Alternative Architectures for Neural Machine Translations

Weiting (Steven) Tan

- RNN + Attention Recap
- Convolutional Neural Net
  - How it works in encoder/decoder
  - CNN + Attention
- Transformer

Feed-Forward Layer

- Classic neural network component
- Given an input vector x, matrix multiplication M with adding a bias vector b

$$Mx + b$$

- Adding a non-linear activation function

$$y = \text{activation}(Mx + b)$$

- Notation

$$y = FF_{\text{activation}}(x) = a(Mx + b)$$

Recurrent Neural Network

- propagate state s
- over time steps t
- receiving an input x_t at each turn

$$s_t = f(s_{t-1}, x_t)$$

(state may computed may as a feed-forward layer)

Recurrent Neural Network

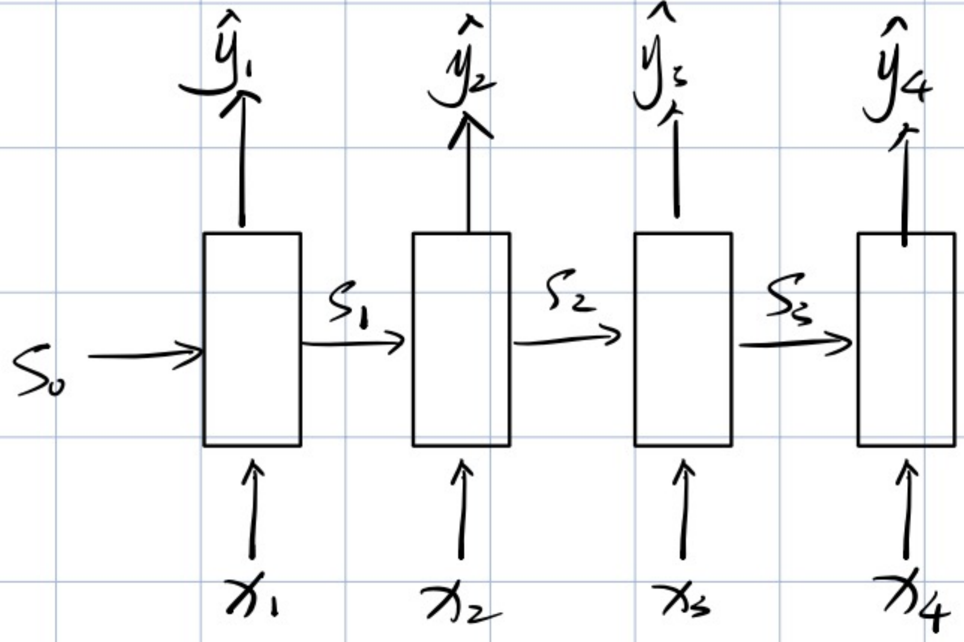
- propagate state s
- over time steps t
- receiving an input x_t at each turn

$$s_t = f(s_{t-1}, x_t)$$

(state may computed may as a feed-forward layer)

$$S_t = f(S_{t-1}, x_t) = \sigma(W_s \vec{S}_{t-1}, W_x \vec{x}_t + b)$$

$$y_t = \mathrm{softmax}\{\sigma(W_y S_t + b')\}$$



$x_t \in R^d$

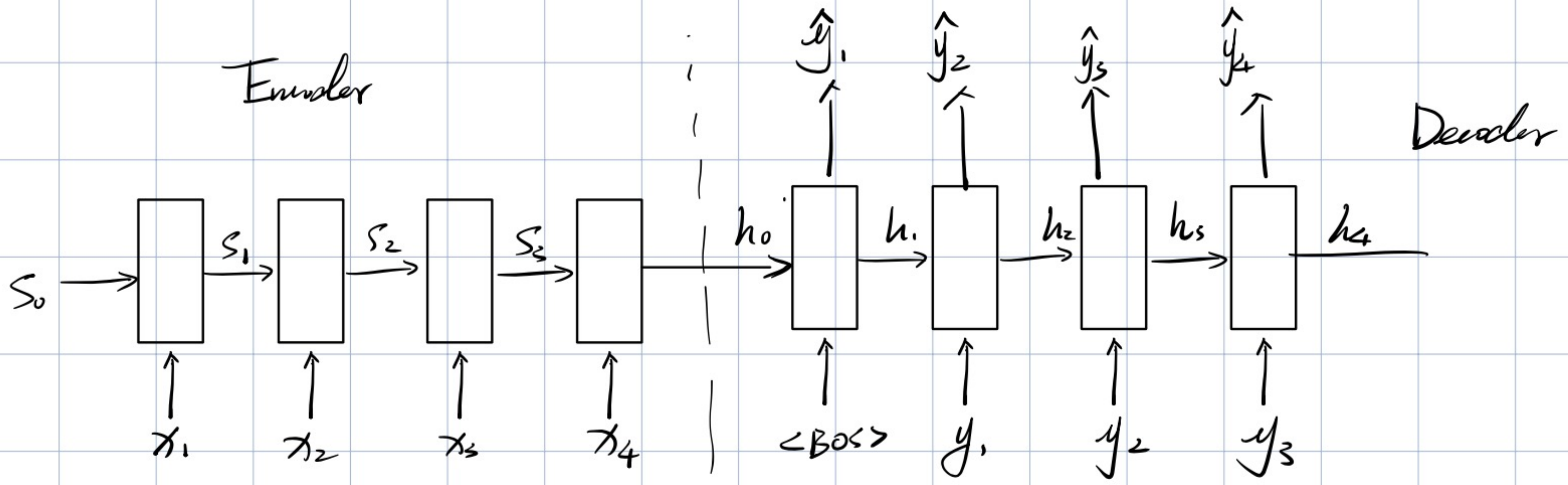
$S_t \in R^s$

$W_x \in R^{s \times d}$

$W_s \in R^{s \times s}$

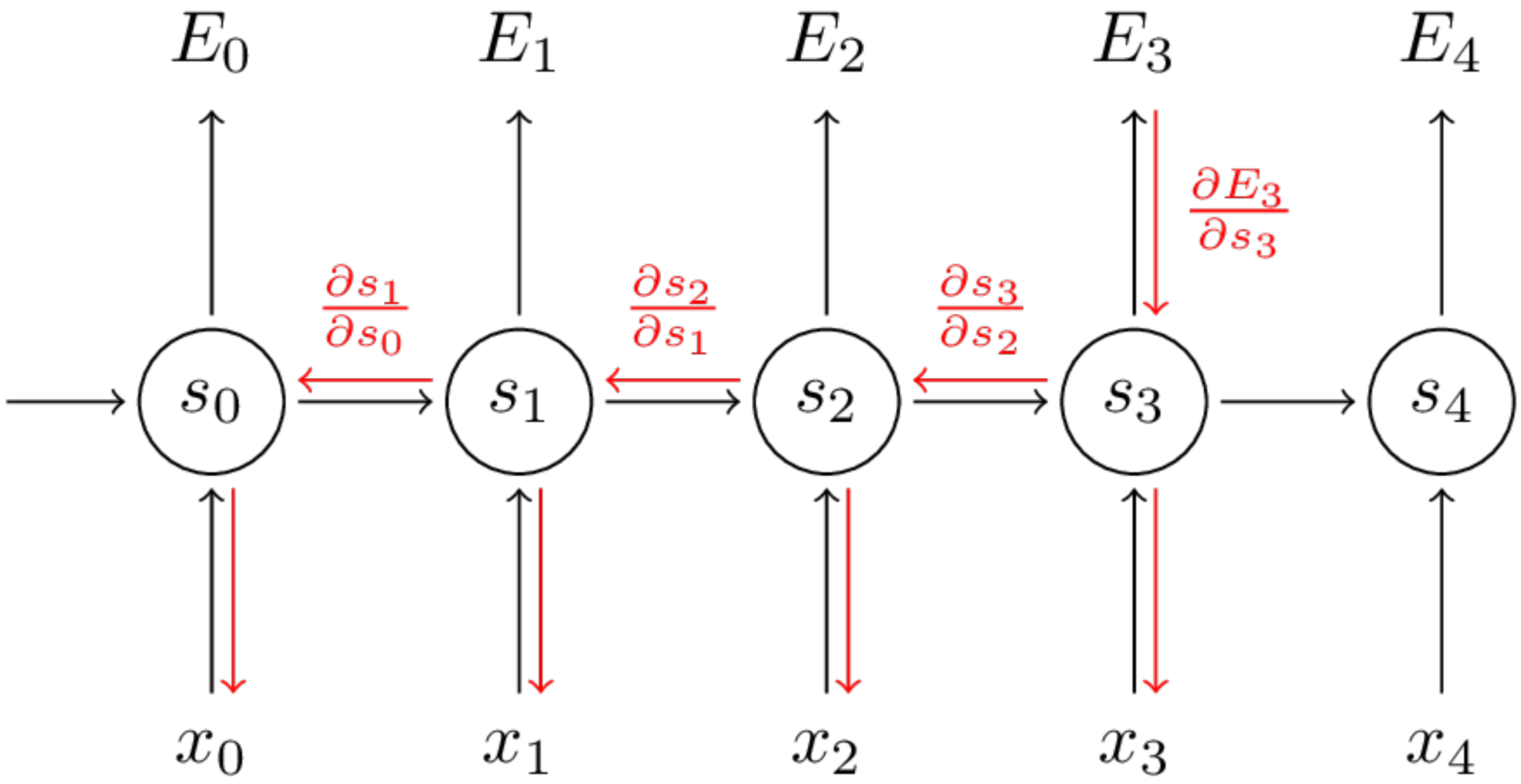
$W_y \in R^{V \times s}$

Encoder - Decoder



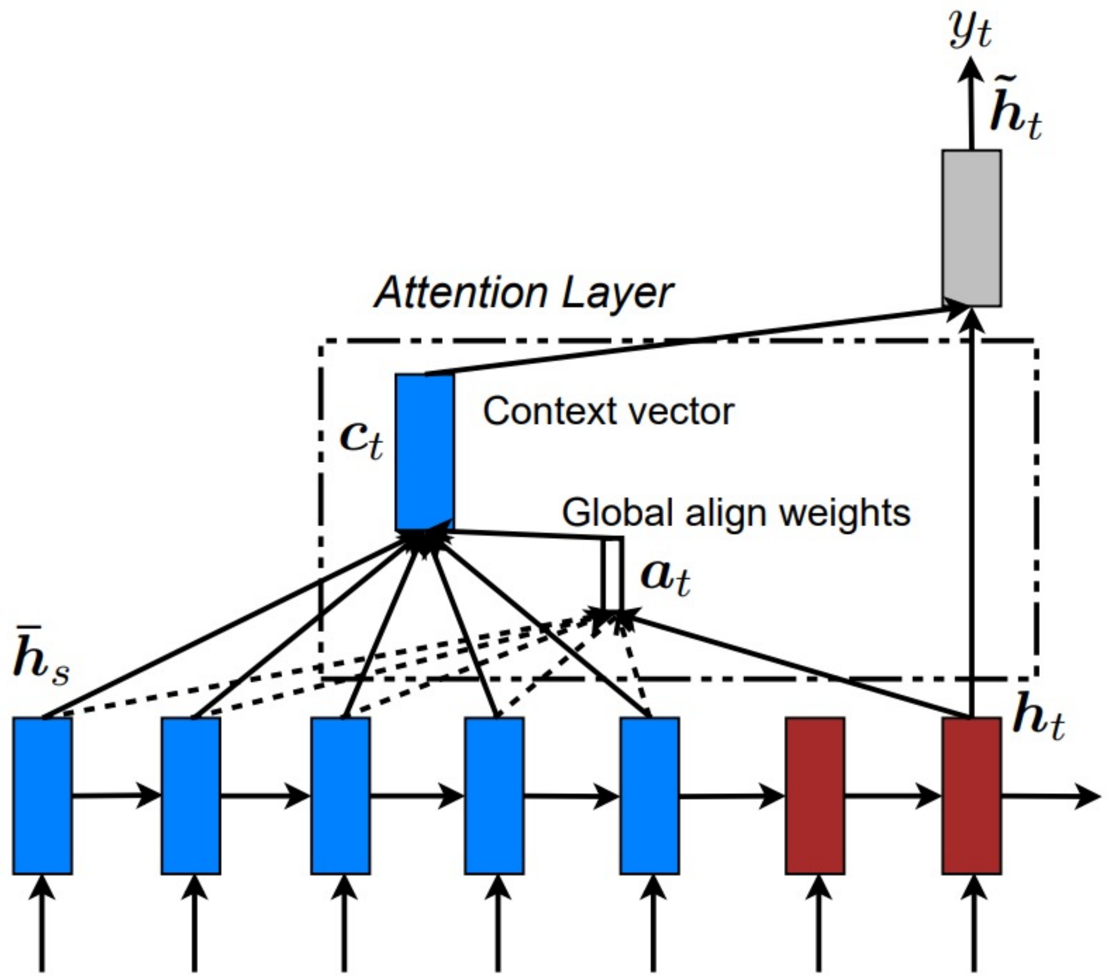
RNN's Issue: Vanishing Gradient

- Mitigation:
  - GRU
  - LSTM



RNN with Attention

- Luong et al 2015
- Encoder-Decoder Attention

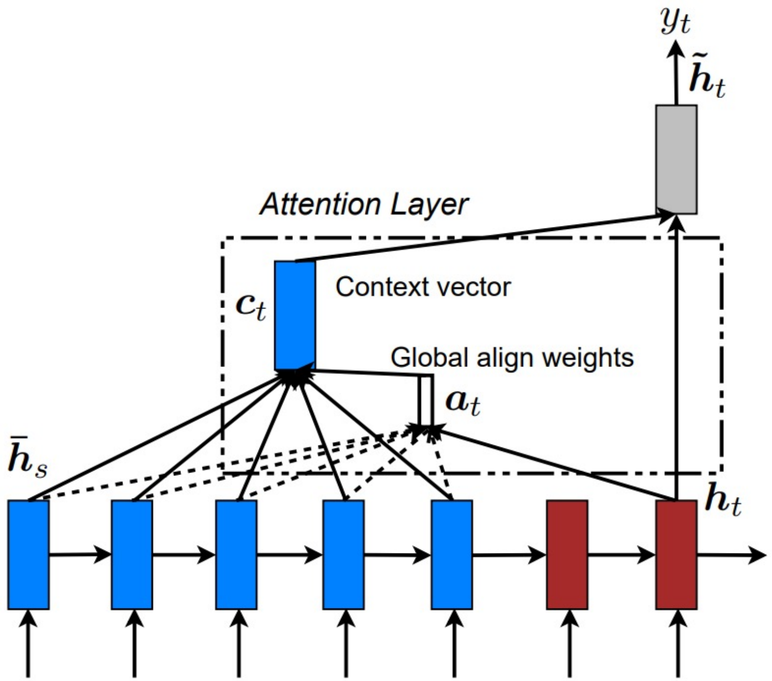


RNN with Attention

- Luong et al 2015
- Encoder-Decoder Attention

Other ways to compute attention

- Dot product: $a(s_{i-1}, h_j) = s_{i-1}^T h_j$
- Scaled dot product: $a(s_{i-1}, h_j) = \frac{1}{\sqrt{|h_j|}} s_{i-1}^T h_j$
- General: $a(s_{i-1}, h_j) = s_{i-1}^T W_a h_j$
- Local: $a(s_{i-1}) = W_a s_{i-1}$



(Dot product)

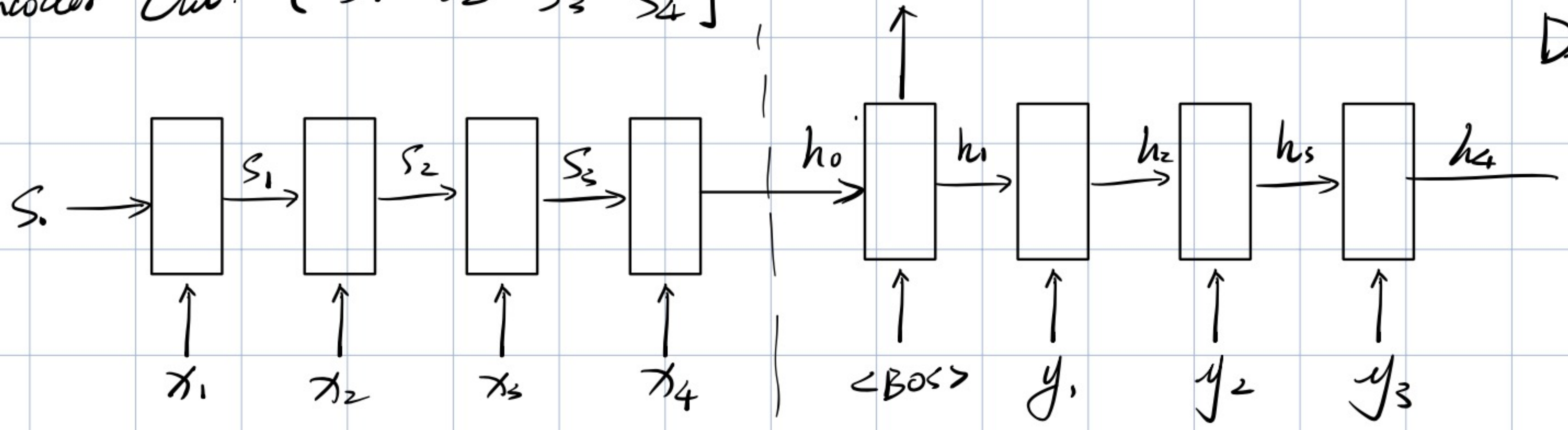
$$\alpha = \text{softmax}([\underbrace{h_1^T S_1}_{\text{scalar}},\ h_1^T S_2,\ h_1^T S_3,\ h_1^T S_4])$$

E-D w. Attention

Encoder Out: $[S_1\ \ S_2\ \ S_3\ \ S_4]$

$C_1 = \sum \alpha_i S_i$

Decoder



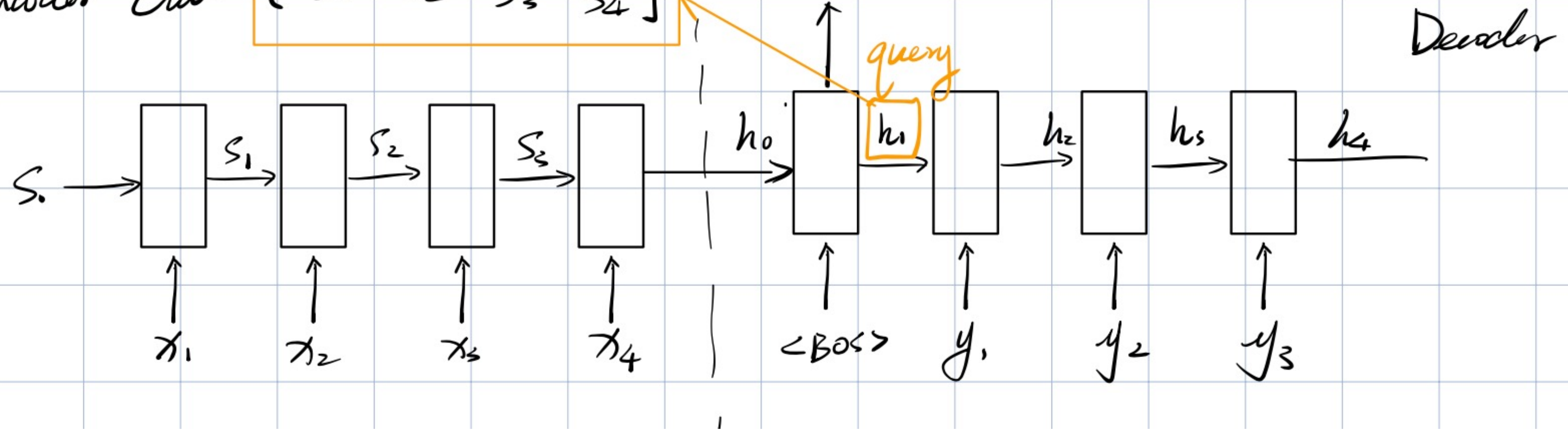
E-D w. Attention

$$\alpha = \text{softmax}([\underbrace{h_1^T s_1}_{\text{scalar}}, h_1^T s_2, h_1^T s_3, h_1^T s_4]) \quad \text{(Dot product)}$$

both keys and Value

Encoder Out: $\boxed{[s_1 \ s_2 \ s_3 \ s_4]}$

$$c_1 = \sum \alpha_i s_i$$



RNN's issue: SLOW

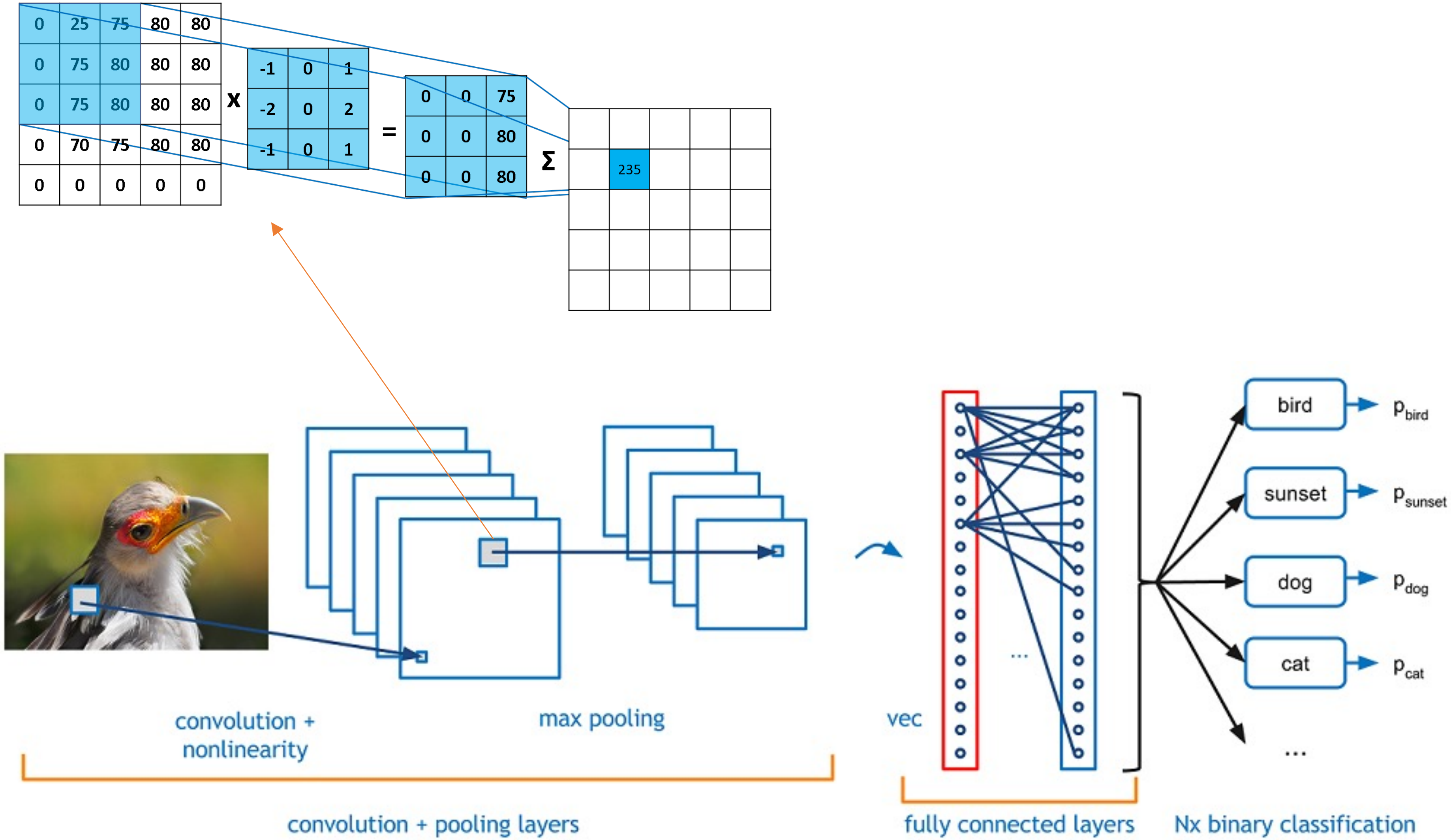
Alternative Sequence Modeling

- Convolutional Neural Networks
- Transformer (Self-Attention)

Encoder: parallelized computation

Decoder: parallelized during training, autoregressive during inference

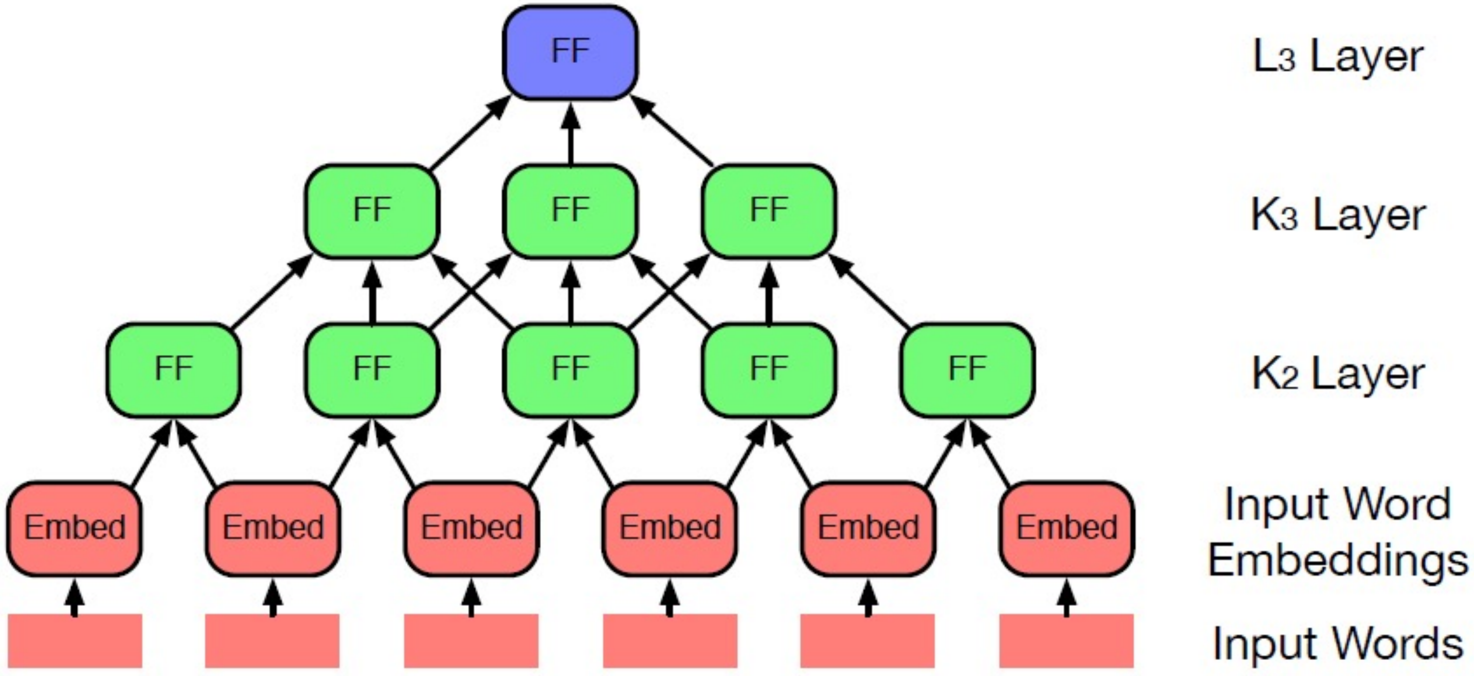
ConvNet for 2D Images



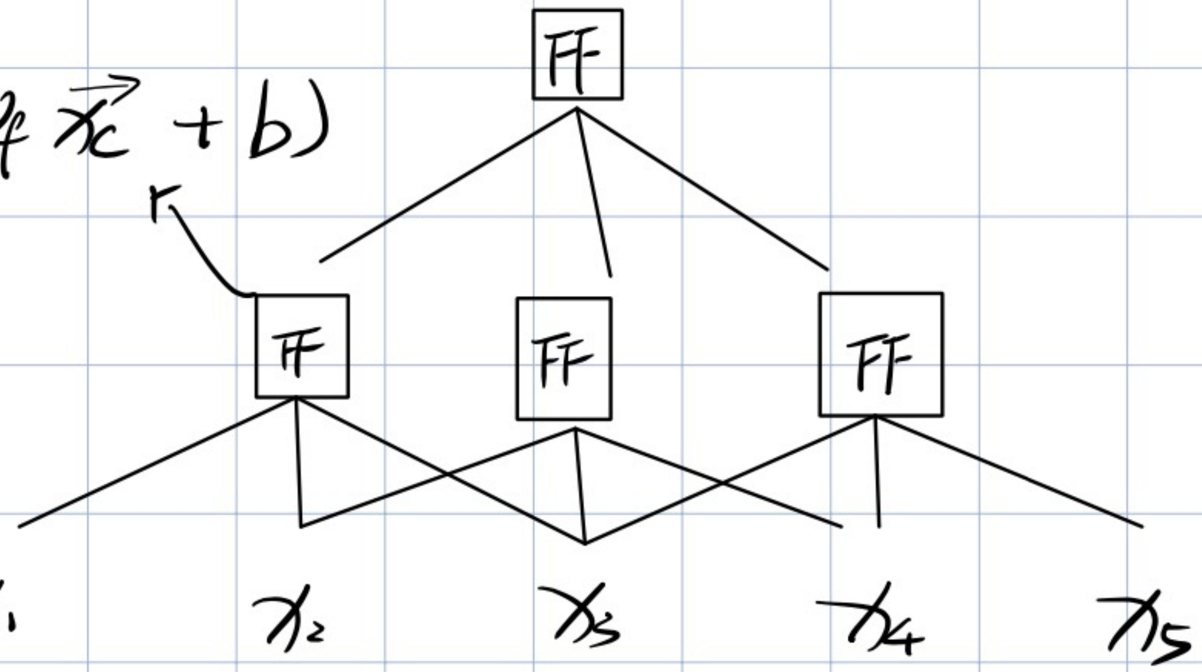
ConvNet for Seq2Seq (Encoder)

- First end-to-end neural machine translation model of the modern era [Kalchbrenner and Blunsom, 2013]

- Encoder



$\sigma(W_f \vec{x_c} + b)$



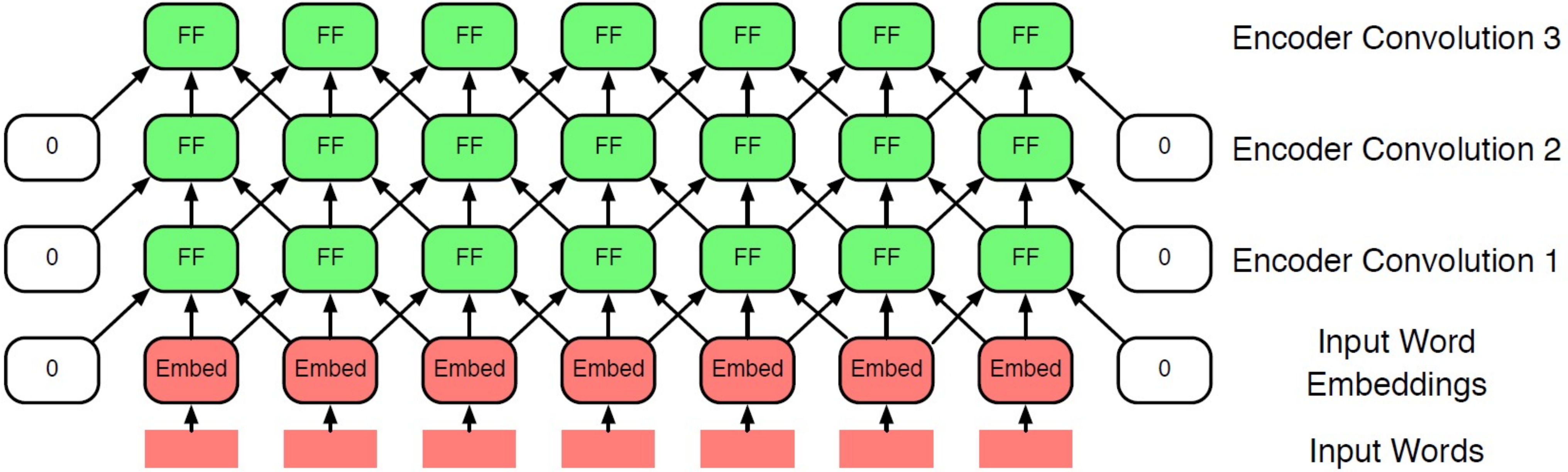
$x \in R^d$

$\vec{x_c} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \in R^{3d}$

$W_f \in R^{d \times 3d}$

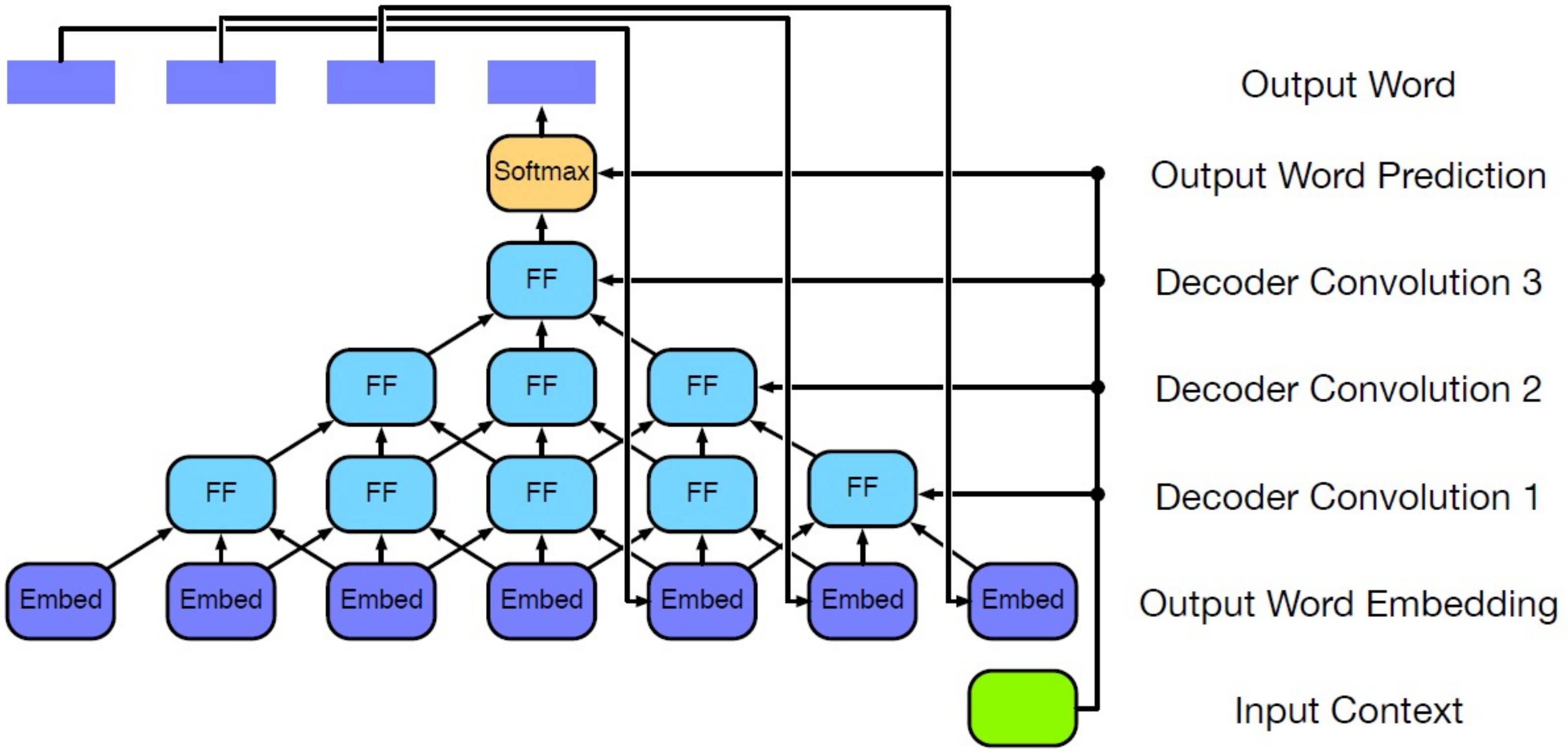
ConvNet with Attention

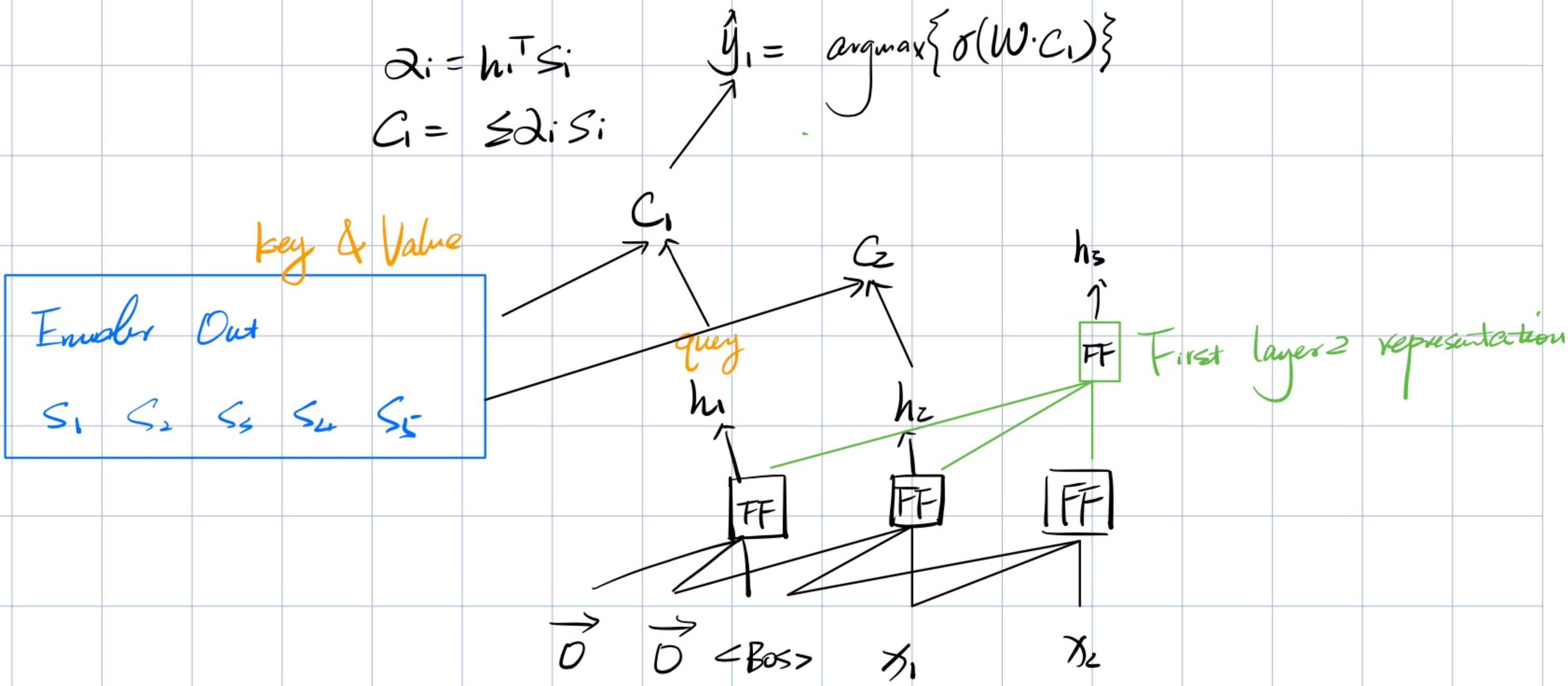
- Gehring et al. 2017
- Encoder:
  - Padding from left and right (keep seq-len)
  - Parallel computation



ConvNet with Attention

- Decoder:
  - Autoregressive
  - Stacking of feed-forward net as decoding is processed

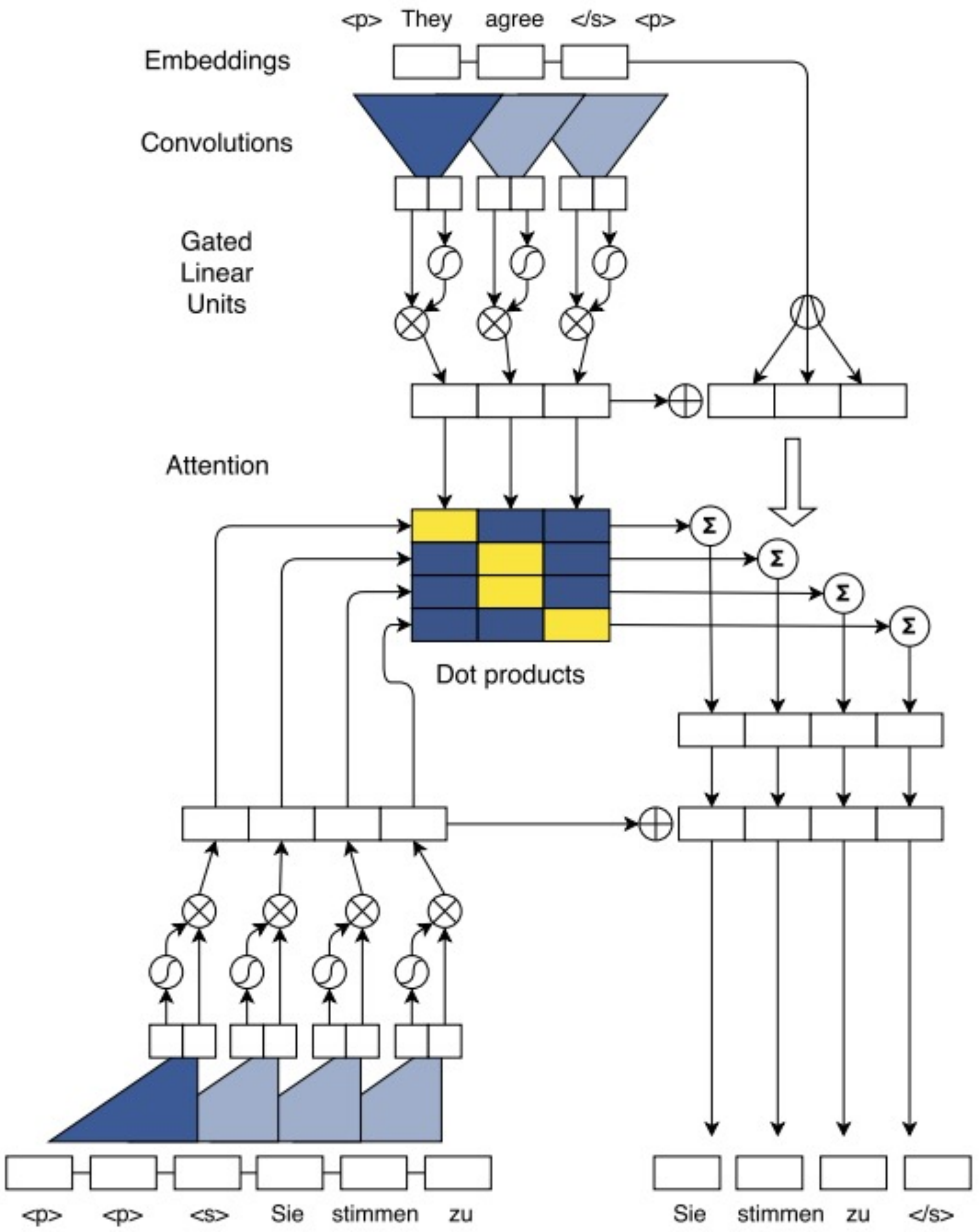




$\alpha_i = h_1^T s_i$
$C_1 = \sum \alpha_i s_i$
$\hat{y}_1 = \text{argmax}\{\sigma(W \cdot C_1)\}$
C_1
C_2
h_3
key & Value
Encoder Out
s_1 s_2 s_3 s_4 s_5
query
h_1
h_2
FF
First layer2 representation
FF FF FF
$\vec{0}$ $\vec{0}$ <Bos> x_1 x_2

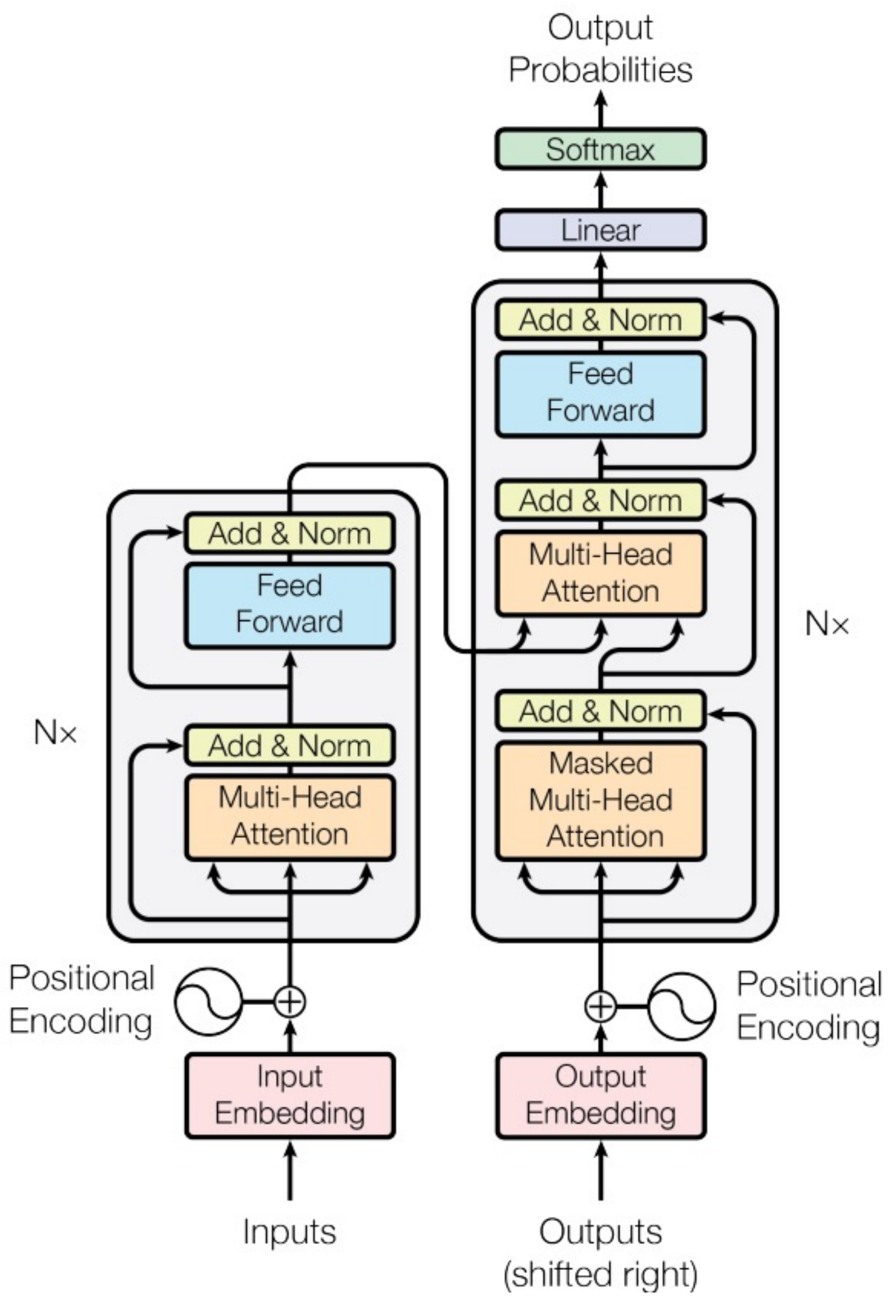
ConvNet with Attention

- Encoder-Decoder
- Attention is the simple dot product



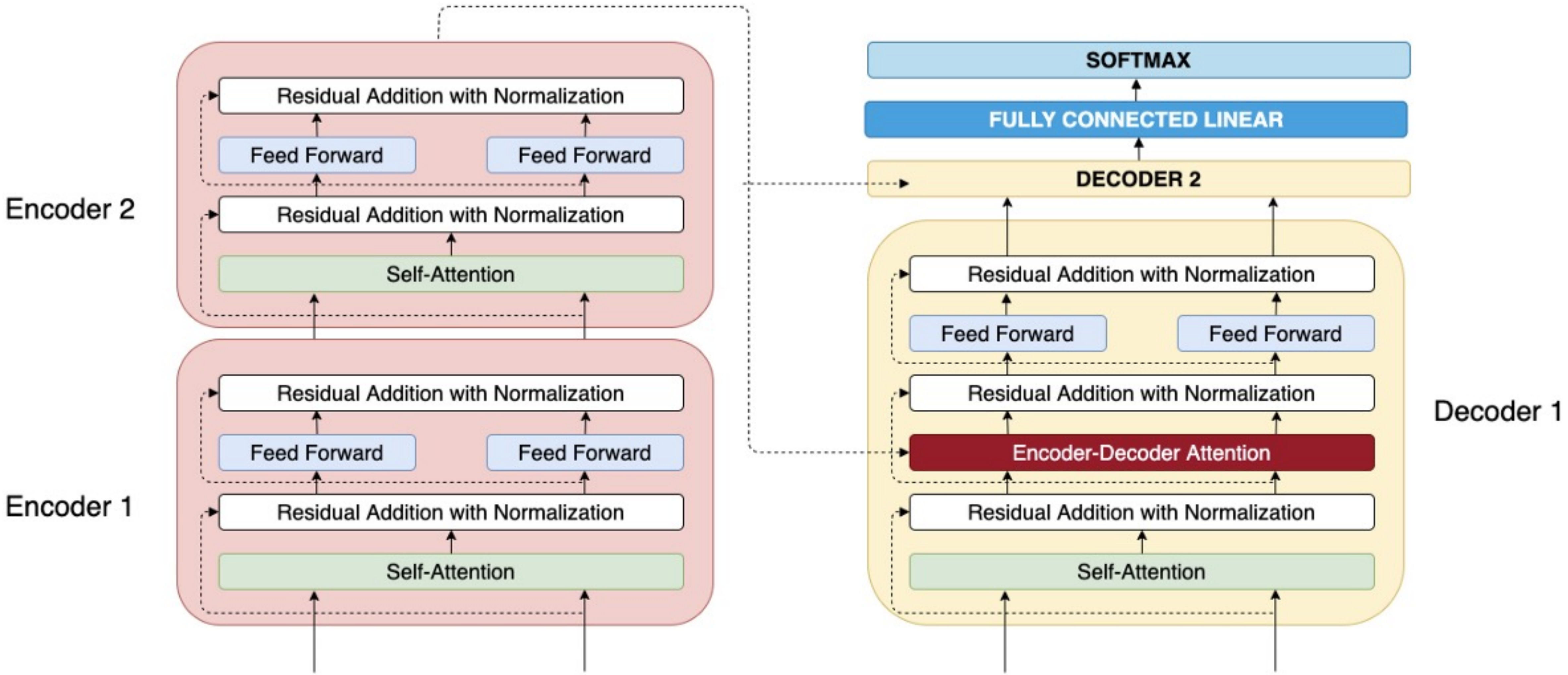
Transformer

1. Self-Attention and Encoder-Decoder Attention
2. Multi-Head Attention
3. Positional Encoding/Embedding

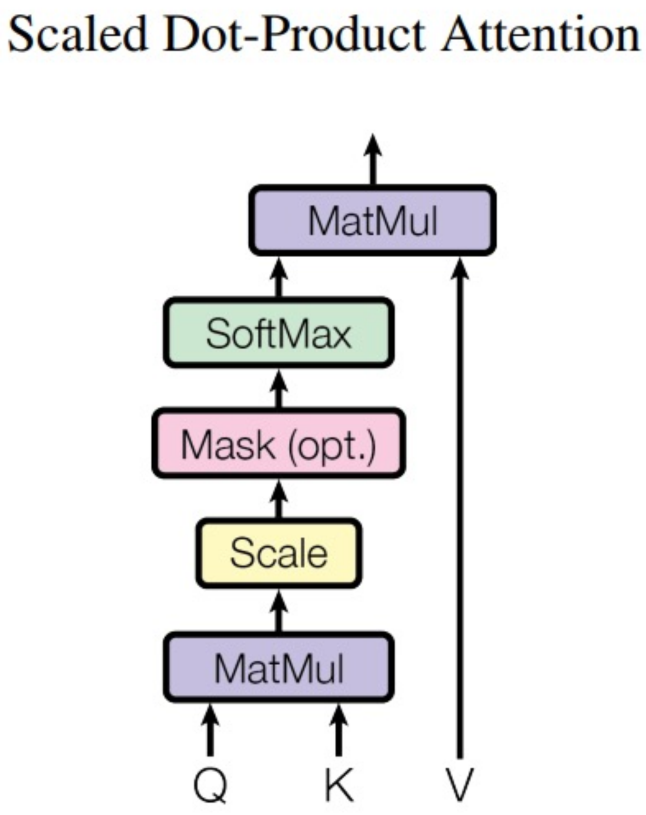


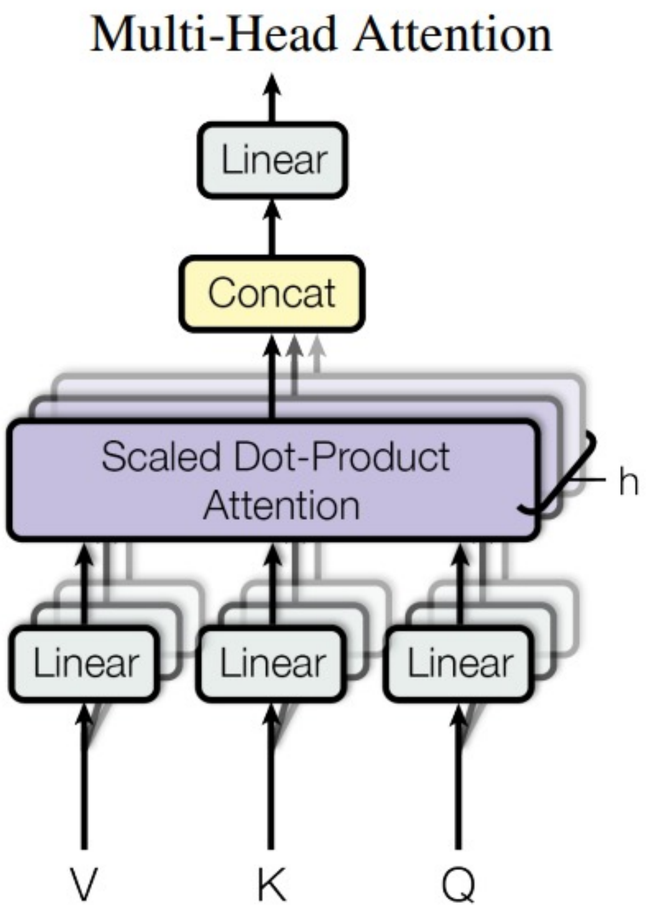
Jay Alammar https://jalammar.github.io/illustrated-transformer/

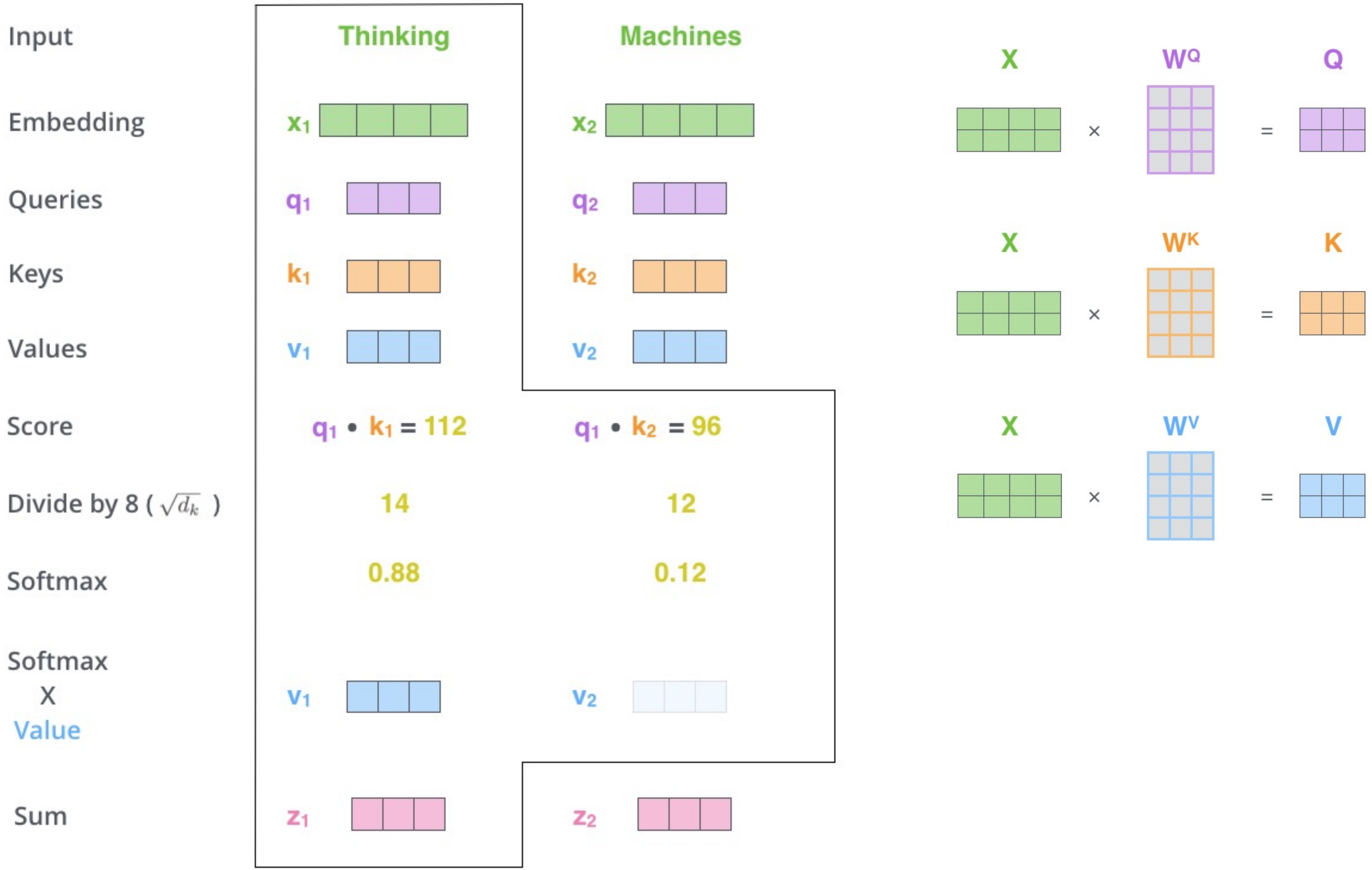
Transformer



Transformer Encoder Attention





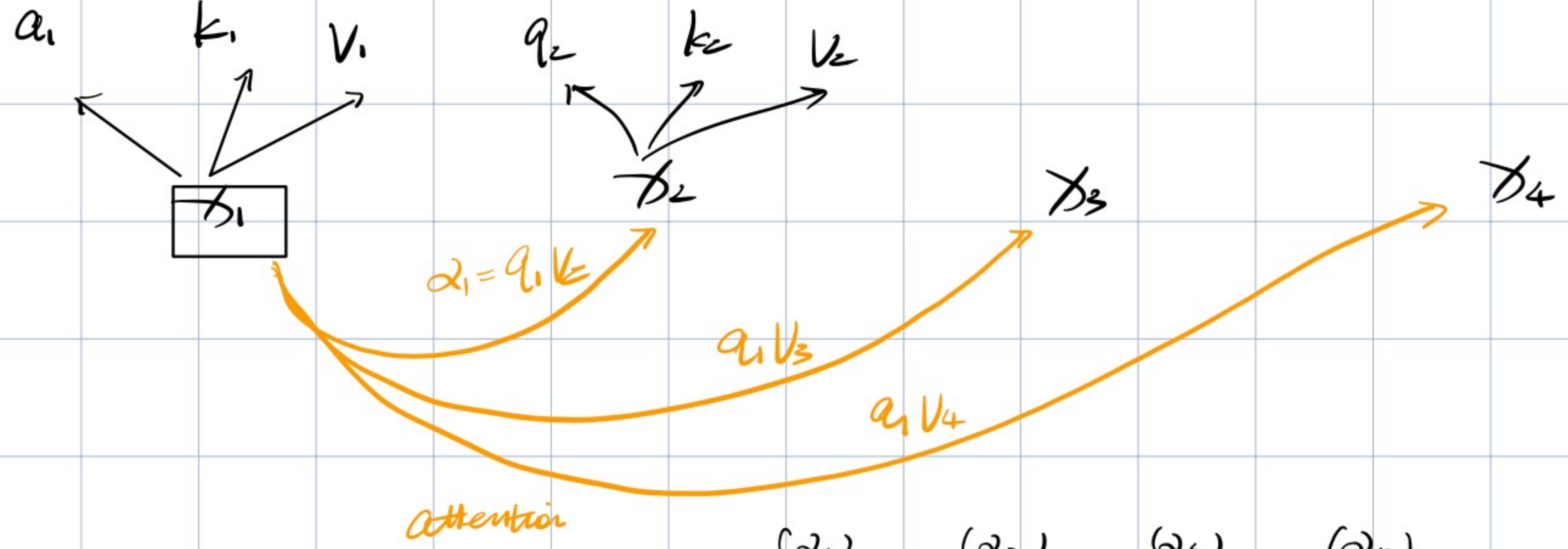


Input
Embedding
Queries
Keys
Values
Score
Divide by 8 ($\sqrt{d_k}$)
Softmax
Softmax
X
Value
Sum
Thinking
Machines
x_1
x_2
q_1
q_2
k_1
k_2
v_1
v_2
$q_1 \cdot k_1 = 112$
$q_1 \cdot k_2 = 96$
14
12
0.88
0.12
v_1
v_2
z_1
z_2
X
W^Q
Q
X
W^K
K
X
W^V
V
×
=
×
=
×
=

Self - Attention

$$\begin{cases} q_1 = W_Q \cdot x_1 \\ k_1 = W_K \cdot x_1 \\ v_1 = W_V \cdot x_1 \end{cases} \in R^h$$

$$x \in R^d$$
$$W_Q \in R^{m \times d}$$
$$W_K \in R^{m \times d}$$
$$W_V \in R^{m \times d}$$



$$\alpha = \mathrm{softmax}\{[\underset{(\alpha_1)}{q_1 v_1},\ \underset{(\alpha_2)}{q_2 v_2},\ \underset{(\alpha_3)}{q_3 v_3},\ \underset{(\alpha_4)}{q_4 v_4}]\}$$

$$z_1 = \sum \alpha_i v_i \quad \in R^h$$

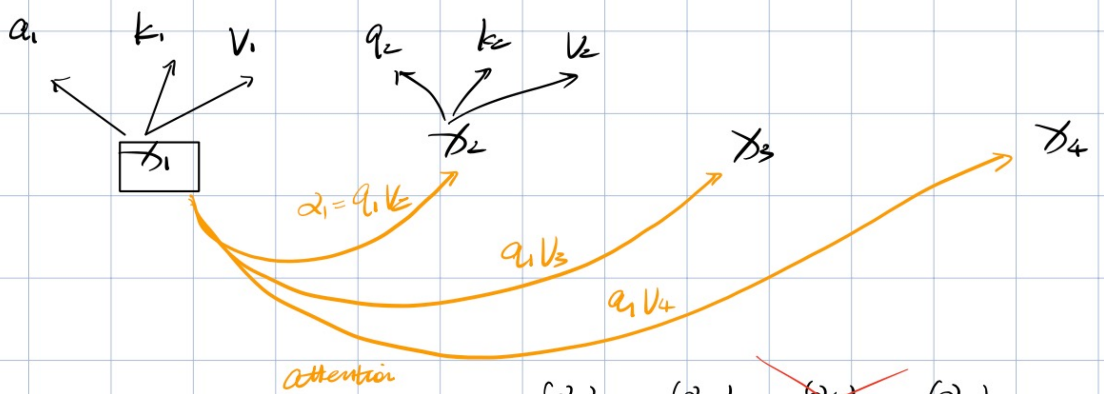
Encoder Output : $[z_1 \quad z_2 \quad z_3 \quad z_4]$

Multihead : $\begin{cases} W_Q^1 \; W_K^1 \; W_V^1 \rightarrow [z_1^1 \; z_2^1 \; z_3^1 \; z_4^1] \\ W_Q^2 \; W_K^2 \; W_V^1 \rightarrow [z_1^2 \; z_2^2 \; z_3^2 \; z_4^2] \\ - - - - \end{cases}$

For x_1 now we have

$$\begin{bmatrix} z_1^1 \\ -- \\ z_1^2 \\ z_1^5 \\ \vdots \\ z_1^8 \end{bmatrix} \in \mathbb{R}^{8h} \xrightarrow[\text{FF layer}]{W_Z \in \mathbb{R}^{h \times 8h}} \mathbb{R}^h$$

Decoder's Self-attention



$$\alpha = \text{softmax}\{[\overset{(\alpha_1)}{q_1 v_1},\ \overset{(\alpha_2)}{q_2 v_2},\ \cancel{\overset{(\alpha_3)}{q_3 v_3}},\ \cancel{\overset{(\alpha_4)}{q_4 v_4}}]\}$$

cannot expose the "futur

$$z_1 = \sum \alpha_i v_i \in \mathbb{R}^h$$

for x_1 $\quad \alpha = \text{softmax}\{[q_1 v_1, -\text{inf}, -\text{inf}, -\text{inf}]\}$

for x_2 $\quad \alpha = \text{softmax}\{[q_2 v_1, q_2 v_2, -\text{inf}, -\text{inf}]\}$

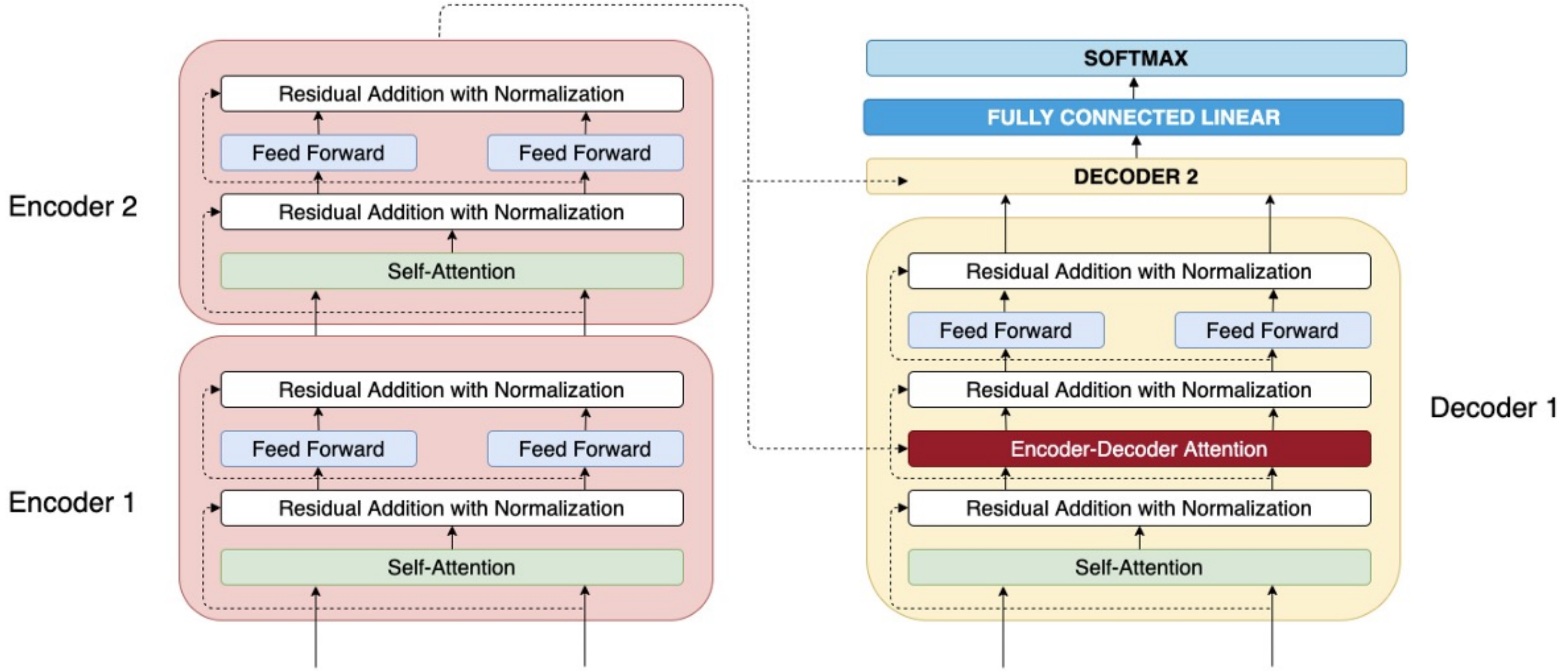
- - -

for x_4 $\quad \alpha = \text{softmax}\{[q_4 v_1, q_4 v_2, q_4 v_3, q_4 v_4]\}$

mask:

Transformer Decoder

- Self-attention with mask
- Encoder-Decoder Attention
  - Query is from decoder
  - Key and Value are from Encoder



Encoder - Decoder Attention in transformer

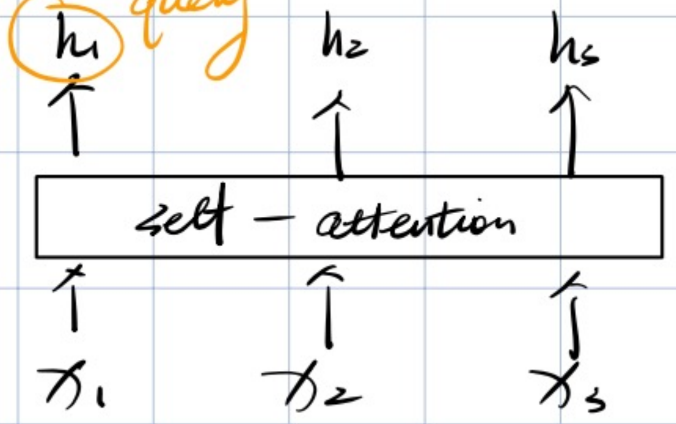
Encoder Output

$$\underbrace{[\, S_1 \quad S_2 \quad S_3 \quad S_4 \,]}_{\text{both key and value}}$$

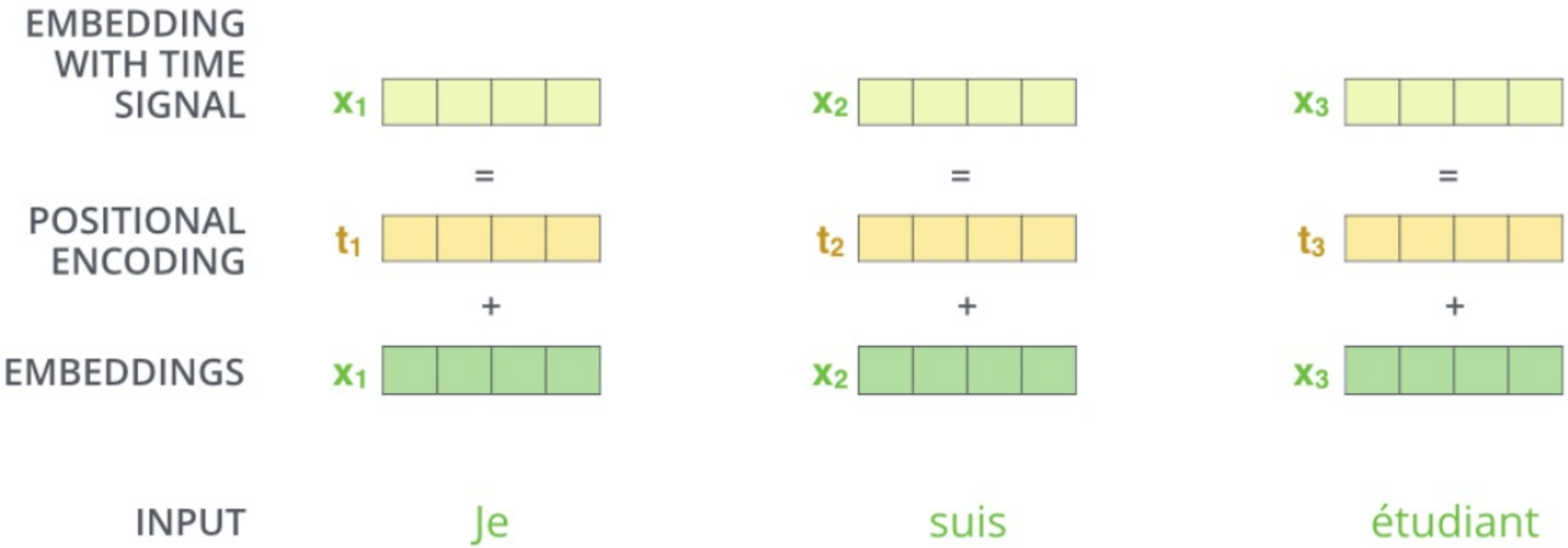
$$\alpha = \text{softmax}(h_1^T S) \rightarrow [h_1^T s_1, h_1^T s_2, \ldots h_1^T s_4]$$

$$z_1 = \sum \alpha_i s_i$$

query



Positional Embedding



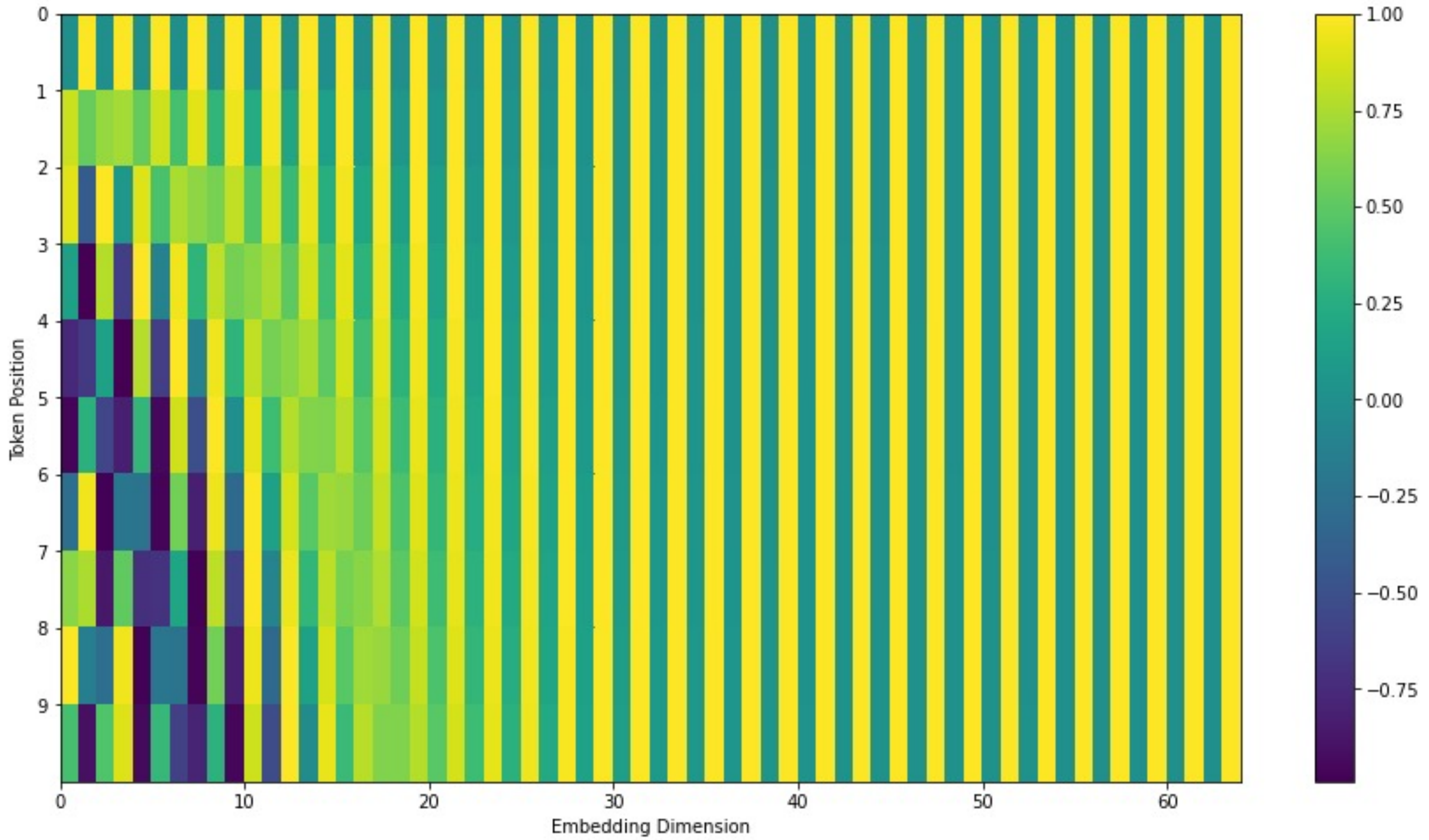
Positional Embedding

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

$$\overrightarrow{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k+1 \end{cases}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$

Positional Embedding



https://github.com/jalammar/jalammar.github.io/blob/master/notebookes/transformer/transformer_positional_encoding_graph.ipynb