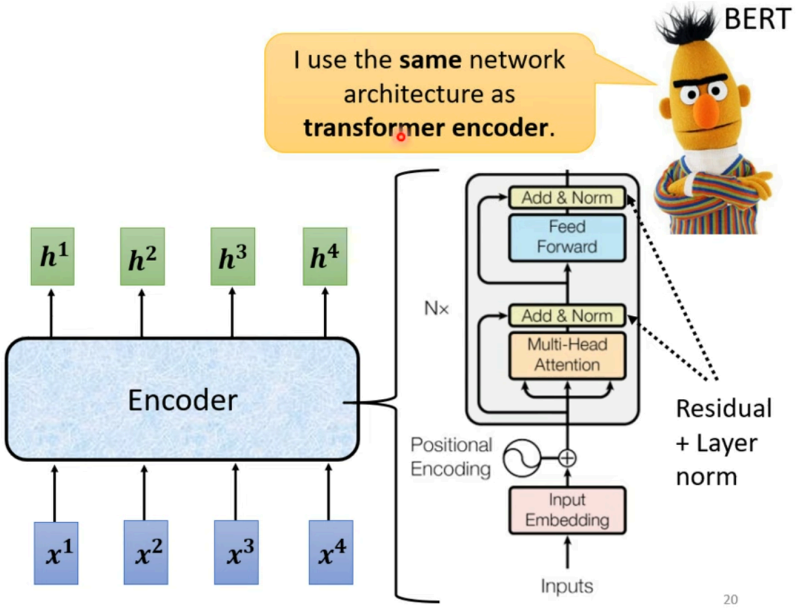
Transformer

Seq2seq Model based on self-attention, usually end-to-end, more efficient than RNN-related models

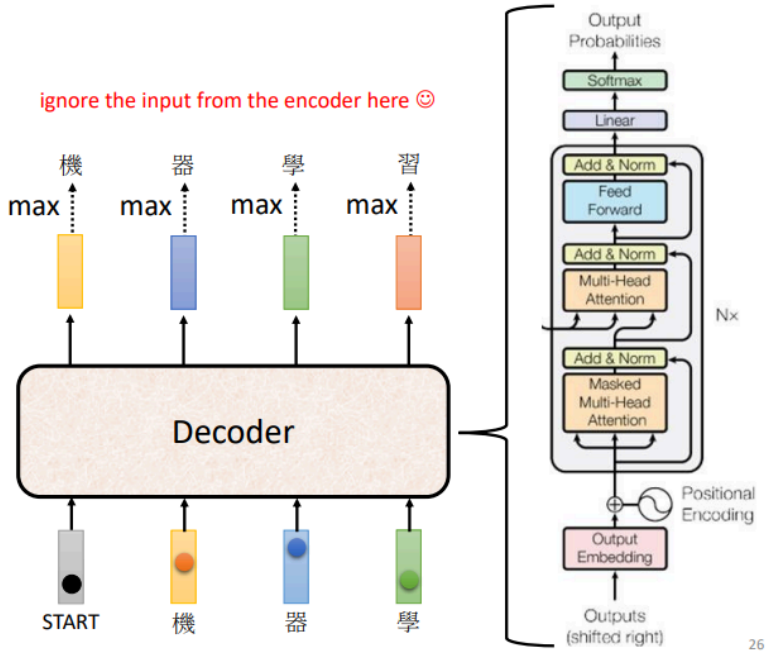
Transformer structure

- Encoder: seq--> another seq of the same length
  The encoder design in the original paper.

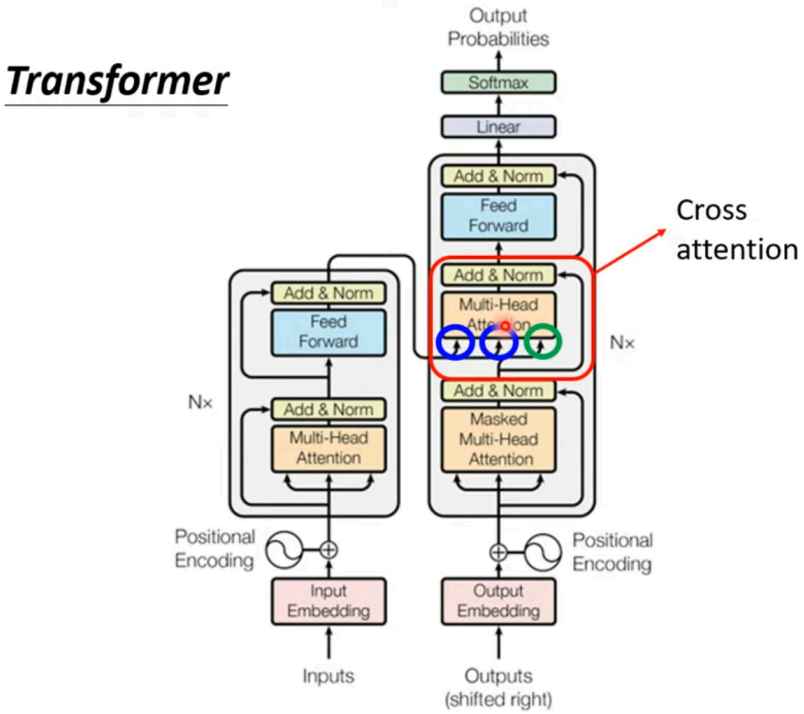


  - Input Embedding: map the tokens (words) into vectors which contrain semantic information
  - Positional Encoding: add positional informations directly instead of reading the sequence in order like RNN
  - Multi-Head Attention: learn the relationships between tokens
  - Add: residual connection (created in ResNet) allows deeper network
  - Norm: Layer normalization speeds up the traning
  - Feed Forward: use FC with activation functions to introduce non-linearity
- Decoder. encoder's output + generated tokens --> the next token

- Autoregressive (in original paper)



  - Masked Multi-Head Attention: decoder process the input sequentially, so we prevent it being influenced by unprocessed tokens
    - when the ouput is a special token END, the process stops
- Non-autoregressive (NAT)
  - output the whole sequence at a time
  - how to determine the length of the ouput seq?
    - use another predictor for output length
    - ouput a very long seq, ignore tokens after END
- Cross attention: the messenger between the encoder and the decoder



- `K, V` from encoder's ouput, `Q` from decoder's current layer input
- allows the decoder to have context from the encoder, focusing on the relevant context for each token
- the actual source of `K, V` varies...

Transformer Extension

Teacher Forcing: In the training, the decoder receives input from the **correct answer**, rather than the sequence it generates.

Copy Mechanism: allow the model to copy words directly from the input to the ouput, useful in tasks like *summarization or chat-bot*

Guided Attention: force the model to focus on the specific parts of the input sequence when predicting the output, often used in tasks like *speech recognition or translation*

Beam Search: Transformer always choose the most possible token as the output, which is called *Greedy Search*. Beam Search evaluates the tokens in beams, allowing for better solutions than Greedy Search.