

Based on GPT and ELMo

Current language models utilize different architectures and downstream application strategies:

- GPT employs a unidirectional (left-to-right) architecture and relies on fine-tuning with downstream data.
- ELMo utilizes an RNN architecture and serves as a feature-based approach (providing embeddings).

BERT introduces a novel approach by focusing on bidirectional information. Unlike GPT, which uses a shallow concatenation of independently trained left-to-right and right-to-left language models, BERT uses Masked Language Models (MLM) to enable pre-trained deep bidirectional representations.

Model Architecture

The architecture is a multi-layer bidirectional Transformer encoder. The scale of the model is defined by three main parameters:

- L: Number of layers
- H: Hidden size
- A: Number of attention heads

Two primary model sizes are mentioned: BERT base and BERT large.

- BERT base : L=12, H=768, A=12, total parameters=110M
- BERT large: L=24, H=1024, A=16, total parameters=340M

two strategies for applying pre-trained language representations to down-stream tasks:

- feature-based -----> embeddings ---> eg. ELMo
- fine-tuning -----> for downstream task ----> eg. GPT

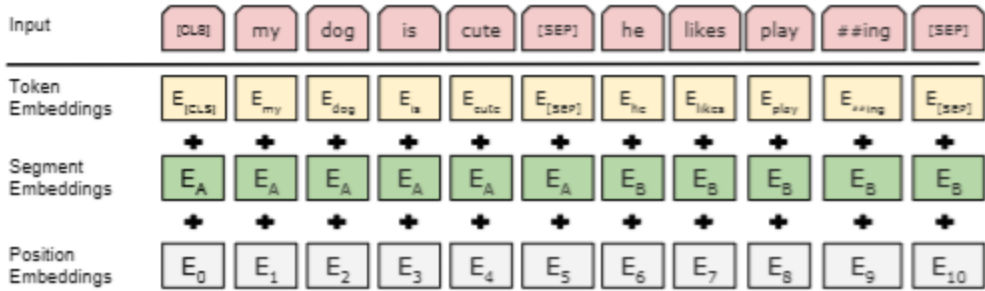
Input Representation

The model accepts either a single sentence or a pair of sentences packed into a single sequence.

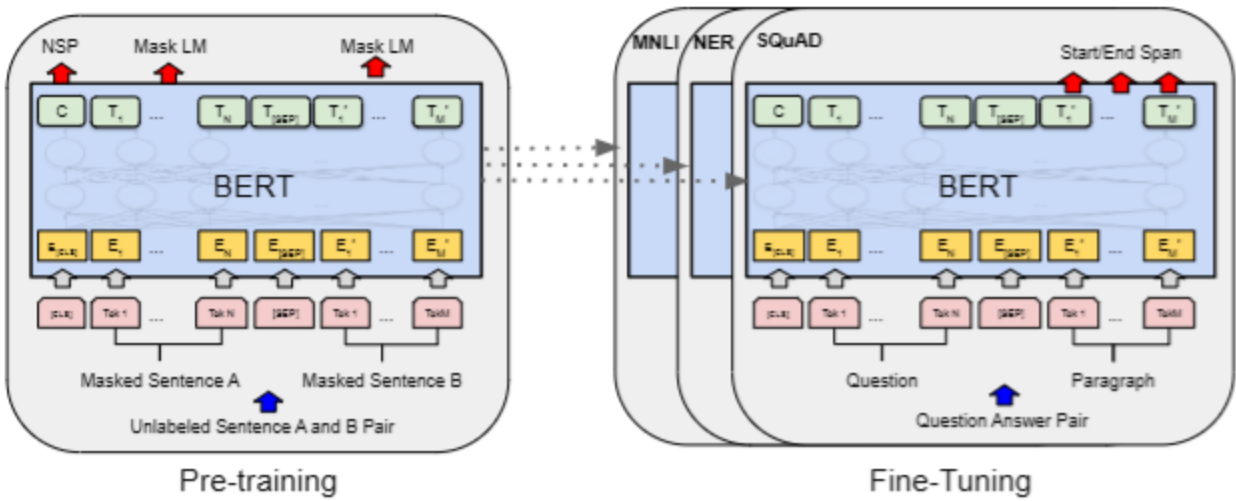
- The first token of every sequence is always [CLS] .
- Sentences within a pair are separated by a [SEP] token.
- A learned embedding is added to every token to indicate whether it belongs to sentence A or sentence B.

For any given token, its final input representation is constructed by summing three components:

- Token embeddings
- Segment embeddings
- Position embeddings Unlike the original Transformer where position representations were manually constructed, all vector representations in BERT are learned.



Pre-training Phrase



BERT is trained on unlabeled data over two different pre-training tasks:

Task 1: Masked Language Model (MLM)

The objective is to predict the original vocabulary ID of the masked word based only on its context. 15% of all WordPiece tokens in each sequence are chosen at random. Unlike denoising auto-encoders, BERT only predicts the masked words rather than reconstructing the entire input.

To address the discrepancy that the [MASK] token never appears during the fine-tuning phase, an 80-10-10 strategy is applied to the chosen 15% tokens:

- 80% of the time: Replaced with the [MASK] token.
- 10% of the time: Replaced with a random token.

- 10% of the time: Kept as the unchanged original token.
- 80% of the time: Replace the word with the [MASK] token, e.g., my dog is hairy → my dog is [MASK]
- 10% of the time: Replace the word with a random word, e.g., my dog is hairy → my dog is apple
- 10% of the time: Keep the word unchanged, e.g., my dog is hairy → my dog is hairy. The purpose of this is to bias the representation towards the actual observed word.

The model then predicts the original token using cross entropy loss. During the loss calculation, only the genuinely masked tokens participate. This is implemented using a mask vector (1 for masked, 0 for unmasked) multiplied by the token predictions and actuals, setting the loss of unmasked tokens to zero.

$$\text{Loss}_{\text{MLM}} = -\frac{1}{N} \sum_{i=1}^N \sum_{c=1}^V y_{i,c} \log(p_{i,c})$$

Task 2: Next Sentence Prediction (NSP)

This task helps the model capture sentence-level relationships.

Input = [CLS] the man went to [MASK] store [SEP]
he bought a gallon [MASK] milk [SEP]
Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]
penguin [MASK] are flight ##less birds [SEP]
Label = NotNext

$$\text{Loss}_{\text{NSP}} = -(y \log(p) + (1 - y) \log(1 - p))$$

Total Pre-training Loss:

$$\text{Loss}_{\text{total}} = \text{Loss}_{\text{MLM}} + \text{Loss}_{\text{NSP}}$$

Fine-tuning Phrase

For downstream tasks (both sentence-level and token-level, such as sentiment analysis and named entity recognition), the model is initialized with the pre-trained parameters.

Task-specific inputs and outputs are plugged into BERT, and a simple output layer (usually a linear projection layer) is added on top. All parameters, including the internal 12 or 24 layers, are fine-tuned using labeled data from the downstream tasks.

This process is conducted end-to-end, meaning the entire workflow from receiving raw input to producing final output occurs continuously within the same system, with all parameters jointly updated.

Experiments

- GLUE: Evaluated by using the final vector of the [CLS] token to learn an output layer.
- SQuAD v1.1: Also used for evaluation.

Limitation in Machine Translation: BERT's bidirectionality makes it unsuitable for Seq2Seq tasks like machine translation, which require both understanding and generation. Forcing BERT to only observe left-side information would negate its bidirectional advantage. Therefore, classic translation models typically adopt an Encoder-Decoder architecture instead.