Chapter 13

Transformers

Transformers stand as a groundbreaking evolution in deep learning, reshaping how machines process and understand sequences. Earlier models like LSTMs and GRUs operated by passing information step by step through a hidden state, which often limited their ability to capture long-range dependencies. Even with enhancements such as attention and teacher forcing, these architectures struggled to retain information across long sentences or documents. The Transformer, however, discards recurrence entirely, using self-attention to examine every element in a sequence simultaneously. This parallelization allows the model to link any token to every other token efficiently, making it both faster and more capable of modeling context over long spans. The encoder-decoder design (figure 1) reflects this innovation: the encoder produces rich latent representations through stacks of attention and feed-forward layers, while the decoder uses masked self-attention and encoder-decoder attention to generate coherent outputs without relying on sequential memory.

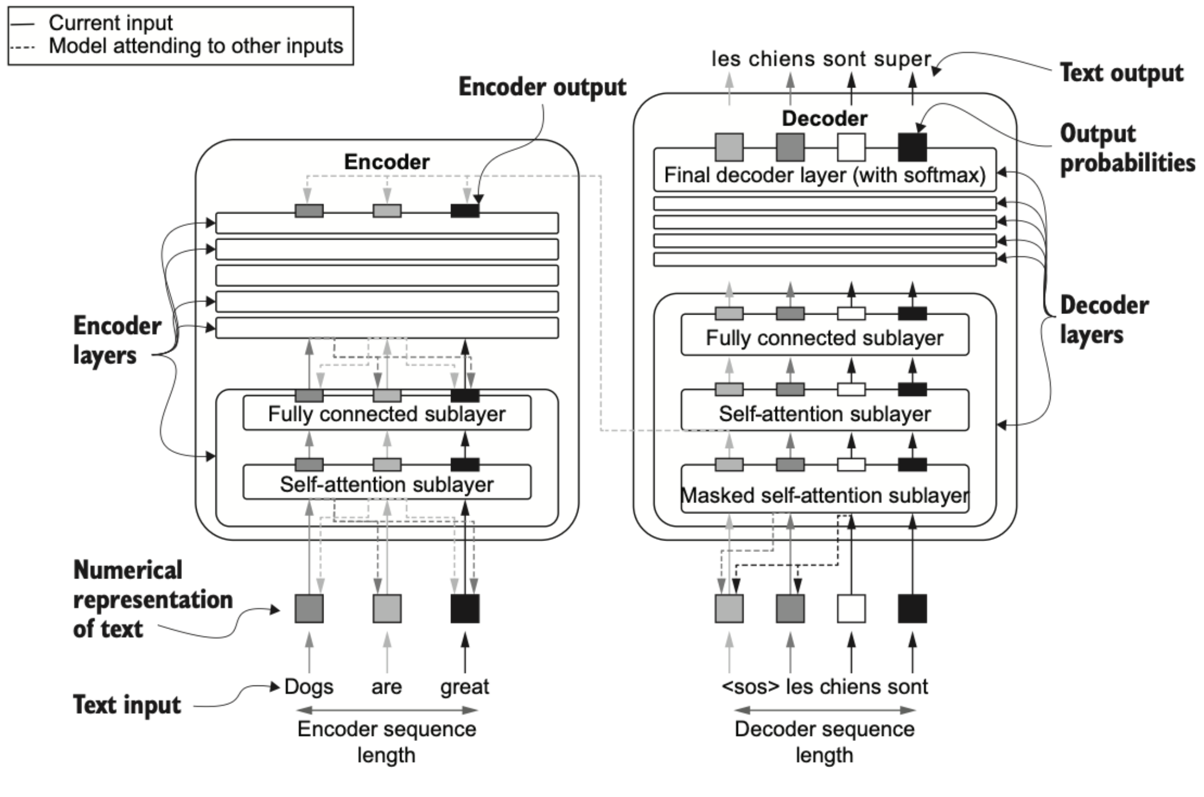


Figure 1 How Transformer models are used to solve NLP problems

At the heart of this mechanism lies self-attention, where tokens generate queries, keys, and values to determine how much they should borrow meaning from surrounding words. Through a weighted aggregation, each token's representation is enriched by the context of the entire sequence. Yet attention alone is not enough; embeddings ensure that words are translated into meaningful vectors. Token embeddings provide semantic representation, while positional embeddings encode order, since the Transformer processes all tokens in parallel and lacks inherent sequentiality. These positional encodings, often sinusoidal in form, vary smoothly with position and feature dimension, creating unique patterns that give the model a sense of sequence (figure 2). By summing token and positional embeddings, final hybrid representations emerge, enabling the model to combine meaning with order (figure 3). Training stability is maintained through residual connections and layer normalization, which improve gradient flow and counteract distribution shifts that otherwise hinder deep networks.

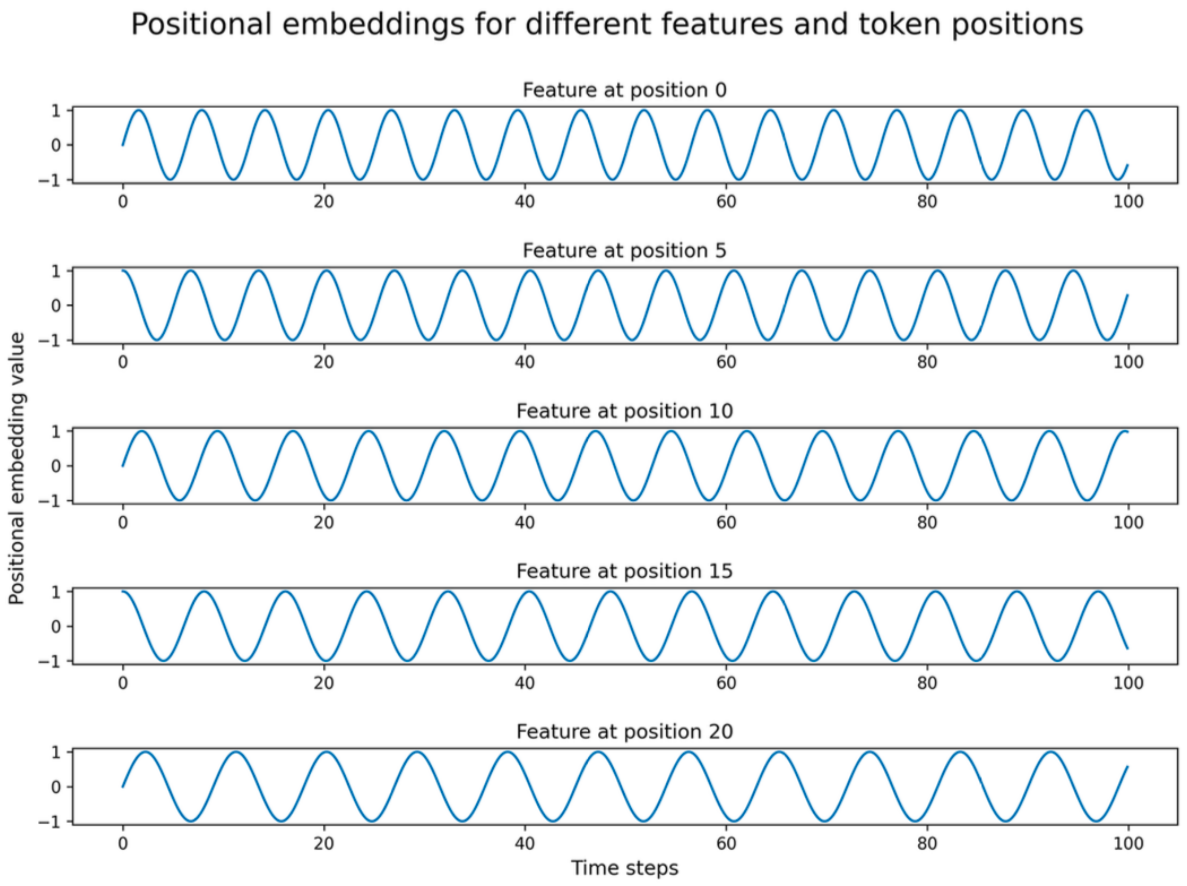


Figure 2 How positional embeddings change with the time step and the feature position. Even numbered feature positions use the sine function, whereas odd numbered positions use the cosine function

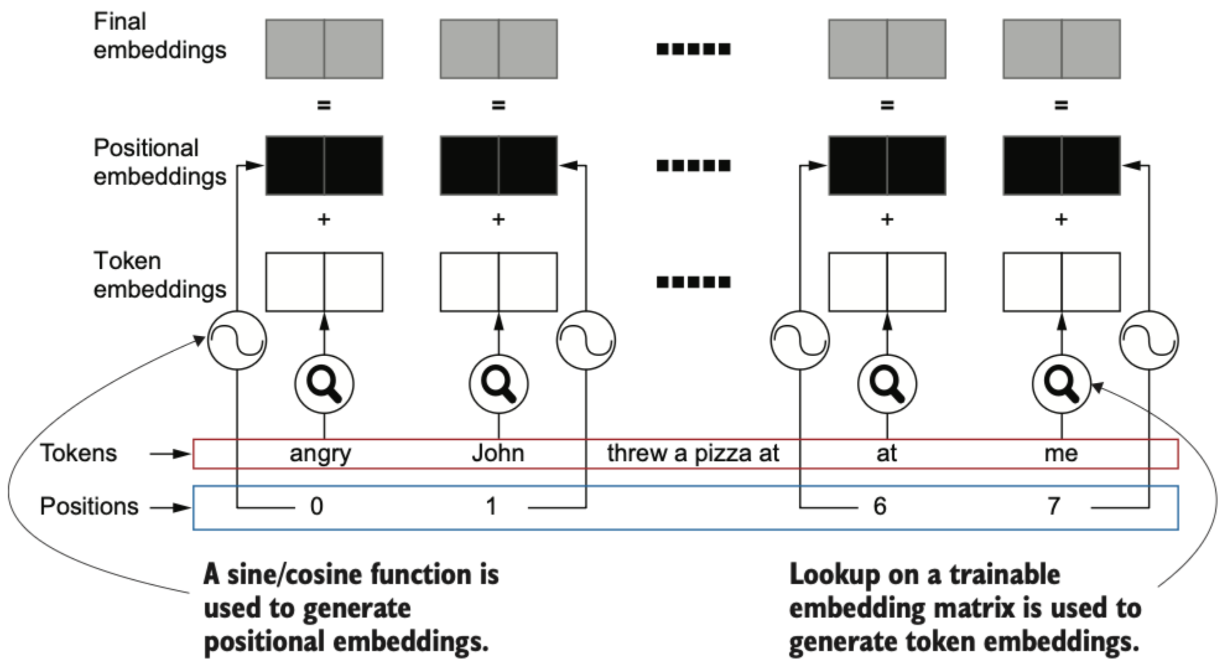

Final embeddings
Positional embeddings
Token embeddings
Tokens
Positions
angry
John
threw a pizza at
at
me
0
1
6
7
A sine/cosine function is used to generate positional embeddings.
Lookup on a trainable embedding matrix is used to generate token embeddings.

Figure 3 The embeddings generated in a Transformer model and how the final embeddings are computed

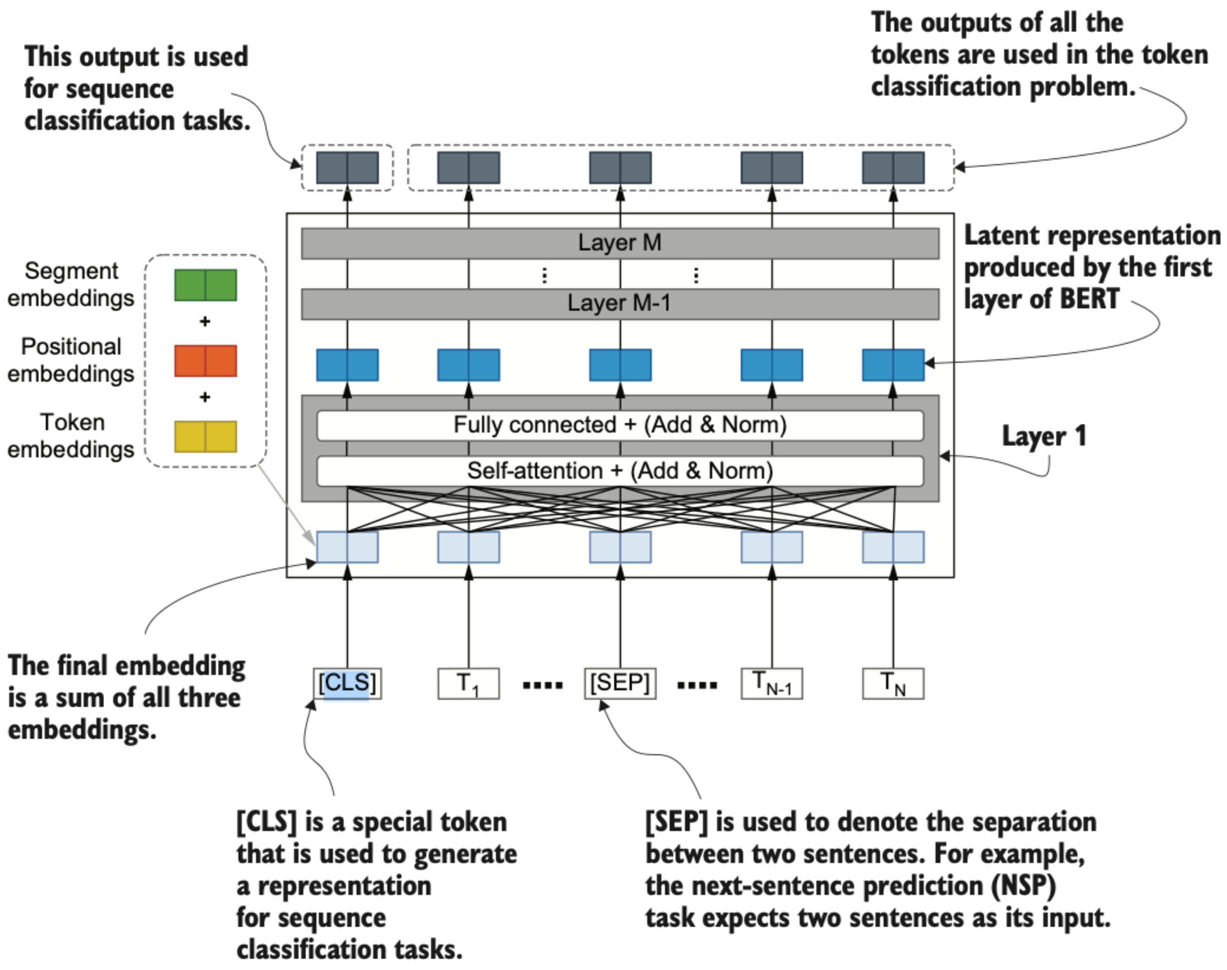
From this foundation, powerful pretrained models like BERT emerged. Built solely on the Transformer's encoder, BERT introduced a new paradigm of transfer learning in language. Instead of training models from scratch for each task, BERT is pretrained on vast corpora using two objectives. The first, masked language modeling, randomly hides tokens in a sentence and requires the model to predict them, teaching deep contextual understanding. The second, next-sentence prediction, trains the model to recognize whether one sentence logically follows another, enabling it to capture discourse-level coherence. The architecture (figure 4) combines token, positional, and segment embeddings, with special markers like [CLS] and [SEP] guiding classification and sequence separation. Once pretrained, BERT can be adapted to a variety of tasks with minimal fine-tuning: the [CLS] vector provides a sequence-level representation for classification, while token-level outputs support labeling and span prediction.

Reproduced code for encoding an input string with BERT's tokenizer:

```
from official.nlp import bert as tfm


def get_bert_inputs(tokenizer, docs, max_seq_len=None):
    packer = tfm.nlp.layers.BertPackInputs(
        seq_length=max_seq_len,
        special_tokens_dict=tokenizer.get_special_tokens_dict()
    )
    packed = packer(tokenizer(docs))


    packed_numpy = {k: v.numpy() for k, v in packed.items()}
    return packed_numpy
```



This output is used for sequence classification tasks.
The outputs of all the tokens are used in the token classification problem.
Layer M
Layer M-1
Latent representation produced by the first layer of BERT
Segment embeddings
Positional embeddings
Token embeddings
Fully connected + (Add & Norm)
Self-attention + (Add & Norm)
Layer 1
The final embedding is a sum of all three embeddings.
[CLS]
T1
[SEP]
TN-1
TN
[CLS] is a special token that is used to generate a representation for sequence classification tasks.
[SEP] is used to denote the separation between two sentences. For example, the next-sentence prediction (NSP) task expects two sentences as its input.

Figure 4 The high-level architecture of BERT. It takes a set of input tokens and produces a sequence of hidden representations generated using several hidden layers

The strength of this approach becomes evident when applied to a real-world problem like spam detection. A dataset of SMS messages labeled as spam or ham reveals a common challenge: severe class imbalance, with far fewer spam samples. To address this, resampling strategies are employed, including random undersampling and more refined techniques like the near-miss algorithm, which selectively removes majority-class samples close to the minority class to sharpen distinctions. Figure 5 illustrates how balanced training, validation, and test sets can be constructed. Tokenization is handled by WordPiece, which splits rare words into sub-words, reducing vocabulary size and ensuring that unseen words can still be represented. Inputs are encoded with token IDs, attention masks, and segment IDs, with special tokens inserted as required. On top of a pretrained BERT encoder from TensorFlow Hub, a simple classification head is added. With only a few epochs of fine-tuning, accuracy climbs rapidly, surpassing 80% on validation and achieving strong generalization on test data. This efficiency stems from BERT's deep pretrained knowledge of language, which frees the model to focus on the specific classification task rather than learning linguistic patterns from scratch.

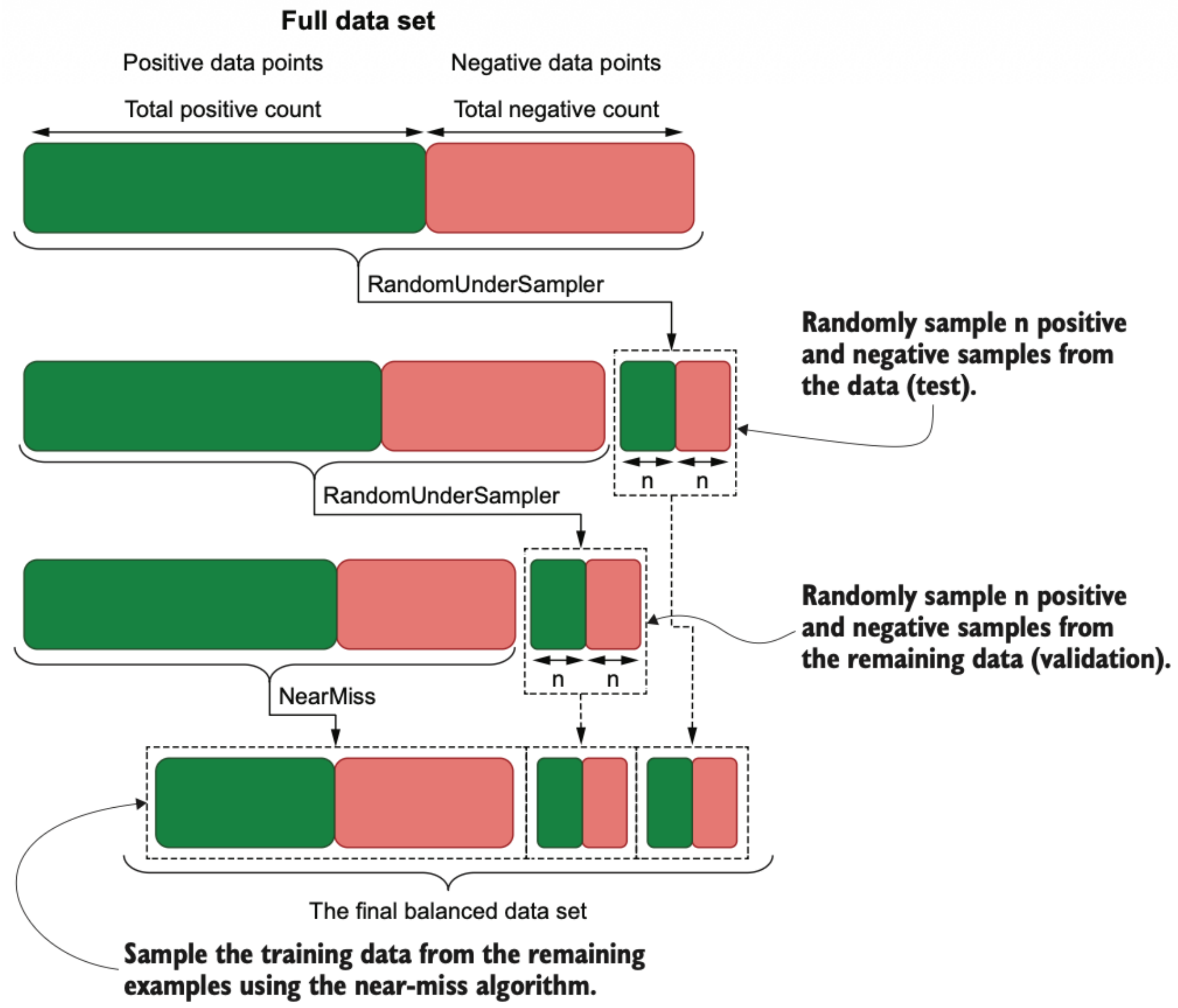

Full data set
Positive data points
Negative data points
Total positive count
Total negative count
RandomUnderSampler
Randomly sample n positive and negative samples from the data (test).
n
n
RandomUnderSampler
Randomly sample n positive and negative samples from the remaining data (validation).
n
n
NearMiss
The final balanced data set
Sample the training data from the remaining examples using the near-miss algorithm.

Figure 5 How the training, validation, and testing data sets are created from the original data set

Beyond spam filtering, the same architecture extends naturally to more complex applications such as question answering. In this setting, the input combines a question and a passage separated by [SEP], and the model predicts start and end indices marking the answer span. Hugging Face's Transformers library streamlines this process, offering pretrained models and tokenizers that can be fine-tuned with minimal effort. A question like "What color is the ball?"

paired with a passage about a dog and its red ball enables the model to attend to both sequences simultaneously, using segment embeddings to distinguish question from context. The result is an extracted answer, directly grounded in the passage. Such flexibility demonstrates the adaptability of the Transformer framework: whether classifying entire sequences, labeling tokens, or extracting spans, the same architecture applies with only slight adjustments.

The influence of Transformers has extended well beyond text. Self-attention mechanisms have proven equally effective in computer vision, where they are used to model relationships across image patches instead of words. By rethinking spatial data as sequences, these models rival and sometimes surpass convolutional networks in classification, detection, and segmentation tasks. This cross-domain success underscores the universality of the approach: self-attention is not merely a language innovation but a fundamental principle for understanding structured data.