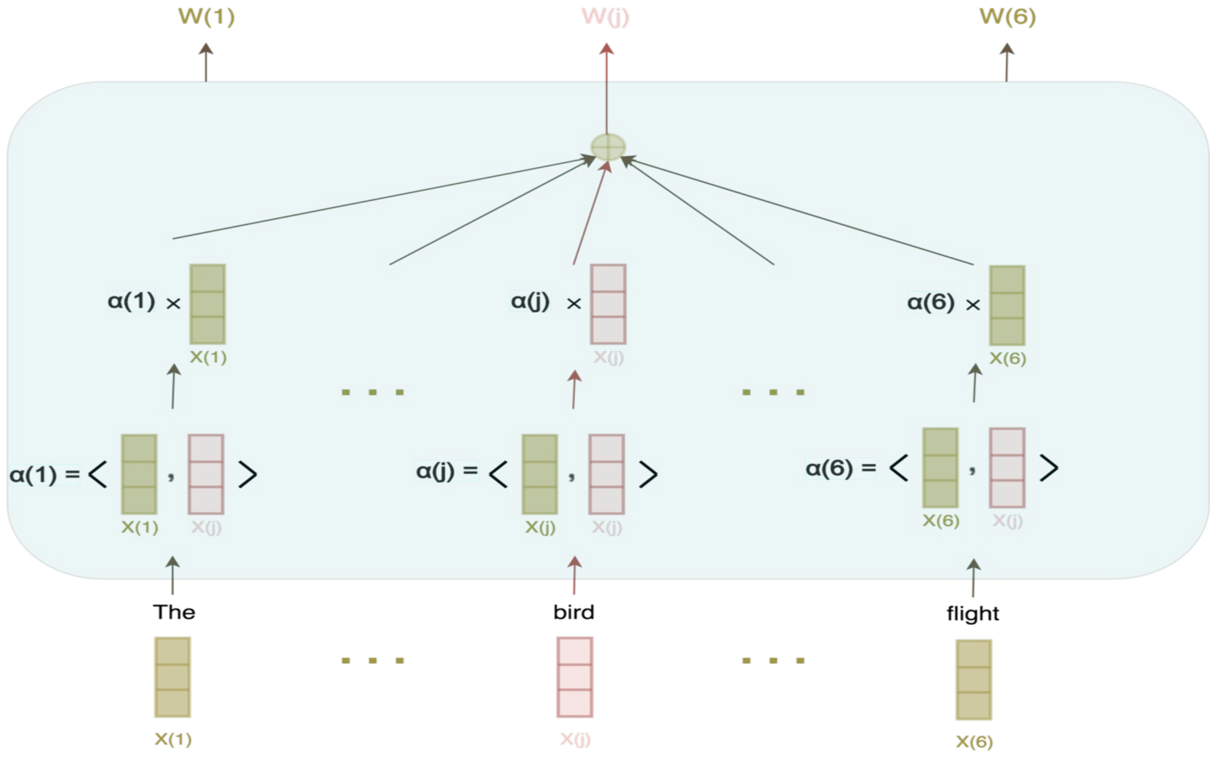


Self-Attention

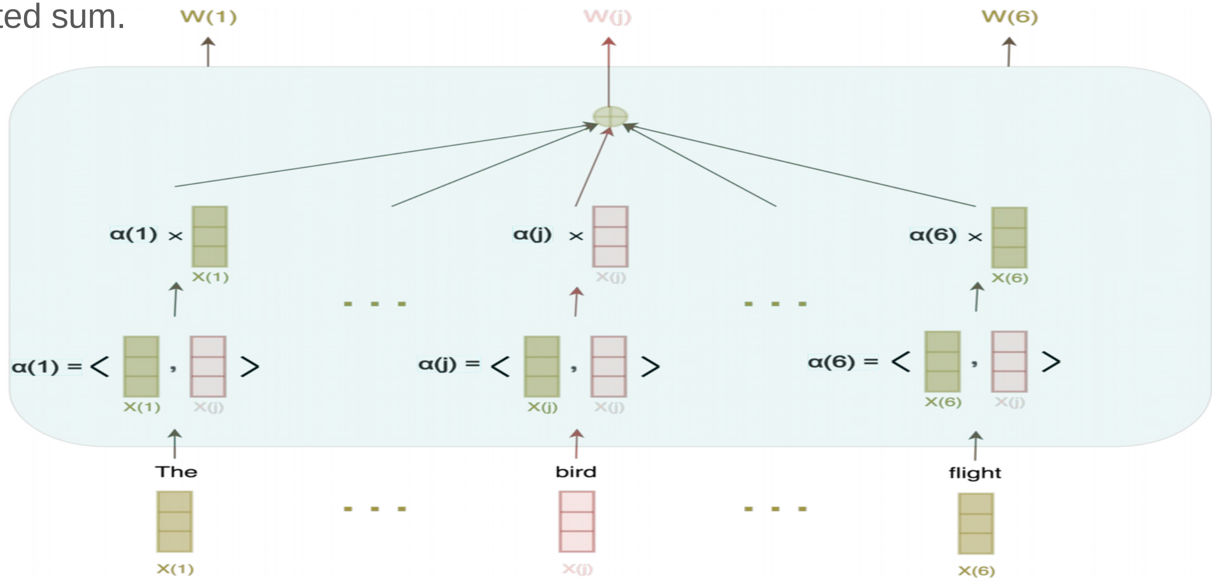
- Self-attention is the method the Transformer uses to bake the “understanding” of other relevant words into the one we’re currently processing.
- Self-attention is first and foremost a sequence-to-sequence transformation, meaning it transforms a sequence of tokens into a new sequence of tokens.
- Say the following sentence is an input sentence we want to translate:
  ”The animal didn't cross the street because it was too tired”
- What does “it” in this sentence refer to? Is it referring to the street or to the animal? It’s a simple question to a human, but not as simple to an algorithm.
- When the model is processing the word “it”, self-attention allows it to associate “it” with “animal”.



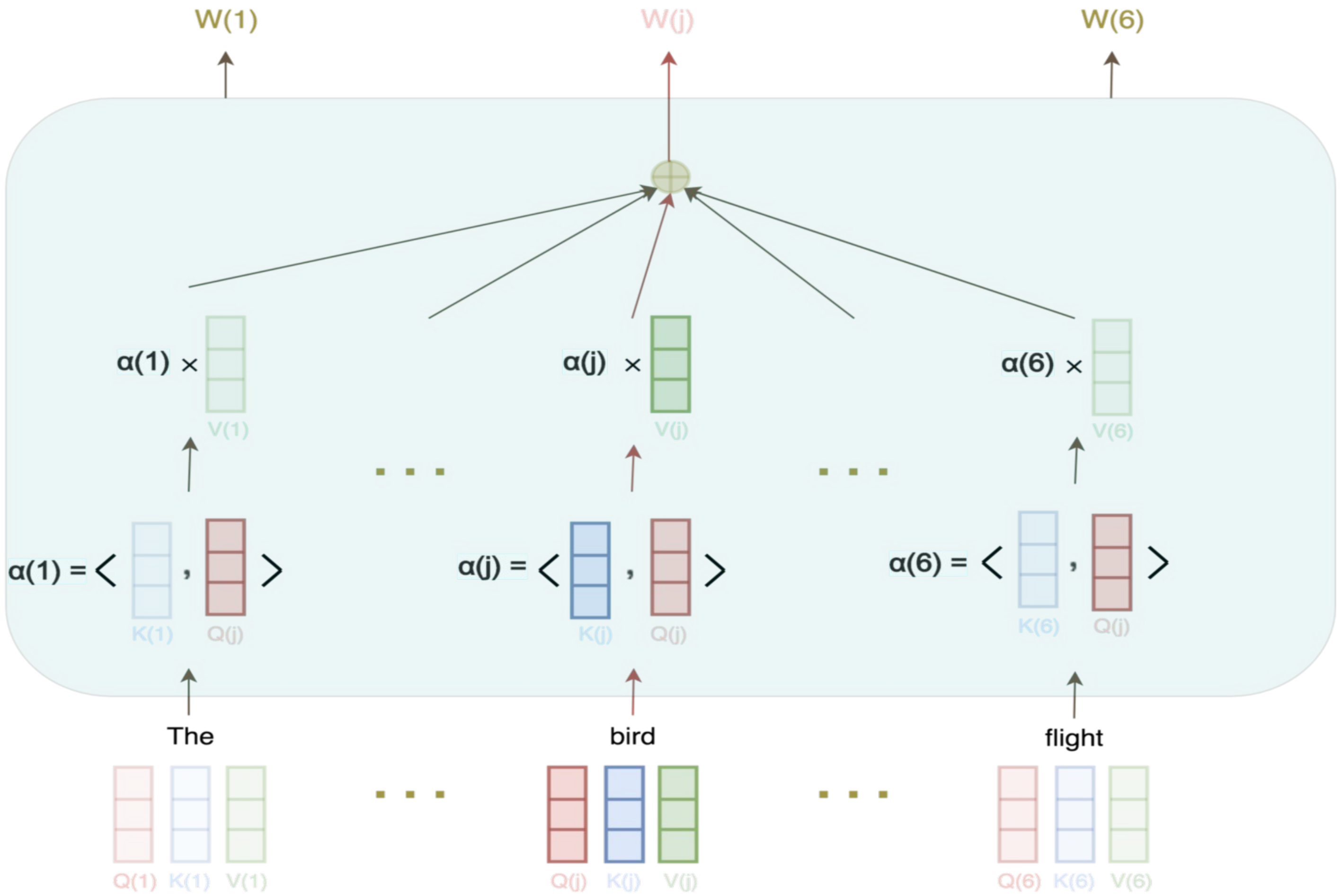
-


W(1)
W(j)
W(6)
α(1) × X(1)
α(j) × X(j)
α(6) × X(6)
α(1) = < X(1) , X(j) >
α(j) = < X(j) , X(j) >
α(6) = < X(6) , X(j) >
The
bird
flight
X(1)
X(j)
X(6)

- **Three Embeddings for the Price of One**
- Now let’s take a step back and think of how the embedding W(i) of word ineeds to behave. Here, I want you to notice that this single vector is actually being used in three different tasks! (In the following, I am paraphrasing this blogpost):
- 1- To compute the new representation of word i: The embedding W(i) will be used in the dot product with the other wordsW(j)of the sentence to compute the weights for each word.
- 2- To compute the new representations of the other words in the sentence j: The embedding W(i) will be used in the dot product with these words to compute the weights for each word.
- 3- To compute the new representation of word i: The embedding W(i) will be used as a summand of the final weighted sum.



•


W(1)
W(j)
W(6)
α(1) ×
V(1)
α(j) ×
V(j)
α(6) ×
V(6)
α(1) = < , >
K(1)
Q(j)
α(j) = < , >
K(j)
Q(j)
α(6) = < , >
K(6)
Q(j)
The
bird
flight
Q(1) K(1) V(1)
Q(j) K(j) V(j)
Q(6) K(6) V(6)

Sentence:

- *The little **bird** took its flight.*
- Measure how bird is related to other words in sentence.
- dot product of its Query vector, Q(bird), with the Key of all the other words, K(the),...,K(flight), yielding a vector of weights for all the words in the sentence: α(the),…,α(flight).
- We then use these weights in the weighted sum of Value vectors of all the words in the sentence V(the),...,V(nest).

$$W(bird) = \alpha(the).V(the) + \alpha(little).V(little) + ... + \alpha(flight).V(flight)$$

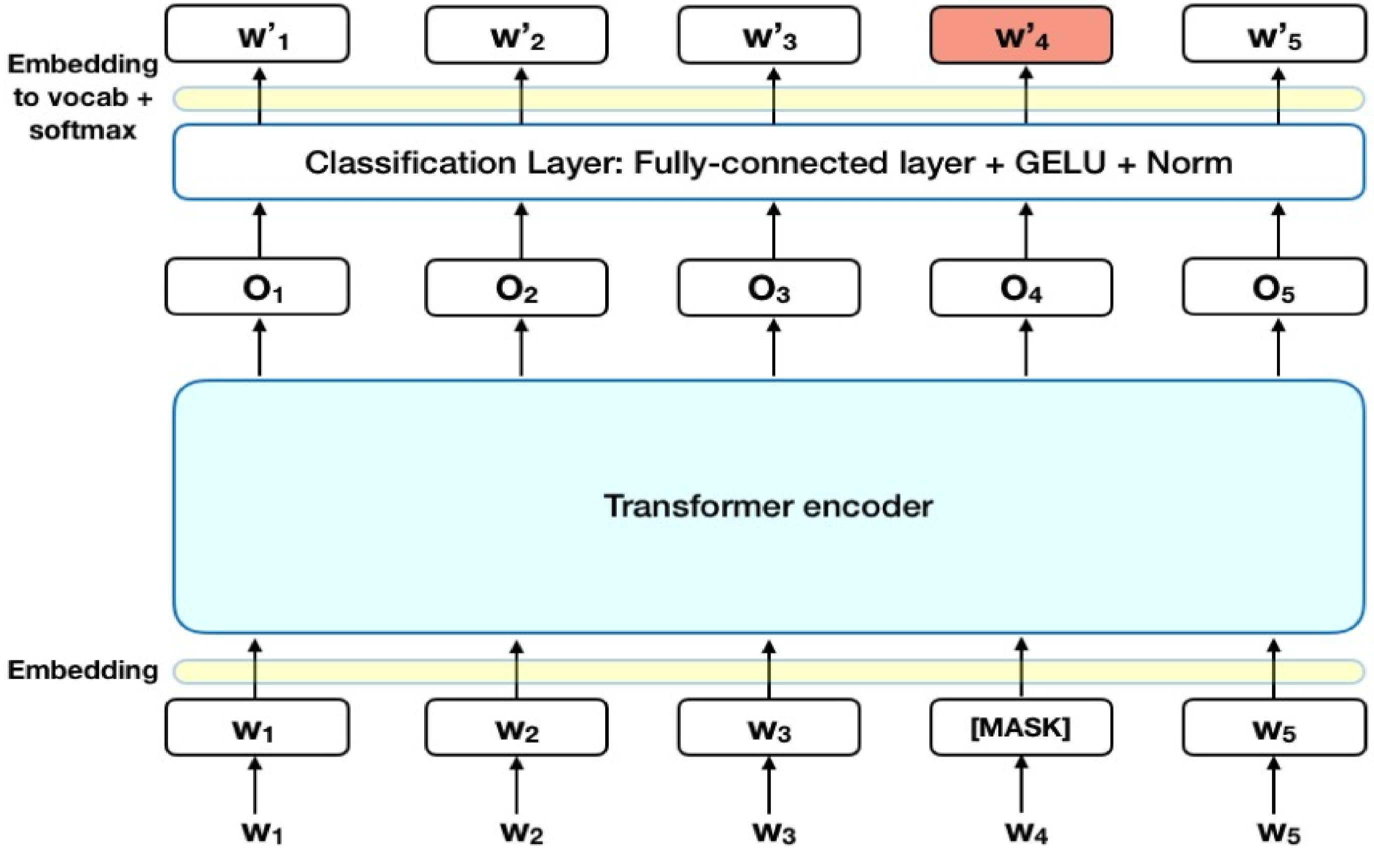
- And since all of these operations are independent, they can be vectorized:

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

BERT(Bidirectional Representation for Transformers)

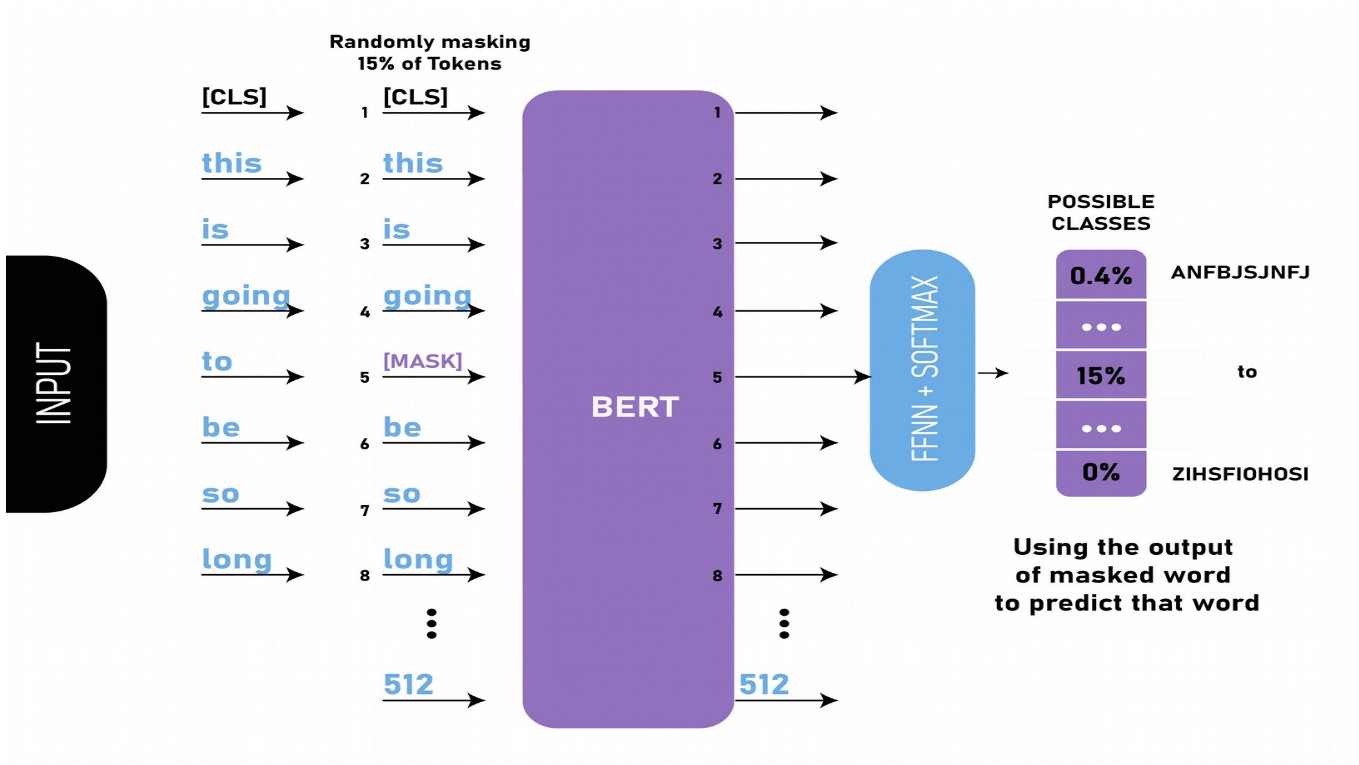
- BERT makes use of Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text.
- When training language models, there is a challenge of defining a prediction goal. Many models predict the next word in a sequence (e.g. “The child came home from ___”), a directional approach which inherently limits context learning. To overcome this challenge, BERT uses two training strategies:
- **Masked LM (MLM)**
- Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token.
- The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence. In technical terms, the prediction of the output words requires:
  1. Adding a classification layer on top of the encoder output.
  2. Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
  3. Calculating the probability of each word in the vocabulary with softmax.
-

BERT


w'1
w'2
w'3
w'4
w'5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O1
O2
O3
O4
O5
Transformer encoder
Embedding
w1
w2
w3
[MASK]
w5
w1
w2
w3
w4
w5

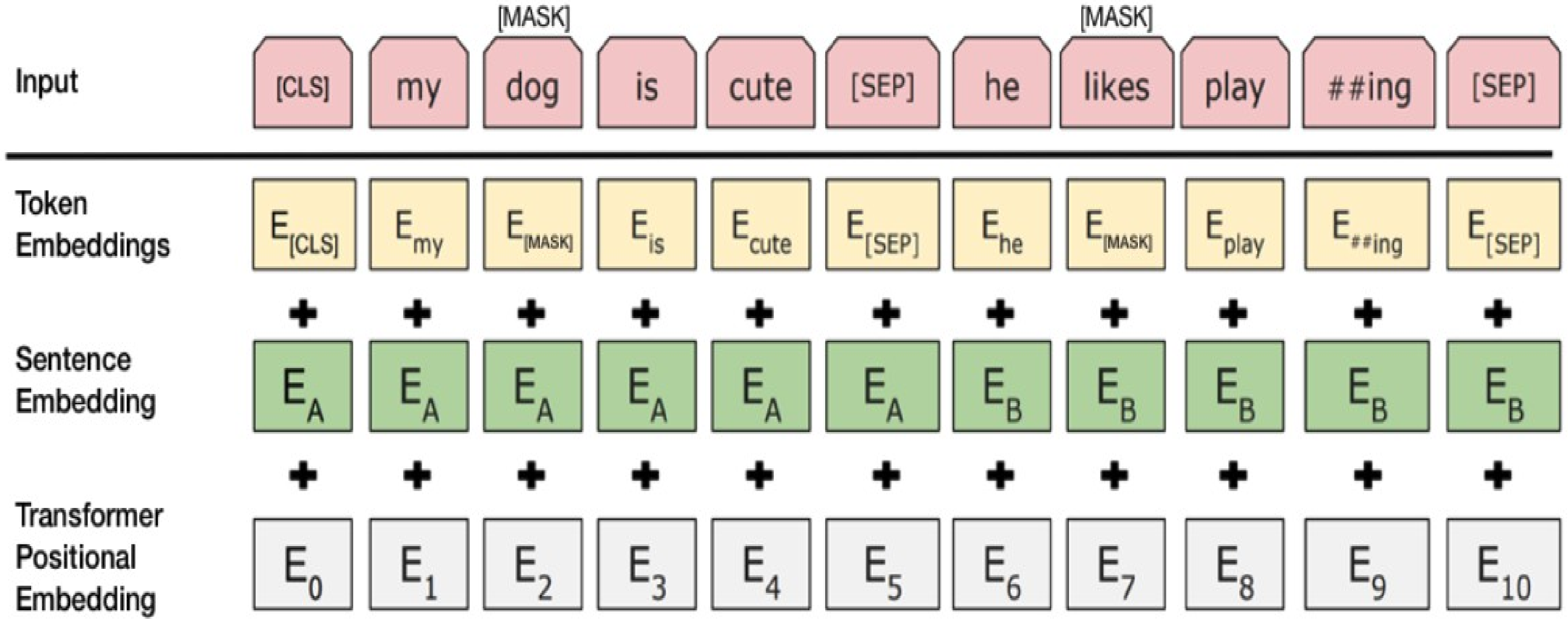
BERT

-



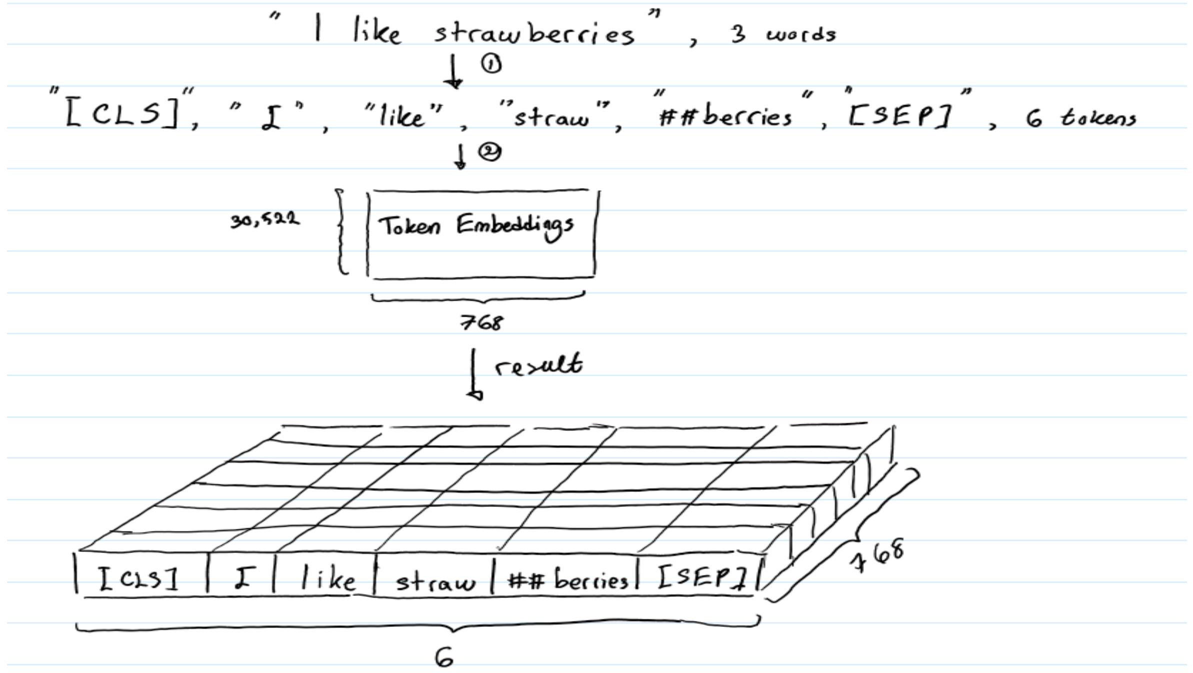
Randomly masking 15% of Tokens
INPUT
[CLS]
this
is
going
to
be
so
long
1 [CLS]
2 this
3 is
4 going
5 [MASK]
6 be
7 so
8 long
512
BERT
1
2
3
4
5
6
7
8
512
FFNN + SOFTMAX
POSSIBLE CLASSES
0.4% ANFBJSJNFJ
•••
15% to
•••
0% ZIHSFIOHOSI
Using the output of masked word to predict that word

Embedding Layers In BERT



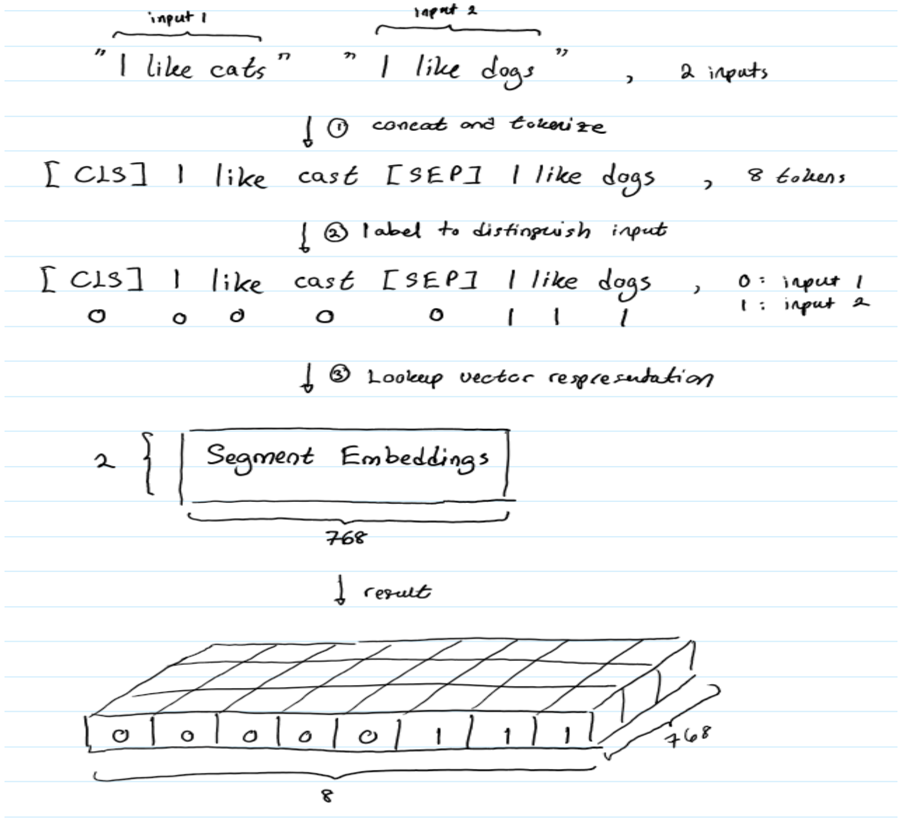
Embedding Layers In BERT

Token Embeddings

- The role of the Token Embeddings layer is to transform words into vector representations of fixed dimension. In the case of BERT, each word is represented as a **768-dimensional** vector.
- 
"I like strawberries", 3 words
①
"[CLS]", "I", "like", "straw", "##berries", "[SEP]", 6 tokens
②
30,522
Token Embeddings
768
result
[CLS]
I
like
straw
##berries
[SEP]
768
6

Segment Embeddings

- The pair of input text are simply concatenated and fed into the model. So how does BERT distinguishes the inputs in a given pair? The answer is Segment Embeddings.



input 1
input 2
" I like cats " " I like dogs " , 2 inputs
① concat and tokenize
[CLS] I like cast [SEP] I like dogs , 8 tokens
② label to distinguish input
[CLS] I like cast [SEP] I like dogs , 0 : input 1
0 0 0 0 0 1 1 1
1 : input 2
③ Lookup vector respresentation
2
Segment Embeddings
768
result
0 0 0 0 0 1 1 1
768
8

Position Embeddings

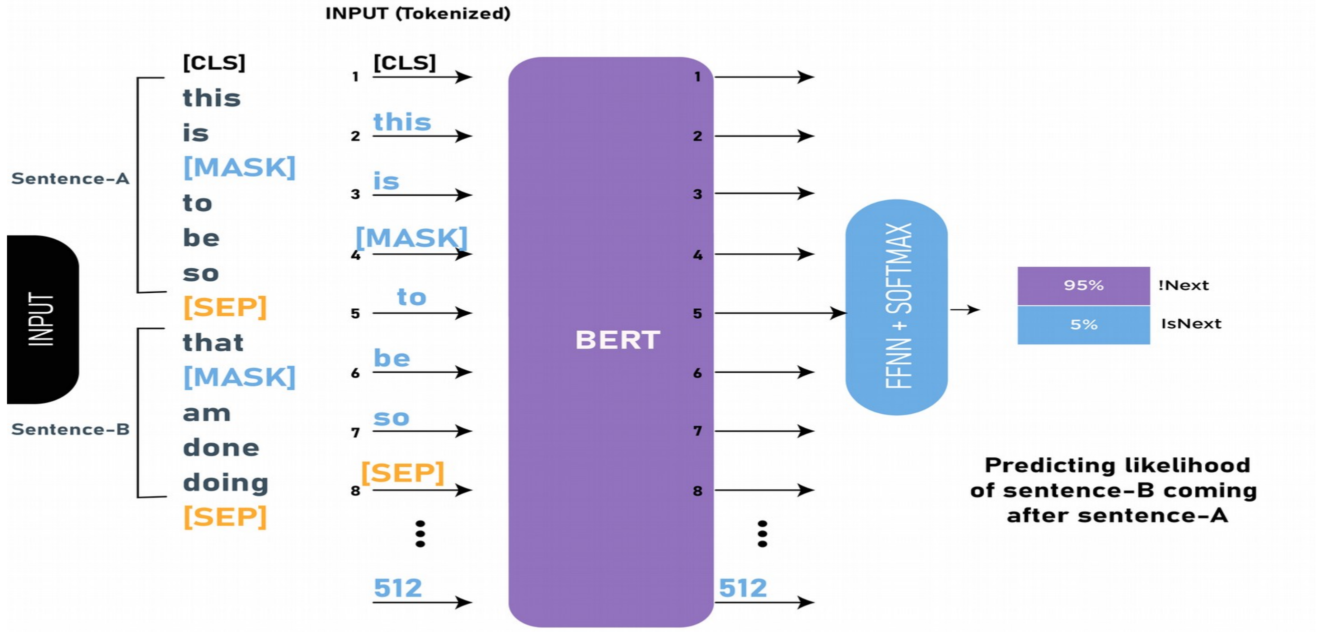
- BERT consists of a stack of Transformers and broadly speaking, Transformers do not encode the sequential nature of their inputs.
- To summarize, having position embeddings will allow BERT to understand that given an input text like:
  I think, therefore I am
  *t*he first “I” should not have the same vector representation as the second “I”.
- BERT was designed to process input sequences of up to length 512.
- Position Embeddings layer is a lookup table of size (512, 768) where the first row is the vector representation of any word in the first position, the second row is the vector representation of any word in the second position, etc.
- Therefore, if we have an input like “Hello world” and “Hi there”, both “Hello” and “Hi” will have identical position embeddings since they are the first word in the input sequence.

- **Token Embeddings** with shape (1, n, 768) which are just vector representations of words
- **Segment Embeddings** with shape (1, n, 768) which are vector representations to help BERT distinguish between paired input sequences.
- **Position Embeddings** with shape (1, n, 768) to let BERT know that the inputs its being fed with have a temporal property.
- These representations are summed element-wise to produce a single representation with shape (1, n, 768). This is the input representation that is passed to BERT’s Encoder layer.

BERT

Next Sentence Prediction (NSP)

- 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 subsequent sentence in the original document.
- During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence.
- To help the model distinguish between the two sentences in training, the input is processed in the following way before entering the model:
- A [CLS] token is inserted at the beginning of the first sentence and a [SEP] token is inserted at the end of each sentence.
- A sentence embedding indicating Sentence A or Sentence B is added to each token. Sentence embeddings are similar in concept to token embeddings with a vocabulary of 2.
- A positional embedding is added to each token to indicate its position in the sequence. The concept and implementation of positional embedding are presented in the Transformer paper.
-



- To predict if the second sentence is indeed connected to the first, the following steps are performed:
- The entire input sequence goes through the Transformer model.
- The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases).
- Calculating the probability of IsNextSequence with softmax.
- When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies.

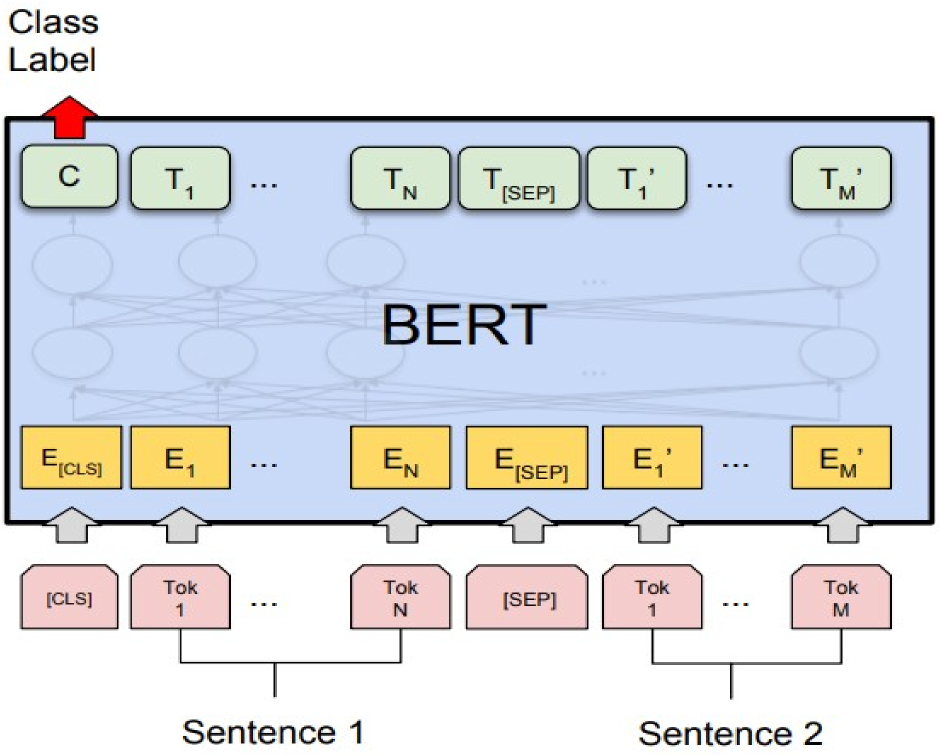
BERT

BERT for Sentence Pair Classification Task:

- BERT has fine-tuned its architecture for a number of sentence pair classification tasks such as:
- **MNLI:** Multi-Genre Natural Language Inference is a large-scale classification task. In this task, we have given a pair of the sentence. The goal is to identify whether the second sentence is entailment, contradiction or neutral with respect to the first sentence.
- **QQP:** Quora Question Pairs, In this dataset, the goal is to determine whether two questions are semantically equal.
- **QNLI:** Question Natural Language Inference, In this task the model needs to determine whether the second sentence is the answer to the question asked in the first sentence.
- **SWAG:** Situations With Adversarial Generations dataset contains 113k sentence classifications. The task is to determine whether the second sentence is the continuation of first or not.
-

BERT

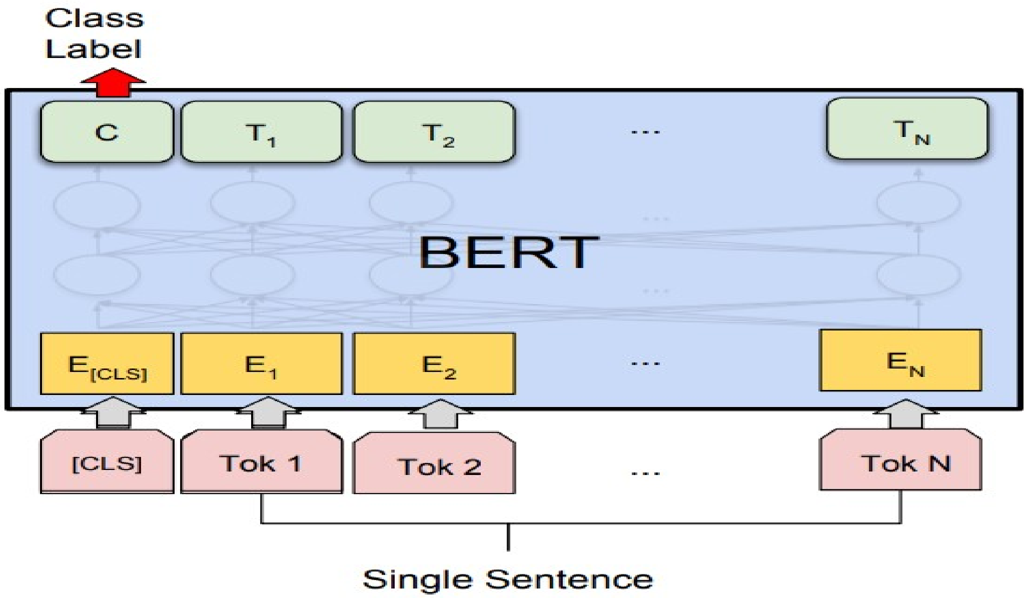
-


Class Label
C
T_1
...
T_N
$T_{[SEP]}$
T_1'
...
T_M'
BERT
$E_{[CLS]}$
E_1
...
E_N
$E_{[SEP]}$
E_1'
...
E_M'
[CLS]
Tok 1
...
Tok N
[SEP]
Tok 1
...
Tok M
Sentence 1
Sentence 2

BERT

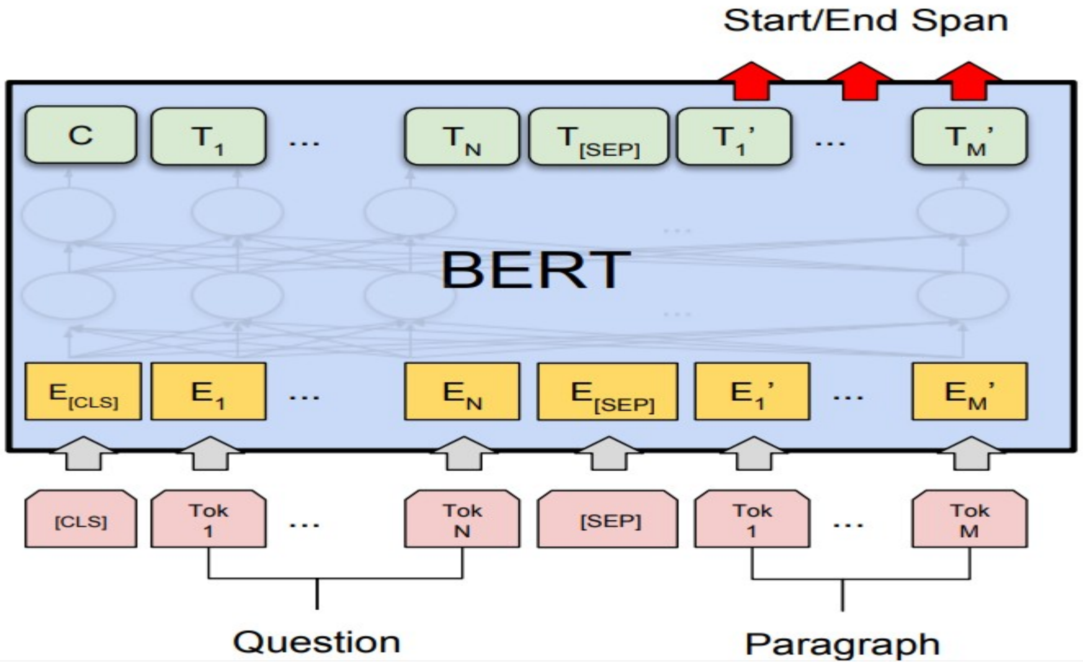
Single Sentence Classification Task :

- SST-2: The Stanford Sentiment Treebank is a binary sentence classification task consisting of sentences extracted from movie reviews with annotations of their sentiment representing in the sentence. BERT generated state-of-the-art results on SST-2.
- CoLA:The Corpus of Linguistic Acceptability is the binary classification task. The goal of this task to predict whether an English sentence that is provided is linguistically acceptable or not.
-



BERT

Question Answer Task: BERT has also generated state-of-the-art results Question Answering Tasks such as Stanford Question Answer Datasets (SQuAD v1.1 and SQuAD v2.0). In these Question Answering task, the model takes a question and passage. The goal is to mark the answer text span in the question.


Start/End Span
C
T_1
...
T_N
$T_{[SEP]}$
T_1'
...
T_M'
BERT
$E_{[CLS]}$
E_1
...
E_N
$E_{[SEP]}$
E_1'
...
E_M'
[CLS]
Tok 1
...
Tok N
[SEP]
Tok 1
...
Tok M
Question
Paragraph