BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

(**B**idirectional **E**ncoder **R**epresentations from **T**ransformers)

Jacob Devlin
Google AI Language

Pre-training in NLP

- Word embeddings are the basis of deep learning for NLP

`king` → `[-0.5, -0.9, 1.4, …]`

`queen` → `[-0.6, -0.8, -0.2, …]`

- Word embeddings (`word2vec`, `GloVe`) are often *pre-trained* on text corpus from co-occurrence statistics

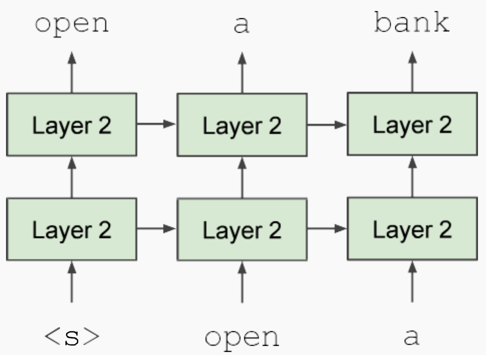
Problem with Previous Methods

- **Problem**: Language models only use left context *or* right context, but language understanding is bidirectional.
- Why are LMs unidirectional?
- Reason 1: Directionality is needed to generate a well-formed probability distribution.
  - We don't care about this.—because BERT is focused on language understanding, not generation.
- Reason 2: Words can "see themselves" in a bidirectional encoder. Information leakage

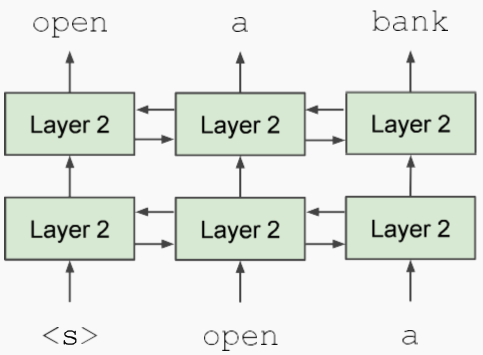
Traditional language models, like GPT or RNN-based models, required a sequential left-to-right or right-to-left approach to create a valid **probability distribution** for predicting the next word.This was essential for tasks like text generation but not as important for tasks requiring **understanding**, like classification or question answering.

Unidirectional vs. Bidirectional Models

Unidirectional context
Build representation incrementally



Bidirectional context
Words can “see themselves”



Masked LM

- **Solution:** Mask out *k*% of the input words, and then predict the masked words
  - We always use *k* = 15%

```
                      store           gallon
                        ↑               ↑
the man went to the [MASK] to buy a [MASK] of milk
```

- Too little masking: Too expensive to train
- Too much masking: Not enough context

Masked LM

- Problem: Mask token never seen at fine-tuning
- Solution: 15% of the words to predict, but don't replace with `[MASK]` 100% of the time. Instead:
- 80% of the time, replace with `[MASK]`

  `went to the store → went to the [MASK]`
- 10% of the time, replace random word

  `went to the store → went to the running`
- 10% of the time, keep same

  `went to the store → went to the store`

If BERT were trained to only predict words in the presence of [MASK] tokens, it might overfit to the idea that predictions are only needed when [MASK] tokens exist.

In real-world tasks, **predictions** are needed for all tokens, even without [MASK]

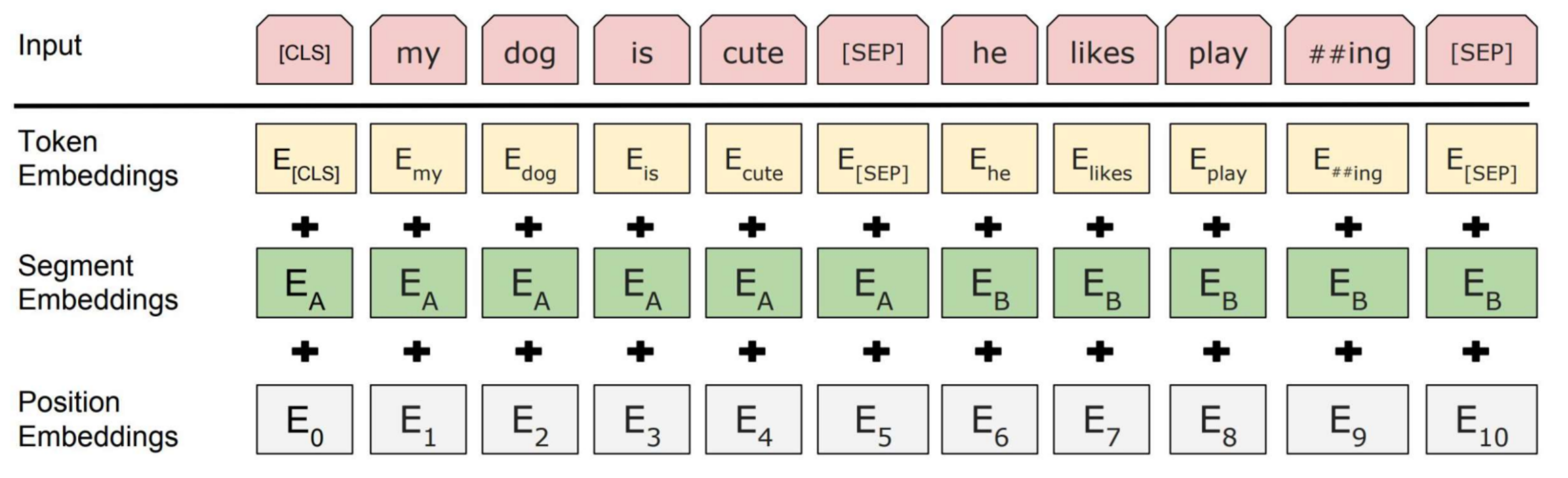
Next Sentence Prediction

- To learn *relationships* between sentences, predict whether Sentence B is actual sentence that proceeds Sentence A, or a random sentence

Sentence A = `The man went to the store.`
Sentence B = `He bought a gallon of milk.`
Label = `IsNextSentence`

Sentence A = `The man went to the store.`
Sentence B = `Penguins are flightless.`
Label = `NotNextSentence`

Input Representation

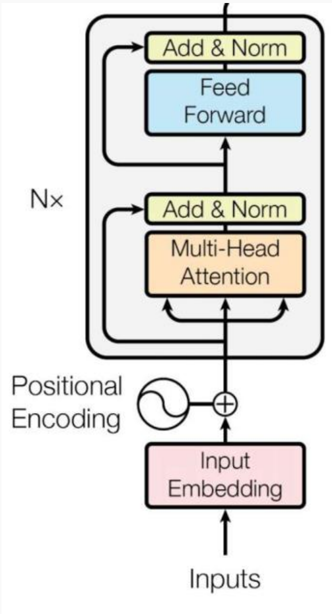


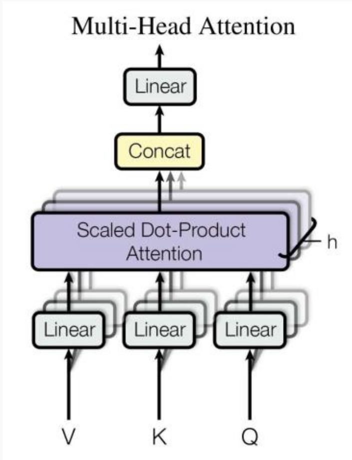
- Use 30,000 WordPiece vocabulary on input.
- Each token is sum of three embeddings
- Single sequence is much more efficient.

Model Architecture

Transformer encoder

- Multi-headed self attention
  - Models context
- Feed-forward layers
  - Computes non-linear hierarchical features
- Layer norm and residuals
  - Makes training deep networks healthy
- Positional embeddings
  - Allows model to learn relative positioning

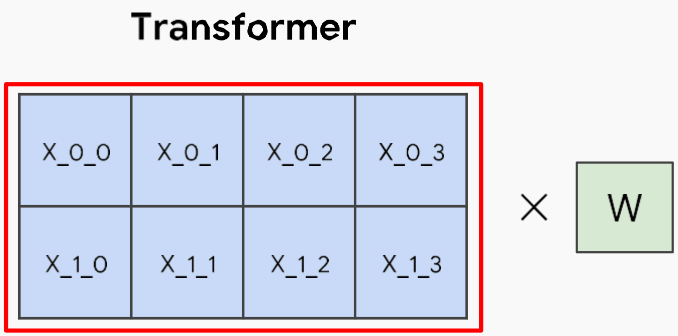




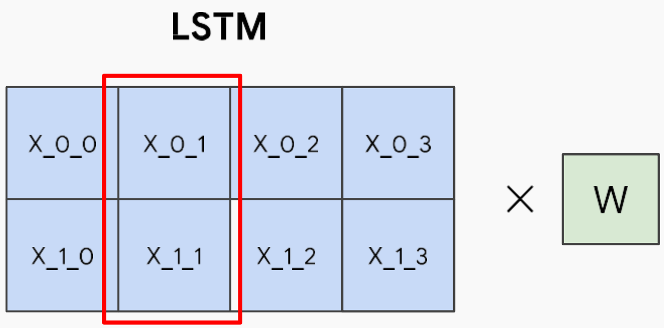
Model Architecture

- Empirical advantages of Transformer vs. LSTM:

1. Self-attention == no locality bias
    - Long-distance context has “equal opportunity”
2. Single multiplication per layer == efficiency on TPU
    - Effective batch size is number of *words*, not *sequences*



Each input has equal access to **all other tokens in the sequence** due to the self-attention mechanism



At each time step, the computation depends only on the current input and the hidden state from the previous time step.
•This introduces a **locality bias**, as tokens farther away from the current token have a diminishing influence due to the vanishing gradient problem.

Model Details

- Data: Wikipedia (2.5B words) + BookCorpus (800M words)
- Batch Size: 131,072 words (1024 sequences * 128 length or 256 sequences * 512 length)
- Training Time: 1M steps (~40 epochs)
- Optimizer: AdamW, 1e-4 learning rate, linear decay
- `BERT-Base`: 12-layer, 768-hidden, 12-head
- `BERT-Large`: 24-layer, 1024-hidden, 16-head
- Trained on 4x4 or 8x8 TPU slice for 4 days

Fine-Tuning Procedure