Attention

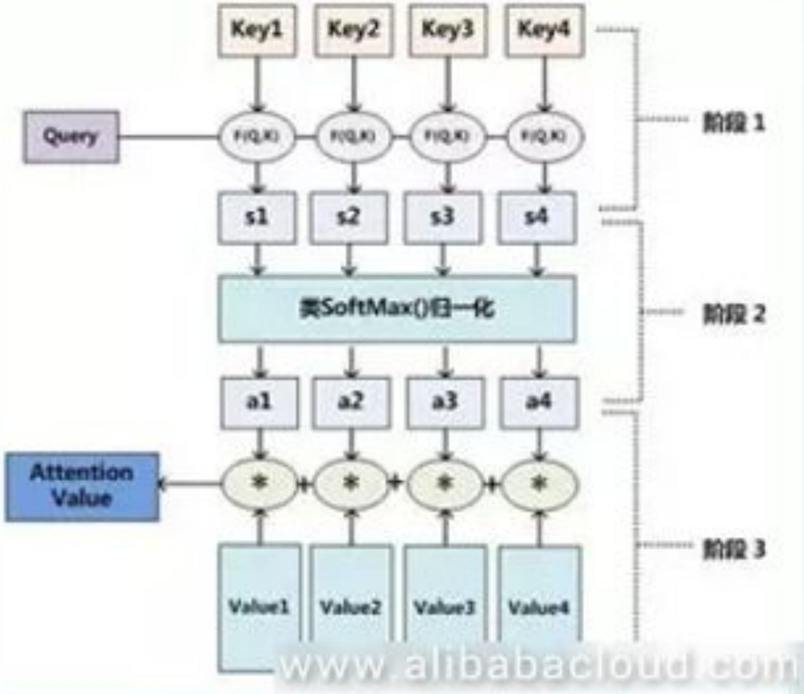
Laboratorio de NLP

Attention

$$f(Q,K_i)=\begin{cases} Q^T K_i & dot \\ Q^T W_a K_i & general \\ W_a[Q;K_i] & concat \\ v_a^T \tanh(W_a Q + U_a K_i) & perceptron \end{cases}$$

$$a_i = softmax(f(Q,K_i)) = \frac{\exp(f(Q,K_i))}{\sum_j \exp(f(Q,K_j))}$$

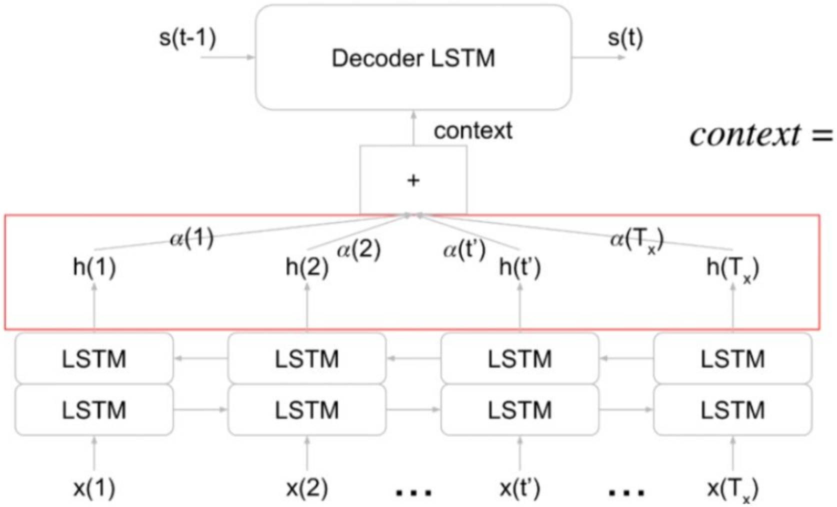
$$\mathrm{Attention}(Q,K,V) = \sum_i a_i V_i$$



Attention en NMT

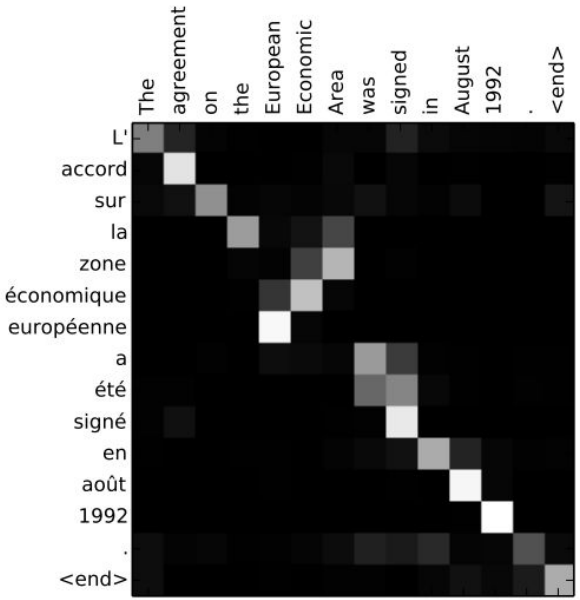
```
##### build the model #####

# Set up the encoder - simple!
encoder_inputs_placeholder = Input(shape=(max_len_input,))
x = embedding_layer(encoder_inputs_placeholder)
encoder = Bidirectional(LSTM(LATENT_DIM, return_sequences=True, dropout=0.5))
encoder_outputs = encoder(x)
```

Attention



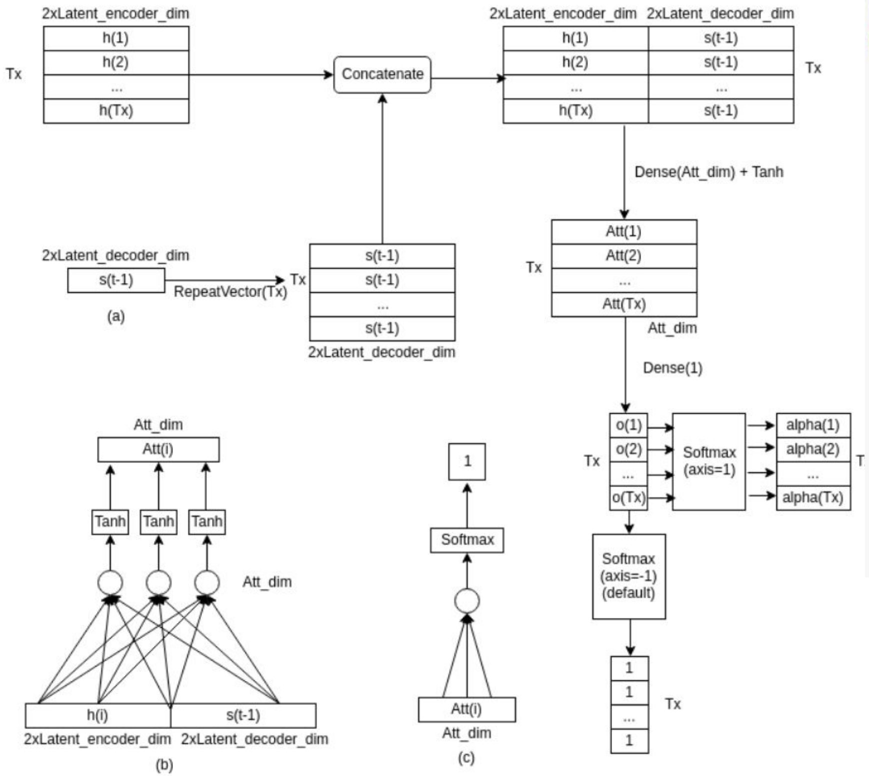
"Attention weights"

$$context = \sum_{t'=1}^{T_x} \alpha(t')h(t')$$



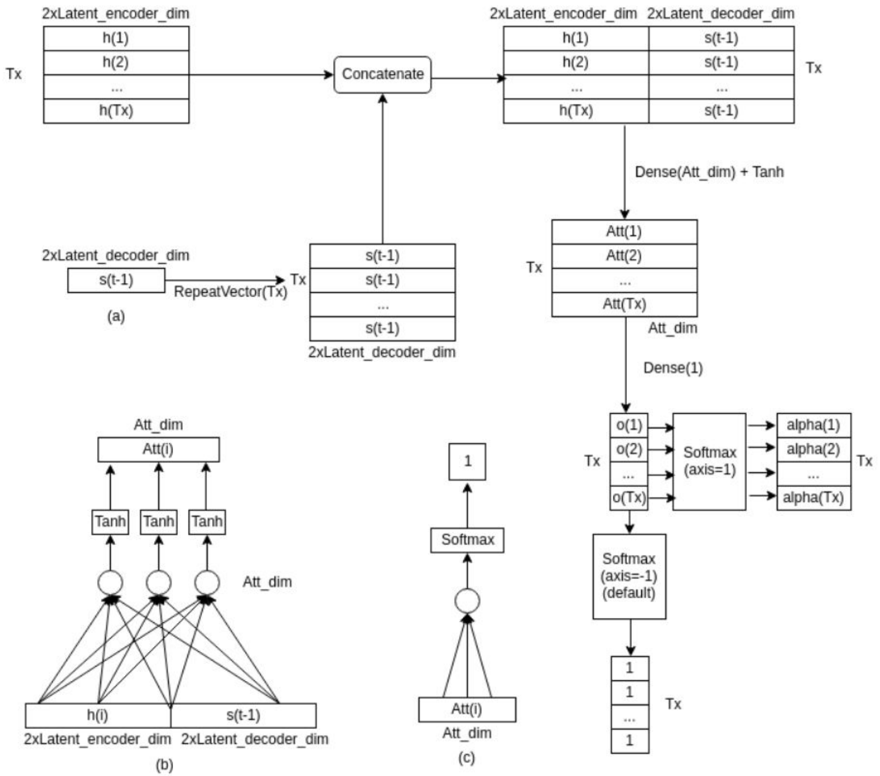

```
##### build the model #####

# Set up the encoder - simple!
encoder_inputs_placeholder = Input(shape=(max_len_input,))
x = embedding_layer(encoder_inputs_placeholder)
encoder = Bidirectional(LSTM(LATENT_DIM, return_sequences=True, dropout=0.5))
encoder_outputs = encoder(x)
```

Attention en NMT




```python
######### Attention #########
# Attention layers need to be global because
# they will be repeated Ty times at the decoder
attn_repeat_layer = RepeatVector(max_len_input)
attn_concat_layer = Concatenate(axis=-1)
attn_dense1 = Dense(10, activation='tanh')
attn_dense2 = Dense(1, activation=softmax_over_time)
attn_dot = Dot(axes=1) # to perform the weighted sum of alpha[t] *

def one_step_attention(h, st_1):
    # h = h(1), ..., h(Tx), shape = (Tx, LATENT_DIM * 2)
    # st_1 = s(t-1), shape = (LATENT_DIM_DECODER,)

    #Paso 1: se repite s(t-1) Tx veces para poder concatenarlo con los h
    st_1 = attn_repeat_layer(st_1)

    # Se concatena la lista de s(t-1) repetido con la lista de los h
    # Now of shape (Tx, LATENT_DIM_DECODER + LATENT_DIM * 2)
    x = attn_concat_layer([h, st_1])

    # Se pasa la lista de vectores concatenados por la primera capa densa
    x = attn_dense1(x)

    # Neural net second layer with special softmax over time
    alphas = attn_dense2(x)
    print(alphas)
    # "Dot" the alphas and the h's
    # Remember a.dot(b) = sum over a[t] * b[t]
    context = attn_dot([alphas, h])
    return [context,alphas]
```

Attention en NMT

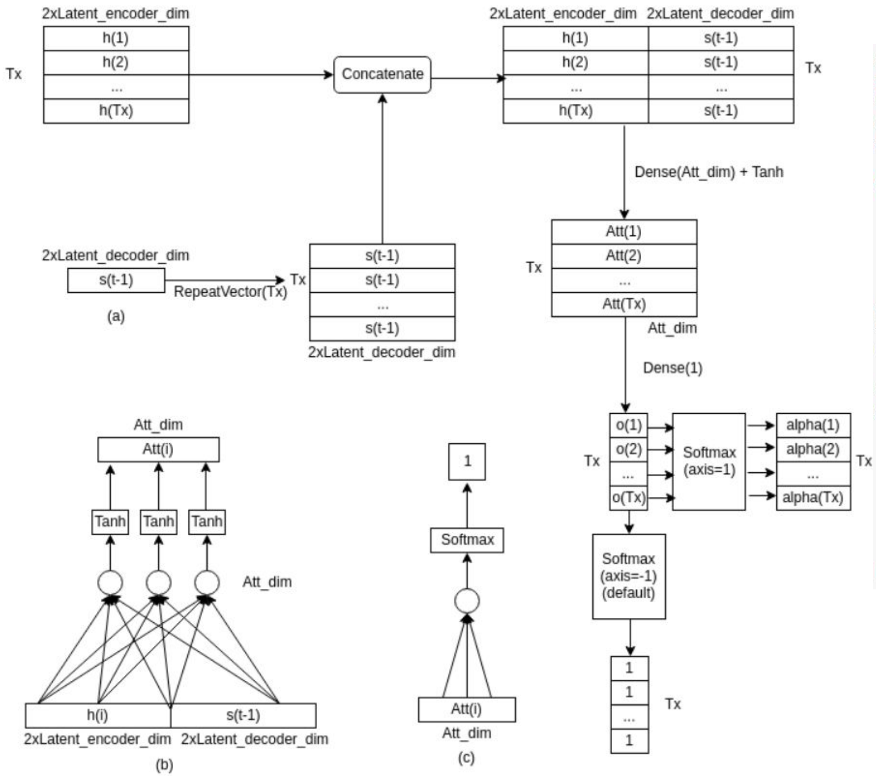



```
# define the rest of the decoder (after attention)
decoder_lstm = LSTM(LATENT_DIM_DECODER, return_state=True)
decoder_dense = Dense(num_words_output, activation='softmax')

initial_s = Input(shape=(LATENT_DIM_DECODER,), name='s0')
initial_c = Input(shape=(LATENT_DIM_DECODER,), name='c0')
context_last_word_concat_layer = Concatenate(axis=2)
```

```
# s, c will be re-assigned in each iteration of the loop
s = initial_s
c = initial_c
```

```
# collect outputs in a list at first
outputs = []
for t in range(max_len_target): # Ty times
    # get the context using attention
    context,alphas = one_step_attention(encoder_outputs, s)

    # we need a different layer for each time step
    selector = Lambda(lambda x: x[:, t:t+1])
    xt = selector(decoder_inputs_x)

    # combine
    decoder_lstm_input = context_last_word_concat_layer([context, xt])

    # pass the combined [context, last word] into the LSTM
    # along with [s, c]
    # get the new [s, c] and output
    o, s, c = decoder_lstm(decoder_lstm_input, initial_state=[s, c])

    # final dense layer to get next word prediction
    decoder_outputs = decoder_dense(o)
    outputs.append(decoder_outputs)
```

Attention en NMT




```python
# 'outputs' is now a list of length Ty
# each element is of shape (batch size, output vocab size)
# therefore if we simply stack all the outputs into 1 tensor
# it would be of shape T x N x D
# we would like it to be of shape N x T x D

def stack_and_transpose(x):
  # x is a list of length T, each element is a batch_size x output_vocab_size tensor
  x = K.stack(x) # is now T x batch_size x output_vocab_size tensor
  x = K.permute_dimensions(x, pattern=(1, 0, 2)) # is now batch_size x T x output_vocab_size
  return x

# make it a layer
stacker = Lambda(stack_and_transpose)
outputs = stacker(outputs)

# create the model
model = Model(
  inputs=[
    encoder_inputs_placeholder,
    decoder_inputs_placeholder,
    initial_s,
    initial_c,
  ],
  outputs=outputs
)
```

Image Captioning

Network Topology

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

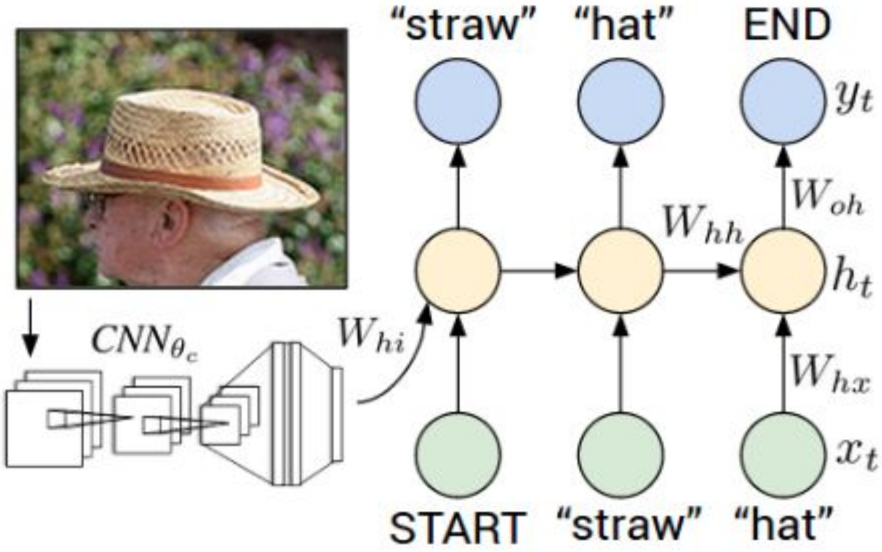
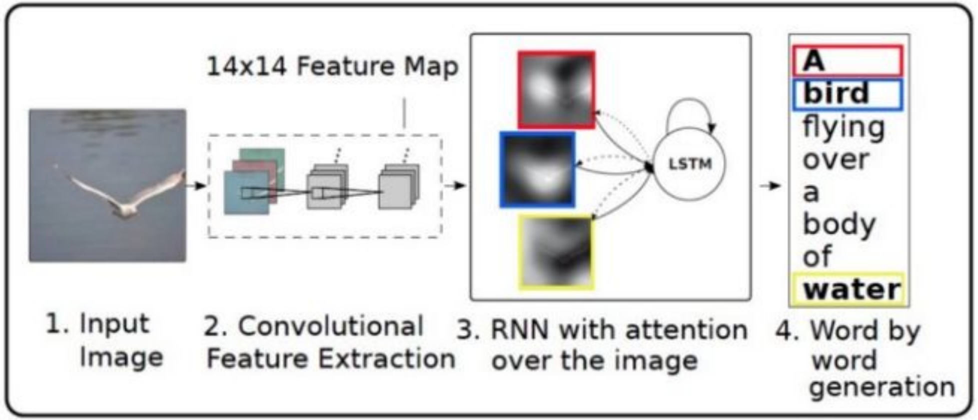


Image Captioning (Dataset)

- Common Objects in Context (COCO). A collection of more than 120 thousand images with descriptions
- Flickr 8K. A collection of 8 thousand described images taken from flickr.com.
- Flickr 30K. A collection of 30 thousand described images taken from flickr.com.
- Exploring Image Captioning Datasets, 2016

Image Captioning con Attention

Show, attend and tell: https://arxiv.org/pdf/1502.03044.pdf



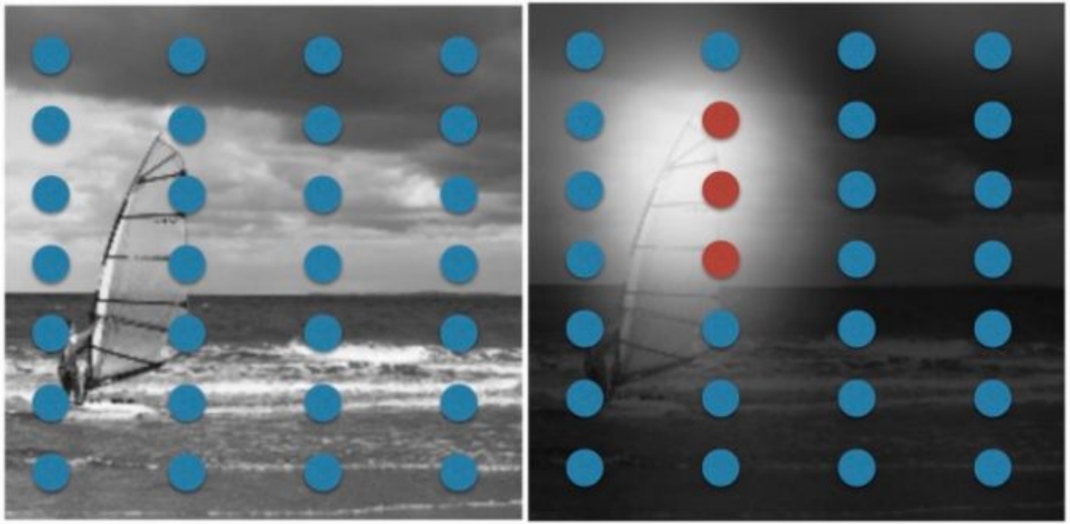
$$e_{ti} = f_{\text{att}}(\mathbf{a}_i, \mathbf{h}_{t-1})$$

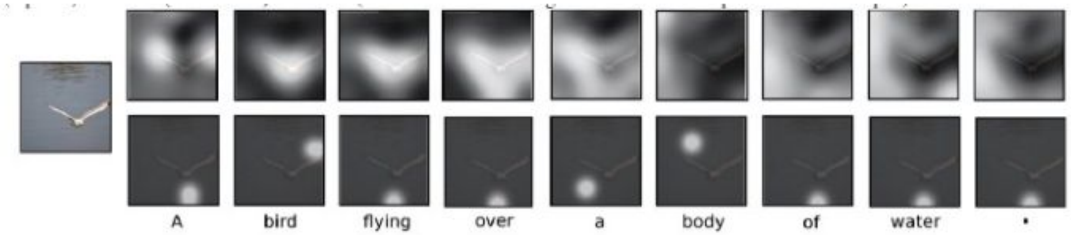
Weight per location is learned using LSTM hidden state and image features

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^{L} \exp(e_{tk})}.$$

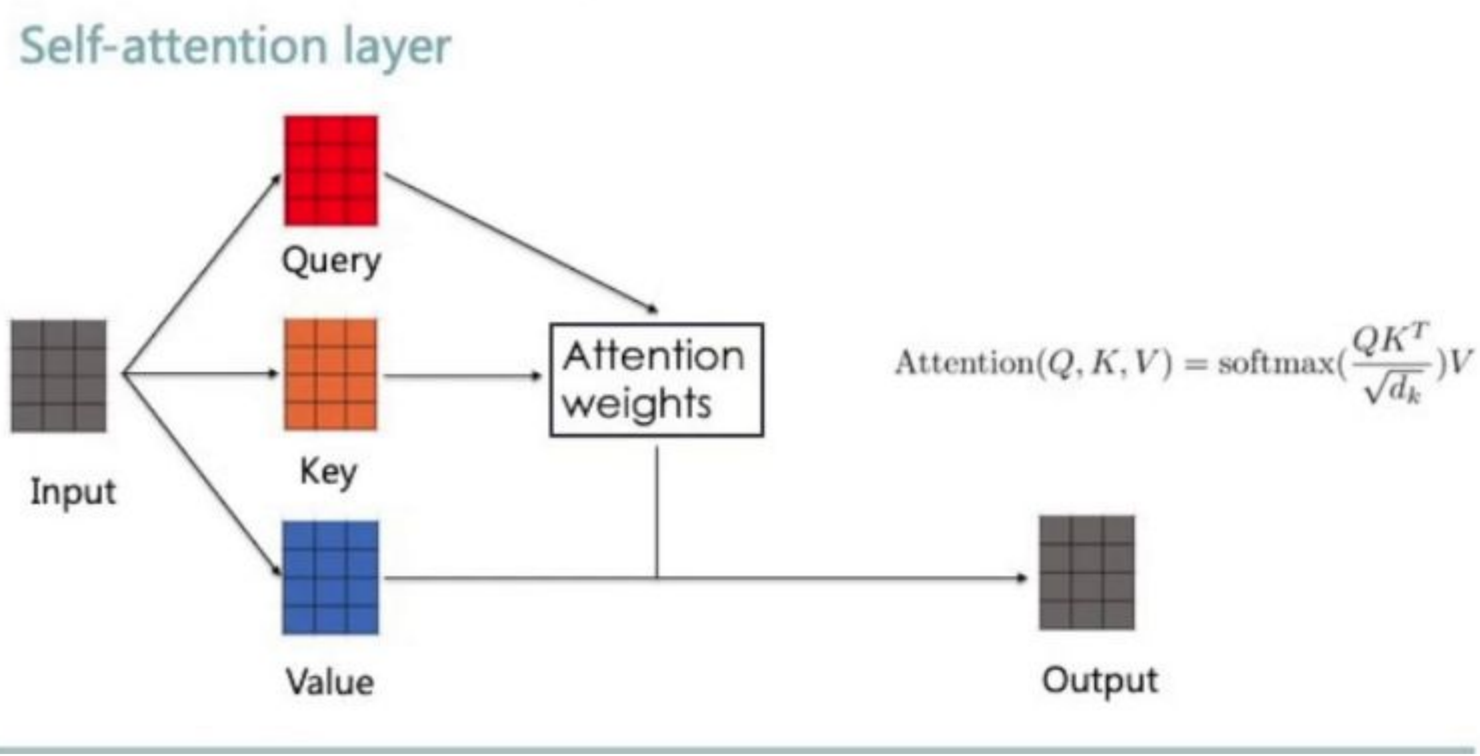
Softmax. Alpha sum to 1. Thus a valid multi-nomial distribution parameter

Image Captioning con Attention

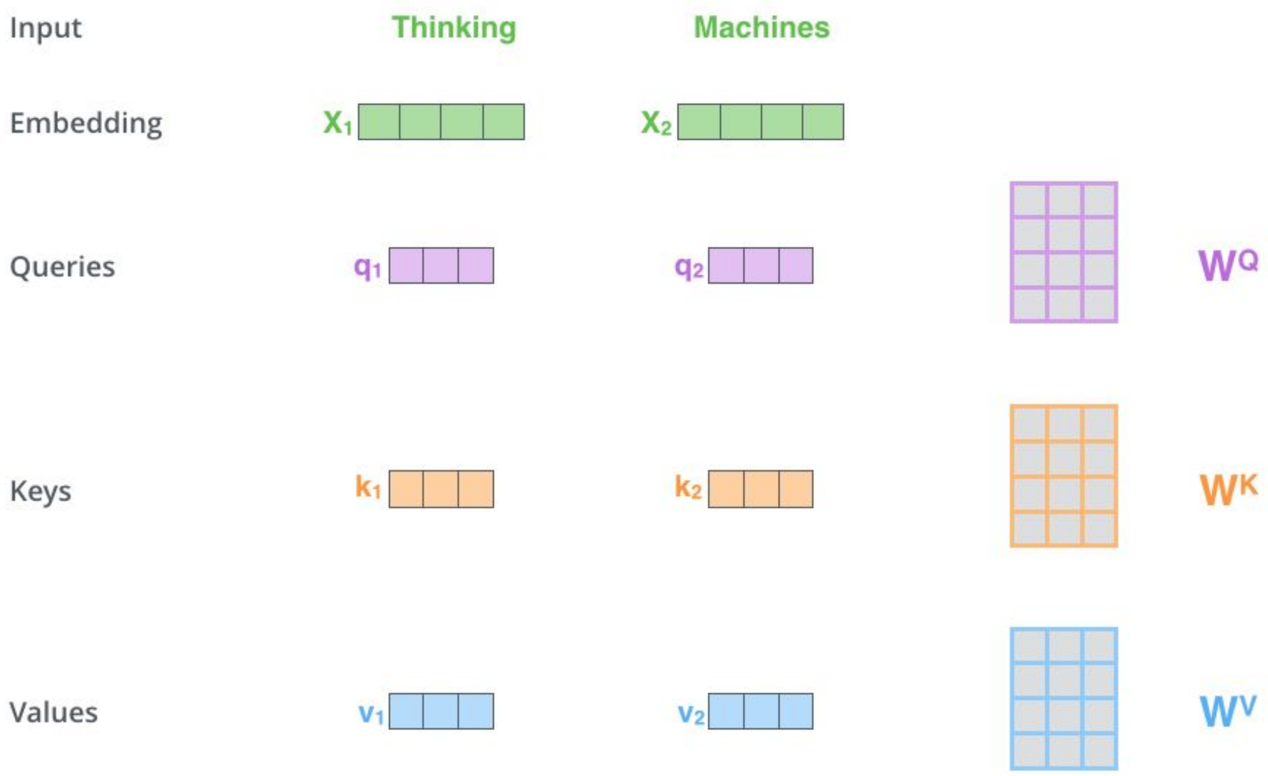




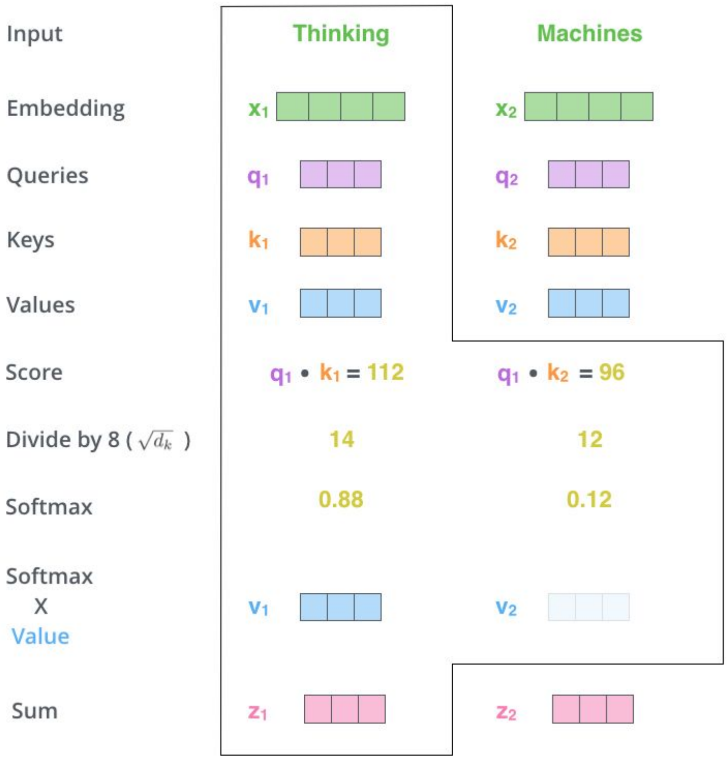
Self Attention



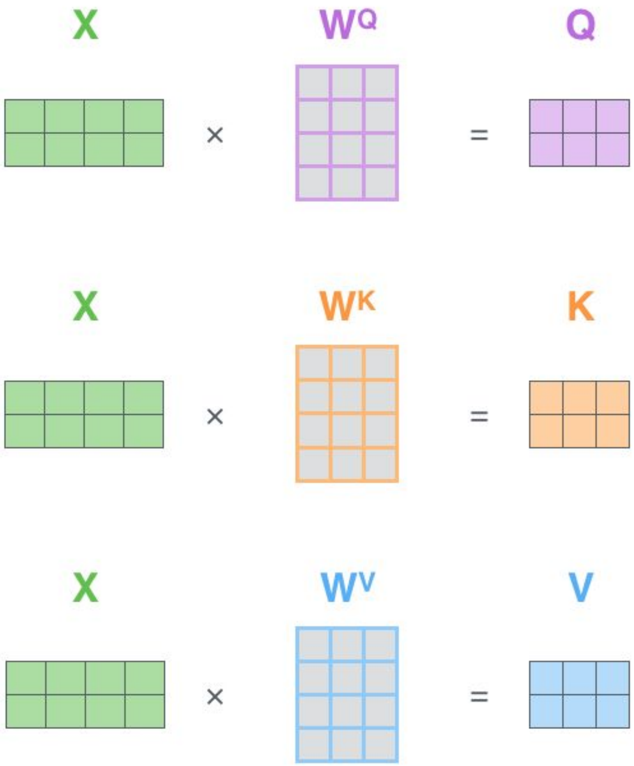
Self Attention (secuencias)



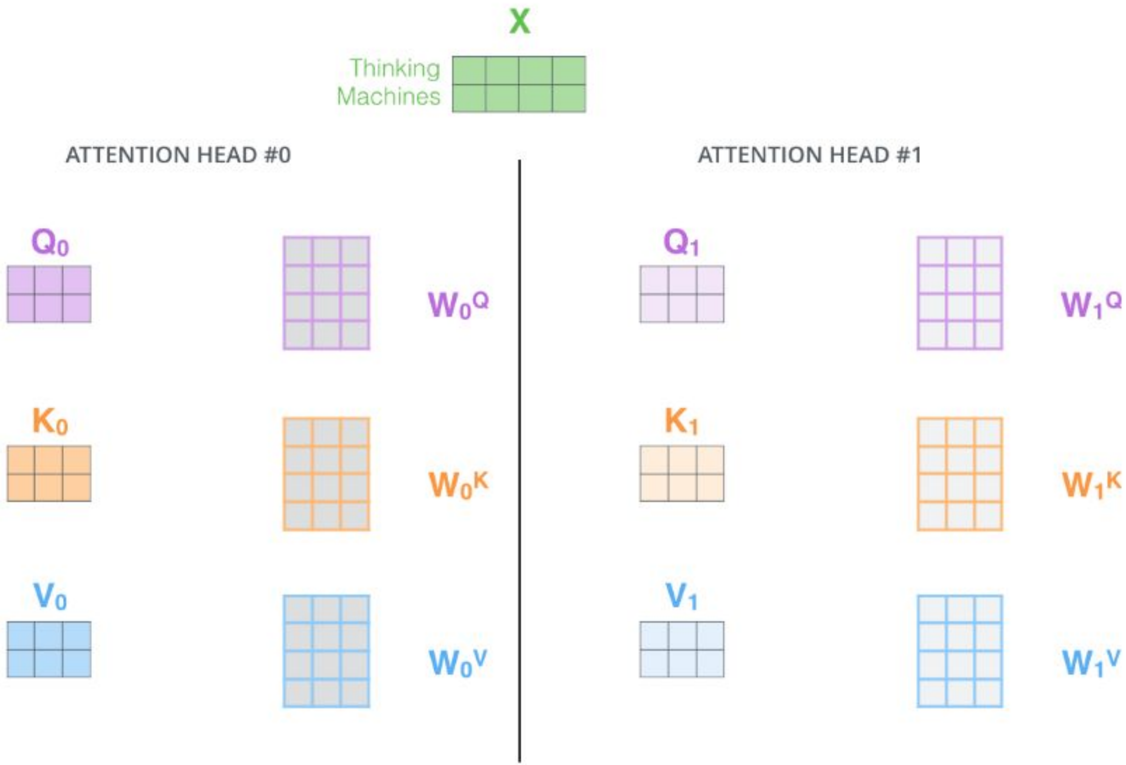
Self Attention (secuencias)



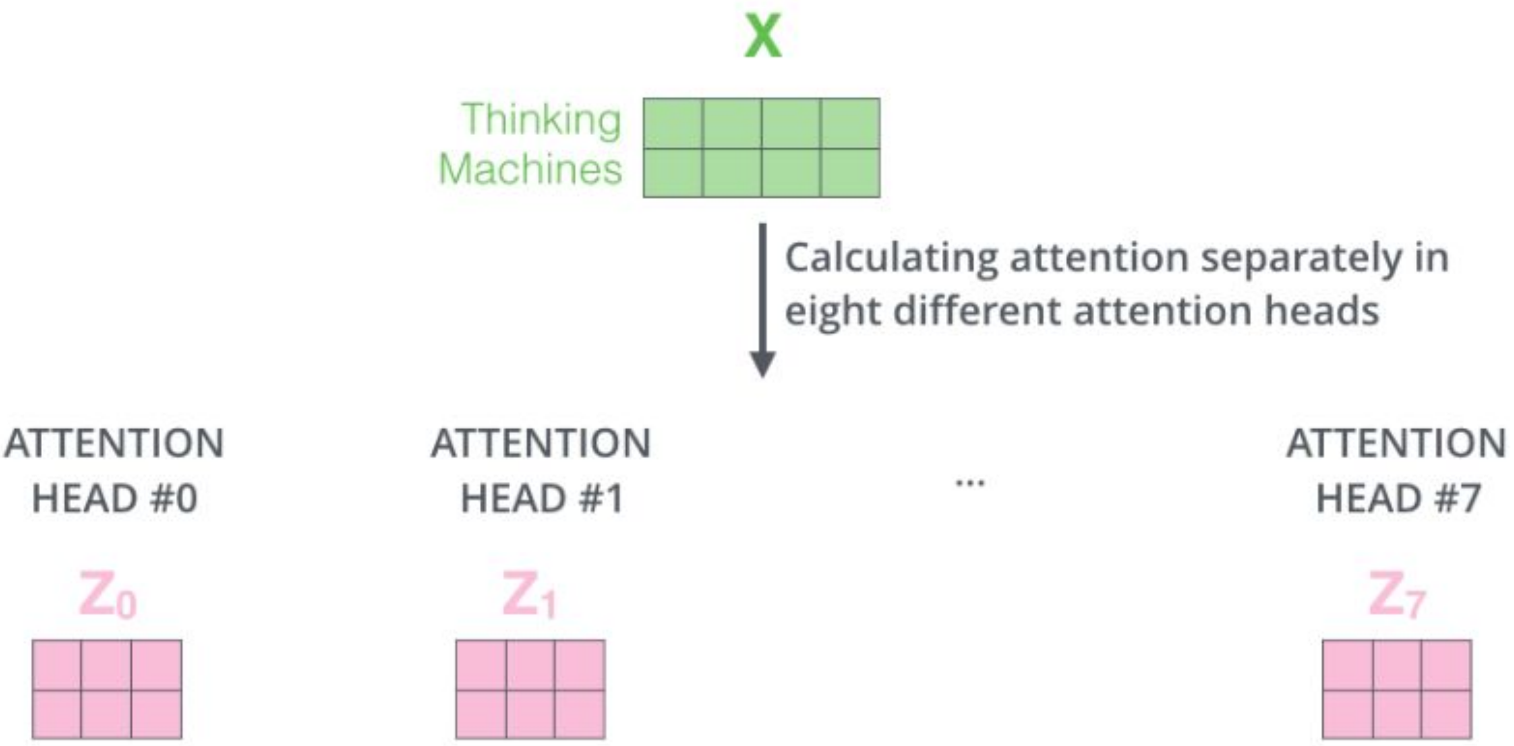
Self Attention (secuencias)



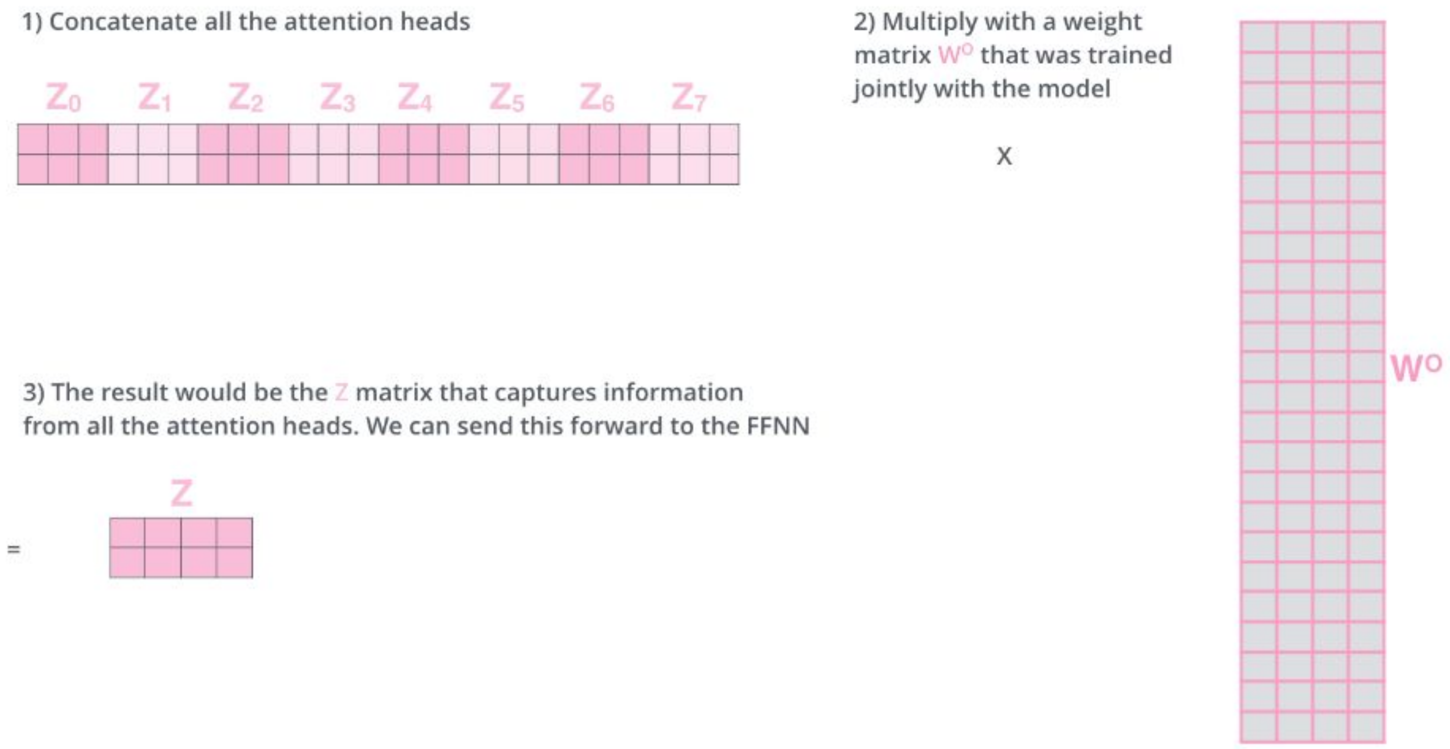
Multihead Attention



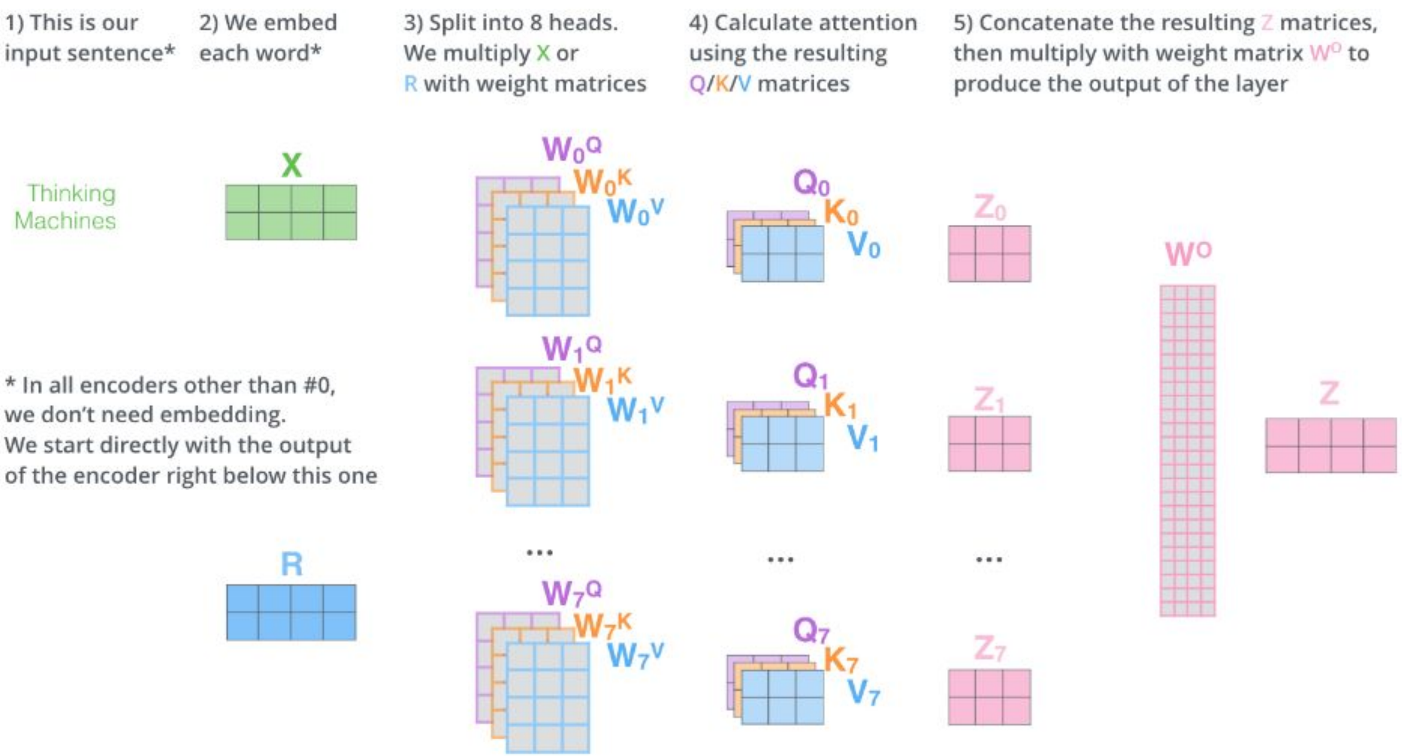
Multihead Attention



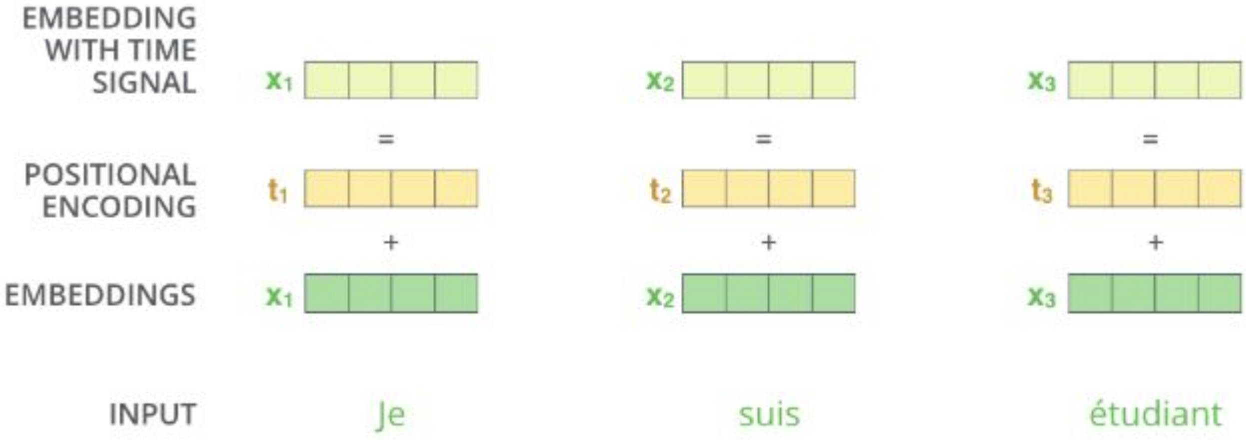
Multihead Attention

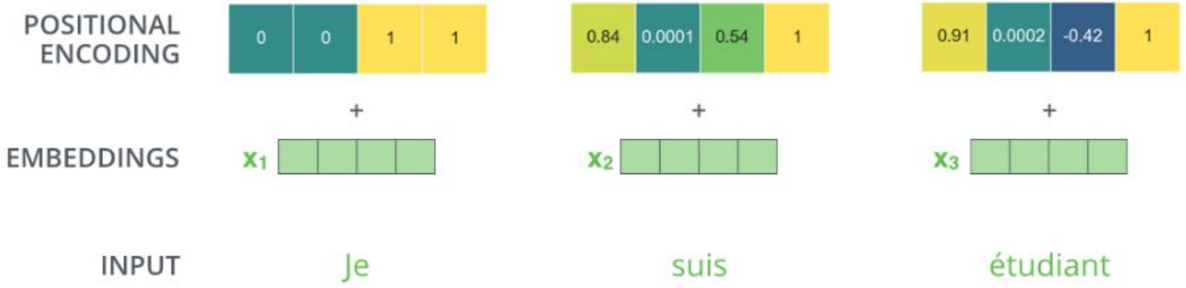


Multihead Attention

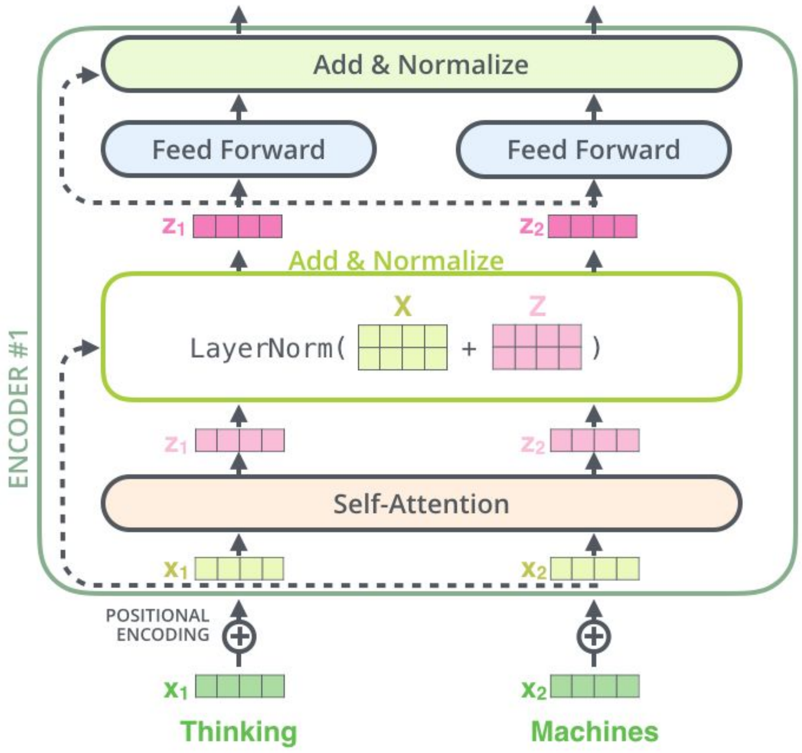


Positional Embeddings

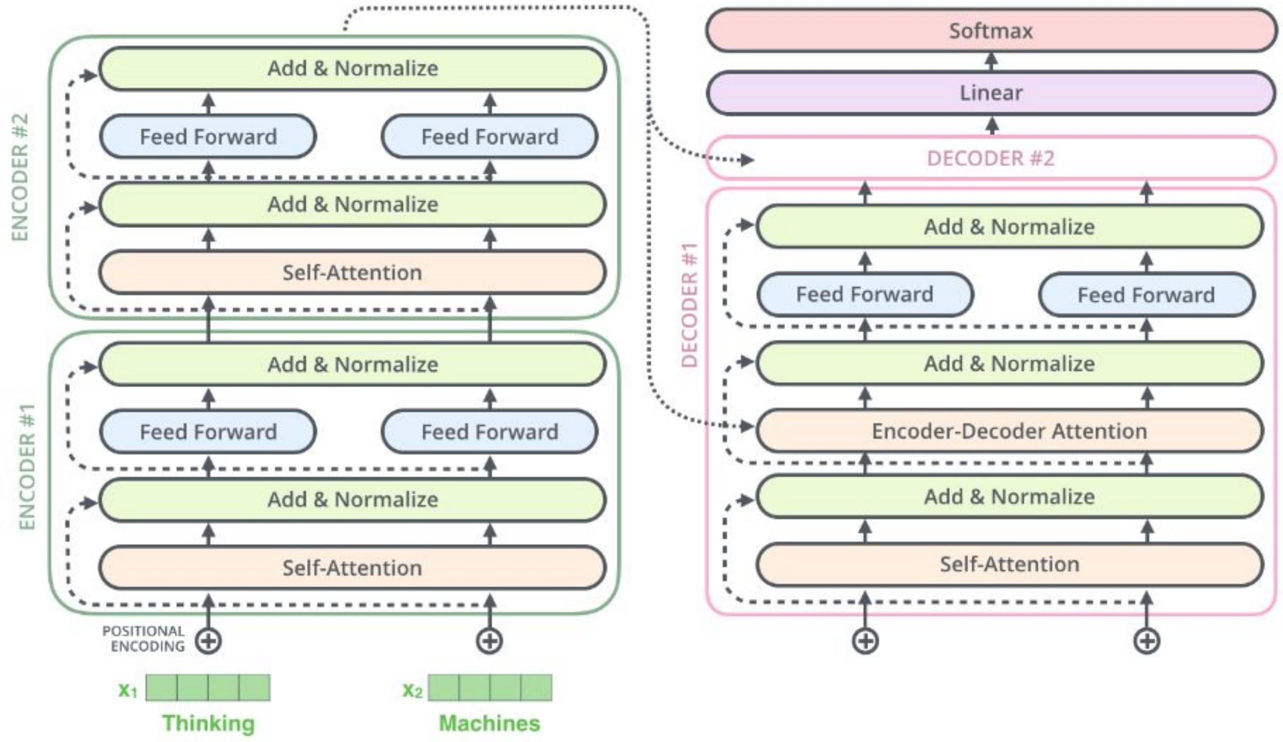




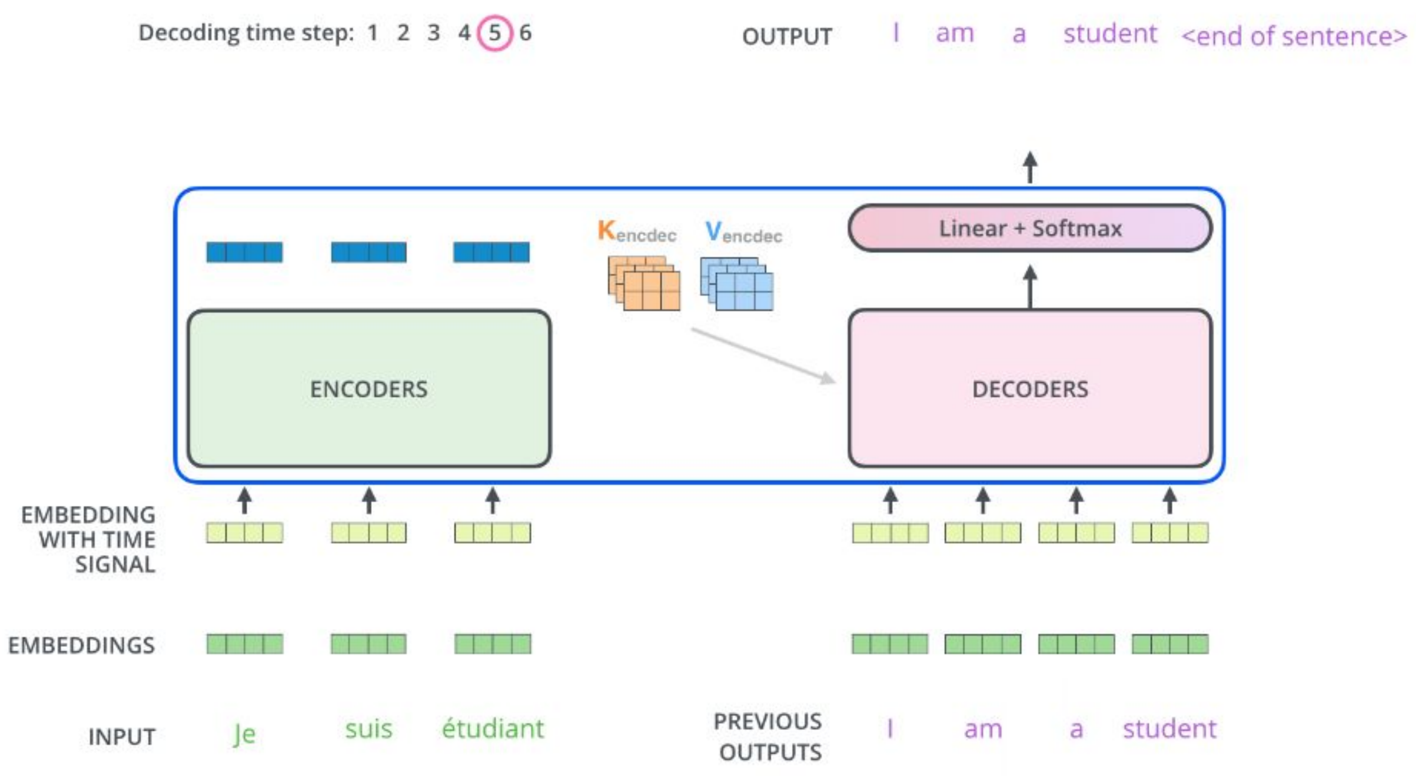
Encoder completo con Multihead Attention



Encoder - Decoder con Multihead Attention



Encoder - Decoder con Multihead Attention



Overview de BERT: Masked Language Model