Chapter 5

State-of-the-art in Deep Learning: Transformers

Transformers, a deep learning architecture that has surpassed CNNs and RNNs in natural language processing (NLP). Introduced by Vaswani et al. in "Attention Is All You Need" (2017), Transformers rely on attention mechanisms instead of recurrence or convolutions. Before diving into the architecture, text must be converted into numbers. Tokenization maps words to IDs, then vectors are constructed through one-hot encoding or embeddings. To handle variable-length sentences, padding and truncation are used. This produces fixed-length matrices suitable for neural networks, as shown in figure 1.

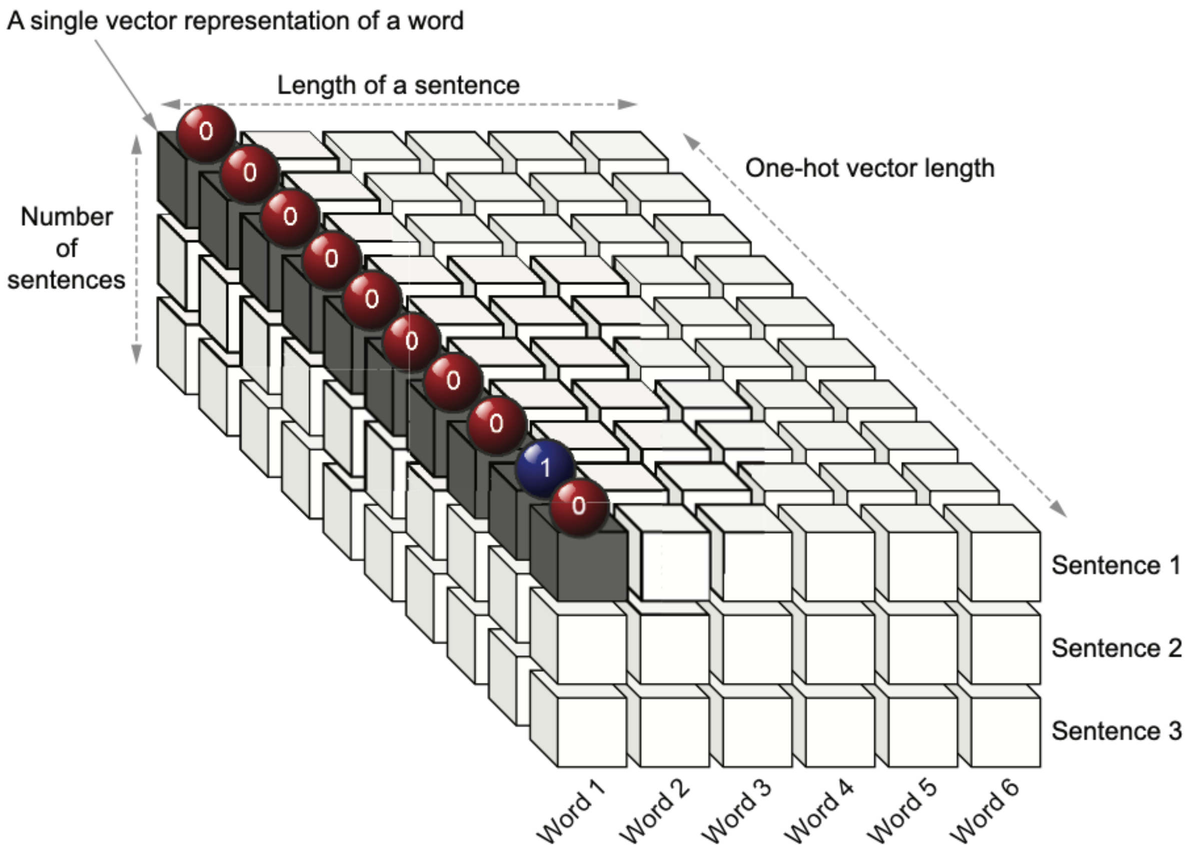

Figure 1 The boxes in the Word Boxes game. The shaded box represent a single word

The Transformer architecture follows an encoder–decoder design figure 2. The encoder converts input sequences (e.g., English text) into latent representations, while the decoder generates outputs step by step (e.g., French translation). Each encoder block has two sublayers: a self-attention mechanism and a fully connected feed-forward network. Self-attention allows each word to consider all other words in a sentence simultaneously, which overcomes the memory limitations of RNNs. The decoder, in contrast, contains three sublayers: masked self-attention, encoder–decoder attention, and a fully connected layer. Masking ensures that during training the decoder cannot “see” future words, which prevents cheating figure 3.

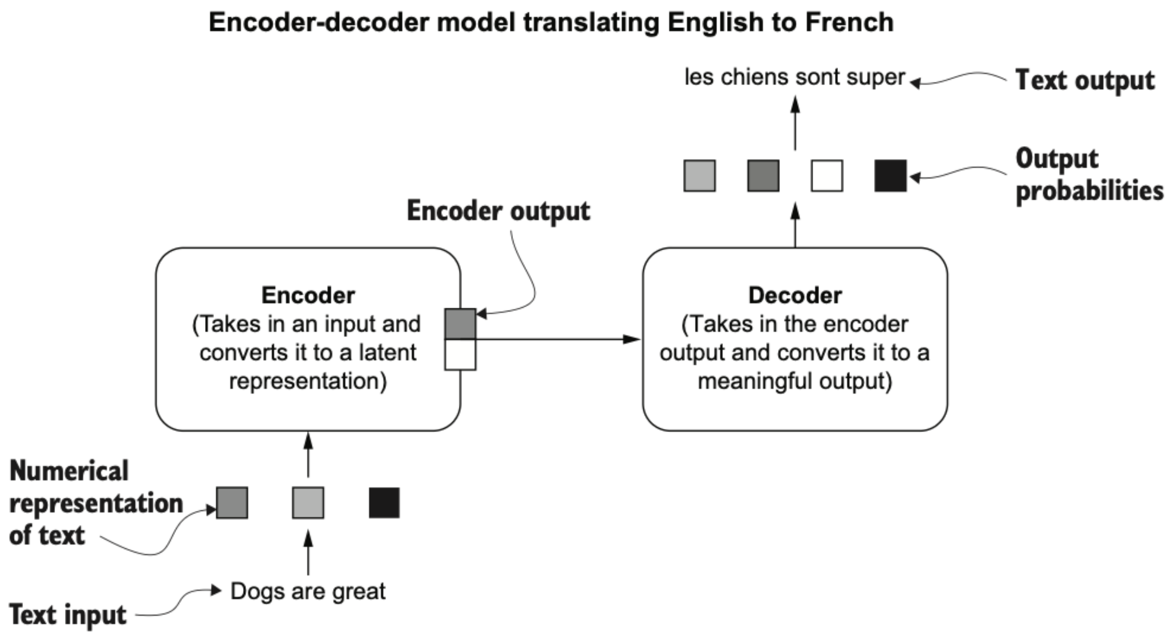
Encoder-decoder model translating English to French
les chiens sont super
Text output
Output probabilities
Encoder output
Encoder
(Takes in an input and converts it to a latent representation)
Decoder
(Takes in the encoder output and converts it to a meaningful output)
Numerical representation of text
Dogs are great
Text input

Figure 2 The encoder-decoder architecture for a machine translation task

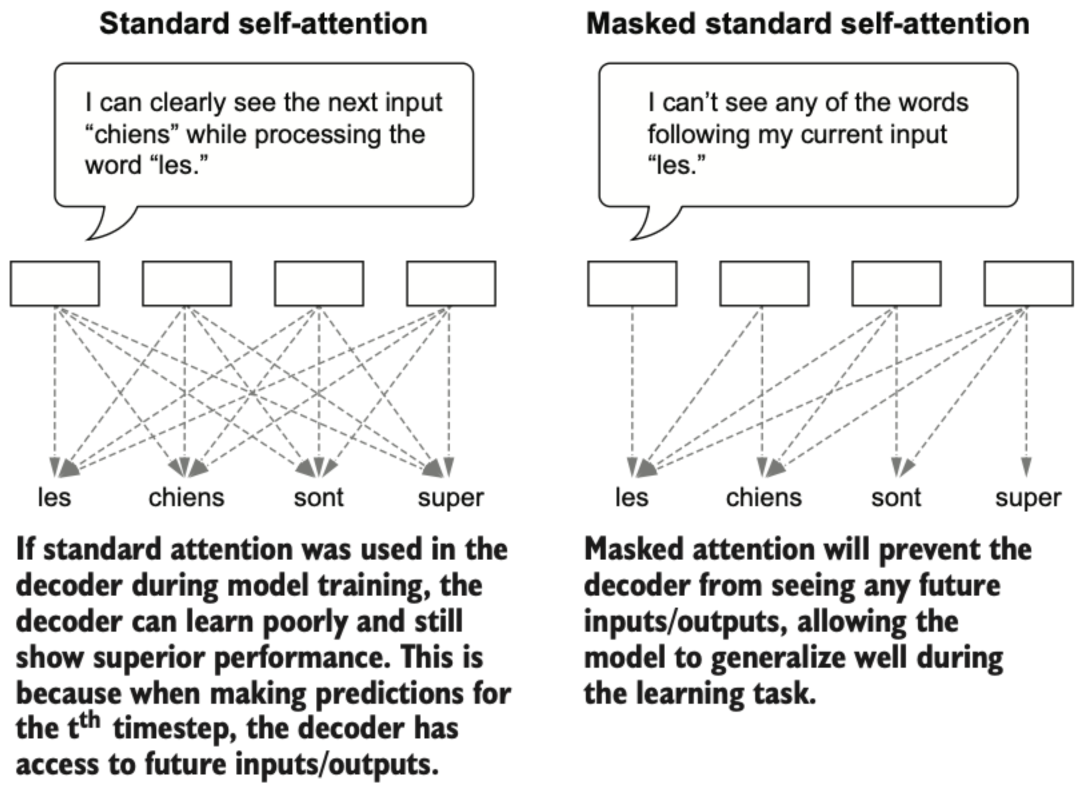
Standard self-attention
Masked standard self-attention
I can clearly see the next input "chiens" while processing the word "les."
I can't see any of the words following my current input "les."
les
chiens
sont
super
les
chiens
sont
super
If standard attention was used in the decoder during model training, the decoder can learn poorly and still show superior performance. This is because when making predictions for the tth timestep, the decoder has access to future inputs/outputs.
Masked attention will prevent the decoder from seeing any future inputs/outputs, allowing the model to generalize well during the learning task.

Figure 3 Standard self-attention versus masked self-attention methods

The self-attention mechanism is the core of the model. Each word is projected into three vectors: query (Q), key (K), and value (V). Attention scores are computed by scaled dot products between queries and keys, followed by a softmax that produces probability weights. These weights determine how much focus each word places on others when generating context-aware representations. Figure 4 provides a clear visualization of this process.

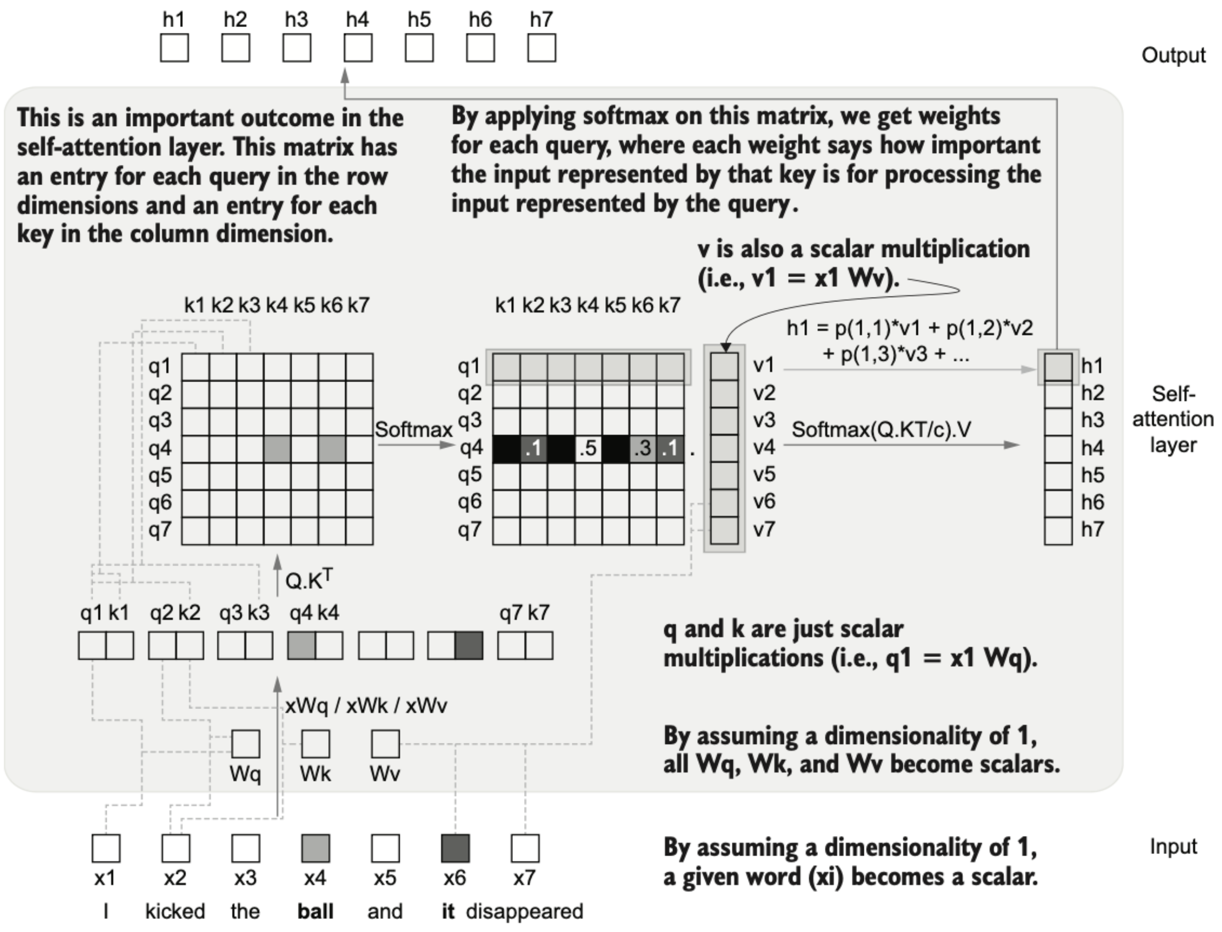
h1 h2 h3 h4 h5 h6 h7
Output
This is an important outcome in the self-attention layer. This matrix has an entry for each query in the row dimensions and an entry for each key in the column dimension.
By applying softmax on this matrix, we get weights for each query, where each weight says how important the input represented by that key is for processing the input represented by the query.
v is also a scalar multiplication (i.e., v1 = x1 Wv).
k1 k2 k3 k4 k5 k6 k7
q1 q2 q3 q4 q5 q6 q7
Softmax
.1 .5 .3 .1
v1 v2 v3 v4 v5 v6 v7
h1 = p(1,1)*v1 + p(1,2)*v2 + p(1,3)*v3 + ...
Softmax(Q.KT/c).V
Self-attention layer
Q.KT
q1 k1 q2 k2 q3 k3 q4 k4 q7 k7
q and k are just scalar multiplications (i.e., q1 = x1 Wq).
xWq / xWk / xWv
Wq Wk Wv
By assuming a dimensionality of 1, all Wq, Wk, and Wv become scalars.
x1 x2 x3 x4 x5 x6 x7
I kicked the ball and it disappeared
By assuming a dimensionality of 1, a given word (xi) becomes a scalar.
Input

Figure 4 The computations in the self-attention layer. The self-attention layers starts with an input sequence and computes sequences of query, key, and value vectors

Reproduced code for self-attention layer:

```
import tensorflow as tf

import tensorflow.keras.layers as layers

import math


class SelfAttentionLayer(layers.Layer):

    def __init__(self, d):

        super(SelfAttentionLayer, self).__init__()

        self.d = d


    def build(self, input_shape):
        self.Wq = self.add_weight(
            shape=(input_shape[-1], self.d),
            initializer='glorot_uniform',
            trainable=True, dtype='float32'
        )
        self.Wk = self.add_weight(
            shape=(input_shape[-1], self.d),
            initializer='glorot_uniform',
            trainable=True, dtype='float32'
        )
        self.Wv = self.add_weight(
            shape=(input_shape[-1], self.d),
            initializer='glorot_uniform',
            trainable=True, dtype='float32'
        )

    def call(self, q_x, k_x, v_x, mask=None):
        q = tf.matmul(q_x, self.Wq)
        k = tf.matmul(k_x, self.Wk)
        v = tf.matmul(v_x, self.Wv)

        p = tf.matmul(q, k, transpose_b=True) / math.sqrt(self.d)

        if mask is not None:
            p += mask * -1e9

        p = tf.nn.softmax(p)
        h = tf.matmul(p, v)

        return h, p
```

To improve performance, Transformers use multi-head attention, which runs several self-attention operations in parallel with reduced dimensions and then concatenates the outputs. This allows the model to capture different relational patterns within the data. Fully connected layers follow attention layers to apply non-linear transformations, making the representations richer.

Finally, to demonstrates building a mini Transformer in TensorFlow/Keras. Using custom layers for self-attention and feed-forward networks, by implements encoder and decoder layers, stacks them, and defines embeddings for input tokens. The details are:

1. Self-attention layer
   - Implements scaled dot-product attention.
   - Each input embedding is projected into Query (Q), Key (K), and Value (V) vectors via trainable weight matrices.
   - Attention scores are computed as:

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

   - Softmax normalizes the scores so they sum to 1, producing weighted combinations of values.
2. Feed-forward layer
   - Each encoder and decoder block also includes a fully connected position-wise feed-forward network (two dense layers with ReLU activation in between).
   - This provides non-linearity and allows more complex feature transformations.
3. Encoder block
   - Each encoder block is composed of:
     - Multi-head self-attention
     - Add & Normalize (residual connection + layer normalization)
     - Feed-forward network
     - Another Add & Normalize
   - Multiple encoder blocks are stacked to form the encoder.
4. Decoder block
   - Each decoder block includes three sublayers:
     - Masked self-attention → prevents attending to future tokens.
     - Encoder–decoder attention → allows the decoder to focus on encoder outputs.
     - Feed-forward network.
   - Each is followed by Add & Normalize.
   - Several decoder blocks are stacked to form the decoder.

5. Embedding and positional encoding
   - Since Transformers have no recurrence or convolution, they use positional encoding (sinusoidal functions or learned embeddings) to encode word order.
   - The model uses embedding layers to map token IDs to dense vectors before feeding them into the encoder and decoder.
6. Final model assembly
   - **Inputs:** tokenized sequences for source (encoder) and target (decoder).
   - **Encoder:** processes source embeddings into context vectors.
   - **Decoder:** generates target sequences step by step, attending to both previous target tokens and encoder outputs.
   - **Output layer:** a dense softmax layer predicts probabilities over the vocabulary.

Figure 5 shows the overall architecture, while the model summary confirms that this small-scale Transformer already contains over 13 million parameters.

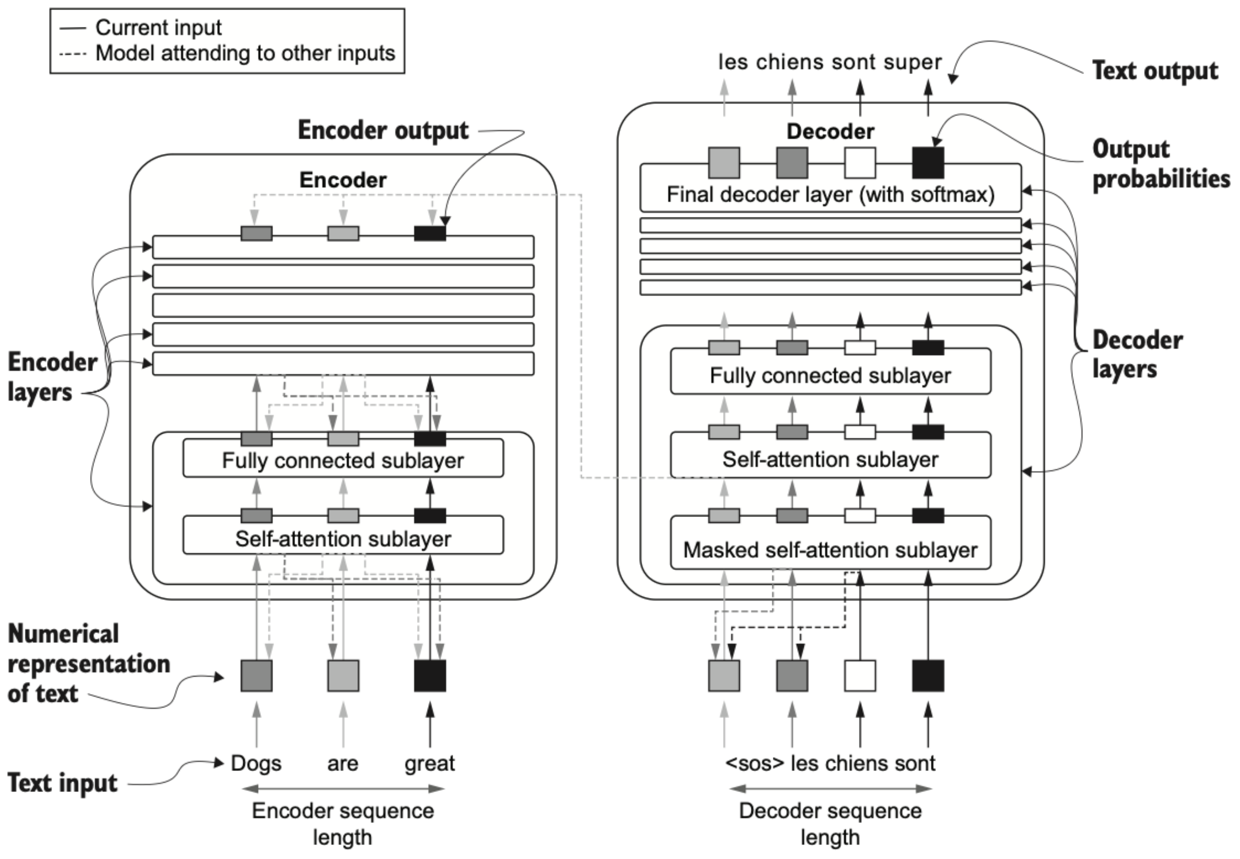

Figure 5 The transformer model architecture

In conclusion, Transformers represent a paradigm shift in NLP by enabling models to efficiently learn long-range dependencies. Key insights include: (1) the encoder–decoder framework, (2) self-attention for global context, (3) masked attention for proper sequence prediction, and (4) multi-head attention for diverse relational learning.