Attention-based Networks

AI+X
INSTITUTE FOR ARTIFICIAL INTELLIGENCE

Attention-based Networks

- The most popular network, the Transformer, was proposed in the paper "Attention is All You Need"
- It is an encoder-decoder architecture, but many applications only use the encoding part
    - Sequence-to-sequence mapping

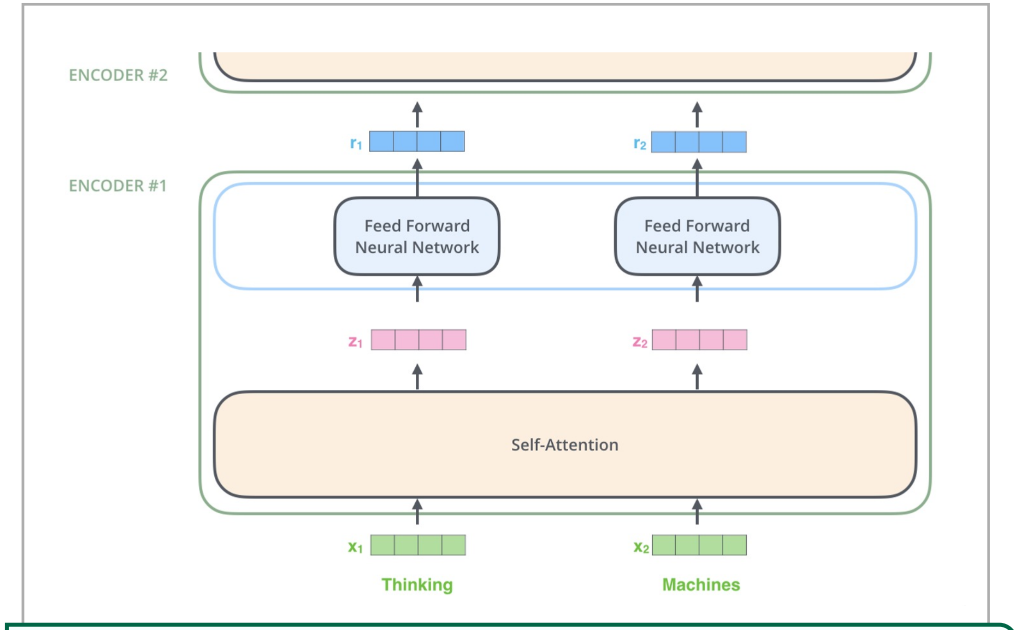

Figure 1. Encoder block of a Transformer network.

Attention-based Networks

- The sequence of word embeddings passes through a self-attention process
- Then, each updated embedding passes through a feed-forward neural network

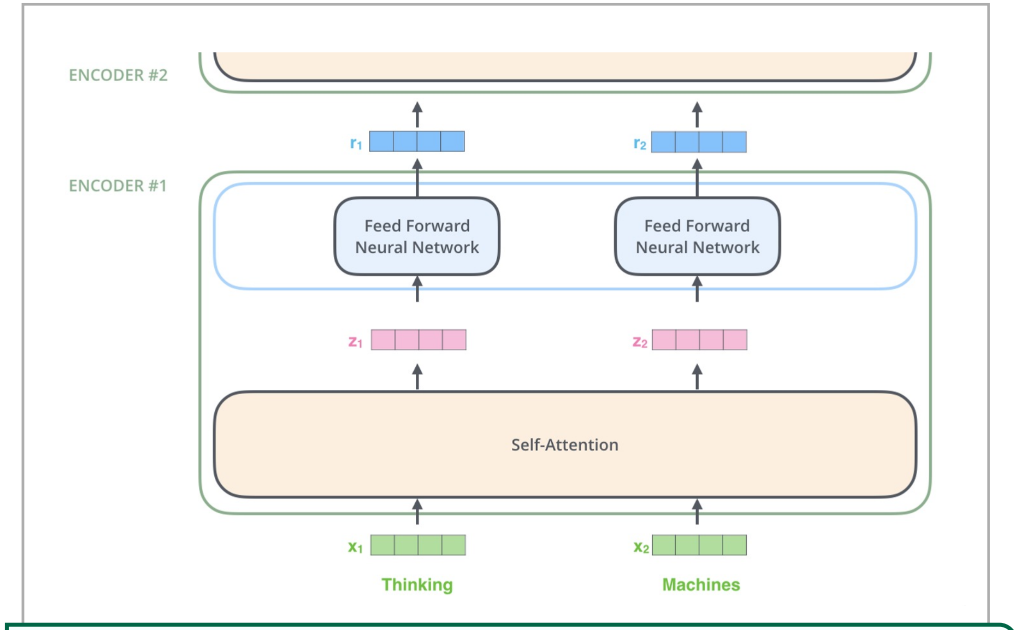

Figure 2. Encoder block of a Transformer network.

Self-attention module

- Self-attention looks at the entire sequence at once
    - RNN looks into one piece of the sequence at a time
- What does "it" refer to? Is it referring to the street or to the animal?
- Self-attention looks at other positions in the input sequence for clues that can help lead to a better encoding for each word

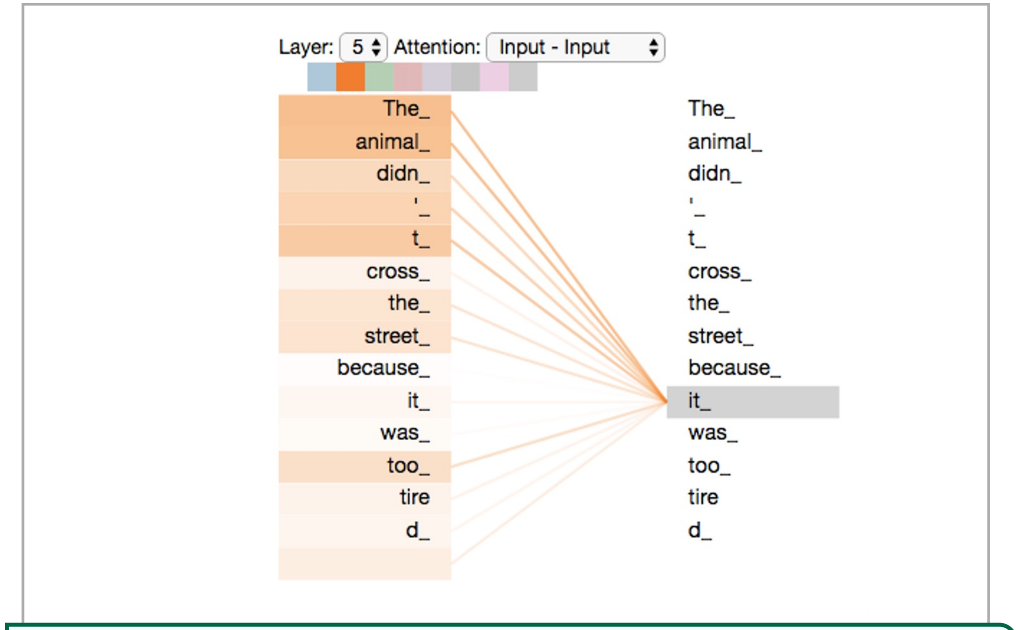

Figure 3. Illustration of the self-attention process.

Self-attention Module

- Create three vectors from each input vector
  - a Query vector
    - What you are looking for to complement this word
  - a Key vector
    - The meaning this word has to offer
  - a Value vector
    - The value this word will contribute to the output vector

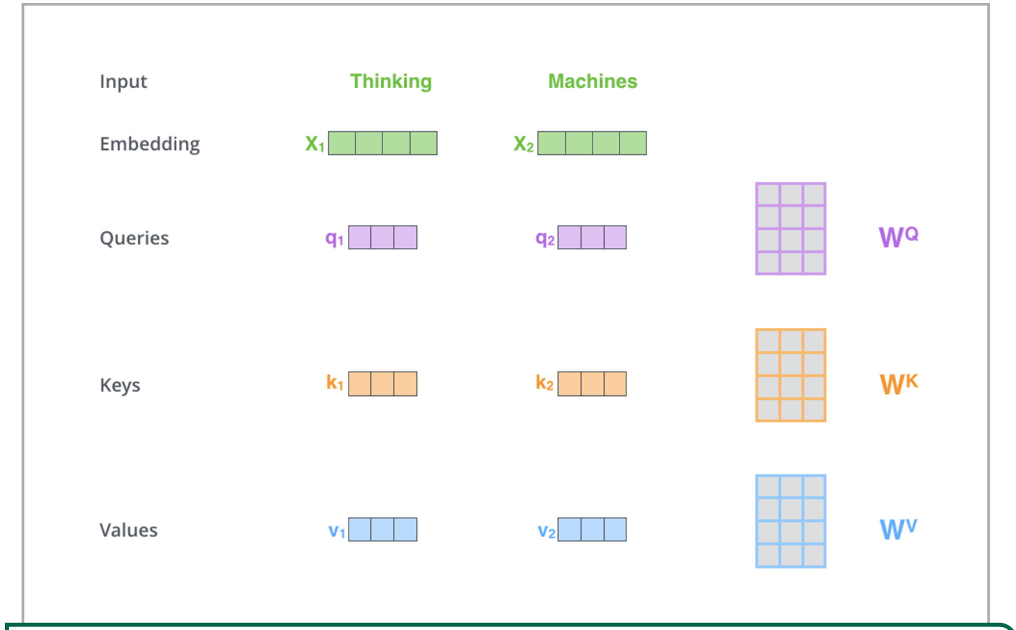

Figure 4. First step of the self-attention module: creation of queries, keys and values for each embedding.

Self-attention Module

- We use dot product between queries and keys to decide how each word will contribute to the output vector of a certain word
- When there is a high similarity between the query of word A and key of word B, the value of word B will have a higher impact in the output vector of word A

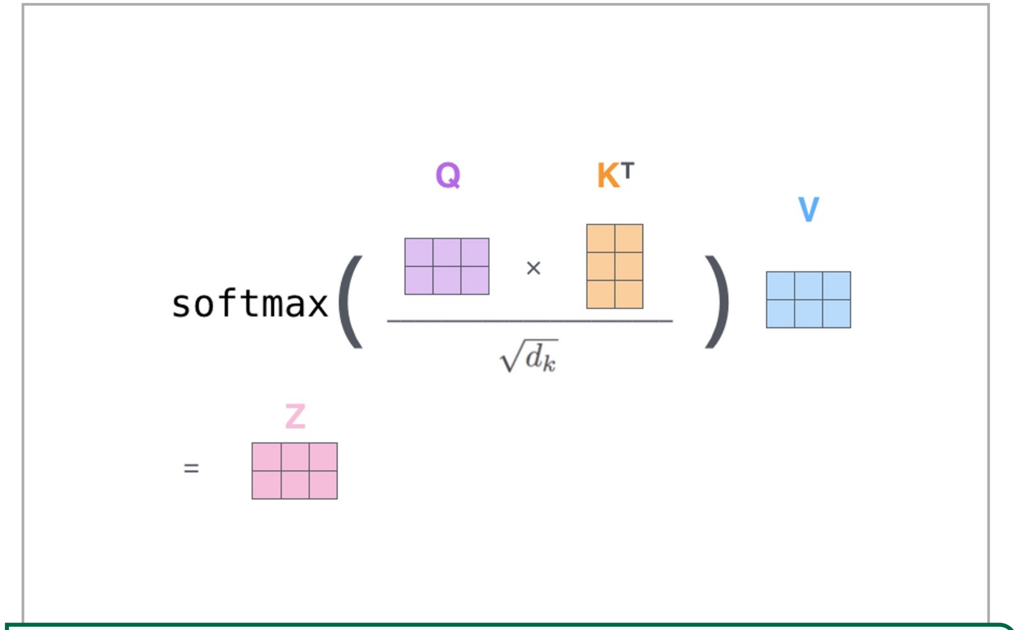

Figure 5. Second step of the self-attention module: use queries, keys and values to update each word embedding.

Self-attention Module

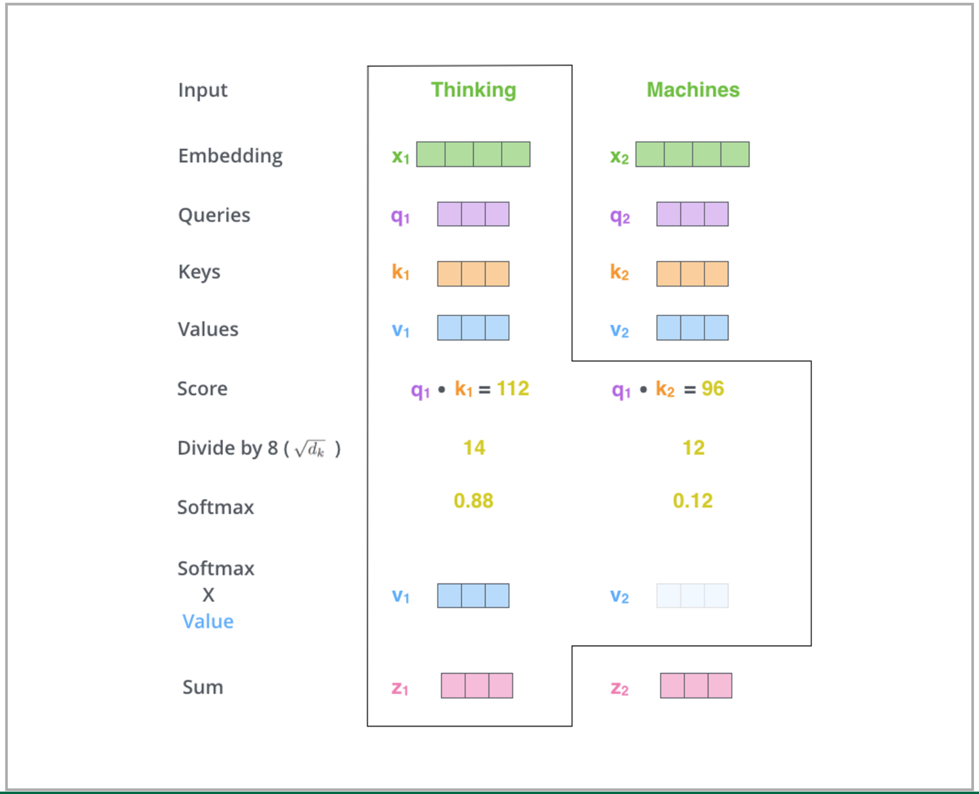

Figure 6. Visual example of the self-attention module operation.

Self-attention Module

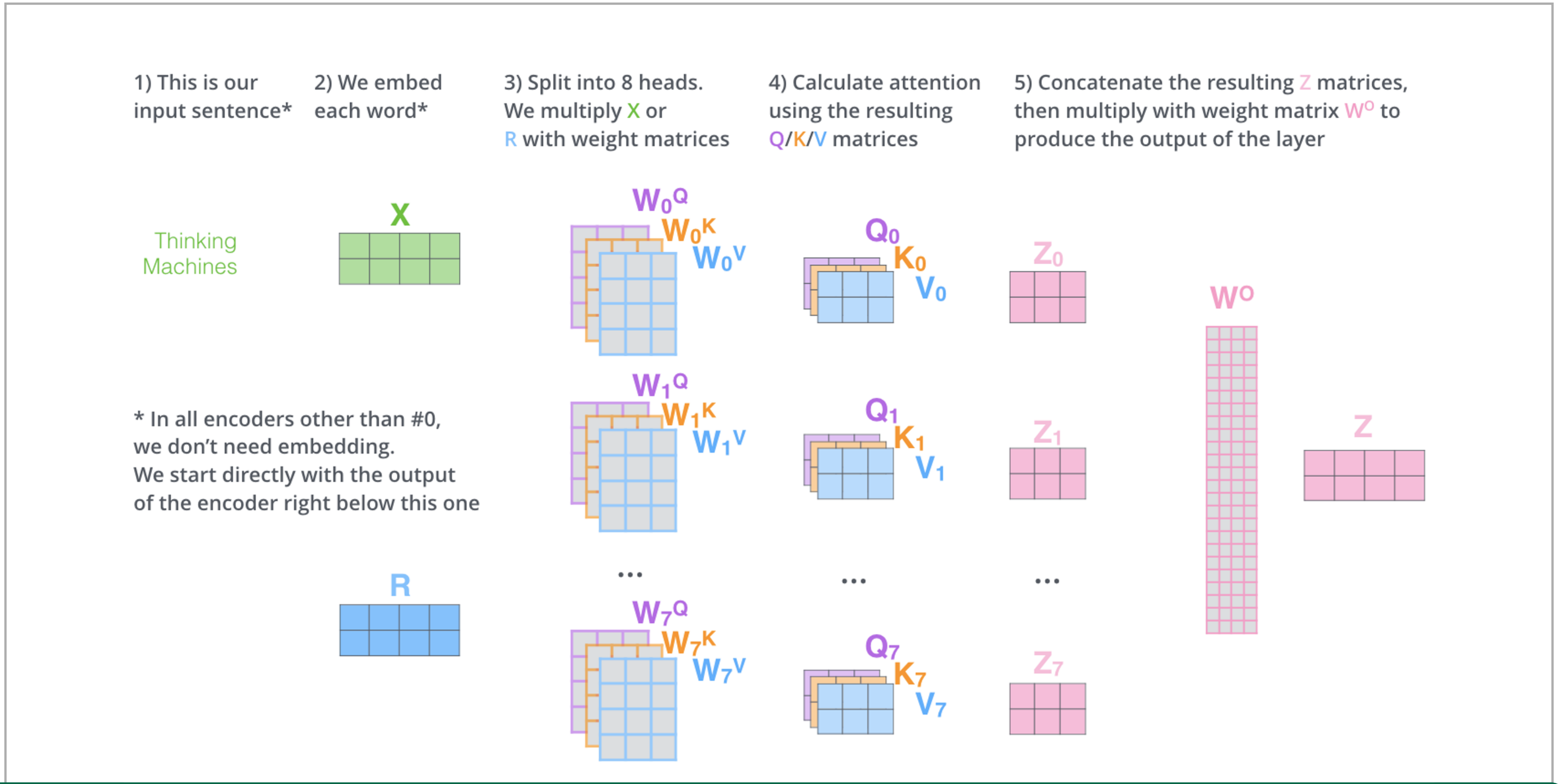

Figure 7. Instead of a single, large self-attention module, use N smaller modules to learn different ways of paying attention to other parts of the sequence.

Knowledge Check 1

As it is, how the self-attention module distinguishes between the word "the" in the beginning of the sentence and the word "the" before the word "street"?

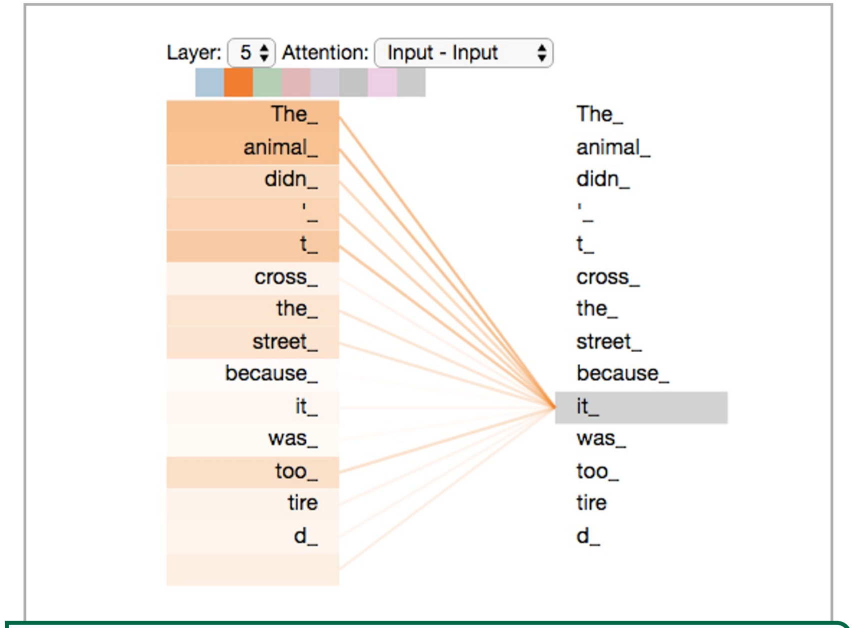

Figure 8. Illustration of the self-attention process.

A It doesn't. As long as they have the same input embedding, they will have the same output embedding.

B As long as the words are in different positions of the input sentence, they will receive different attention weights and will be mapped into different output embeddings.

C As long as they are followed by different words (e.g. "animal" and "street"), they will receive different attention weights and will be mapped into different output embeddings.

D As the first "the" has an uppercase T and the second a lowercase one, they will have different input embeddings and thus will be mapped into different output embeddings.

Positional encoding

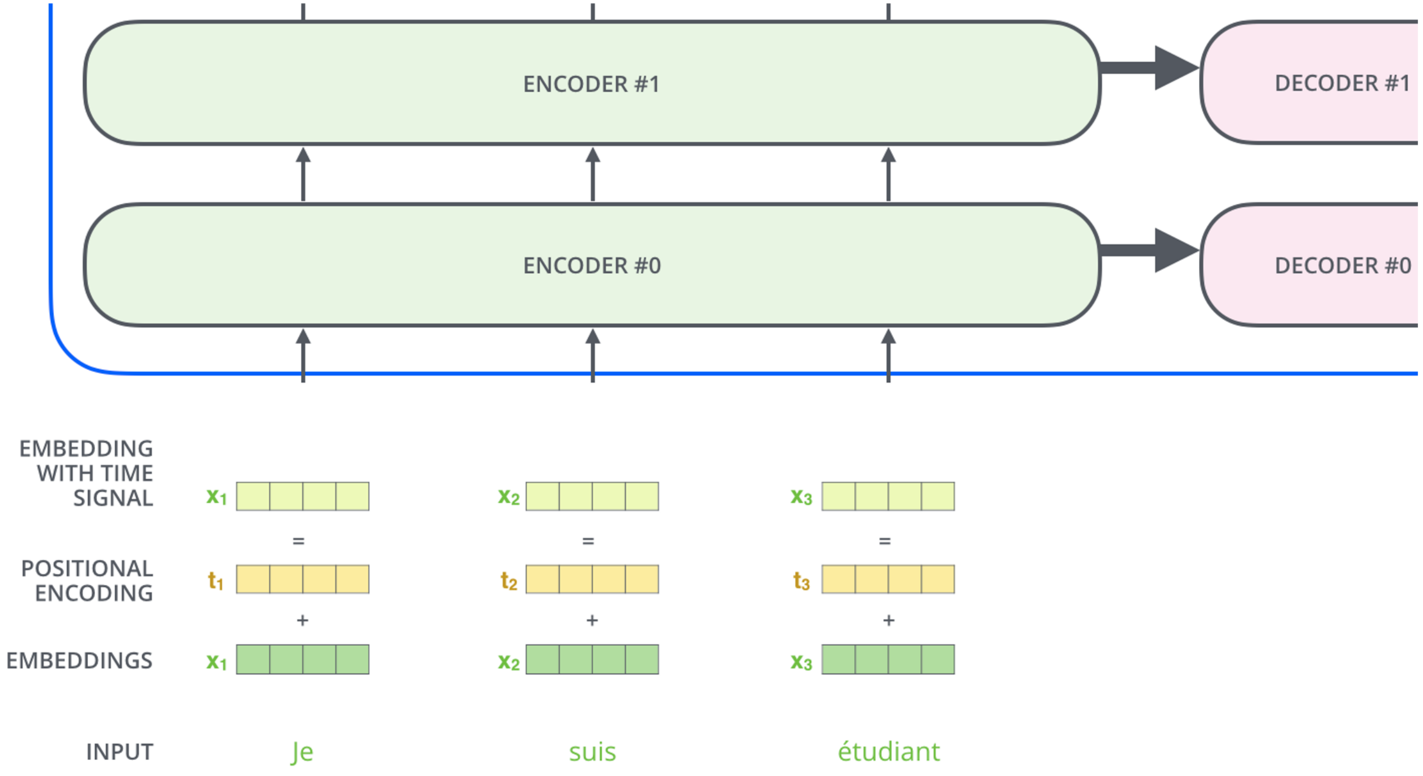

Figure 9. Different shifts applied to the distribution of word embeddings in each position of a sentence so that the network can learn the distance between different words.

Text classification with transformers

- Add one "class" token at the beginning of every sequence
- Use a Transformer model to project the input sequence into an output sequence
- Use the corresponding "class" token at the output sequence for the classification task

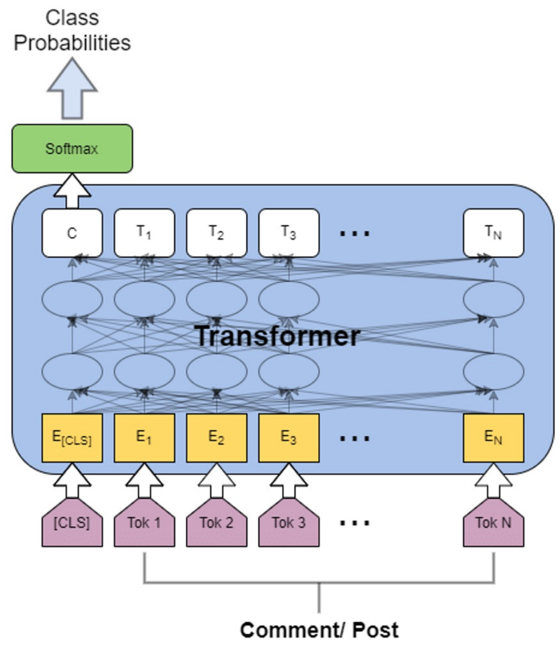

Figure 10. Illustration of a sentiment analysis network using a Transformer encoder.

Implementation

```python
# source: https://keras.io/examples/nlp/text_classification_with_transformer/
class TransformerBlock(tf.keras.layers.Layer):
   def __init__(self, embed_dim, num_heads, ff_dim, flag=False, rate=0.1):
       super(TransformerBlock, self).__init__()
       self.att = tf.keras.layers.MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)
       self.ffn = tf.keras.Sequential([
           tf.keras.layers.Dense(ff_dim, activation="relu"), tf.keras.layers.Dense(embed_dim),
       ])
       self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
       self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
       self.dropout1 = tf.keras.layers.Dropout(rate)
       self.dropout2 = tf.keras.layers.Dropout(rate)
       self.flag = flag

   def call(self, inputs, training):
       attn_output = self.att(inputs, inputs)                      # self-attention layer
       attn_output = self.dropout1(attn_output, training=training)
       out1 = self.layernorm1(inputs + attn_output)                # layer norm
       ffn_output = self.ffn(out1)                                 # feed-forward layer
       ffn_output = self.dropout2(ffn_output, training=training)
       if self.flag:                                               # layer norm
         return self.layernorm2(out1 + ffn_output)[:,0,:]
       else:
         return self.layernorm2(out1 + ffn_output)
```

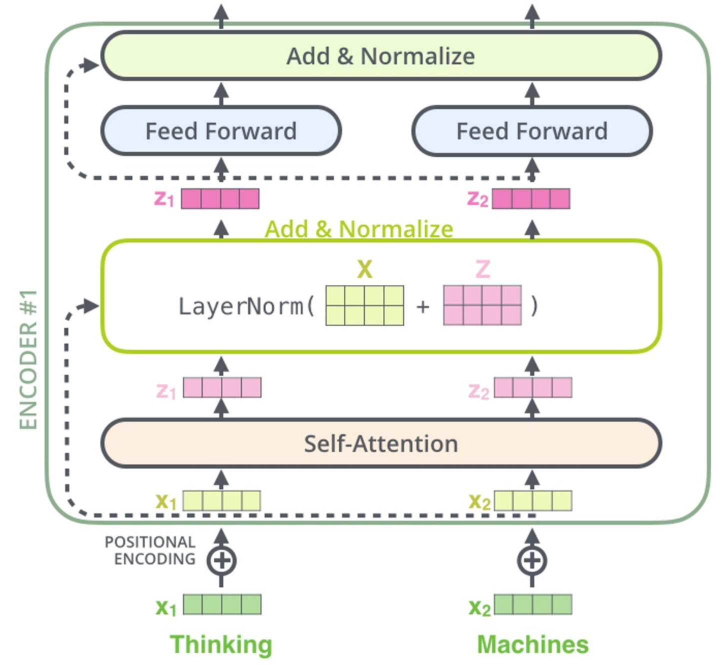

Figure 11. Detailed encoder block.

Implementation

```
# source: https://keras.io/examples/nlp/text_classification_with_transformer/
class TokenAndPositionEmbedding(tf.keras.layers.Layer):
   def __init__(self, maxlen, vocab_size, embed_dim):
       super(TokenAndPositionEmbedding, self).__init__()
       self.token_emb = tf.keras.layers.Embedding(input_dim=vocab_size, output_dim=embed_dim)
       self.pos_emb = tf.keras.layers.Embedding(input_dim=maxlen, output_dim=embed_dim)

   def call(self, x):
       maxlen = tf.shape(x)[-1]
       positions = tf.range(start=0, limit=maxlen, delta=1)
       positions = self.pos_emb(positions)
       x = self.token_emb(x)
       return x + positions
```

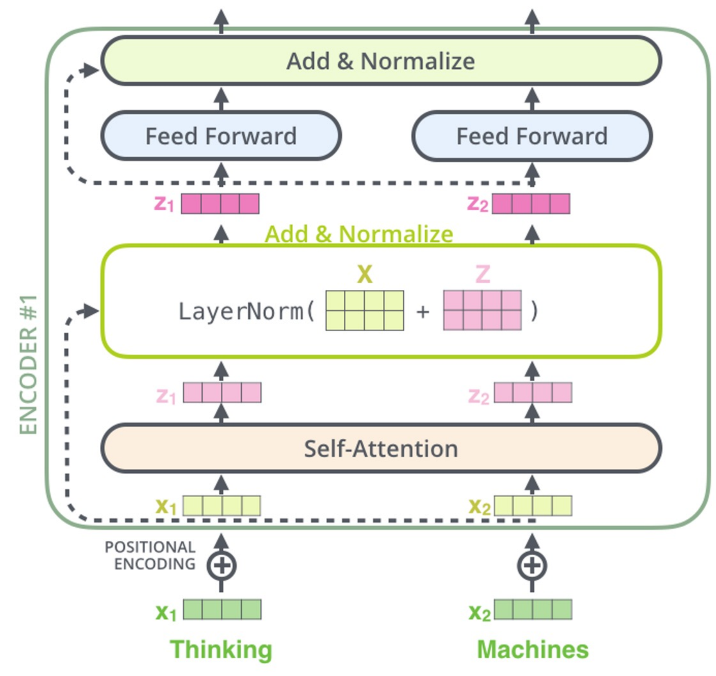

Figure 12. Detailed encoder block.

Implementation

```python
embed_dim = 64 # Embedding size for each token
num_heads = 8  # Number of attention heads
ff_dim = 128   # Hidden layer size in feed forward inside transformer

model = tf.keras.Sequential()
model.add(tf.keras.layers.Input(shape=(maxlen, )))
model.add(tf.keras.layers.Masking(mask_value=0))
model.add(TokenAndPositionEmbedding(maxlen, vocab_size, embed_dim))
model.add(TransformerBlock(embed_dim, num_heads, ff_dim, True))
model.add(tf.keras.layers.Dense(ff_dim, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

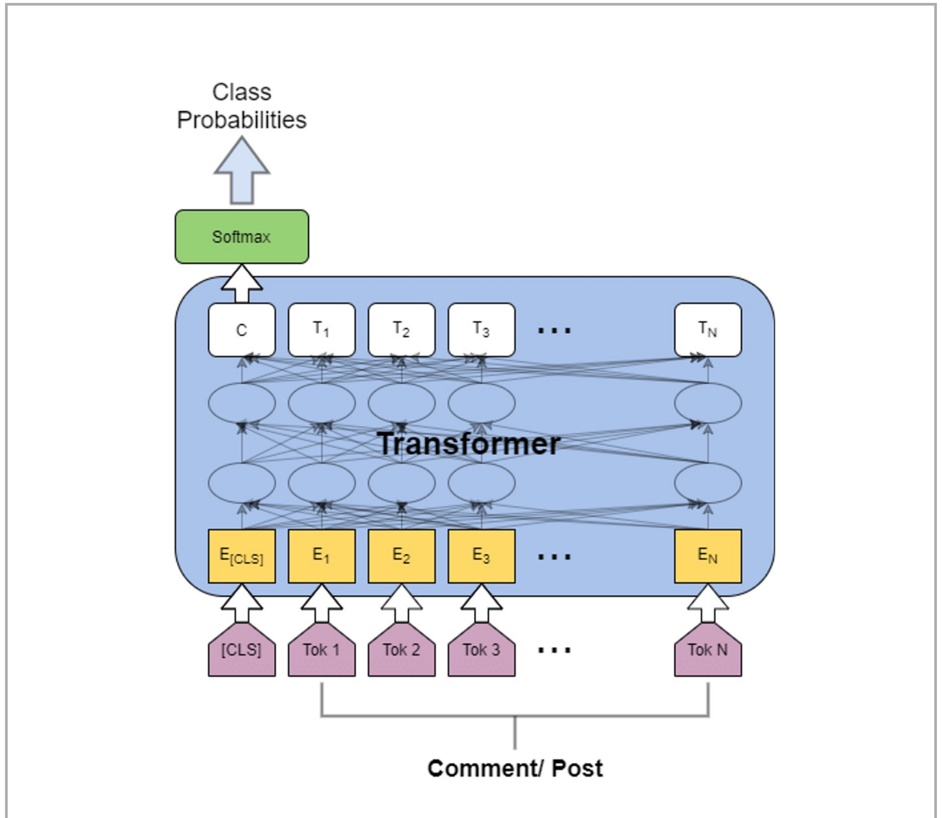

Figure 13. Illustration of a sentiment analysis network using a Transformer encoder.

Knowledge Check 2

What is the advantage of Transformers over RNNs?

A — Transformers look at the entire sequence at once, so words in the beginning of the sentence can look up at words at the end of the sentence for contextualization.

B — Transformers do not suffer from long-term dependency issues.

C — Transformers process the entire sentence in parallel, while RNNs must process sentences in a word-by-word fashion.

D — All the above.

You have reached the end of the lecture.

Image/Figure References

Figure 1-2. Encoder block of a Transformer network. Retrieved from: https://jalammar.github.io/illustrated-transformer/
Figure 3. Illustration of the self-attention process Retrieved from: : https://jalammar.github.io/illustrated-transformer/
Figure 4. First step of the self-attention module: creation of queries, keys and values for each embedding. Retrieved from: https://jalammar.github.io/illustrated-transformer/
Figure 5. Second step of the self-attention module: use queries, keys and values to update each word embedding. Retrieved from: https://jalammar.github.io/illustrated-transformer/
Figure 6. Visual example of the self-attention module operation. Retrieved from: https://jalammar.github.io/illustrated-transformer/
Figure 7. Instead of a single, large self-attention module, use N smaller modules to learn different ways of paying attention to other parts of the sequence. Retrieved from: https://jalammar.github.io/illustrated-transformer/
Figure 8. Illustration of the self-attention process. Retrieved from: https://jalammar.github.io/illustrated-transformer/
Figure 9. Different shifts applied to the distribution of word embeddings in each position of a sentence so that the network can learn the distance between different words. Retrieved from: https://jalammar.github.io/illustrated-transformer/
Figure 10. Illustration of a sentiment analysis network using a Transformer encoder.
Figure 11-12. Detailed encoder block. Retrieved from: https://jalammar.github.io/illustrated-transformer/
Figure 13. Illustration of a sentiment analysis network using a Transformer encoder.
Other images were purchased from Getty Images and with permission to use.