Recitation 5: Attention & Transformers

Ekin Akyürek & Wei Fang

MIT 6.806-6.864 Spring 2021

Outline

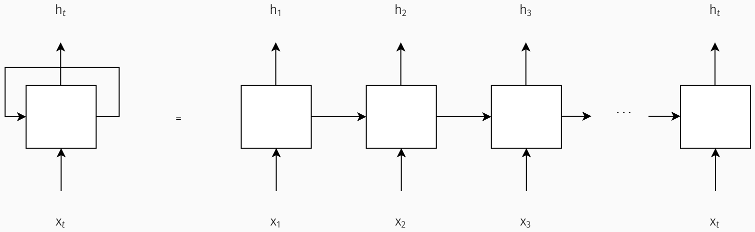
Quick Review - RNN & seq2seq

Attention with RNN & seq2seq

Attention Variants

Transformers: Attention is all you need!

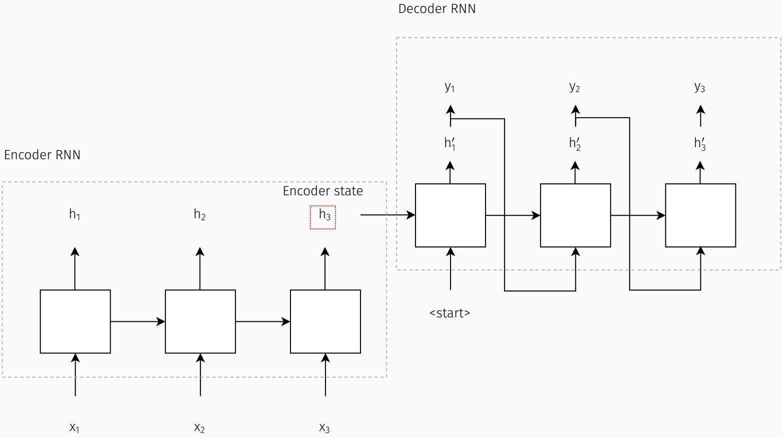
RNN



- Produces hidden state h_t at each time step; can be viewed as summary up to time t
- Recurrent cell can contain gating mechanisms (e.g. GRU or LSTM)

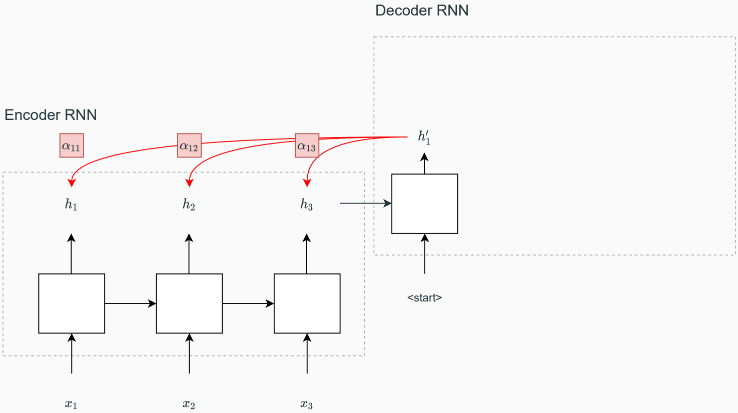
seq2seq: Encoder-Decoder Framework

- An RNN to summarize inputs (encoder)
- Another RNN to produce predictions based on encoded summary

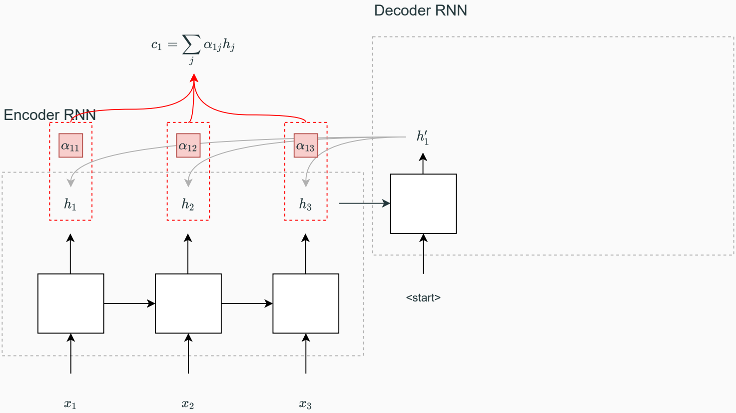


Outline

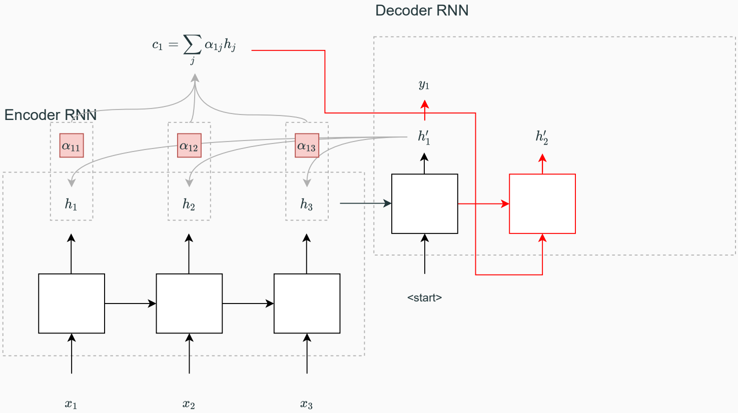
Attention with seq2seq



Attention with seq2seq



Attention with seq2seq



Outline

Quick Review - RNN & seq2seq

Attention with RNN & seq2seq

Attention Variants

Transformers: Attention is all you need!

Attention

Attention: a function $f_{att}(h_i, h'_j)$ that produces an alignment/similarity score, could be parametrized (more commonly used, see below) or non-parametrized (eg. cosine similarity).

$$
\begin{aligned}
e_{ij} &= f_{att}(h_i, h'_j) && \text{alignment/similarity score} \\
\alpha_{i:} &= \mathrm{softmax}(e_{i:}) && \text{normalization} \\
c_i &= \sum_j \alpha_{ij} h_j && \text{pooled input by convex combination}
\end{aligned}
$$

Attention Variants

Additive Attention (MLP)

1-layer MLP to calculate attention:

$$f_{att}(h_i, h'_j) = v^\top \tanh(W[h_i; h'_j]) = v^\top \tanh(W_{\text{left}} h_i + W_{\text{right}} h'_j),$$

where v, W are trainable.

Attention Variants

Additive Attention (MLP)

1-layer MLP to calculate attention:

$$f_{att}(h_i, h'_j) = v^\top \tanh(W[h_i; h'_j]) = v^\top \tanh(W_{\text{left}} h_i + W_{\text{right}} h'_j),$$

where v, W are trainable.

Multiplicative Attention

bilinear function:

$$f_{att}(h_i, h'_j) = h_i^\top W h'_j,$$

where W is trainable. When $W = I$ it becomes dot product.

Attention Variants

Additive Attention (MLP)

1-layer MLP to calculate attention:

$$f_{att}(h_i, h'_j) = v^\top \tanh(W[h_i; h'_j]) = v^\top \tanh(W_{\text{left}} h_i + W_{\text{right}} h'_j),$$

where v, W are trainable.

Multiplicative Attention

bilinear function:

$$f_{att}(h_i, h'_j) = h_i^\top W h'_j,$$

where W is trainable. When $W = I$ it becomes dot product.

- Complexity is similar, but in practice multiplicative is more efficient.

Attention Variants

Additive Attention (MLP)

1-layer MLP to calculate attention:

$$f_{att}(h_i, h'_j) = v^\top \tanh(W[h_i; h'_j]) = v^\top \tanh(W_{\text{left}} h_i + W_{\text{right}} h'_j),$$

where v, W are trainable.

Multiplicative Attention

bilinear function:

$$f_{att}(h_i, h'_j) = h_i^\top W h'_j,$$

where W is trainable. When $W = I$ it becomes dot product.

- Complexity is similar, but in practice multiplicative is more efficient.
- For small values of dimension d_h the two mechanisms perform similarly, additive attention outperforms dot product attention for larger values of d_h

Attention Variants

Additive Attention (MLP)

1-layer MLP to calculate attention:

$$f_{att}(h_i, h'_j) = v^\top \tanh(W[h_i; h'_j]) = v^\top \tanh(W_{\text{left}} h_i + W_{\text{right}} h'_j),$$

where v, W are trainable.

Multiplicative Attention

bilinear function:

$$f_{att}(h_i, h'_j) = h_i^\top W h'_j,$$

where W is trainable. When $W = I$ it becomes dot product.

- Complexity is similar, but in practice multiplicative is more efficient.
- For small values of dimension d_h the two mechanisms perform similarly, additive attention outperforms dot product attention for larger values of d_h
- One trick: scale multiplicative attention: $f_{att}(h_i, h'_j) = \frac{1}{\sqrt{d_h}} h_i^\top W h'_j$

Attention Variants (cont.)

Multi-head Attention

Multiple attentions in parallel:

$$
\begin{aligned}
f^{a}_{att}(h_i, h'_j) &= h_i^{\top} W^{a} h'_j, \\
f^{b}_{att}(h_i, h'_j) &= h_i^{\top} W^{b} h'_j, \\
&\vdots
\end{aligned}
$$

$W^{a}, W^{b}, \ldots$ are trainable.

Attention Variants (cont.)

Multi-head Attention

Multiple attentions in parallel:

$$f^a_{att}(h_i, h'_j) = h_i^\top W^a h'_j,$$
$$f^b_{att}(h_i, h'_j) = h_i^\top W^b h'_j,$$
$$\vdots$$

$W^a, W^b, \ldots$ are trainable.

Self-attention

Attend to lower layers (instead of decoder $\rightarrow$ encoder)

Attention Variants (cont.)

Key-value attention

Splits each hidden state h_i into key k_i and value v_i: $h_i = [k_i; v_i]$. *Keys* are used to calculate attention, and *values* are used for pooling.

$$e_{ij} = f_{att}(k_i, k'_j)$$

$$\alpha_{i:} = \text{softmax}(e_{i:})$$

$$c_i = \sum_j \alpha_{ij} v_j$$

Attention Variants (cont.)

Key-value attention

Splits each hidden state h_i into key k_i and value v_i: $h_i = [k_i; v_i]$. *Keys* are used to calculate attention, and *values* are used for pooling.

$$e_{ij} = f_{att}(k_i, k'_j)$$
$$\alpha_{i:} = \mathrm{softmax}(e_{i:})$$
$$c_i = \sum_j \alpha_{ij} v_j$$

Copy mechanism

Similarity scores e_{ij} or α_{ij} used directly for predicting output y_j.

$s_j = f_{out}(h'_j, c_i)$	output layer
$s'_j = \mathrm{softmax}([s_j; e_{ij}])$	predict with concat of pred and attn scores

Can also use only attention scores (pointer network)

Outline

Quick Review - RNN & seq2seq

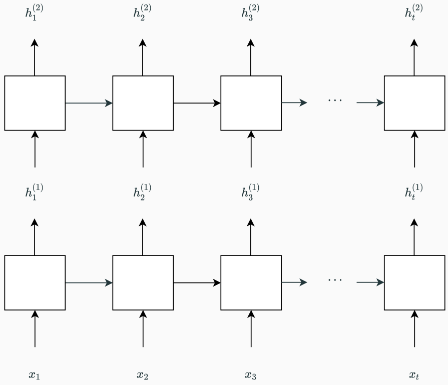
Attention with RNN & seq2seq

Attention Variants

Transformers: Attention is all you need!

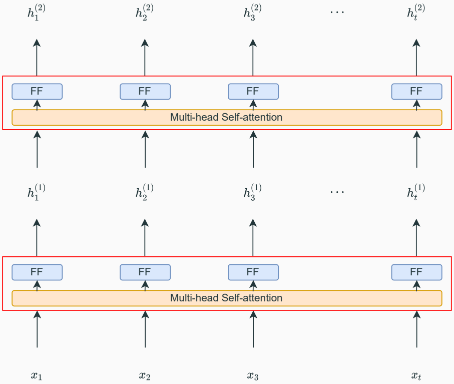
Transformers

- No recurrent layers; replace with self-attention layers



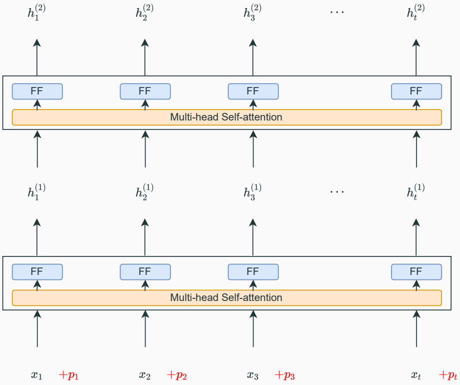
Transformers

- No recurrent layers; replace with self-attention layers



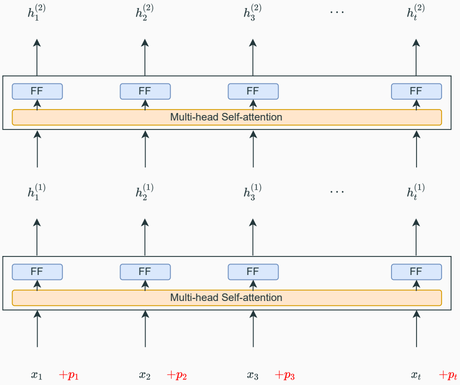
Transformers

- No recurrent layers; replace with self-attention layers
- Without recurrence we have no ordering information, need to add positional encodings



Transformers

- No recurrent layers; replace with self-attention layers
- Without recurrence we have no ordering information, need to add positional encodings
- Multi-head self-attention layer



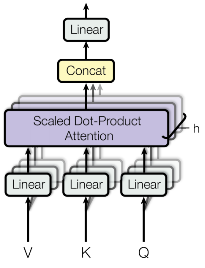
Multi-head Self-attention

Core idea: multi-head + key-value + *scaled* dot-product attention

Multi-head Self-attention

Core idea: multi-head + key-value + *scaled* dot-product attention

1. Do 2-3 in with h heads in parallel (params are not shared)

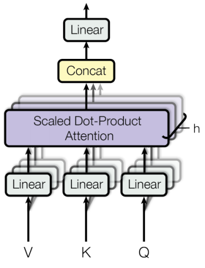


Multi-head Self-attention

Core idea: multi-head + key-value + *scaled* dot-product attention

1. Do 2-3 in with h heads in parallel (params are not shared)
2. Instead of splitting input (from prev layer) $h_i^{(l-1)}$ into key-values, *project* to get key/value and, additionally, queries

$$q_i = h_i^{(l-1)} W_Q, k_i = h_i^{(l-1)} W_K, v_i = h_i^{(l-1)} W_V$$



Multi-head Self-attention

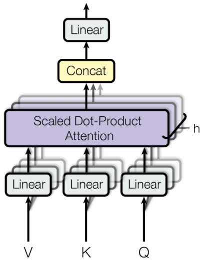
Core idea: multi-head + key-value + *scaled* dot-product attention

1. Do 2-3 in with h heads in parallel (params are not shared)
2. Instead of splitting input (from prev layer) $h_i^{(l-1)}$ into key-values, *project* to get key/value and, additionally, queries

$$q_i = h_i^{(l-1)} W_Q, k_i = h_i^{(l-1)} W_K, v_i = h_i^{(l-1)} W_V$$

3. Recall scaled dot-product attention, but with q_i, k_i, v_i

$$e_{ij} = f_{att}(k_i, k_j'), \alpha_{i:} = \mathrm{softmax}(e_{i:}), c_i = \sum_j \alpha_{ij} v_j$$



Multi-head Self-attention

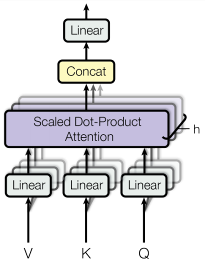
Core idea: multi-head + key-value + *scaled* dot-product attention

1. Do 2-3 in with h heads in parallel (params are not shared)
2. Instead of splitting input (from prev layer) $h_i^{(l-1)}$ into key-values, *project* to get key/value and, additionally, queries

$$q_i = h_i^{(l-1)} W_Q, k_i = h_i^{(l-1)} W_K, v_i = h_i^{(l-1)} W_V$$

3. Recall scaled dot-product attention, but with q_i, k_i, v_i

$$e_{ij} = f_{att}(k_i, q_j'), \alpha_{i:} = \mathrm{softmax}(e_{i:}), c_i = \sum_j \alpha_{ij} v_j$$



Multi-head Self-attention

Core idea: multi-head + key-value + *scaled* dot-product attention

1. Do 2-3 in with h heads in parallel (params are not shared)
2. Instead of splitting input (from prev layer) $h_i^{(l-1)}$ into key-values, *project* to get key/value and, additionally, queries

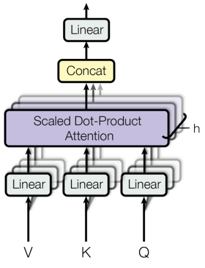
$$q_i = h_i^{(l-1)} W_Q, k_i = h_i^{(l-1)} W_K, v_i = h_i^{(l-1)} W_V$$

3. Recall scaled dot-product attention, but with q_i, k_i, v_i

$$e_{ij} = f_{att}(k_i, q_j'), \alpha_{i:} = \text{softmax}(e_{i:}), c_i = \sum_j \alpha_{ij} v_j$$

Can be written in matrix form:

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



Multi-head Self-attention

Core idea: multi-head + key-value + *scaled* dot-product attention

1. Do 2-3 in with h heads in parallel (params are not shared)
2. Instead of splitting input (from prev layer) $h_i^{(l-1)}$ into key-values, *project* to get key/value and, additionally, queries

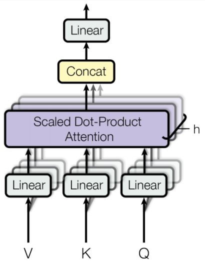
$$q_i = h_i^{(l-1)} W_Q, k_i = h_i^{(l-1)} W_K, v_i = h_i^{(l-1)} W_V$$

3. Recall scaled dot-product attention, but with q_i, k_i, v_i

$$e_{ij} = f_{att}(k_i, q_j'), \alpha_{i:} = \text{softmax}(e_{i:}), c_i = \sum_j \alpha_{ij} v_j$$

Can be written in matrix form:

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

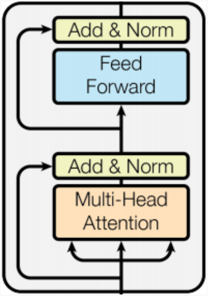


4. Concat pooled inputs *from all heads* and pass through linear

$$\text{MultiHead}(\cdot) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h) W_O$$

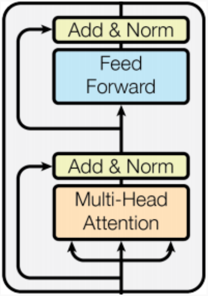
More Details

- Residual connections & layer normalization



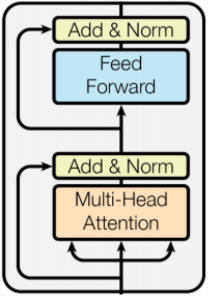
More Details

- Residual connections & layer normalization
- Encoder layer can be substituted in for recurrent layers; can be stacked



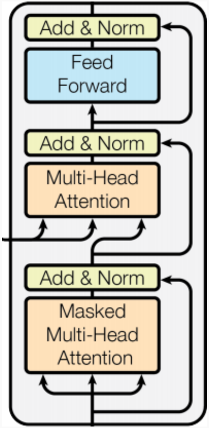
More Details

- Residual connections & layer normalization
- Encoder layer can be substituted in for recurrent layers; can be stacked
- For sequence tasks we only need encoders



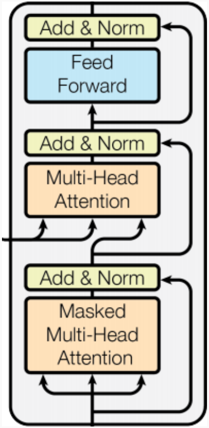
More Details

- Residual connections & layer normalization
- Encoder layer can be substituted in for recurrent layers; can be stacked
- For sequence tasks we only need encoders
- For seq2seq architectures, we need an additional *decoder* transformer with decoder layers



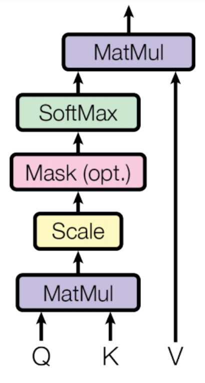
More Details

- Residual connections & layer normalization
- Encoder layer can be substituted in for recurrent layers; can be stacked
- For sequence tasks we only need encoders
- For seq2seq architectures, we need an additional *decoder* transformer with decoder layers
- Decoding is auto-regressive; first multi-head attention in each layer needs to be *masked*

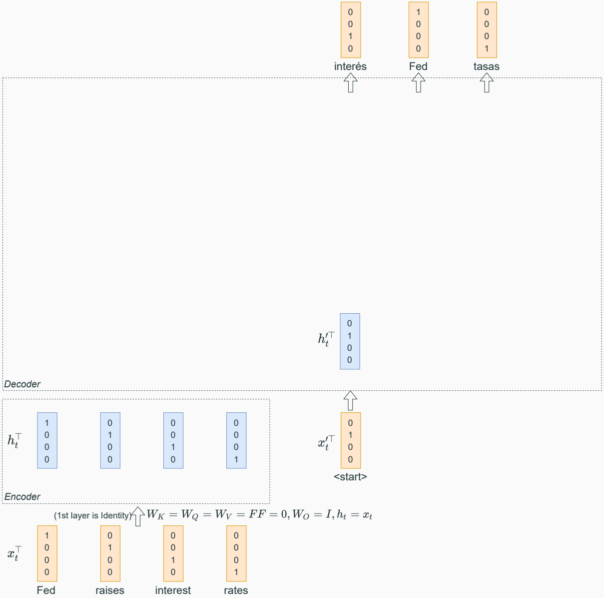


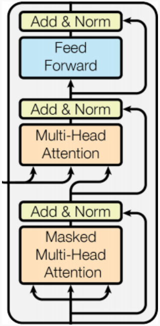
More Details

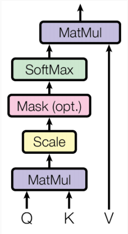
- Residual connections & layer normalization
- Encoder layer can be substituted in for recurrent layers; can be stacked
- For sequence tasks we only need encoders
- For seq2seq architectures, we need an additional *decoder* transformer with decoder layers
- Decoding is auto-regressive; first multi-head attention in each layer needs to be *masked*



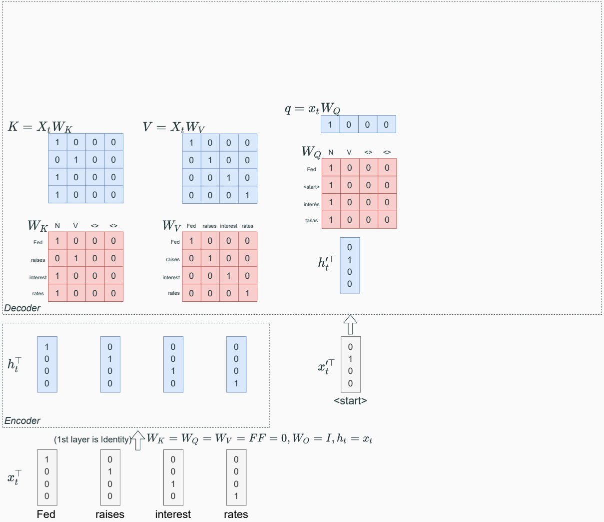
Transformer in action: Toy example (Noun translation)

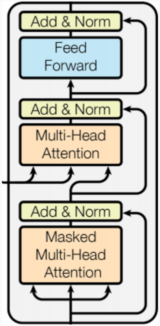


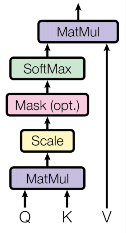




Transformer in action: Toy example (Noun translation)







Transformer in action: Toy example (Noun translation)

$\alpha = \mathrm{softmax}(qK^T)$

$$\begin{bmatrix} 0.3 & 0.1 & 0.3 & 0.3 \end{bmatrix}$$

$c = \mathrm{softmax}(qK^T)V; W_O = I$

$$\begin{bmatrix} 0.3 & 0.1 & 0.3 & 0.3 \end{bmatrix}$$

$$e = qK^T \quad \begin{bmatrix} 1 & 0 & 1 & 1 \end{bmatrix}$$

$$q = x_t W_Q \quad \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$$

$$K = X_t W_K \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \qquad V = X_t W_V \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$W_Q \quad \begin{array}{c|cccc} & \text{N} & \text{V} & \text{<>} & \text{<>} \\ \hline \text{Fed} & 1 & 0 & 0 & 0 \\ \text{<start>} & 1 & 0 & 0 & 0 \\ \text{interés} & 1 & 0 & 0 & 0 \\ \text{tasas} & 1 & 0 & 0 & 0 \end{array}$$

$$W_K \quad \begin{array}{c|cccc} & \text{N} & \text{V} & \text{<>} & \text{<>} \\ \hline \text{Fed} & 1 & 0 & 0 & 0 \\ \text{raises} & 0 & 1 & 0 & 0 \\ \text{interest} & 1 & 0 & 0 & 0 \\ \text{rates} & 1 & 0 & 0 & 0 \end{array} \qquad W_V \quad \begin{array}{c|cccc} & \text{Fed} & \text{raises} & \text{interest} & \text{rates} \\ \hline \text{Fed} & 1 & 0 & 0 & 0 \\ \text{raises} & 0 & 1 & 0 & 0 \\ \text{interest} & 0 & 0 & 1 & 0 \\ \text{rates} & 0 & 0 & 0 & 1 \end{array}$$

$$h_t'^{\top} \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Decoder

$$h_t^{\top} \quad \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

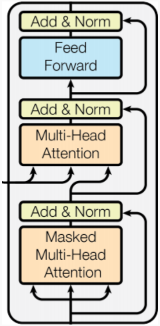
Encoder

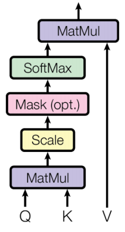
$$x_t'^{\top} \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

<start>

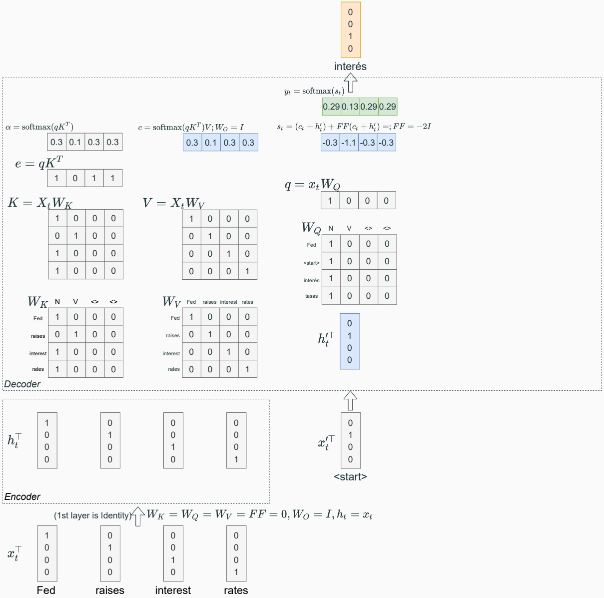
(1st layer is Identity) $W_K = W_Q = W_V = FF = 0, W_O = I, h_t = x_t$

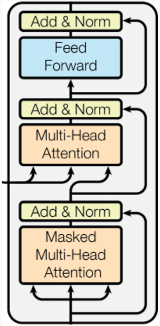
$$x_t^{\top} \quad \underset{\text{Fed}}{\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}} \quad \underset{\text{raises}}{\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}} \quad \underset{\text{interest}}{\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}} \quad \underset{\text{rates}}{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}$$

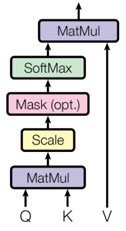




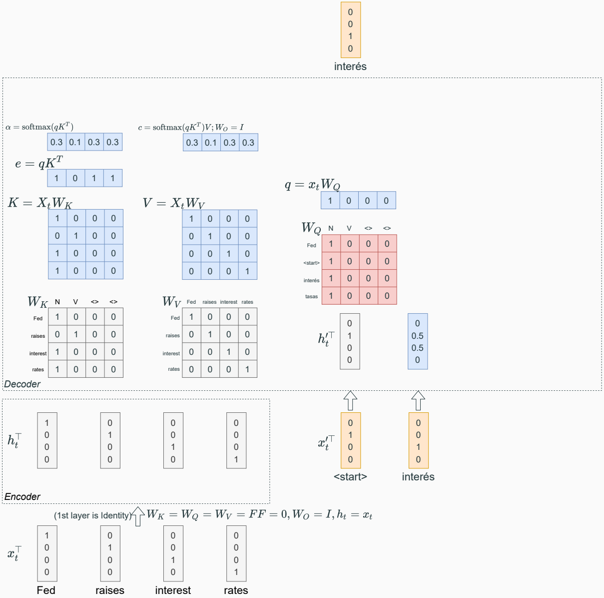
Transformer in action: Toy example (Noun translation)

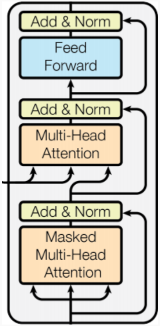


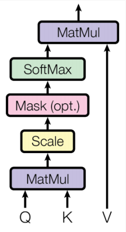




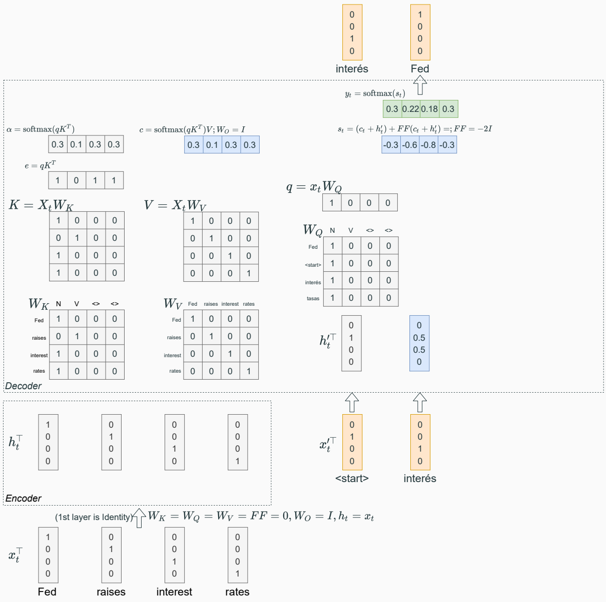
Transformer in action: Toy example (Noun translation)

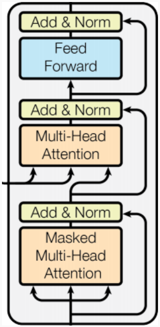


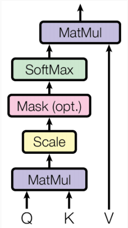




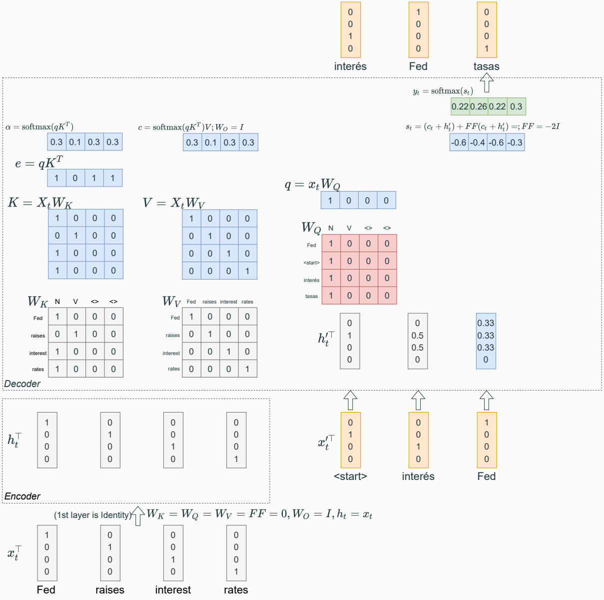
Transformer in action: Toy example (Noun translation)

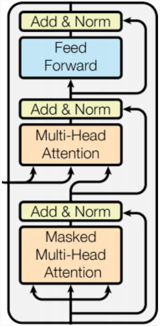


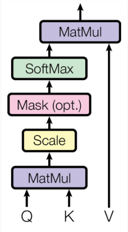




Transformer in action: Toy example (Noun translation)







Useful Resources

- Blog post by Sebastian Ruder
- Blog post by Lilian Weng
- The Annotated Transformer by Alexander Rush
- The Illustrated Transformer by Jay Alammar