CS 4644-DL / 7643-A: LECTURE 17
DANFEI XU

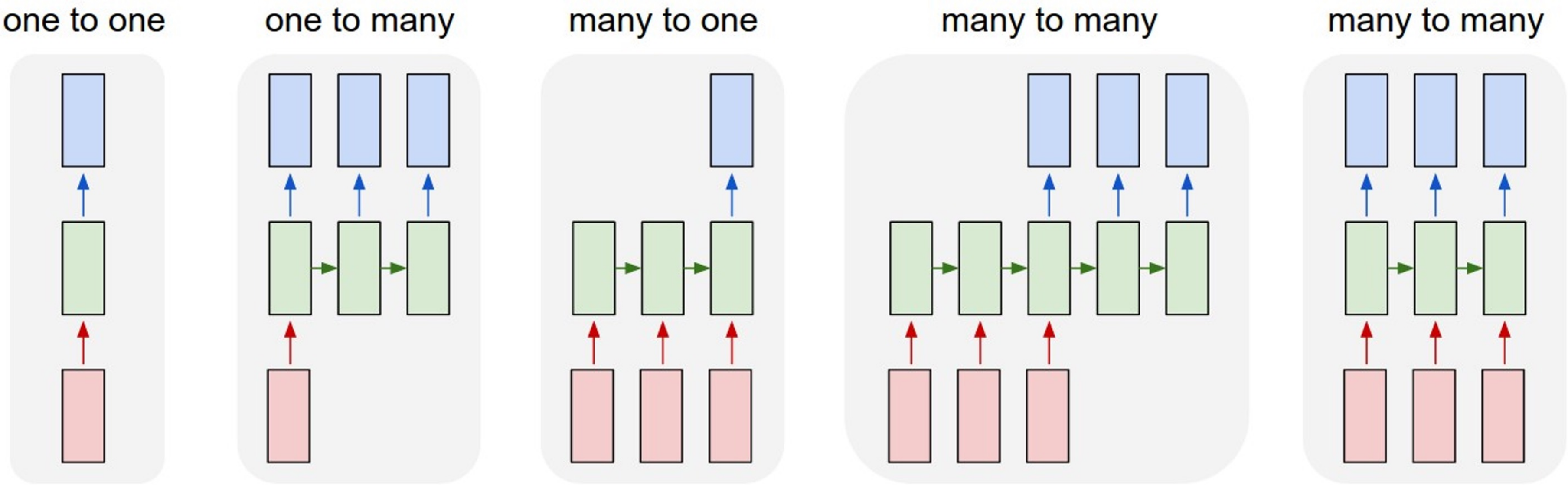
Attention for Sequence Modeling

Attention is (Mostly) All you Need: Transformers

Administrative:

- HW3 due 10/25
- Milestone Report due 11/3

Recurrent Neural Networks: Process Sequences



one to one

one to many

many to one

many to many

many to many

RNN hidden state update

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

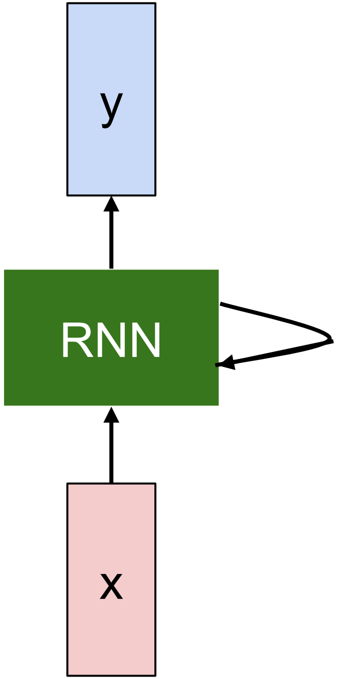
$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state (vector)

some function with parameters W

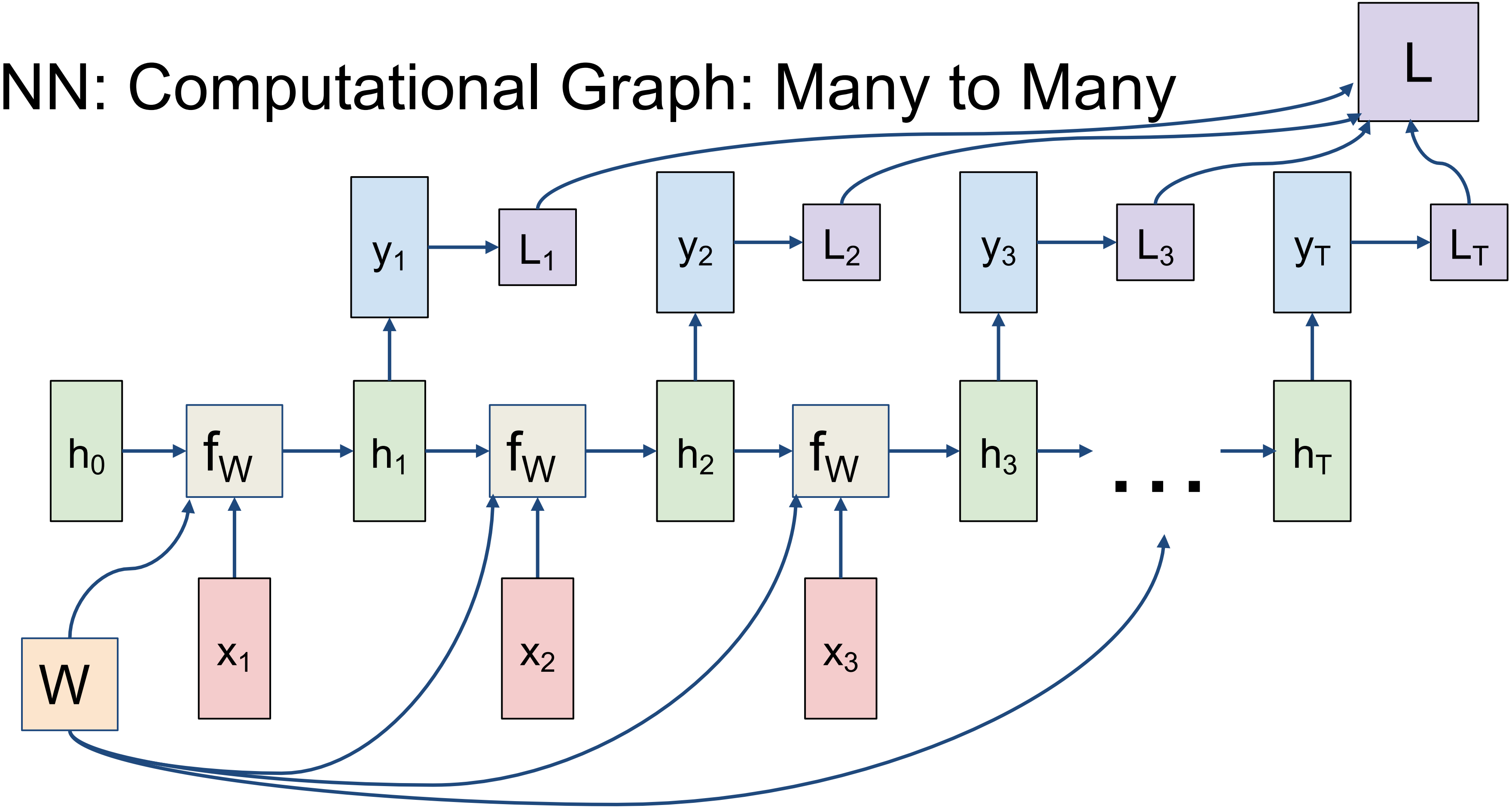
old state (vector)

input vector at some time step

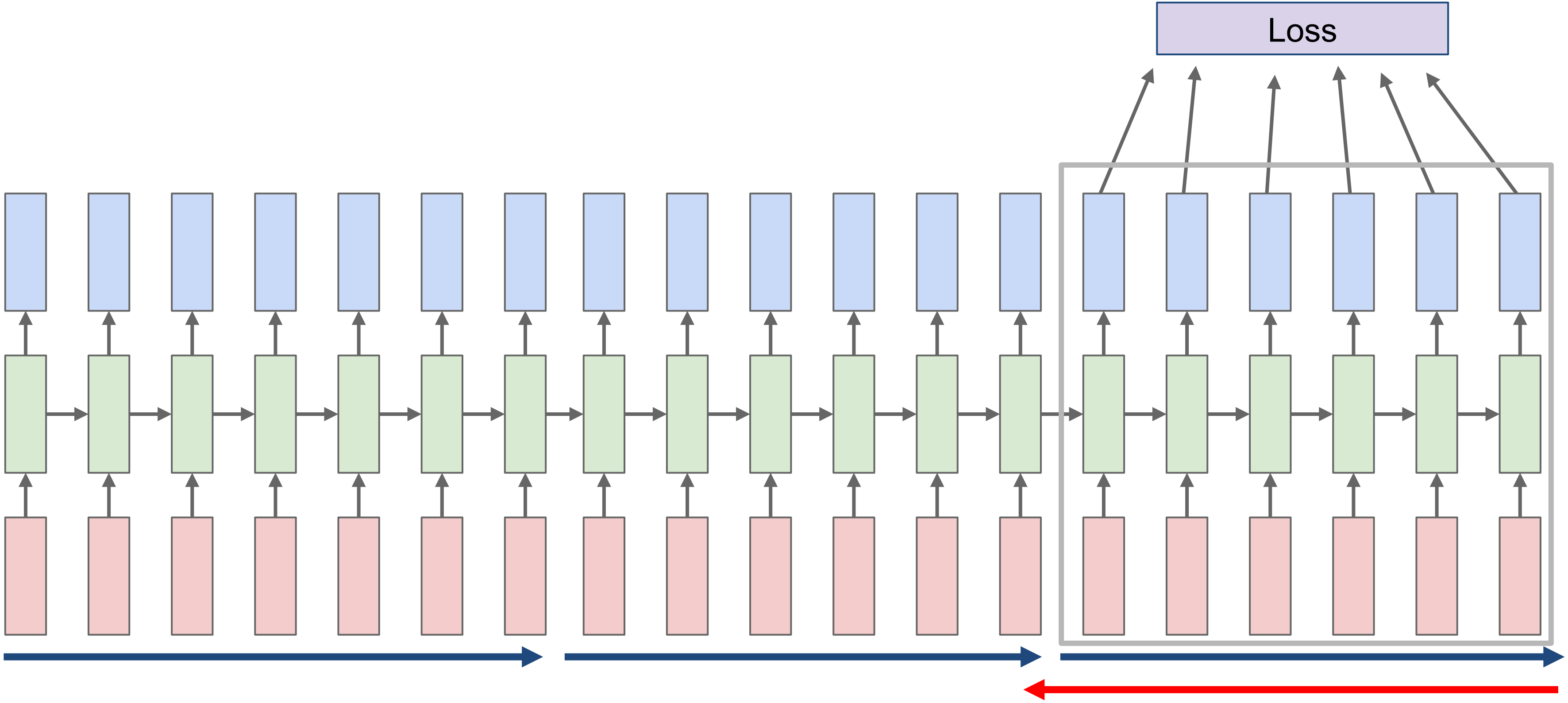


Can set initial state h_0 to all 0's

RNN: Computational Graph: Many to Many



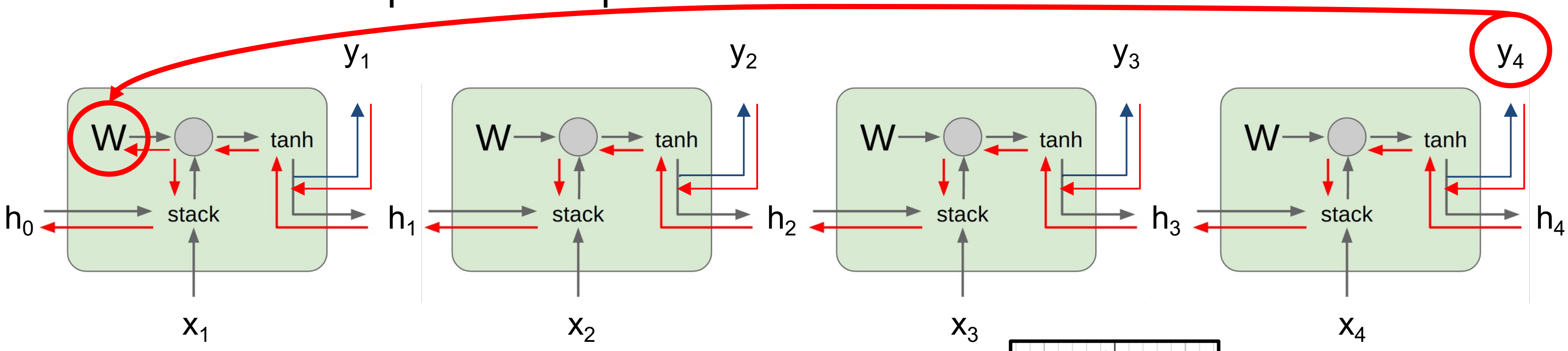
Truncated Backpropagation through time



Vanilla RNN Gradient Flow

Gradients over multiple time steps:

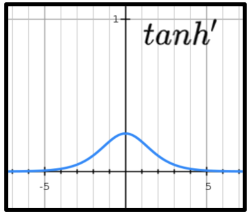
Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Always < 1

Vanishing gradients



$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T}\left(\prod_{t=2}^{T} \boxed{tanh'(W_{hh}h_{t-1} + W_{xh}x_t)}\right) W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Long Short Term Memory (LSTM)

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

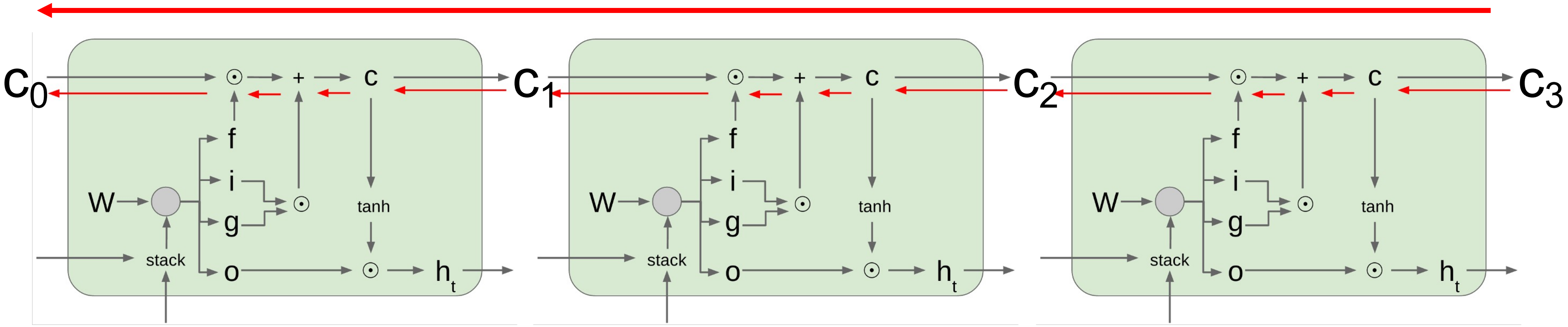
Learn to control information flow from previous state to the next state

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!



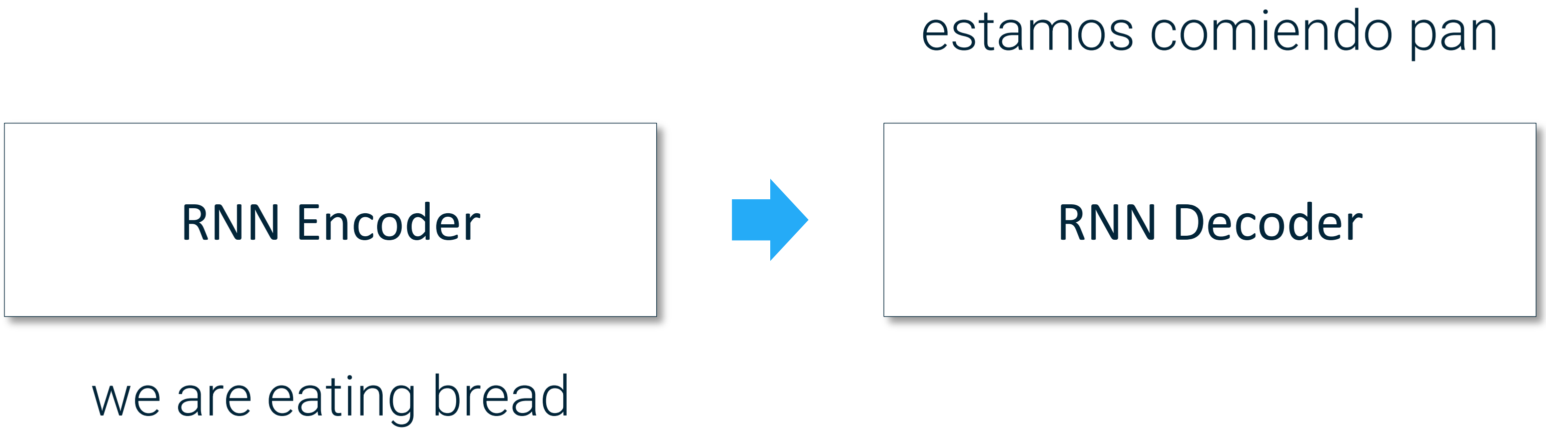
Notice that the gradient contains the **f** gate’s vector of activations

- allows better control of gradients values, using suitable parameter updates of the forget gate.

Also notice that are added through the **f, i, g,** and **o** gates

- better balancing of gradient values

Machine Translation



estamos comiendo pan

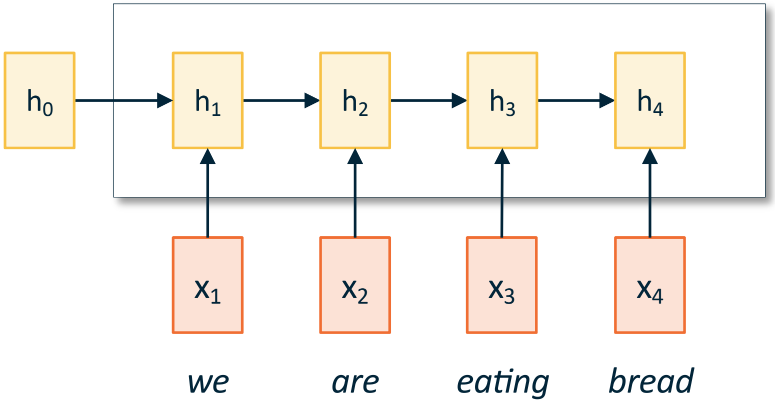
RNN Encoder

RNN Decoder

we are eating bread

Machine Translation with RNNs

Encoder: $h_t = f_W(x_t, h_{t-1})$

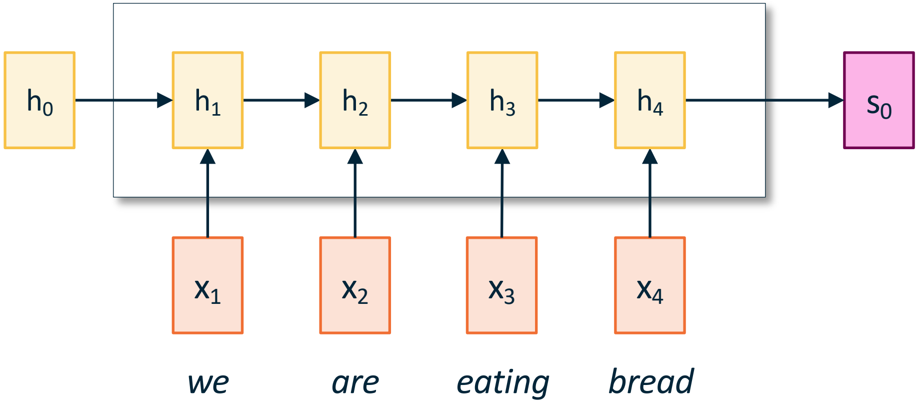


Slide credit: Justin Johnson

Machine Translation with RNNs

Encoder: $h_t = f_W(x_t, h_{t-1})$

$\mathbf{s_0 = h_4}$

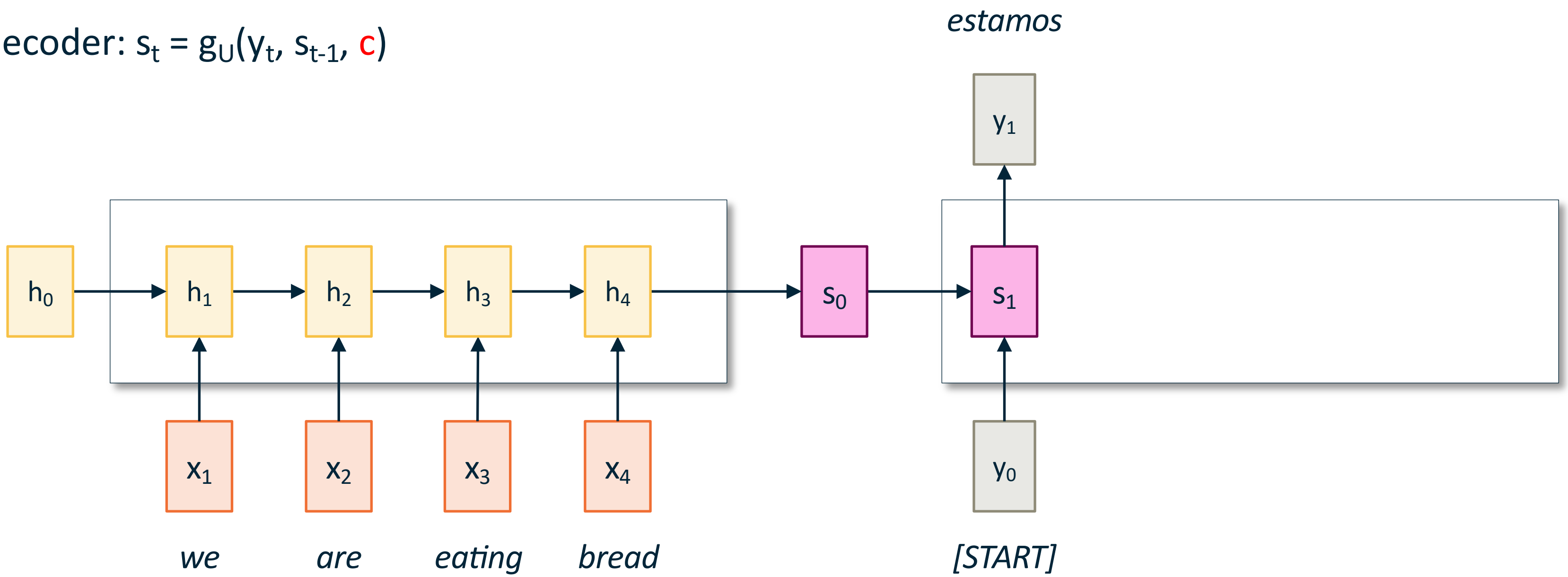


Slide credit: Justin Johnson

Machine Translation with RNNs

Encoder: $h_t = f_W(x_t, h_{t-1})$

Decoder: $s_t = g_U(y_t, s_{t-1}, c)$

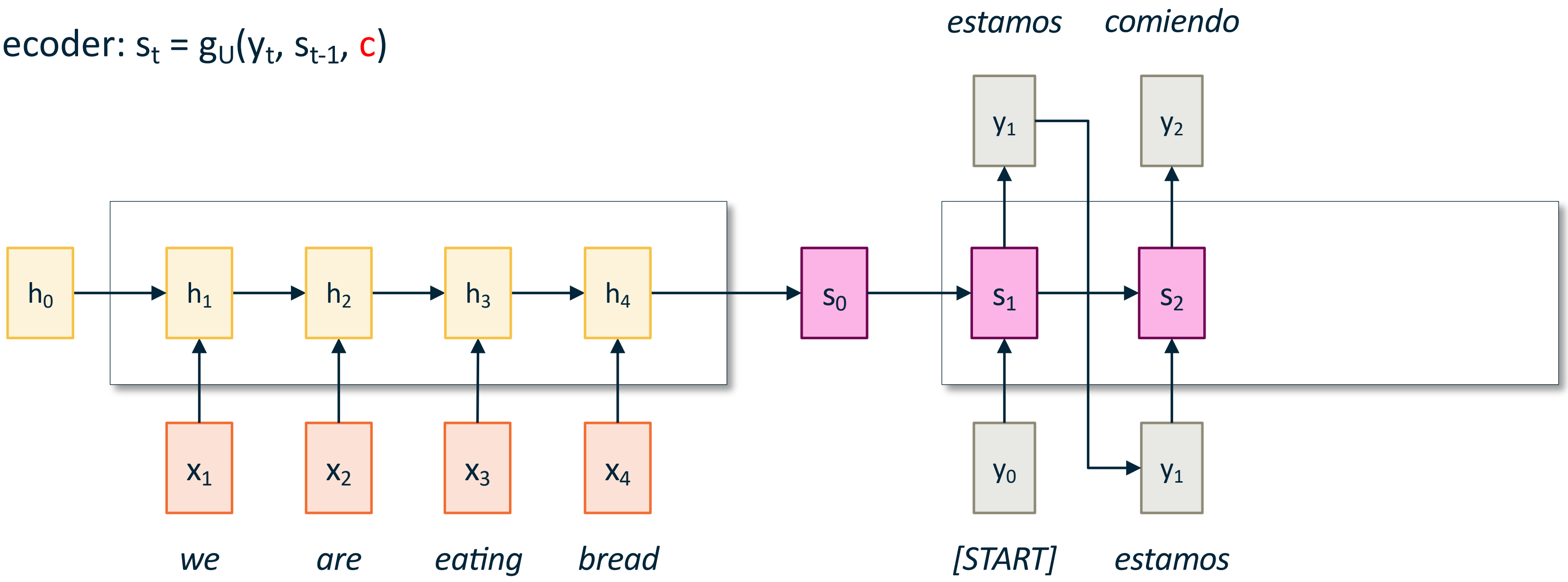


Slide credit: Justin Johnson

Machine Translation with RNNs

Encoder: $h_t = f_W(x_t, h_{t-1})$

Decoder: $s_t = g_U(y_t, s_{t-1}, c)$

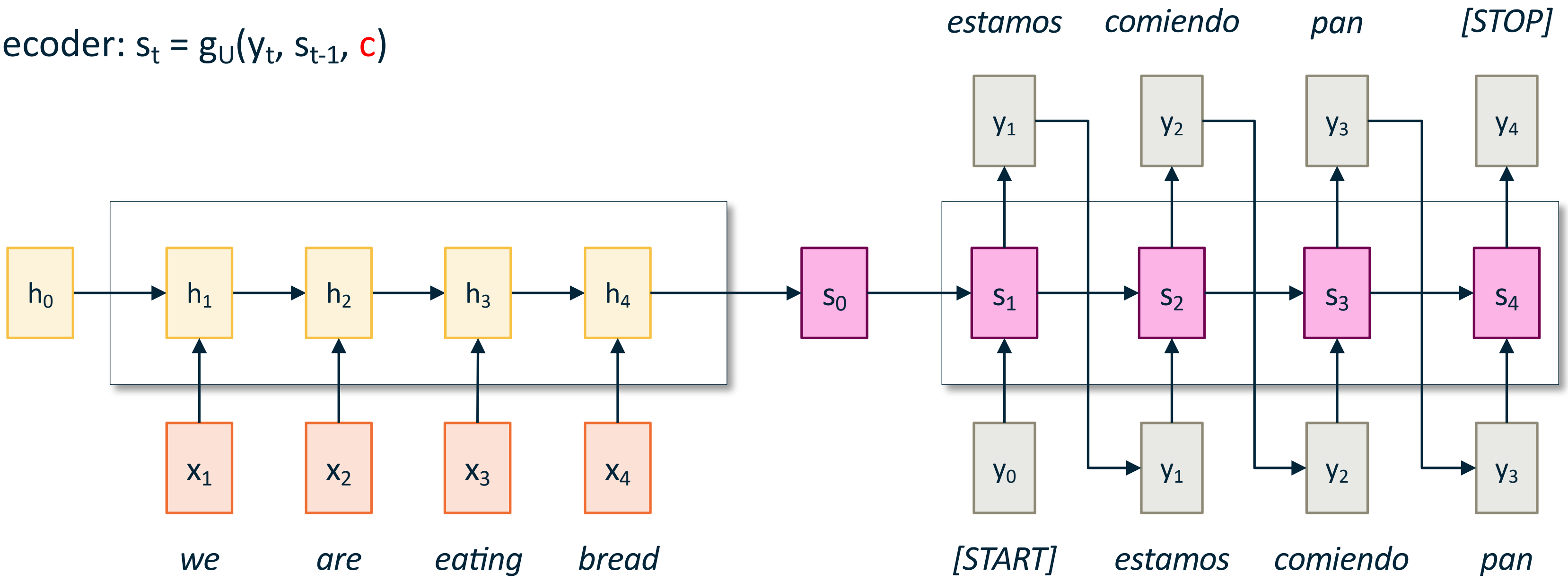


Slide credit: Justin Johnson

Machine Translation with RNNs

Encoder: $h_t = f_W(x_t, h_{t-1})$

Decoder: $s_t = g_U(y_t, s_{t-1}, c)$



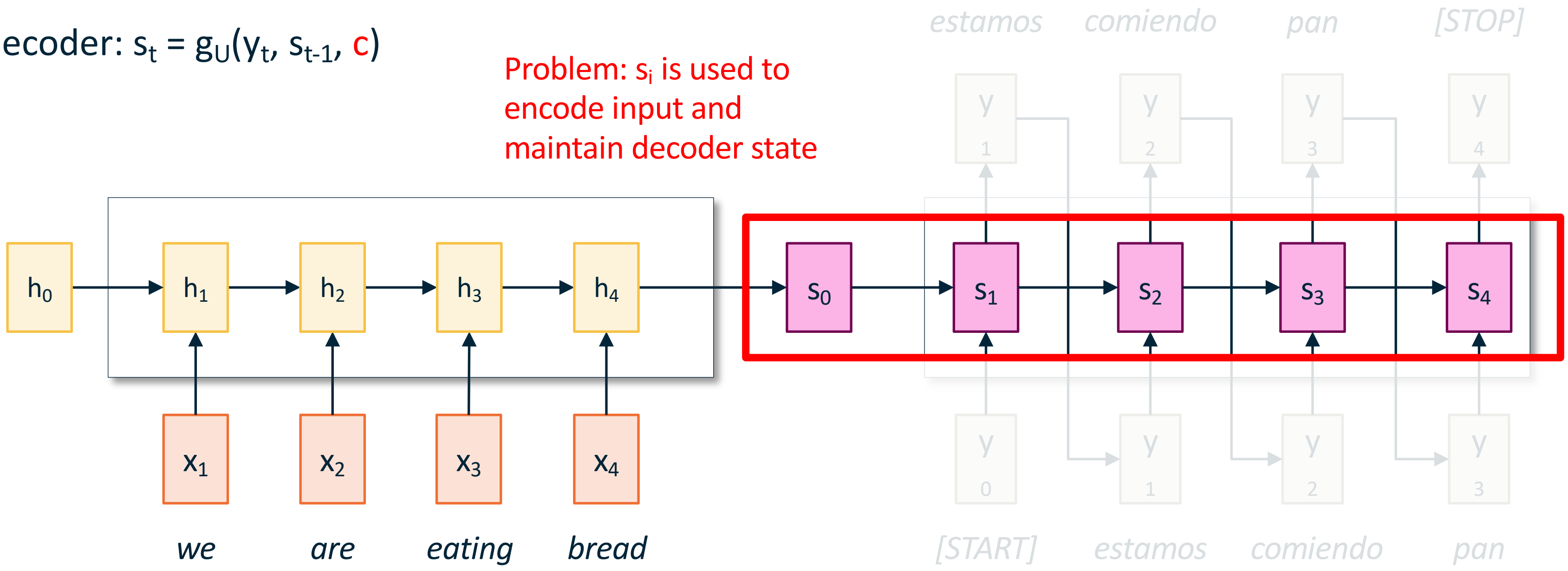
Slide credit: Justin Johnson

Machine Translation with RNNs

Encoder: $h_t = f_W(x_t, h_{t-1})$

Decoder: $s_t = g_U(y_t, s_{t-1}, c)$

Problem: s_i is used to encode input and maintain decoder state



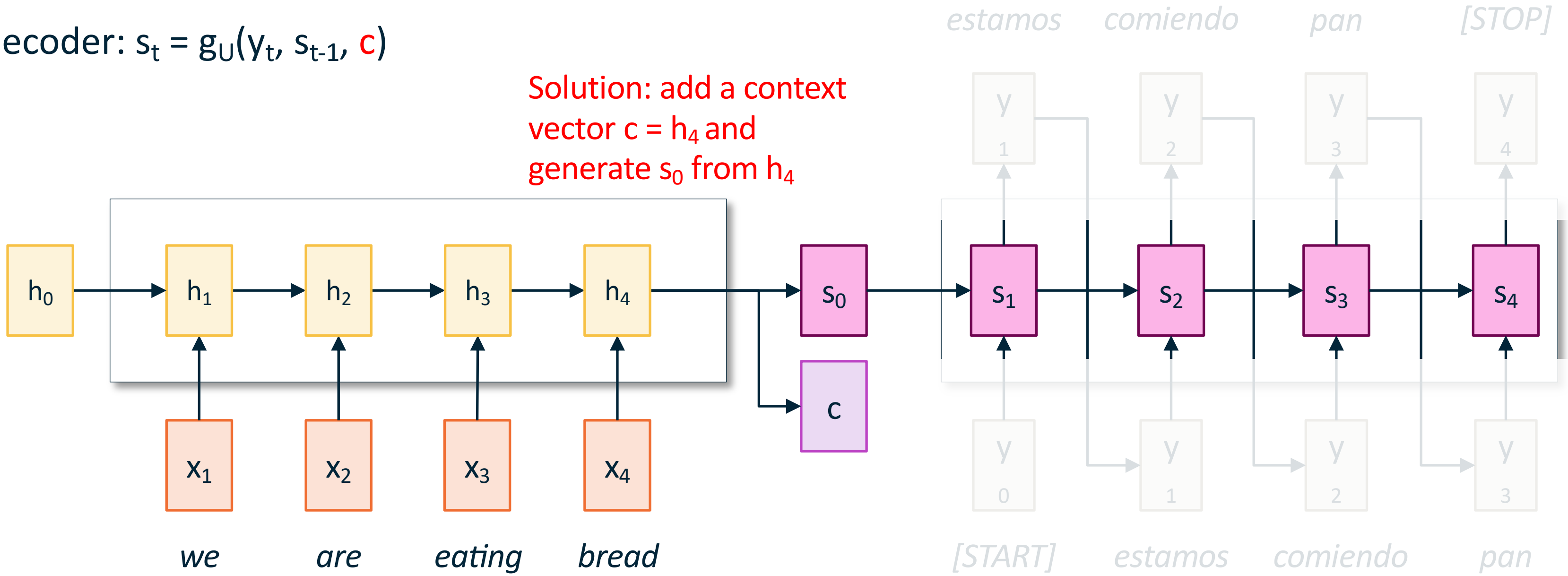
Slide credit: Justin Johnson

Machine Translation with RNNs

Encoder: $h_t = f_W(x_t, h_{t-1})$

Decoder: $s_t = g_U(y_t, s_{t-1}, c)$

Solution: add a context vector $c = h_4$ and generate s_0 from h_4

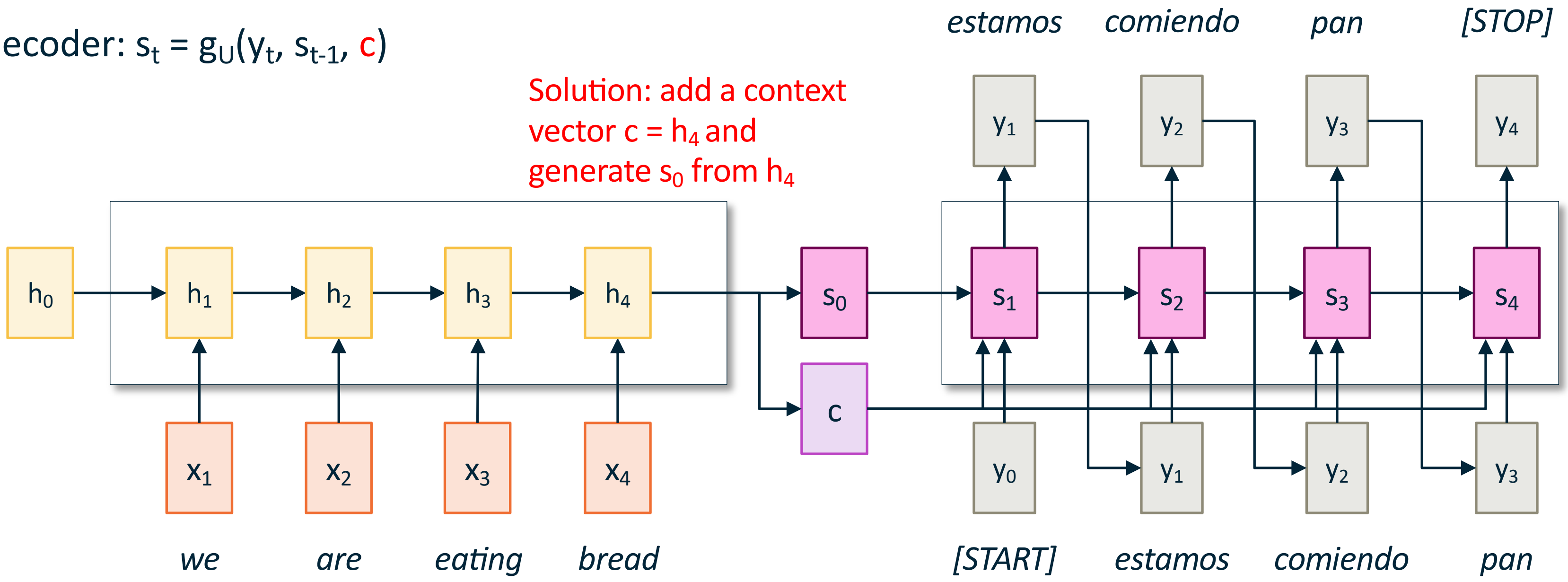


Slide credit: Justin Johnson

Machine Translation with RNNs

Encoder: $h_t = f_W(x_t, h_{t-1})$

Decoder: $s_t = g_U(y_t, s_{t-1}, c)$



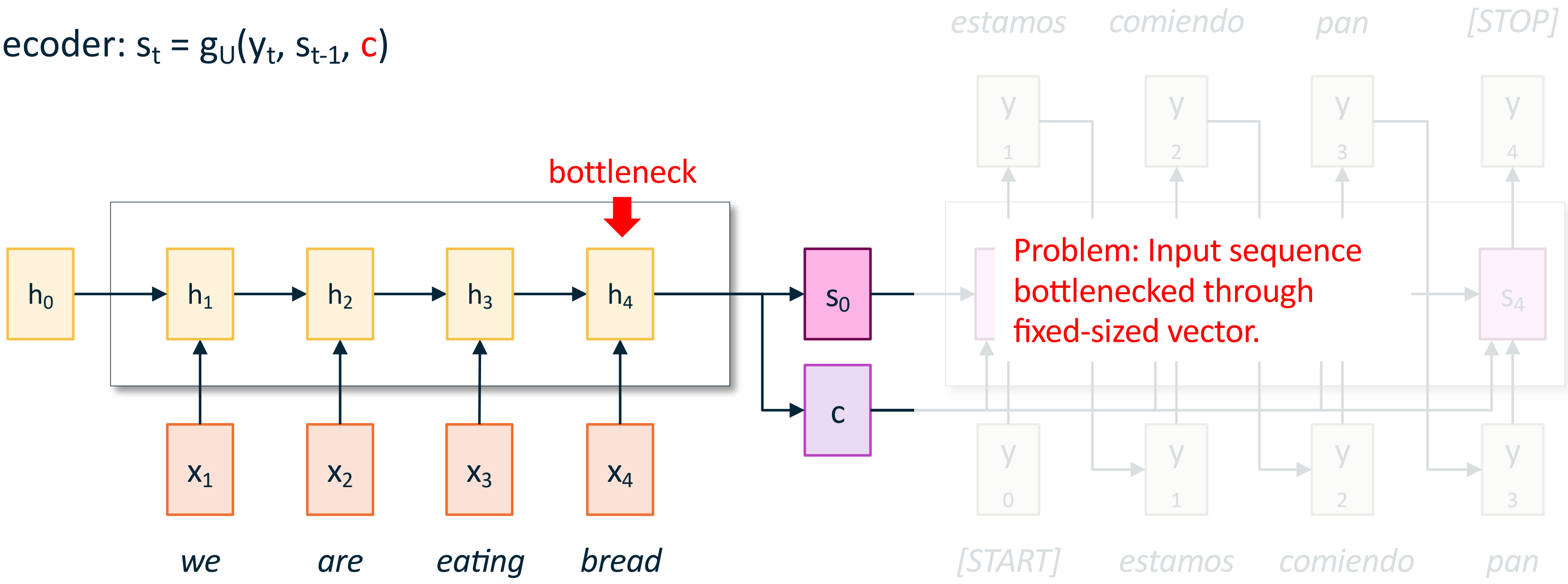
Solution: add a context vector $c = h_4$ and generate s_0 from h_4

Slide credit: Justin Johnson

Machine Translation with RNNs

Encoder: $h_t = f_W(x_t, h_{t-1})$

Decoder: $s_t = g_U(y_t, s_{t-1}, c)$



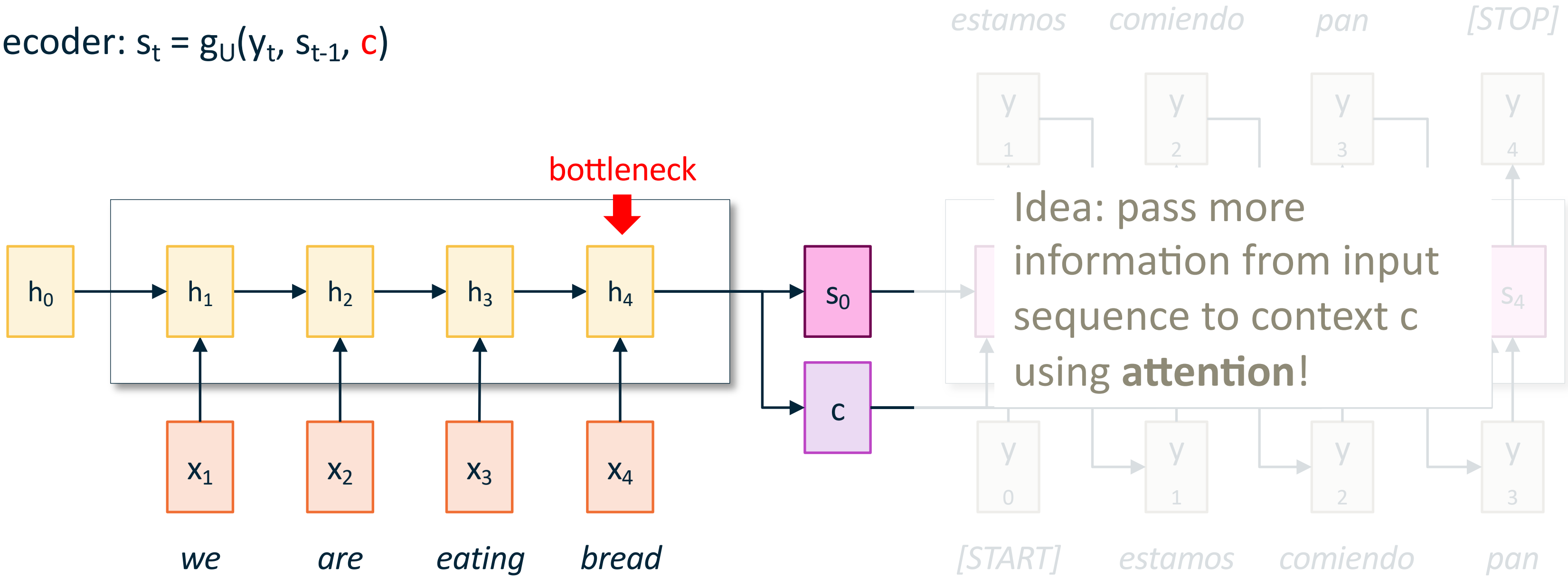
Problem: Input sequence bottlenecked through fixed-sized vector.

Slide credit: Justin Johnson

Machine Translation with RNNs

Encoder: $h_t = f_W(x_t, h_{t-1})$

Decoder: $s_t = g_U(y_t, s_{t-1}, c)$

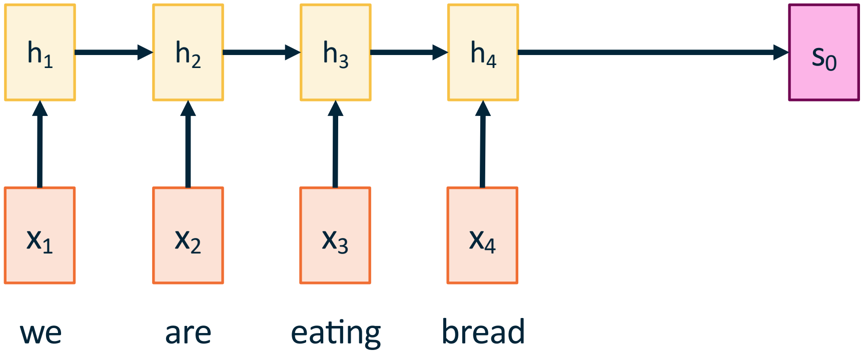


Idea: pass more information from input sequence to context c using **attention**!

Slide credit: Justin Johnson

Machine Translation with RNNs **and Attention**

From final hidden state:
Initial decoder state s_0



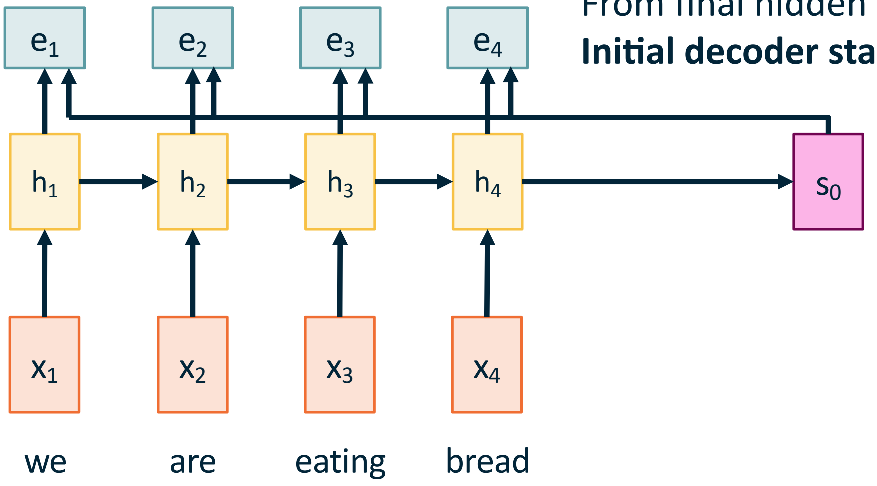
Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Slide credit: Justin Johnson

Machine Translation with RNNs **and Attention**

Compute **affinity scores**

$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

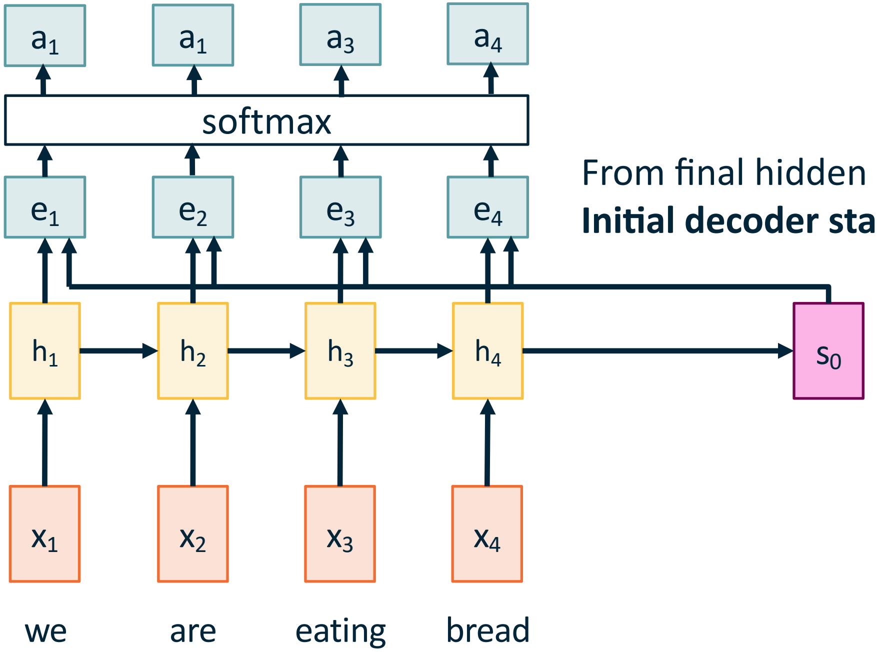


From final hidden state:
Initial decoder state s_0

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Slide credit: Justin Johnson

Machine Translation with RNNs **and Attention**



From final hidden state:
Initial decoder state s_0

Compute **affinity scores**

$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

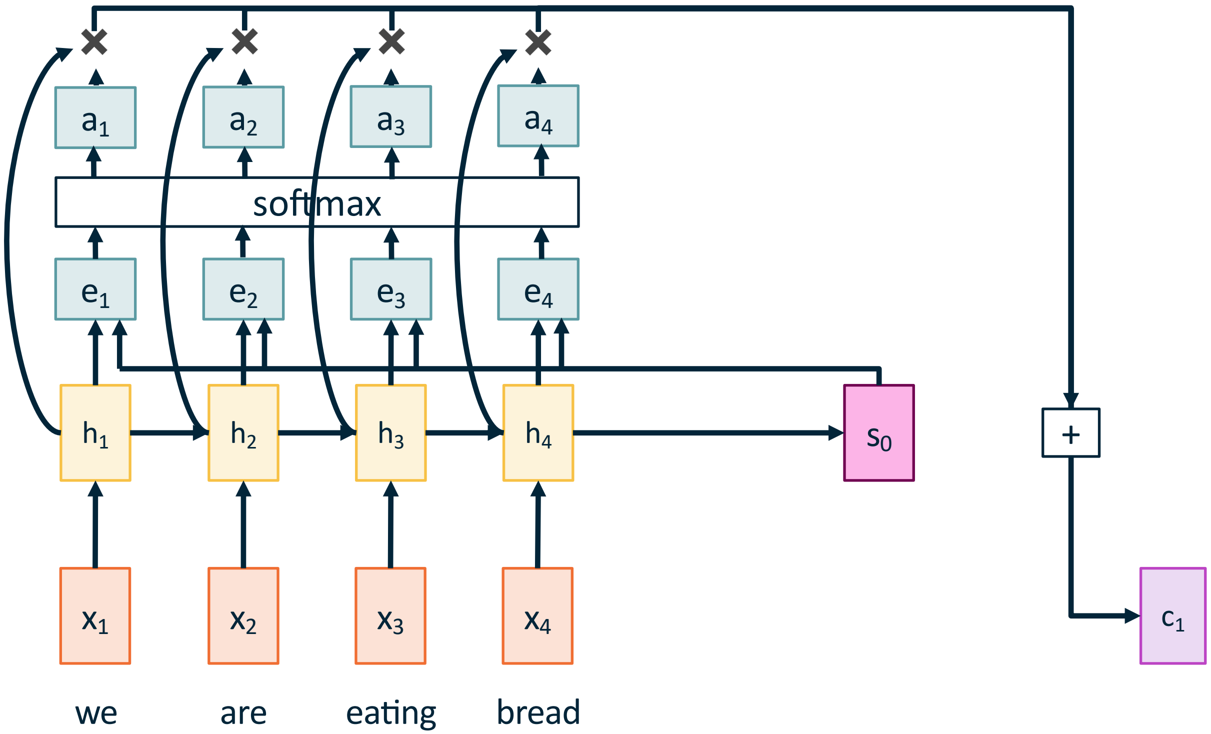
Normalize to get
attention weights

$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Slide credit: Justin Johnson

Machine Translation with RNNs **and Attention**



Compute **affinity scores**

$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize to get **attention weights**

$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

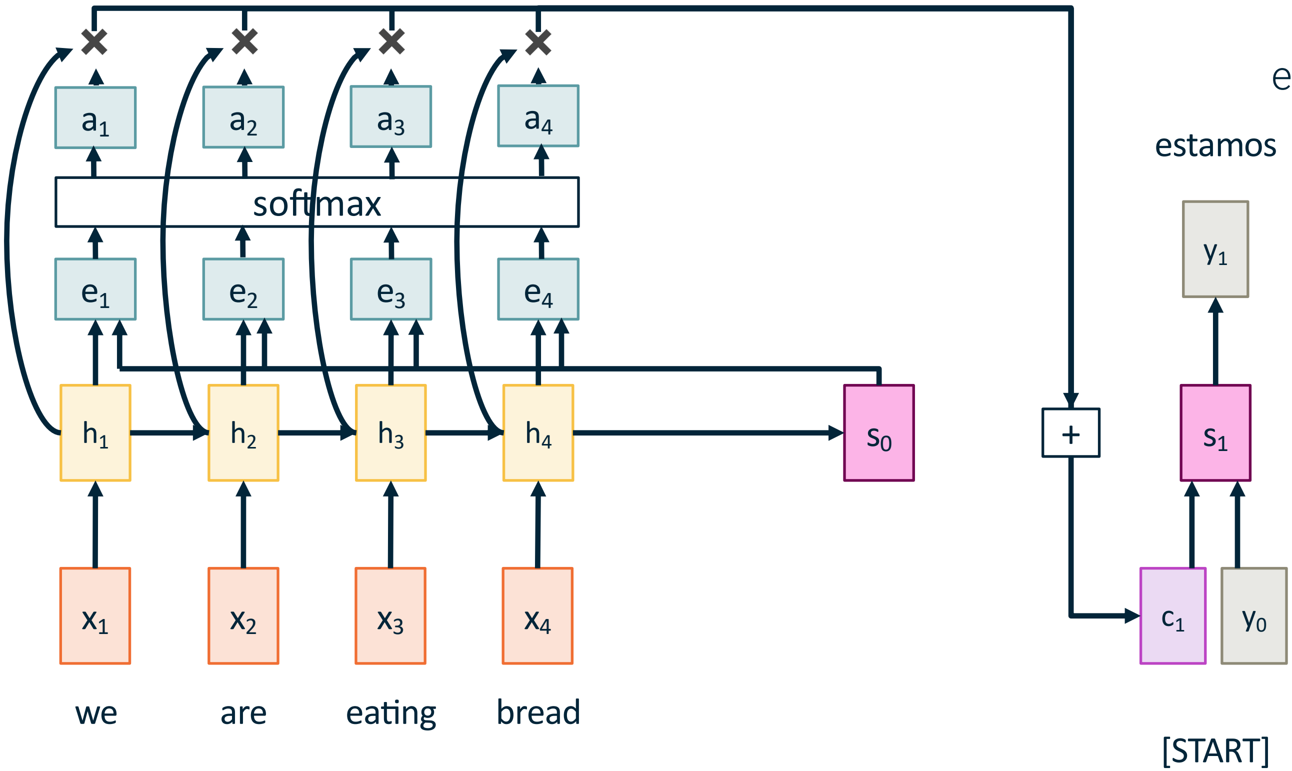
Set context vector **c** to a linear combination of hidden states

$c_t = \sum_i a_{t,i} h_i$

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Slide credit: Justin Johnson

Machine Translation with RNNs **and Attention**



Compute **affinity scores**

$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize to get **attention weights**

$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

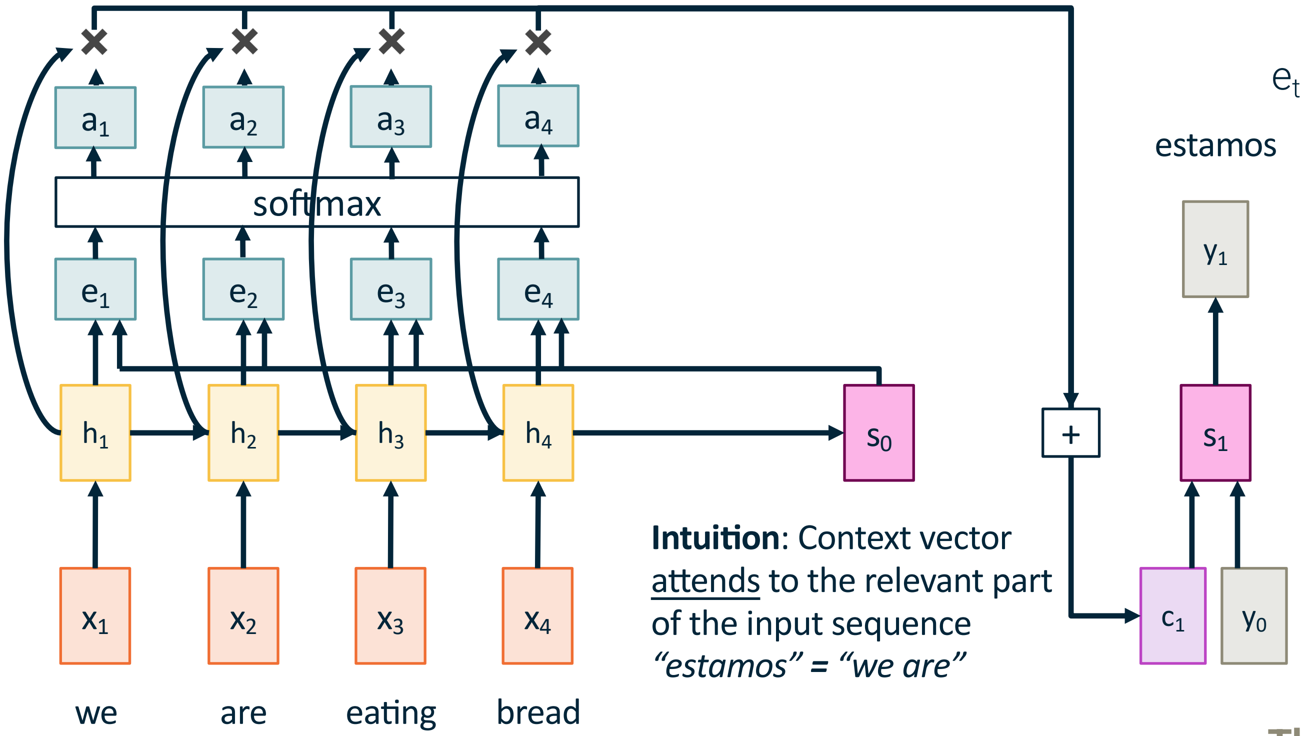
Set context vector **c** to a linear combination of hidden states

$c_t = \sum_i a_{t,i} h_i$

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Slide credit: Justin Johnson

Machine Translation with RNNs **and Attention**



Compute **affinity scores**

$e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize to get **attention weights**

$0 < a_{t,i} < 1$ $\sum_i a_{t,i} = 1$

Set context vector **c** to a linear combination of hidden states

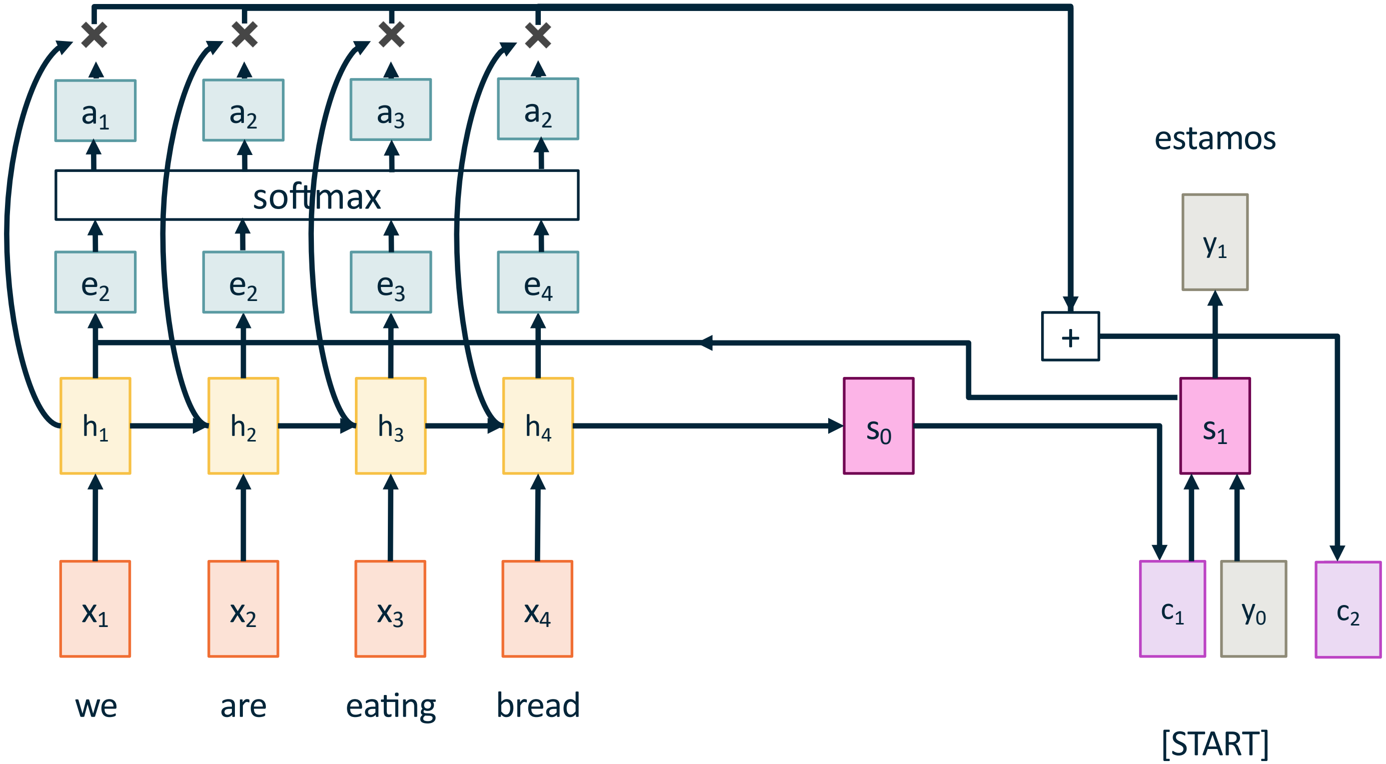
$c_t = \sum_i a_{t,i} h_i$

Intuition: Context vector attends to the relevant part of the input sequence *“estamos” = “we are”*

This is all differentiable! Do not supervise attention weights – backprop through everything

Slide credit: Justin Johnson

Machine Translation with RNNs **and Attention**

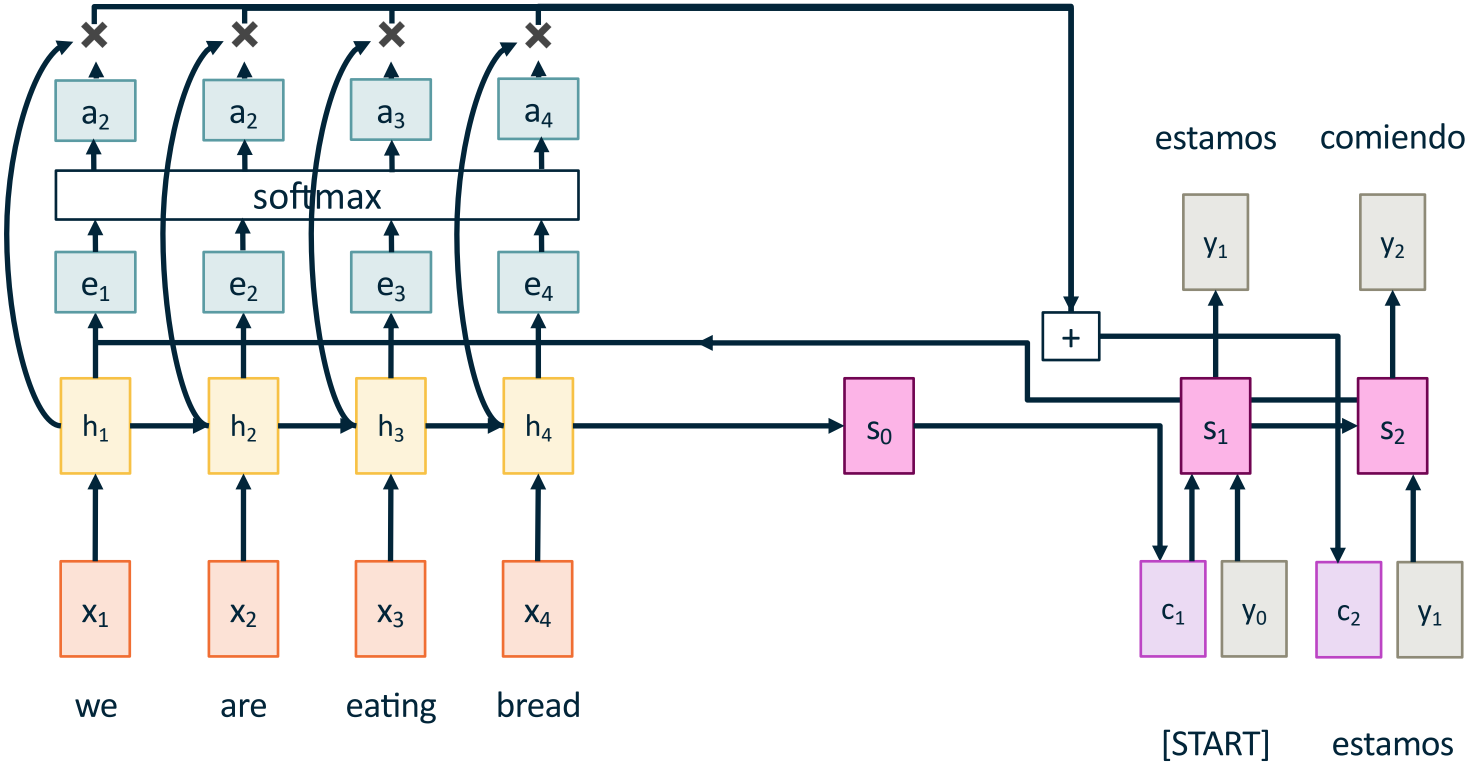


Repeat: Use s_1 to compute new context vector c_2

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Slide credit: Justin Johnson

Machine Translation with RNNs **and Attention**



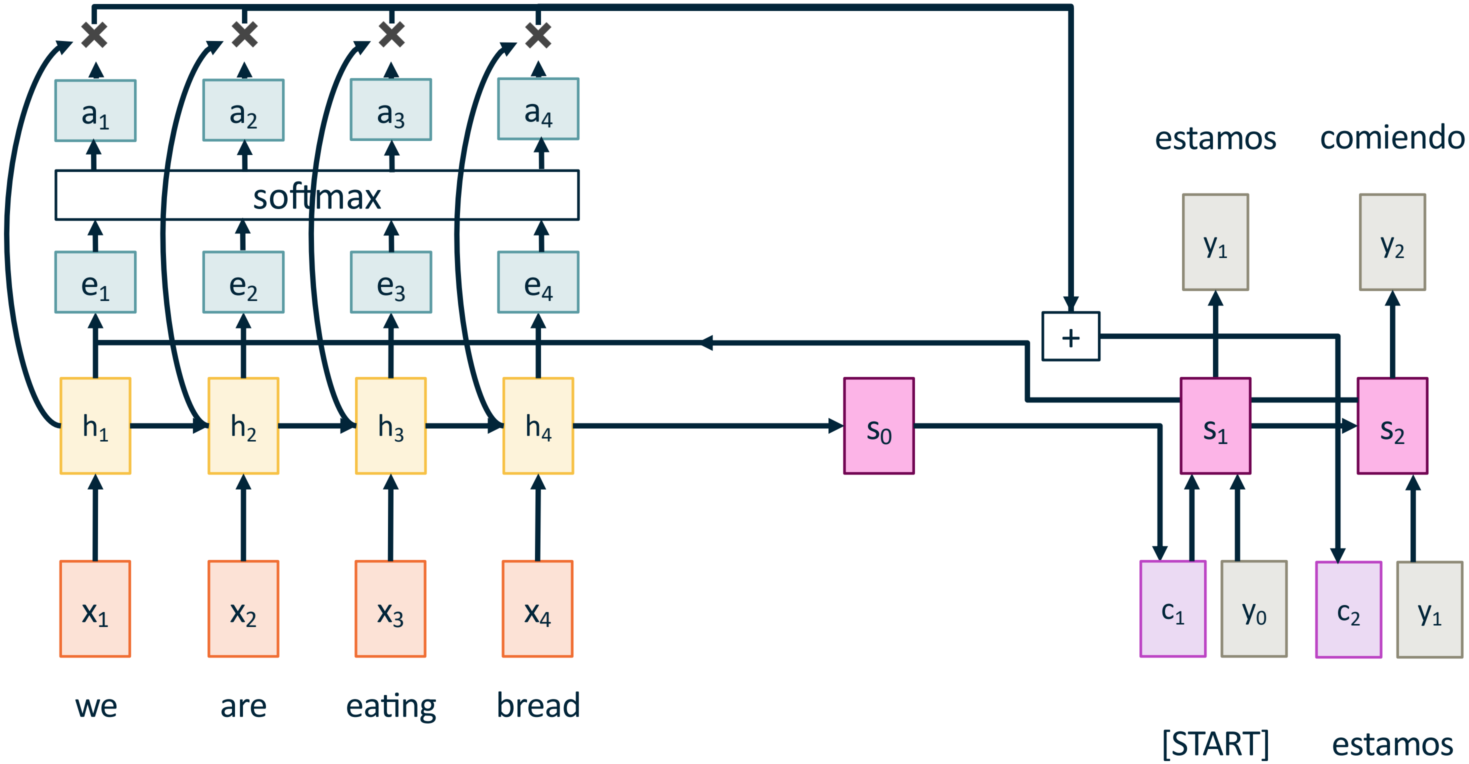
Repeat: Use s_1 to compute new context vector c_2

Use c_2 to compute s_2, y_2

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Slide credit: Justin Johnson

Machine Translation with RNNs **and Attention**



Repeat: Use s_1 to compute new context vector c_2

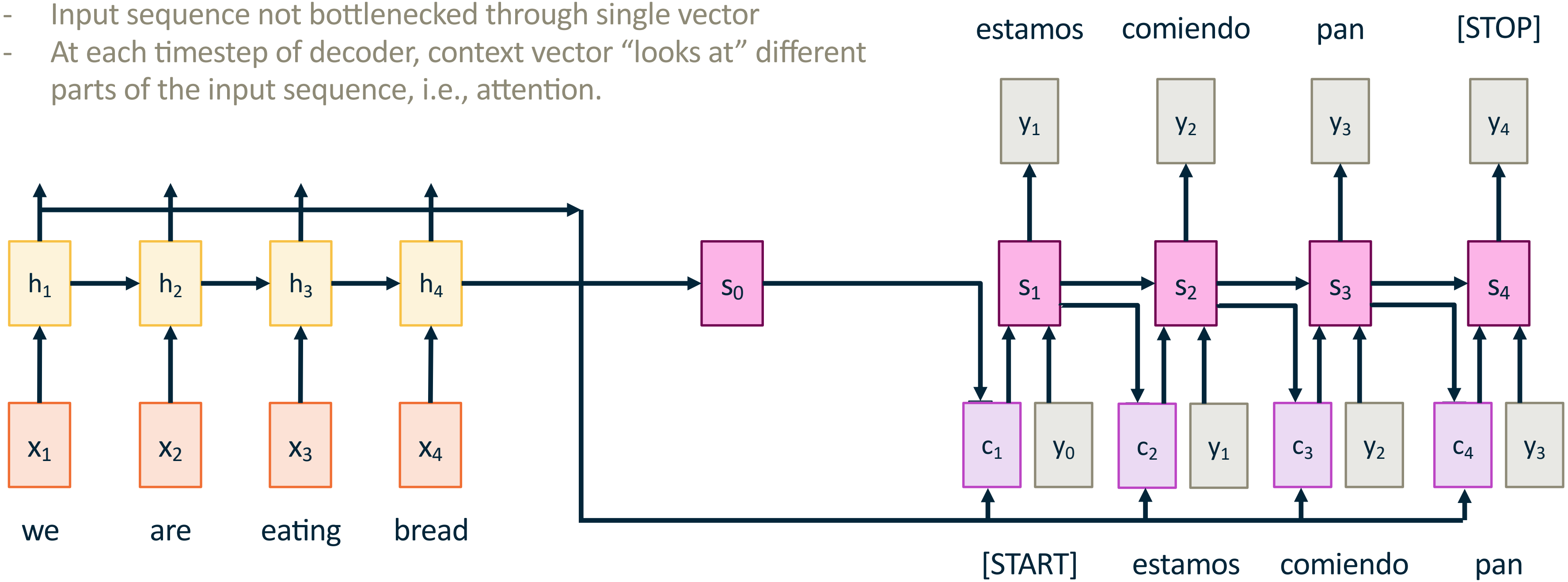
Use c_2 to compute s_2, y_2

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Slide credit: Justin Johnson

Machine Translation with RNNs **and Attention**

Use a different context vector in each timestep of decoder
- Input sequence not bottlenecked through single vector
- At each timestep of decoder, context vector "looks at" different parts of the input sequence, i.e., attention.



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Slide credit: Justin Johnson

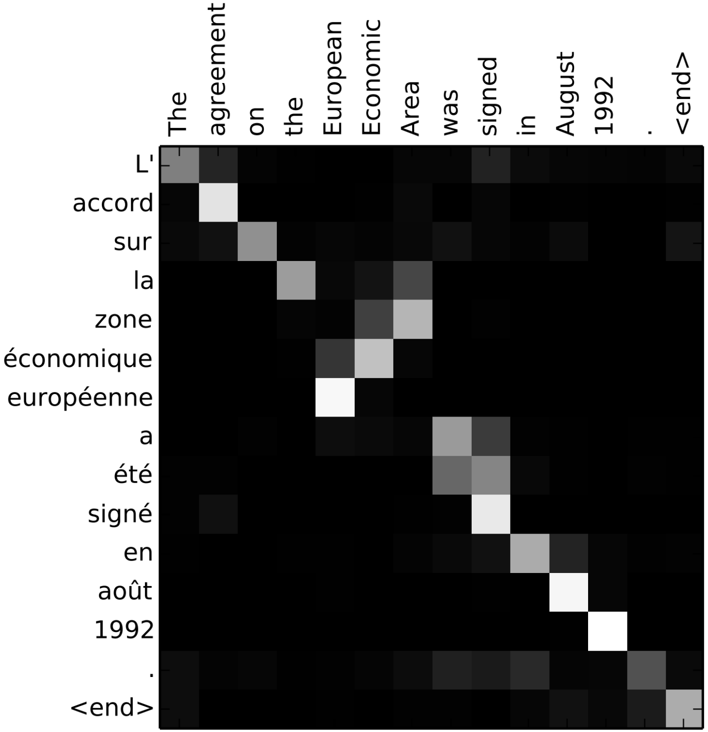
Machine Translation with RNNs **and Attention**

Example: English to French translation

Input: “The agreement on the European Economic Area was signed in August 1992.”

Output: “L’accord sur la zone économique européenne a été signé en août 1992.”

Visualize attention weights $a_{t,i}$



Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Slide credit: Justin Johnson

Machine Translation with RNNs **and Attention**

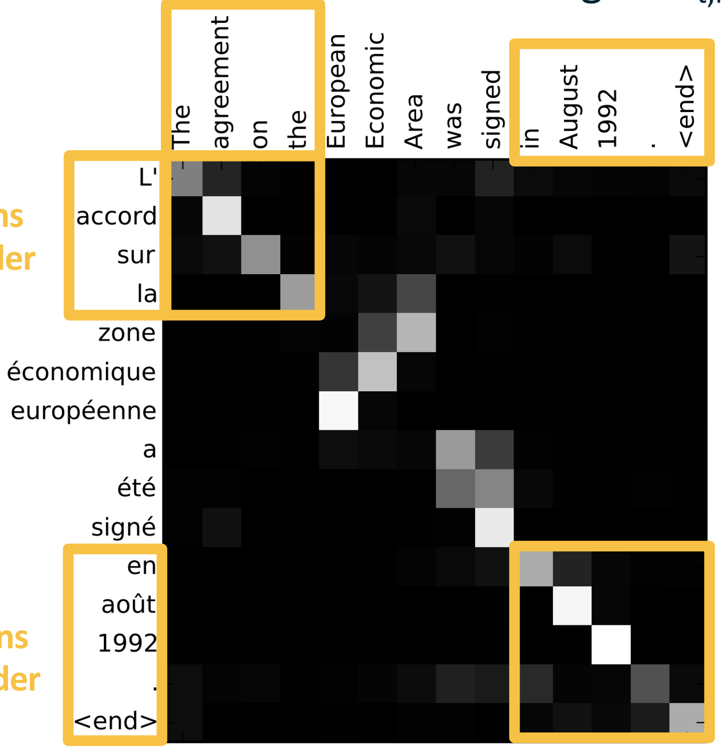
Example: English to French translation

Input: "**The agreement on the** European Economic Area was signed **in August 1992**."

Output: "**L'accord sur la** zone économique européenne a été signé **en août 1992**."

Visualize attention weights $a_{t,i}$

Diagonal attention means words correspond in order



Diagonal attention means words correspond in order

Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Slide credit: Justin Johnson

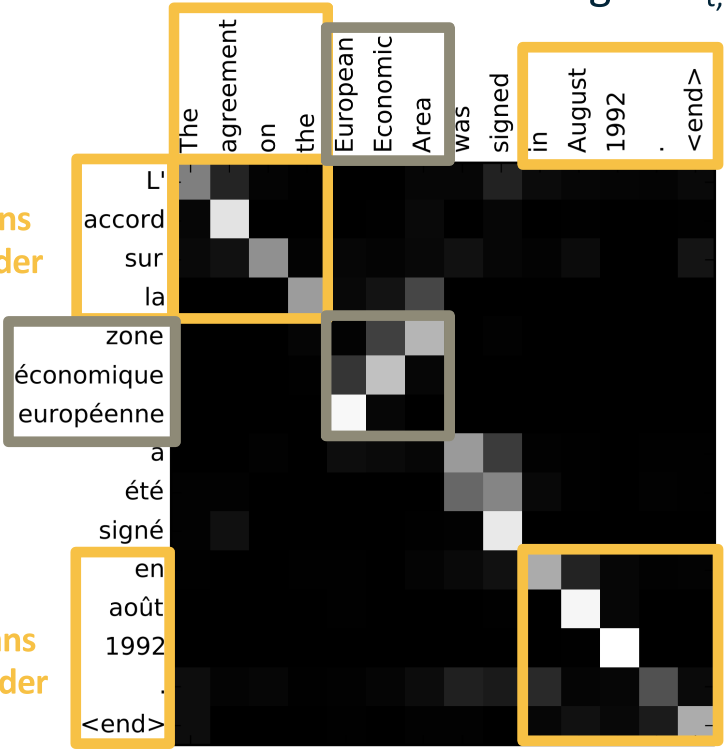
Machine Translation with RNNs **and Attention**

Example: English to French translation

Input: “The agreement on the European Economic Area was signed in August 1992.”

Output: “L’accord sur la zone économique européenne a été signé en août 1992.”

Visualize attention weights $a_{t,i}$



Diagonal attention means words correspond in order

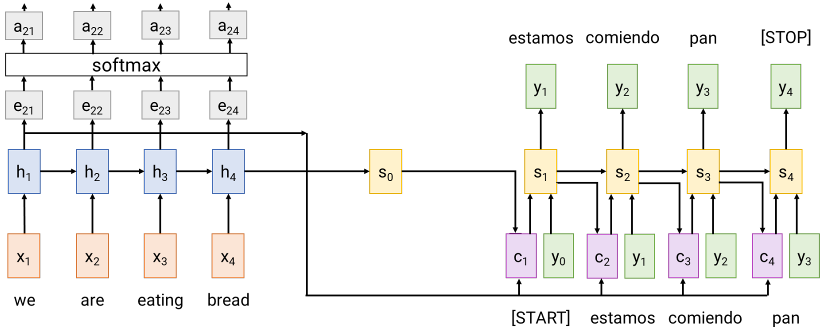
Attention figures out different word orders

Diagonal attention means words correspond in order

Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Slide credit: Justin Johnson

Attention Layer



Inputs:

State vector: s_i (Shape: D_Q)

Hidden vectors: h_i (Shape: N_X x D_H)

Similarity function: f_{att}

Computation:

Similarities: e (Shape: N_X) $e_i = f_{att}(s_{t-1}, h_i)$

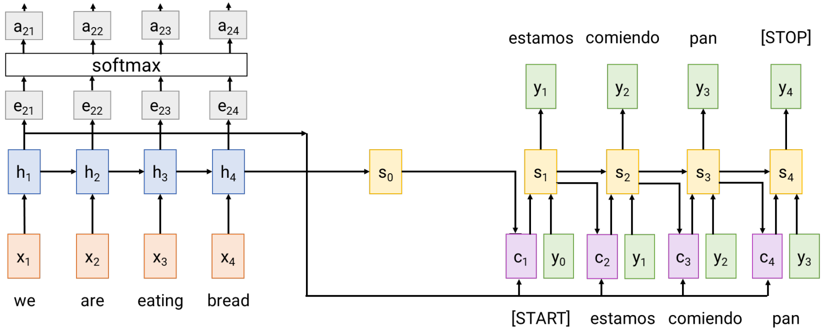
Attention weights: a = softmax(e) (Shape: N_X)

Output vector: $y = \sum_i a_i h_i$ (Shape: D_X)

Slide credit: Justin Johnson

Attention Layer

Inputs:
Query vector: **q** (Shape: D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Similarity function: f_{att}

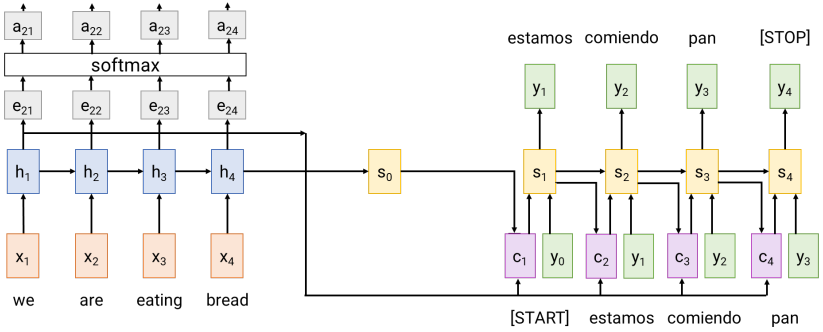


Computation:
Similarities: e (Shape: N_X) $e_i = f_{att}(\mathbf{q}, \mathbf{X}_i)$
Attention weights: a = softmax(e) (Shape: N_X)
Output vector: $y = \sum_i a_i \mathbf{X}_i$ (Shape: D_X)

Slide credit: Justin Johnson

Attention Layer

Inputs:
Query vector: q (Shape: D_Q)
Input vectors: X (Shape: N_X x D_Q)
Similarity function: dot product



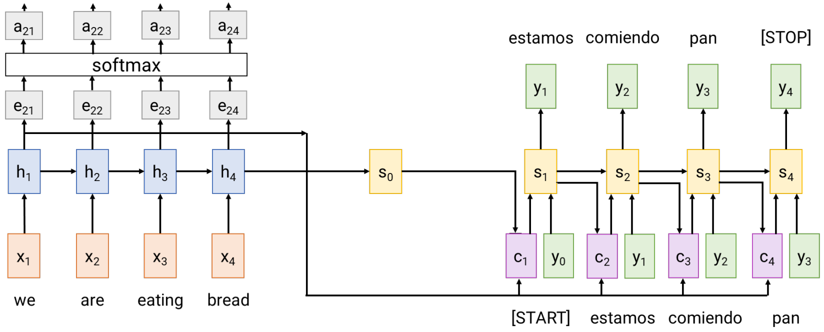
Computation:
Similarities: e (Shape: N_X) $e_i = \mathbf{q} \cdot \mathbf{X}_i$
Attention weights: a = softmax(e) (Shape: N_X)
Output vector: $y = \sum_i a_i \mathbf{X}_i$ (Shape: D_X)

Changes:
- Use dot product for similarity

Slide credit: Justin Johnson

Attention Layer

Inputs:
Query vector: $\mathbf{q}$ (Shape: D_Q)
Input vectors: $\mathbf{X}$ (Shape: N_X x D_Q)
Similarity function: scaled dot product



Computation:
Similarities: e (Shape: N_X) $e_i = \mathbf{q} \cdot \mathbf{X}_i / \mathrm{sqrt}(D_Q)$
Attention weights: a = softmax(e) (Shape: N_X)
Output vector: $y = \sum_i a_i \mathbf{X}_i$ (Shape: D_X)

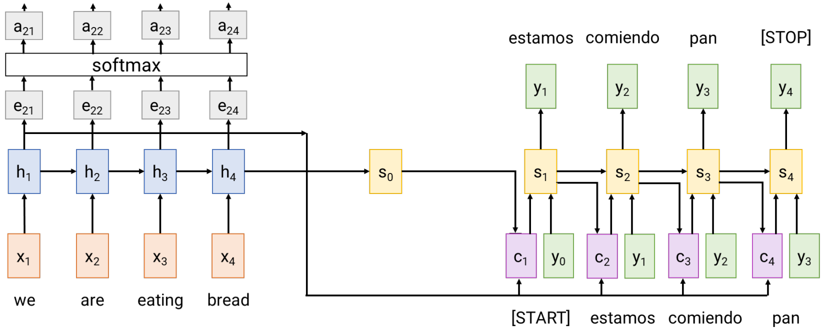
Changes:

- Use **scaled** dot product for similarity

Slide credit: Justin Johnson

Attention Layer

Inputs:
Query vectors: Q (Shape: $N_Q \times D_Q$)
Input vectors: X (Shape: $N_X \times D_Q$)



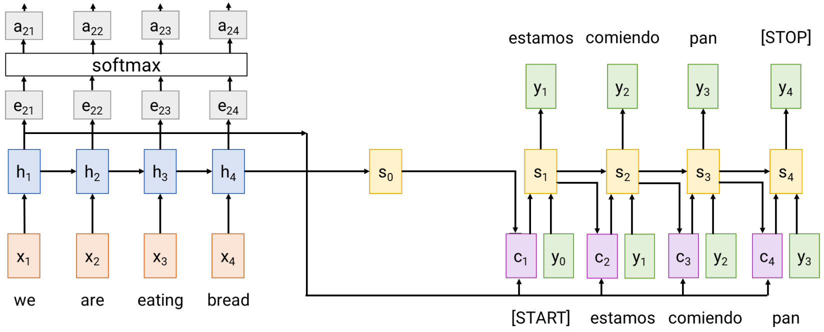
Computation:
Similarities: $E = QX^T$ (Shape: $N_Q \times N_X$) $E_{i,j} = Q_i \cdot X_j / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: $N_Q \times N_X$)
Output vectors: $Y = AX$ (Shape: $N_Q \times D_X$) $Y_i = \sum_j A_{i,j} X_j$

Changes:
- Use dot product for similarity
- Multiple **query** vectors

Slide credit: Justin Johnson

Attention Layer

Inputs:
Query vectors: Q (Shape: N_Q x D_Q)
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)



Computation:
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_Q x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_Q x N_X)
Output vectors: $Y = AV$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} V_j$

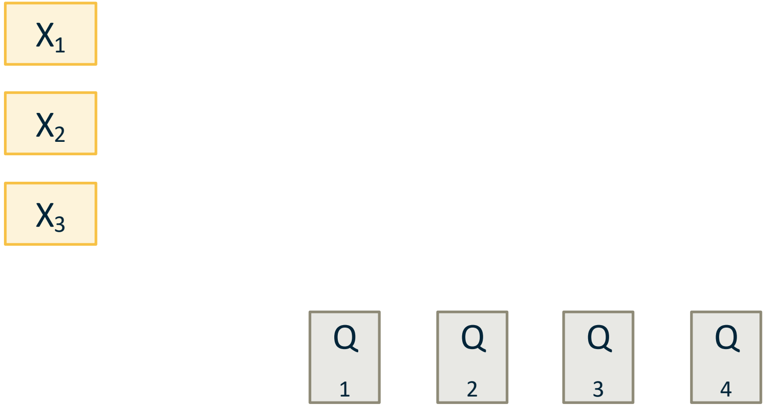
Changes:
- Use dot product for similarity
- Multiple **query** vectors
- Separate **key** and **value**

Slide credit: Justin Johnson

Attention Layer

Inputs:
Query vectors: Q (Shape: $N_Q \times D_Q$)
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)

Computation:
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T$ (Shape: $N_Q \times N_X$) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: $N_Q \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

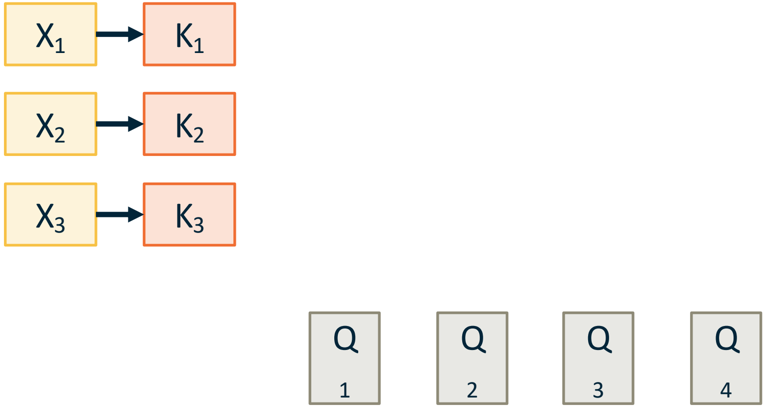


Slide credit: Justin Johnson

Attention Layer

Inputs:
Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **W_K** (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)

Computation:
Key vectors: **K** = **XW_K** (Shape: N_X x D_Q)
Value vectors: **V** = **XW_V** (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_Q x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = AV (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} V_j$

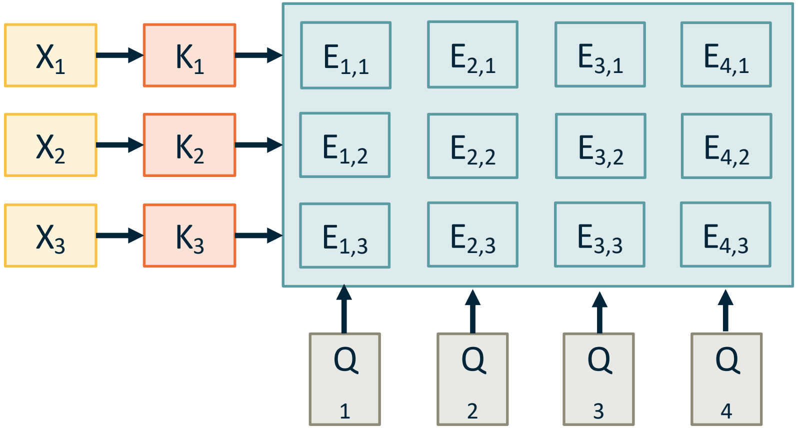


Slide credit: Justin Johnson

Attention Layer

Inputs:
Query vectors: Q (Shape: N_Q x D_Q)
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)

Computation:
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_Q x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: $Y = AV$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} V_j$

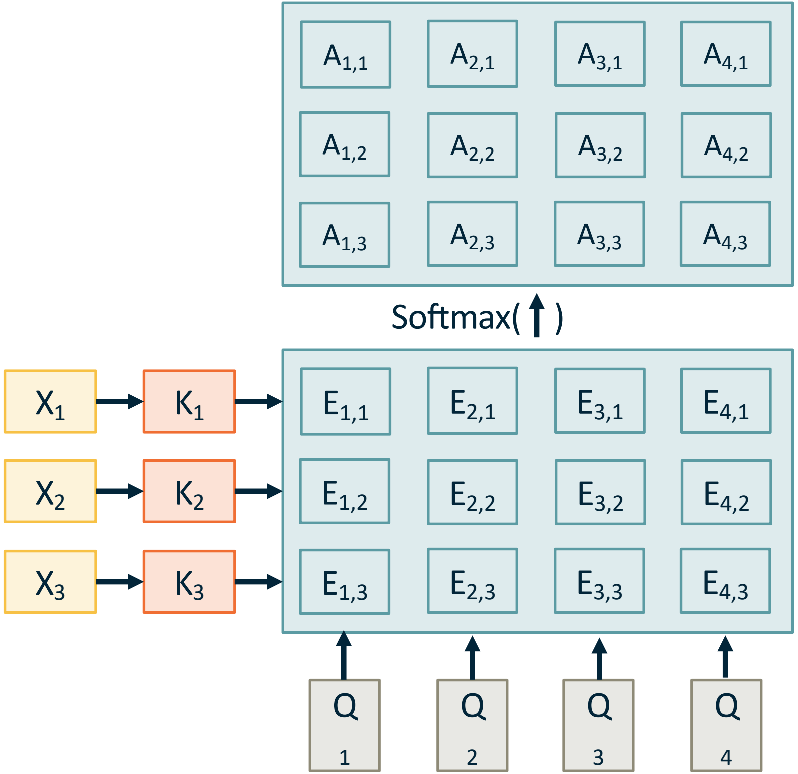


Slide credit: Justin Johnson

Attention Layer

Inputs:
Query vectors: Q (Shape: N_Q x D_Q)
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)

Computation:
Key vectors: K = XW_K (Shape: N_X x D_Q)
Value vectors: V = XW_V (Shape: N_X x D_V)
Similarities: E = QK^T (Shape: N_Q x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = AV (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

Attention Layer

Inputs:

Query vectors: Q (Shape: $N_Q \times D_Q$)

Input vectors: X (Shape: $N_X \times D_X$)

Key matrix: W_K (Shape: $D_X \times D_Q$)

Value matrix: W_V (Shape: $D_X \times D_V$)

Computation:

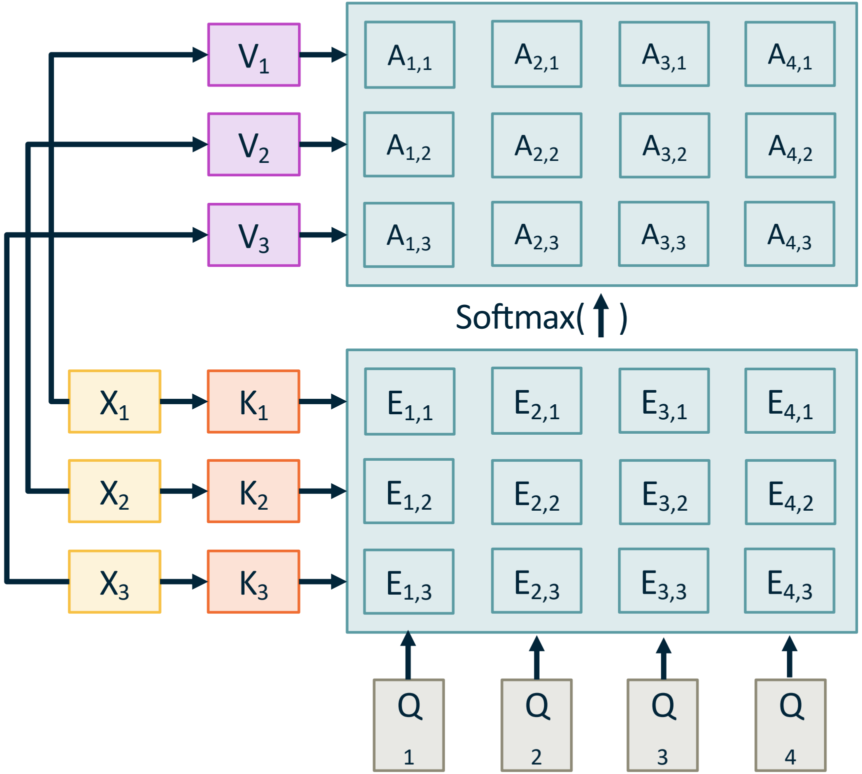
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)

Value vectors: $V = XW_V$ (Shape: $N_X \times D_V$)

Similarities: $E = QK^T$ (Shape: $N_Q \times N_X$) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$

Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: $N_Q \times N_X$)

Output vectors: $Y = AV$ (Shape: $N_Q \times D_V$) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

Attention Layer

Inputs:

Query vectors: Q (Shape: N_Q x D_Q)

Input vectors: X (Shape: N_X x D_X)

Key matrix: W_K (Shape: D_X x D_Q)

Value matrix: W_V (Shape: D_X x D_V)

Computation:

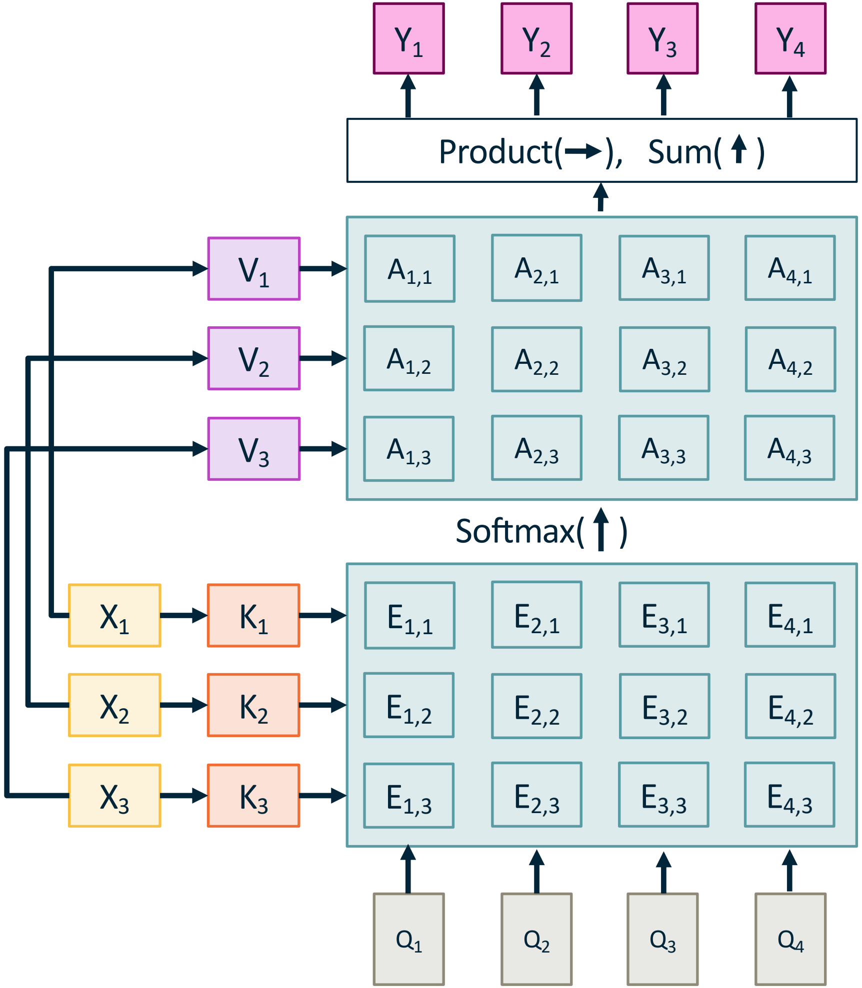
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)

Value vectors: $V = XW_V$ (Shape: N_X x D_V)

Similarities: $E = QK^T$ (Shape: N_Q x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$

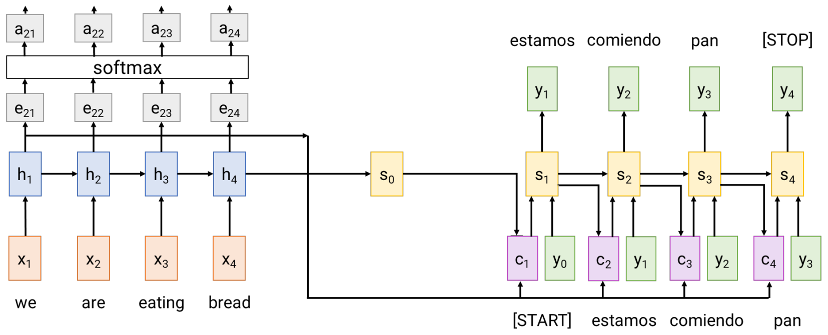
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)

Output vectors: $Y = AV$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

Attention Layer



Inputs:
Query vectors: Q (Shape: N_Q x D_Q)
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)

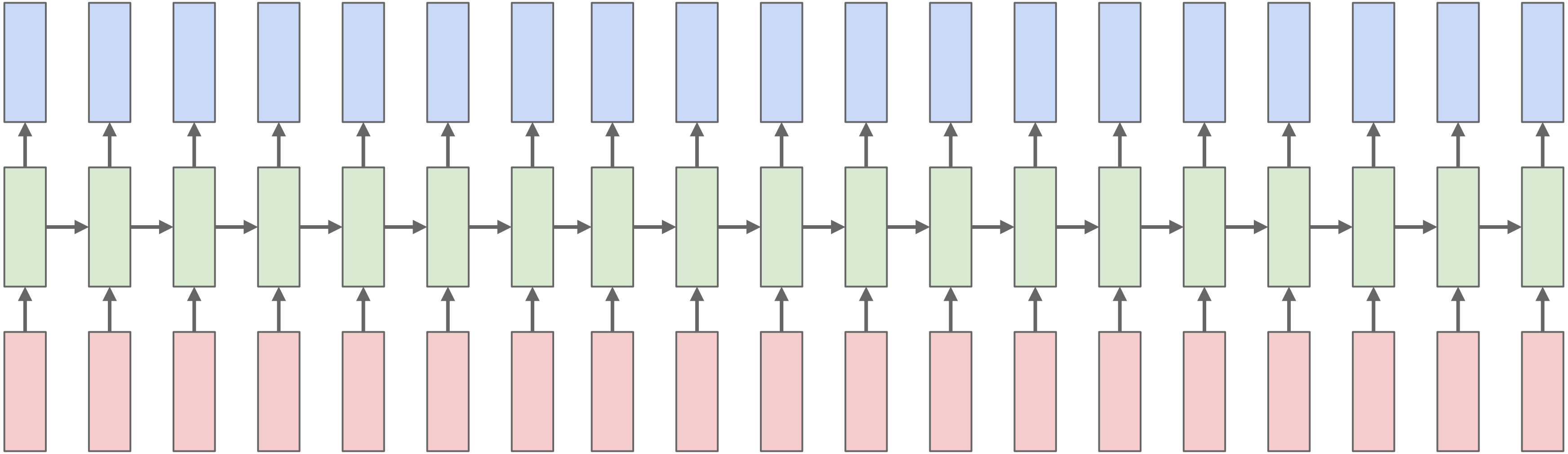
Computation:
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_Q x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: $Y = AV$ (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} V_j$

Attention seems to be really powerful …

Do we still need RNN?

Slide credit: Justin Johnson

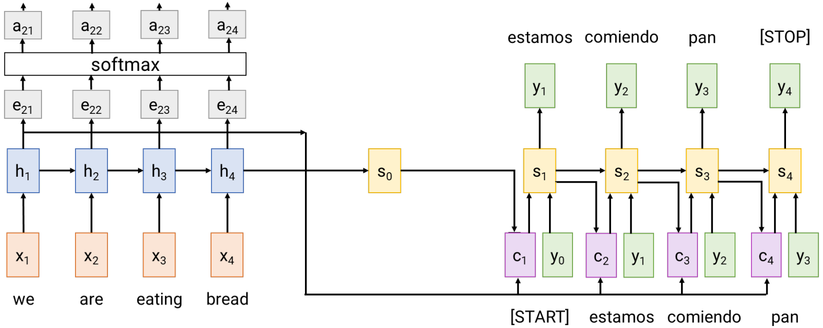
RNN is bad at encoding long-range relationships!



Recurrent update can easily “forget” information

Attention Layer

Inputs:
Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: **W_K** (Shape: D_X x D_Q)
Value matrix: **W_V** (Shape: D_X x D_V)



Attention seems to be really powerful …
Do we still need RNN?

Can we use attention for sequence encoding?

Computation:
Key vectors: **K** = **XW_K** (Shape: N_X x D_Q)
Value vectors: **V** = **XW_V** (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_Q x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_Q x N_X)
Output vectors: Y = AV (Shape: N_Q x D_V) $Y_i = \sum_j A_{i,j} V_j$

Slide credit: Justin Johnson

Self-Attention Layer

Sequence encode -> use each input element as query!

Inputs:

Input vectors: X (Shape: N_X x D_X)

Key matrix: W_K (Shape: D_X x D_Q)

Value matrix: W_V (Shape: D_X x D_V)

Query matrix: W_Q (Shape: D_X x D_Q)

Computation:

Query vectors: $Q = XW_Q$

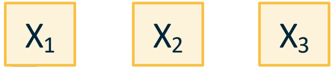
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)

Value vectors: $V = XW_V$ (Shape: N_X x D_V)

Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$

Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_X x N_X)

Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

Self-Attention Layer

Sequence encode -> use each input element as query!

Inputs:

Input vectors: X (Shape: N_X x D_X)

Key matrix: W_K (Shape: D_X x D_Q)

Value matrix: W_V (Shape: D_X x D_V)

Query matrix: W_Q (Shape: D_X x D_Q)

Computation:

Query vectors: $Q = XW_Q$

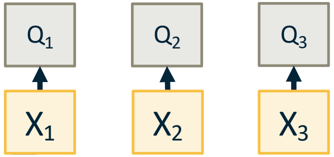
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)

Value vectors: $V = XW_V$ (Shape: N_X x D_V)

Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$

Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_X x N_X)

Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

Self-Attention Layer

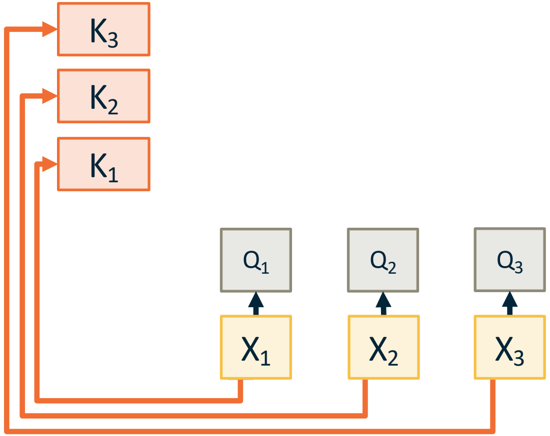
Sequence encode -> use each input element as query!

Inputs:

Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:

Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



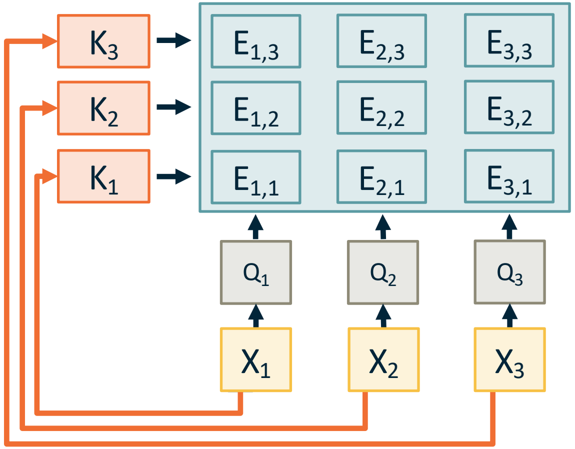
Slide credit: Justin Johnson

Self-Attention Layer

Sequence encode -> use each input element as query!

Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

Self-Attention Layer

Sequence encode -> use each input element as query!

Inputs:

Input vectors: X (Shape: N_X x D_X)

Key matrix: W_K (Shape: D_X x D_Q)

Value matrix: W_V (Shape: D_X x D_V)

Query matrix: W_Q (Shape: D_X x D_Q)

Computation:

Query vectors: $Q = XW_Q$

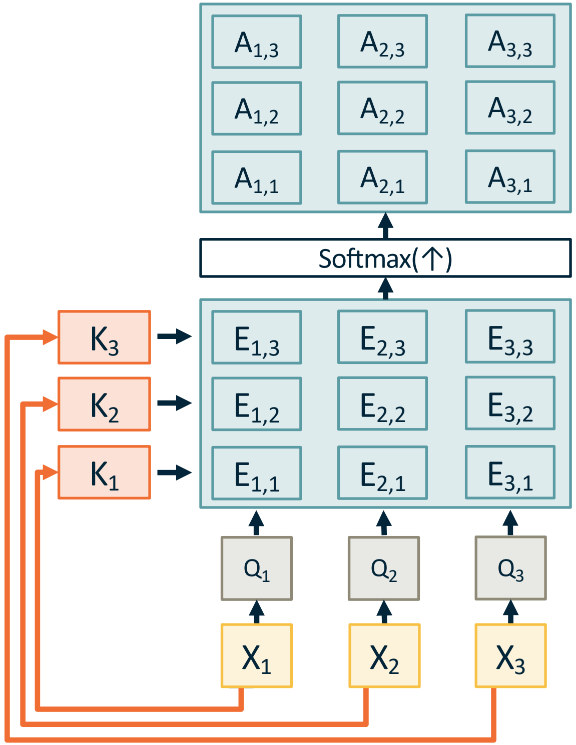
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)

Value vectors: $V = XW_V$ (Shape: N_X x D_V)

Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$

Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_X x N_X)

Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

Self-Attention Layer

Sequence encode -> use each input element as query!

Inputs:

Input vectors: X (Shape: N_X x D_X)

Key matrix: W_K (Shape: D_X x D_Q)

Value matrix: W_V (Shape: D_X x D_V)

Query matrix: W_Q (Shape: D_X x D_Q)

Computation:

Query vectors: $Q = XW_Q$

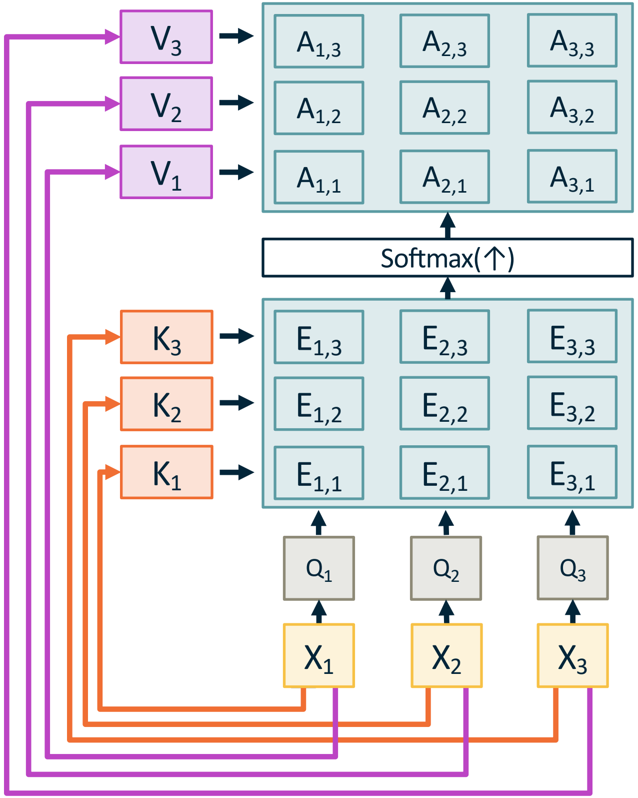
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)

Value vectors: $V = XW_V$ (Shape: N_X x D_V)

Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$

Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_X x N_X)

Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



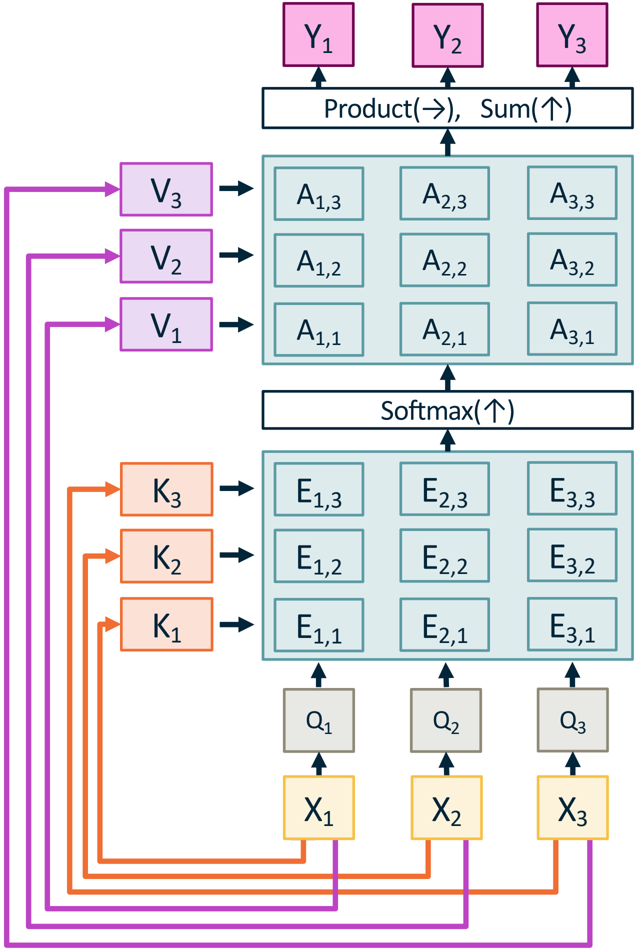
Slide credit: Justin Johnson

Self-Attention Layer

Sequence encode -> use each input element as query!

Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

Self-Attention Layer

Inputs:

Input vectors: X (Shape: N_X x D_X)

Key matrix: W_K (Shape: D_X x D_Q)

Value matrix: W_V (Shape: D_X x D_V)

Query matrix: W_Q (Shape: D_X x D_Q)

Computation:

Query vectors: $Q = XW_Q$

Key vectors: $K = XW_K$ (Shape: N_X x D_Q)

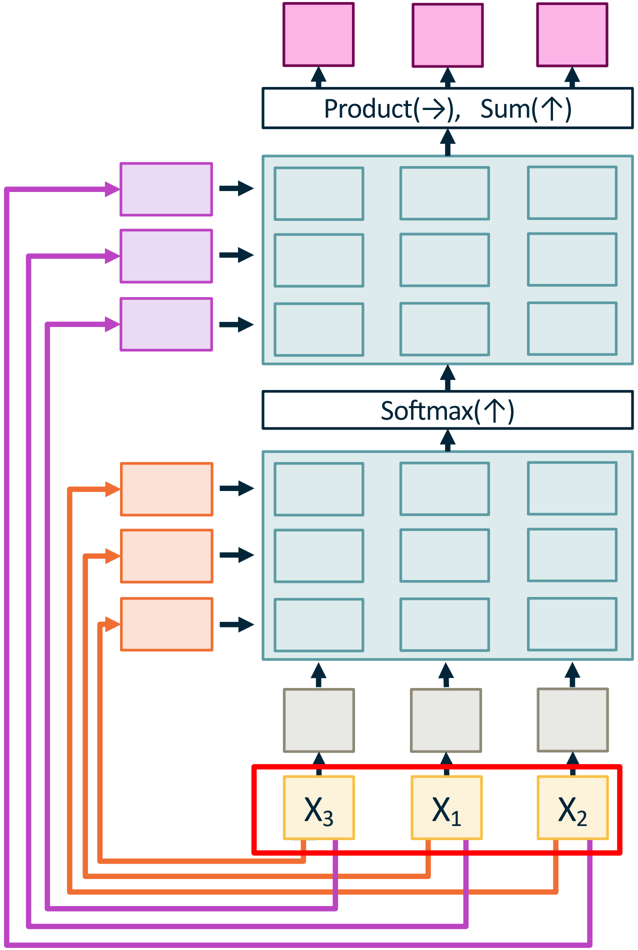
Value vectors: $V = XW_V$ (Shape: N_X x D_V)

Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$

Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)

Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

Consider **permuting** the input vectors:



Slide credit: Justin Johnson

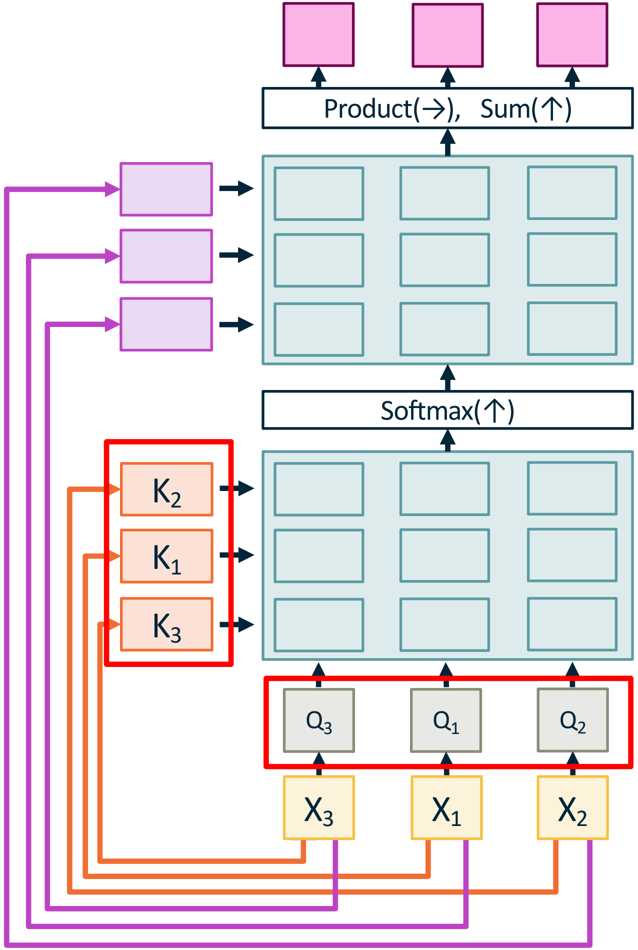
Self-Attention Layer

Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Consider **permuting** the input vectors:

Queries and Keys will be the same, but permuted

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value Vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

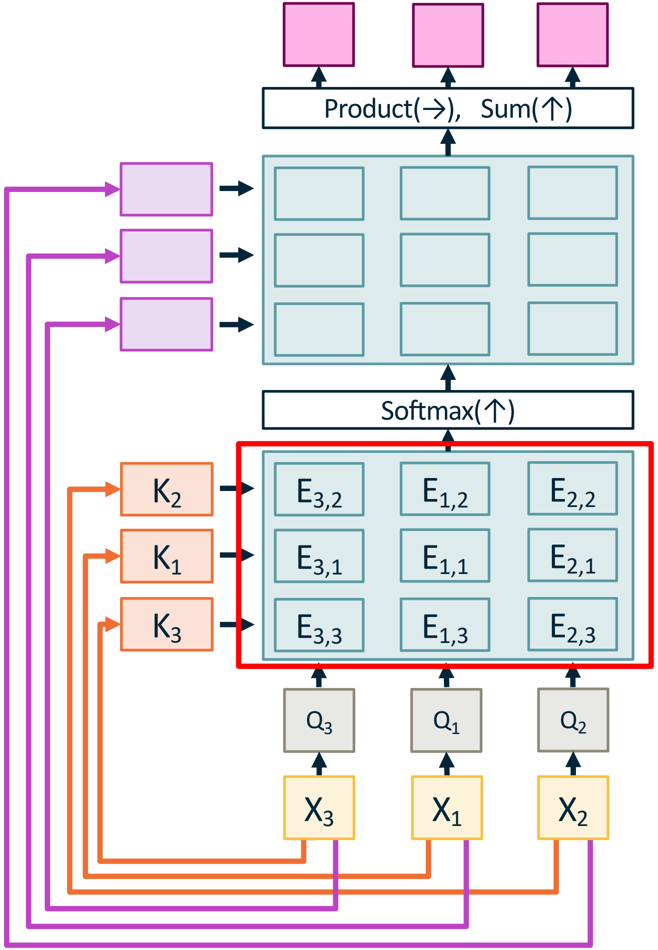
Self-Attention Layer

Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Consider **permuting** the input vectors:

Similarities will be the same, but permuted

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

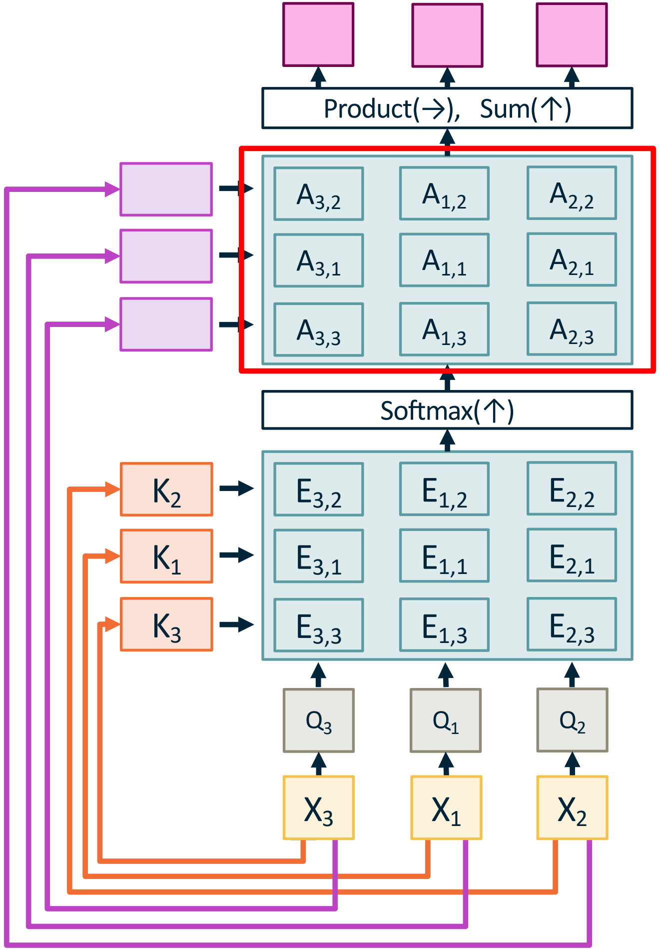
Self-Attention Layer

Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

Consider **permuting** the input vectors:

Attention weights will be the same, but permuted



Slide credit: Justin Johnson

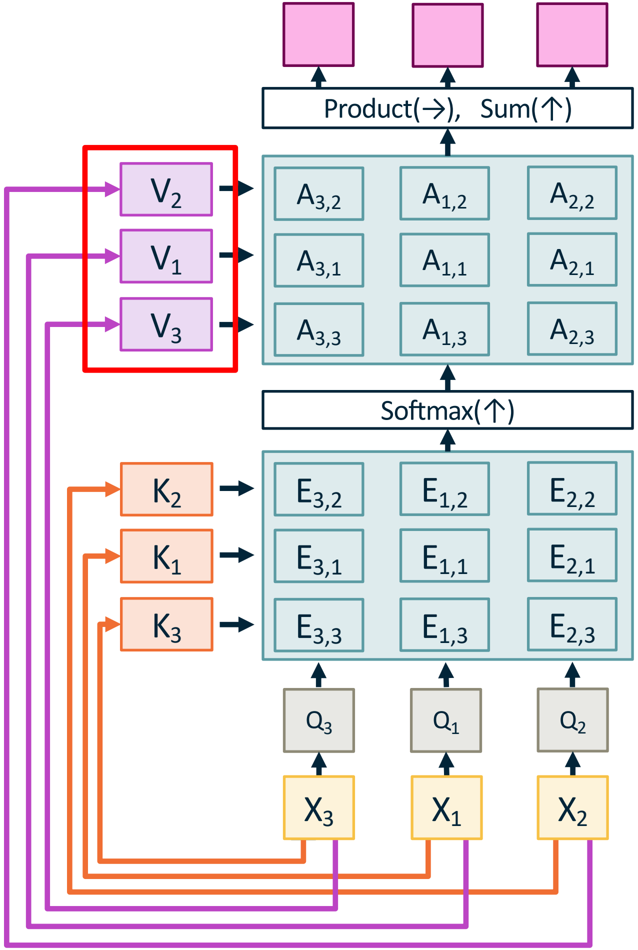
Self-Attention Layer

Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Consider **permuting** the input vectors:

Values will be the same, but permuted

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

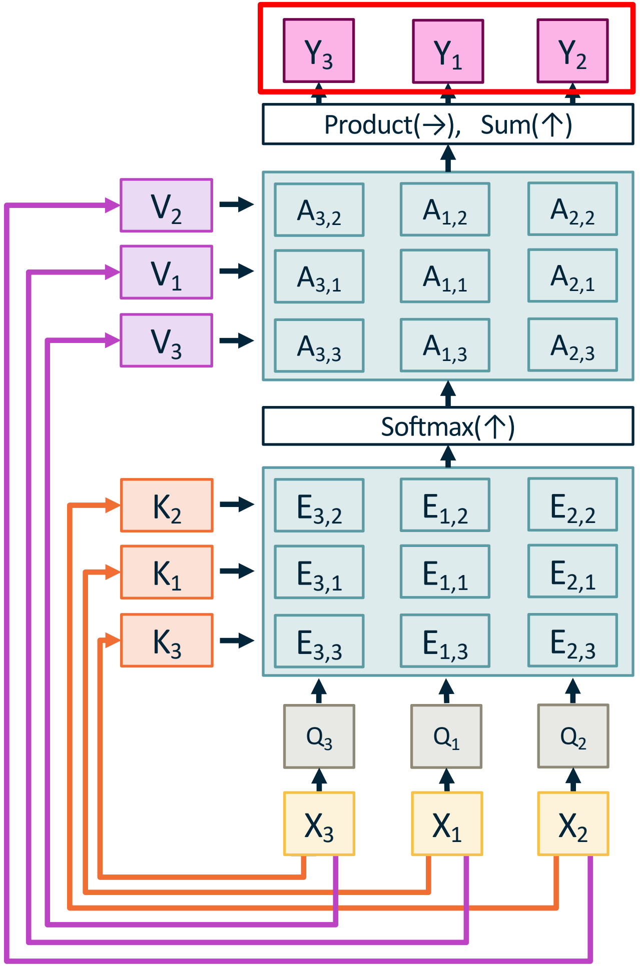
Self-Attention Layer

Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

Consider **permuting** the input vectors:

Outputs will be the same, but permuted



Slide credit: Justin Johnson

Self-Attention Layer

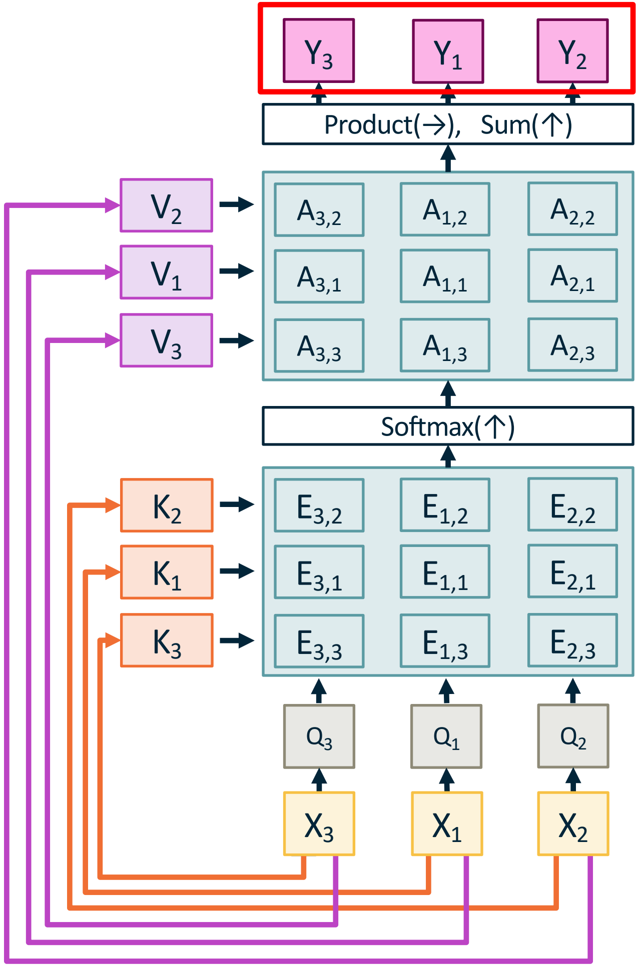
Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

Consider **permuting** the input vectors:

Outputs will be the same, but permuted

Self-attention layer is **Permutation Equivariant**
f(s(x)) = s(f(x))



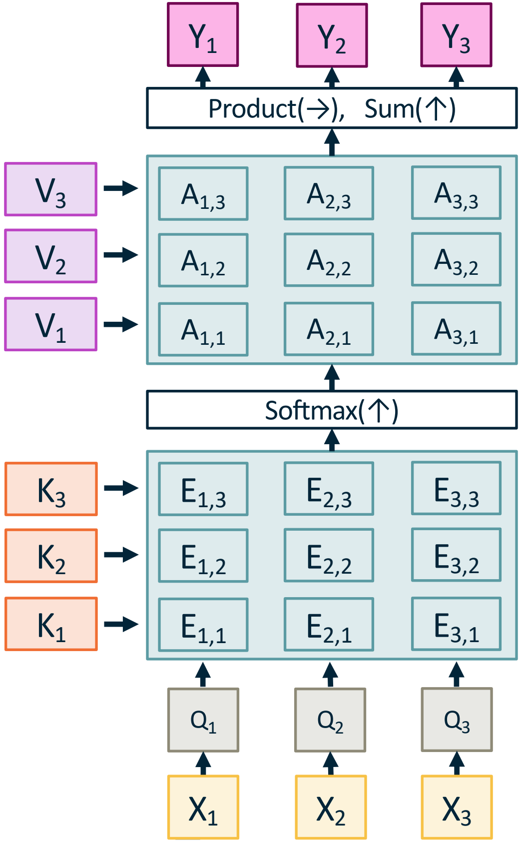
Slide credit: Justin Johnson

Self-Attention Layer

Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Self attention doesn't "know" the order of the vectors it is processing! Not good for sequence encoding.

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

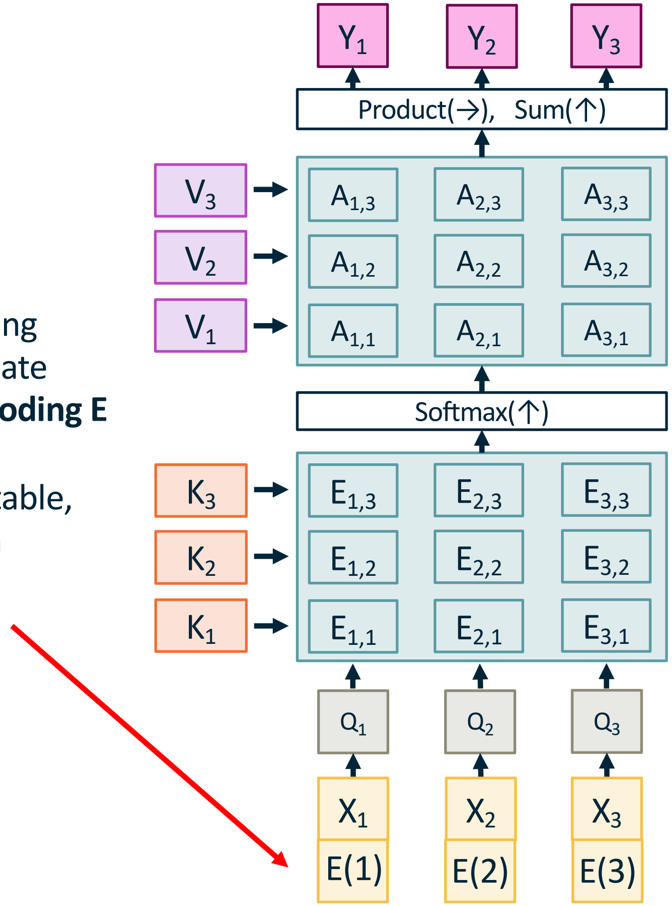
Self-Attention Layer

Inputs:
Input vectors: X (Shape: $N_X \times D_X$)
Key matrix: W_K (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)
Query matrix: W_Q (Shape: $D_X \times D_Q$)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: $N_X \times D_Q$)
Value vectors: $V = XW_V$ (Shape: $N_X \times D_V$)
Similarities: $E = QK^T$ (Shape: $N_X \times N_X$) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

In order to make processing position-aware, concatenate input with **positional encoding E**

E can be learned lookup table, or fixed function (e.g., sin function)



Slide credit: Justin Johnson

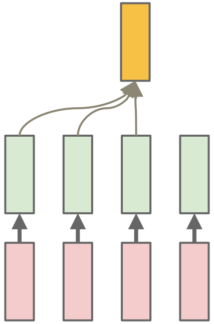
Masked Self-Attention Layer

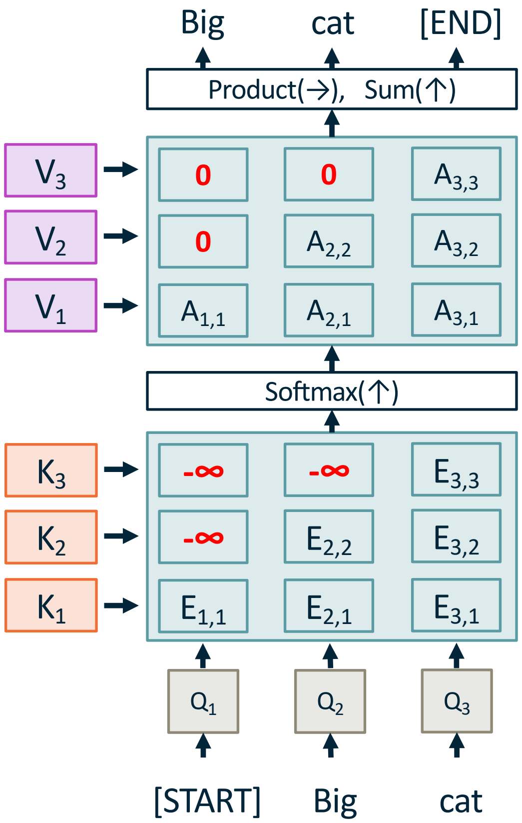
Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Don’t let vectors “look ahead” in the sequence

Used for language modeling (predict next word)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: $A = \mathrm{softmax}(E, \mathrm{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$





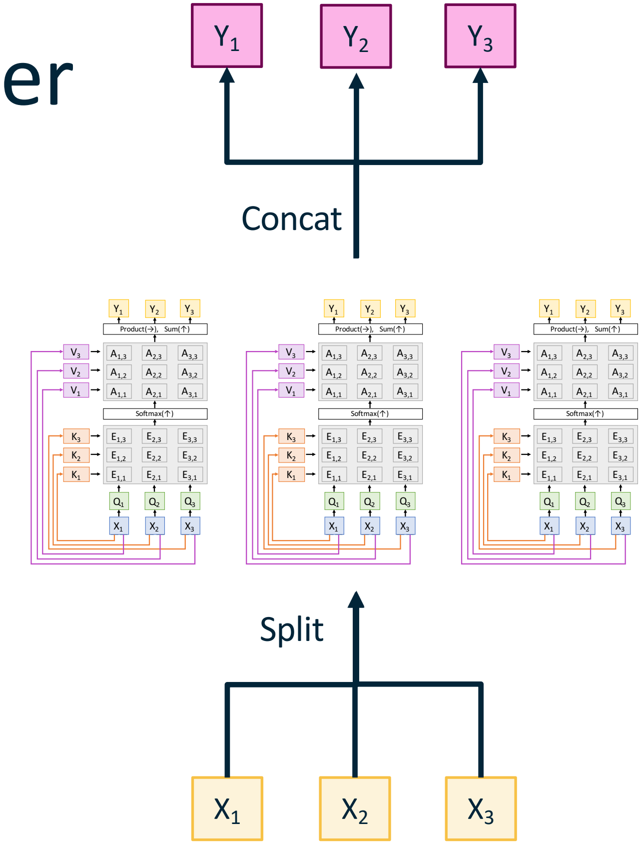
Slide credit: Justin Johnson

Multi-headed Self-Attention Layer

Inputs:
Input vectors: X (Shape: N_X x D_X)
Key matrix: W_K (Shape: D_X x D_Q)
Value matrix: W_V (Shape: D_X x D_V)
Query matrix: W_Q (Shape: D_X x D_Q)

Computation:
Query vectors: $Q = XW_Q$
Key vectors: $K = XW_K$ (Shape: N_X x D_Q)
Value vectors: $V = XW_V$ (Shape: N_X x D_V)
Similarities: $E = QK^T$ (Shape: N_X x N_X) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$

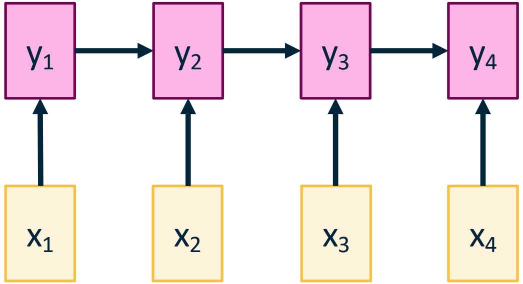
Use H independent “Attention Heads” in parallel



Slide credit: Justin Johnson

Three Ways of Processing Sequences

Recurrent Neural Network



Works on **Ordered Sequences**

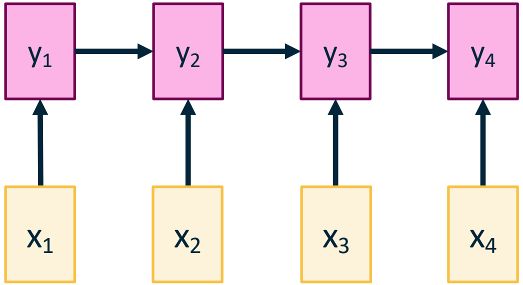
(+) Good at long sequences: After one RNN layer, h_T "sees" the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

Slide credit: Justin Johnson

Three Ways of Processing Sequences

Recurrent Neural Network

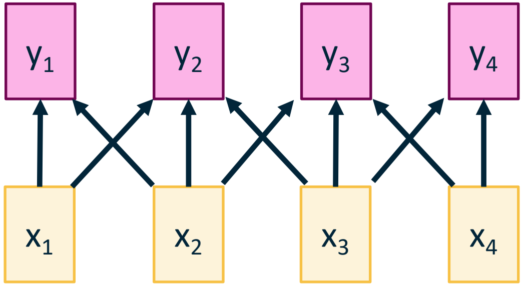


Works on **Ordered Sequences**

(+) Good at long sequences: After one RNN layer, h_T ”sees” the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

1D Convolution



Works on **Multidimensional Grids**

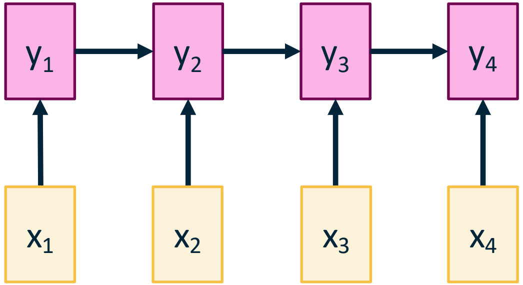
(-) Bad at long sequences: Need to stack many conv layers for outputs to “see” the whole sequence

(+) Highly parallel: Each output can be computed in parallel

Slide credit: Justin Johnson

Three Ways of Processing Sequences

Recurrent Neural Network

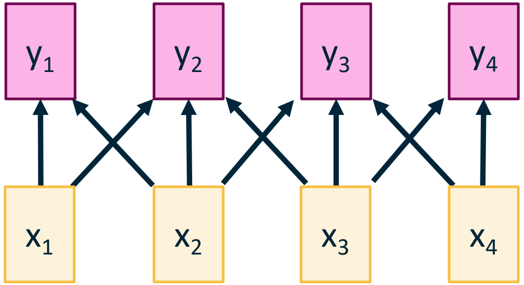


Works on **Ordered Sequences**

(+) Good at long sequences: After one RNN layer, h_T ”sees” the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

1D Convolution

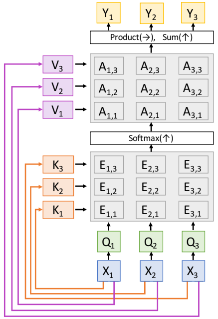


Works on **Multidimensional Grids**

(-) Bad at long sequences: Need to stack many conv layers for outputs to “see” the whole sequence

(+) Highly parallel: Each output can be computed in parallel

Self-Attention



Works on **Sets of Vectors**

(+) Good at long sequences: after one self-attention layer, each output “sees” all inputs!

(+) Highly parallel: Each output can be computed in parallel

(-) Very memory intensive

Slide credit: Justin Johnson

Three Ways of Processing Sequences

Recurrent Neural Network

1D Convolution

Self-Attention

Attention is all you need

Vaswani et al, NeurIPS 2017

Works on **Ordered Sequences**

(+) Good at long sequences: After one RNN layer, h_T "sees" the whole sequence

(-) Not parallelizable: need to compute hidden states sequentially

Works on **Multidimensional Grids**

(-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence

(+) Highly parallel: Each output can be computed in parallel

Works on **Sets of Vectors**

(+) Good at long sequences: after one self-attention layer, each output "sees" all inputs!

(+) Highly parallel: Each output can be computed in parallel

(-) Very memory intensive

Slide credit: Justin Johnson

The Transformer Block

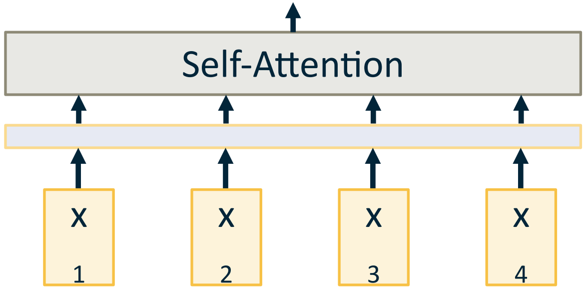
x_1 x_2 x_3 x_4

Vaswani et al, “Attention is all you need”, NeurIPS 2017

Slide credit: Justin Johnson

The Transformer Block

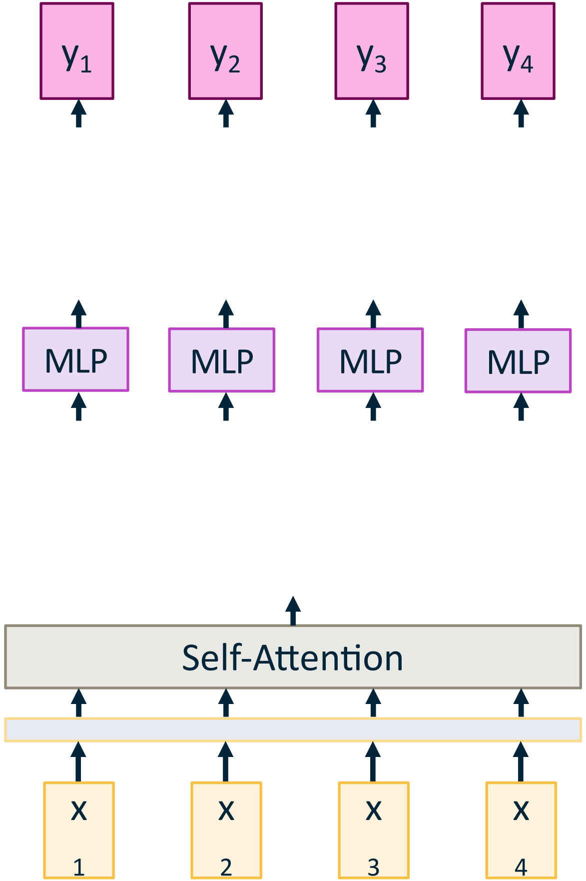
All vectors interact with each other



Vaswani et al, "Attention is all you need", NeurIPS 2017

Slide credit: Justin Johnson

The Transformer Block



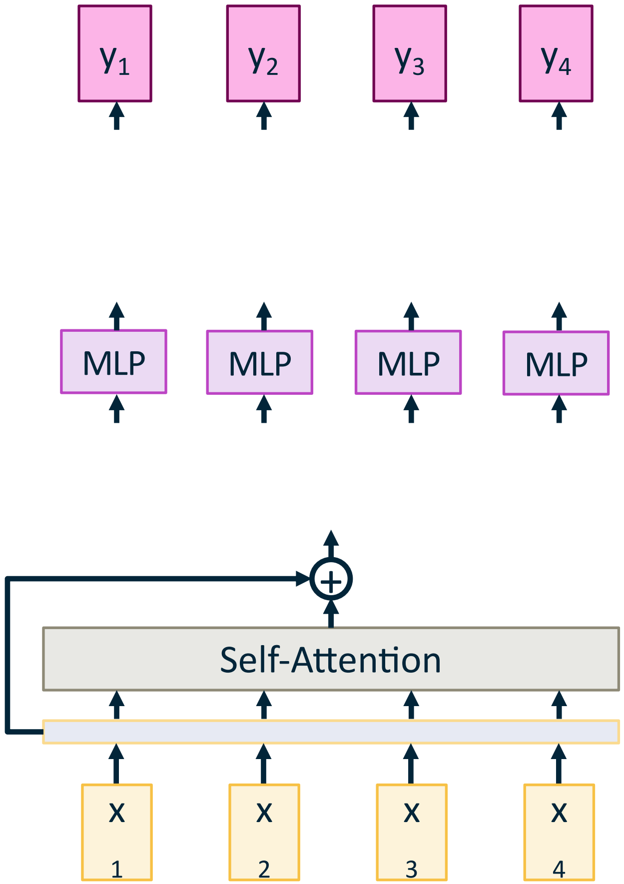
MLP independently on each vector

All vectors interact with each other

Vaswani et al, "Attention is all you need", NeurIPS 2017

Slide credit: Justin Johnson

The Transformer Block



MLP independently on each vector

Residual connection

All vectors interact with each other

Vaswani et al, "Attention is all you need", NeurIPS 2017

Slide credit: Justin Johnson

The Transformer Block

Recall **Layer Normalization**:

Given $h_1, \ldots, h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

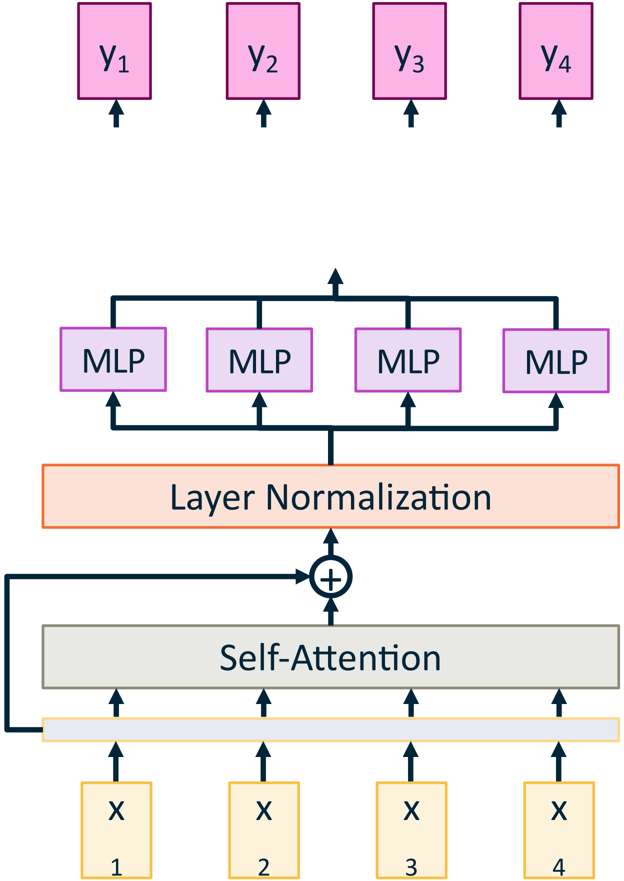
$\mu_i = (1/D)\sum_j h_{i,j}$ (scalar)

$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

$y_i = \gamma * z_i + \beta$

Ba et al, 2016



MLP independently on each vector

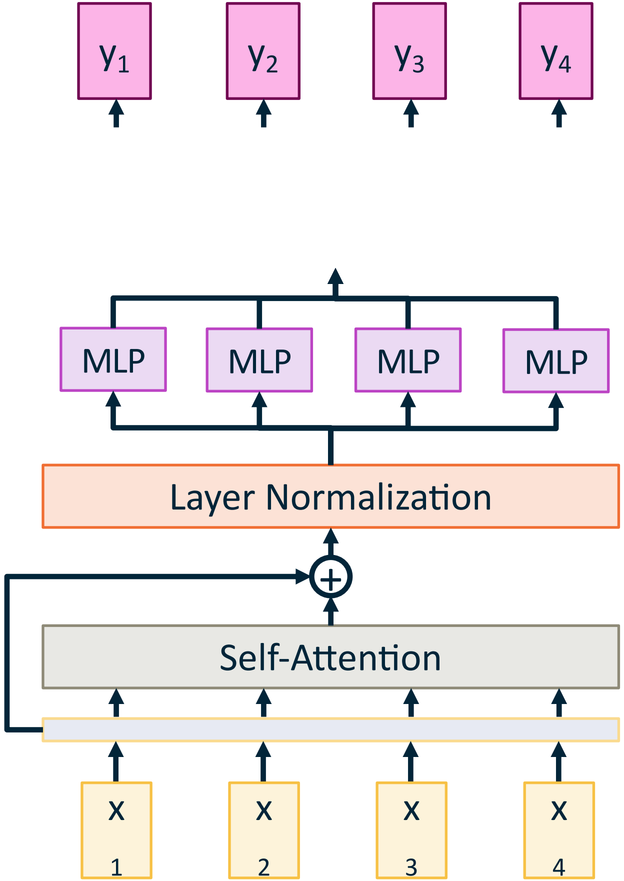
Residual connection

All vectors interact with each other

Vaswani et al, "Attention is all you need", NeurIPS 2017

Slide credit: Justin Johnson

The Transformer Block



MLP independently on each vector

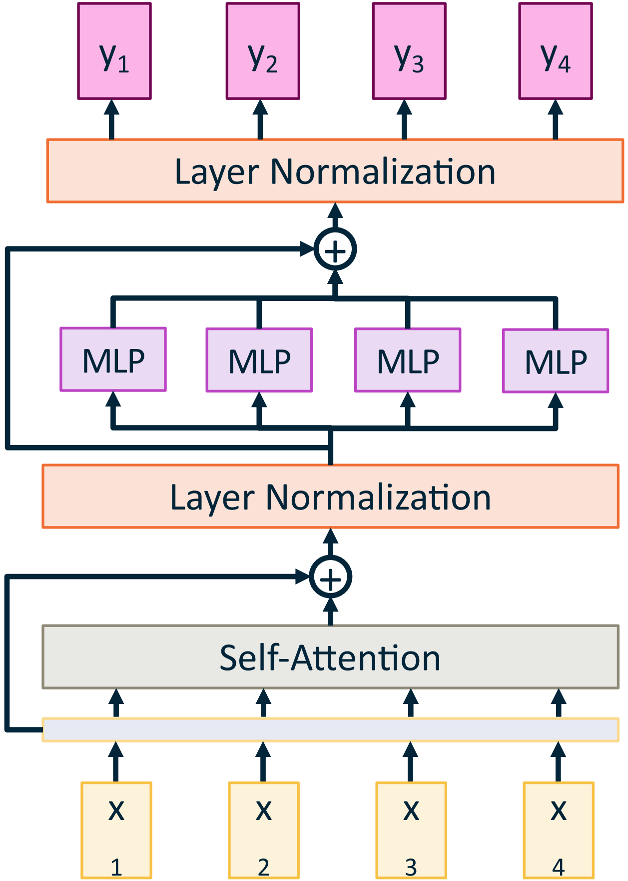
Residual connection

All vectors interact with each other

Vaswani et al, "Attention is all you need", NeurIPS 2017

Slide credit: Justin Johnson

The Transformer Block



Residual connection

MLP independently on each vector

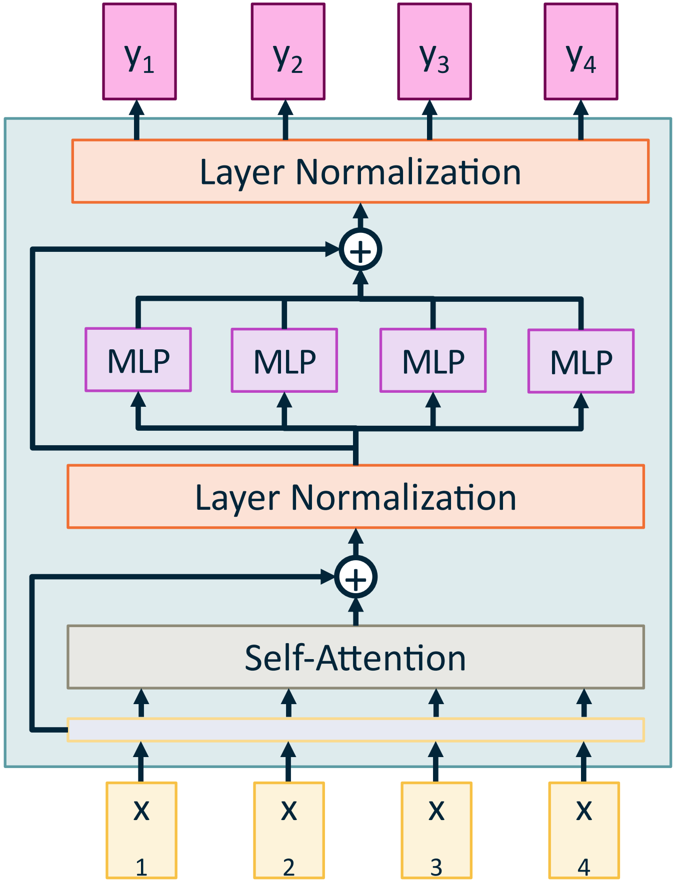
Residual connection

All vectors interact with each other

Vaswani et al, "Attention is all you need", NeurIPS 2017

Slide credit: Justin Johnson

The Transformer Block



<u>Transformer Block</u>:
Input: Set of vectors x
Output: Set of vectors y

Self-attention is the only interaction among vectors!

Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable

Vaswani et al, "Attention is all you need", NeurIPS 2017

Slide credit: Justin Johnson

The Transformer

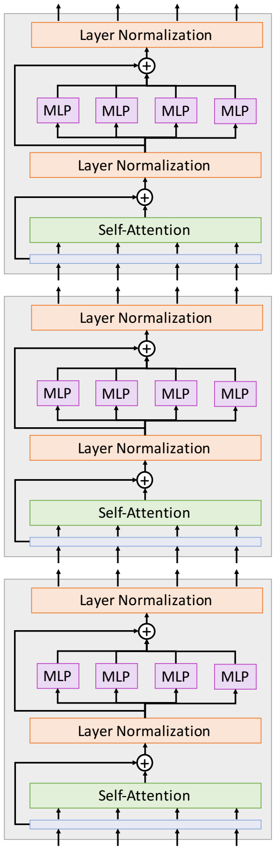
Transformer Block:
Input: Set of vectors x
Output: Set of vectors y

Self-attention is the only interaction between vectors!

Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable

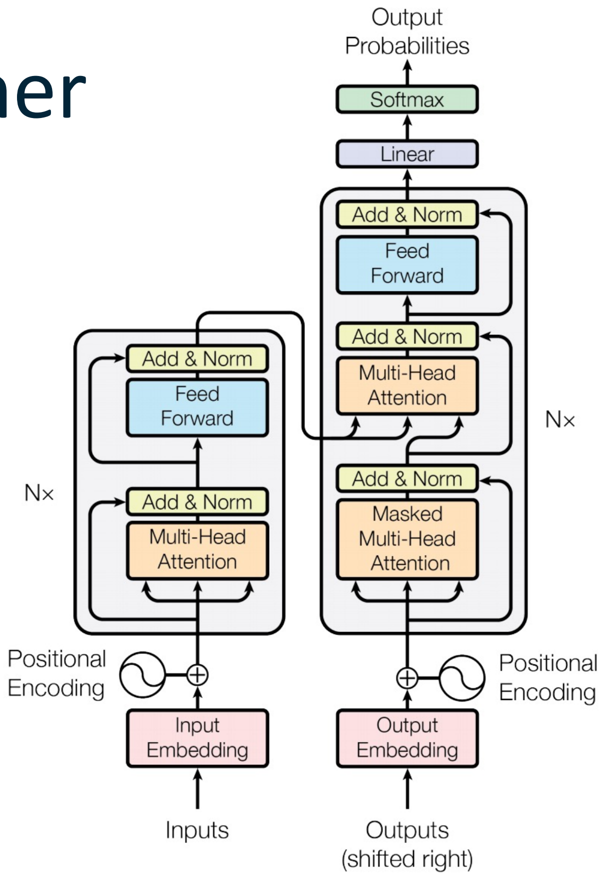
A **Transformer** is a sequence of transformer blocks



Vaswani et al, "Attention is all you need", NeurIPS 2017

Slide credit: Justin Johnson

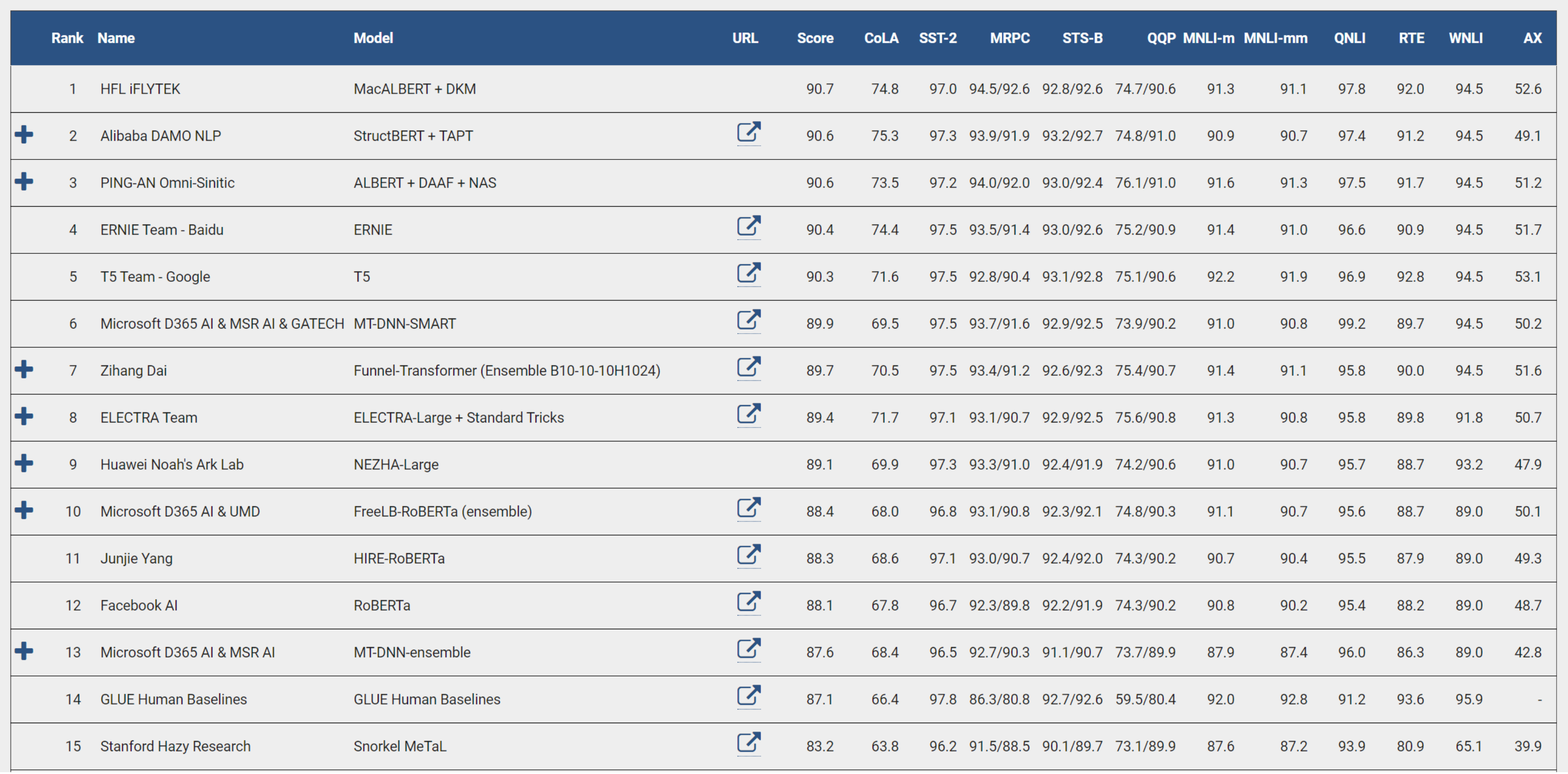
The Transformer



Encoder-Decoder

Vaswani et al, “Attention is all you need”, NeurIPS 2017

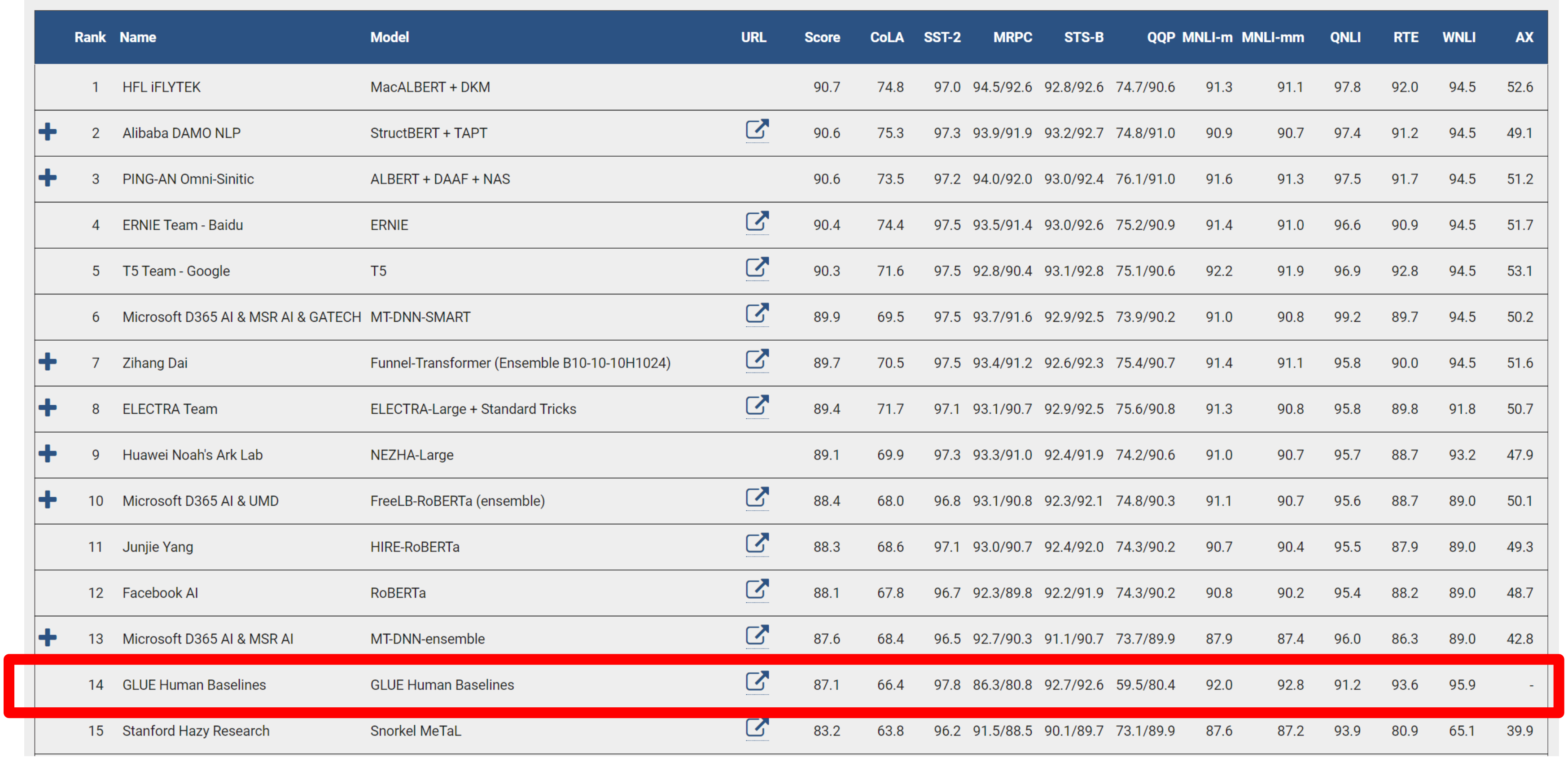
GLUE Benchmark



	Rank	Name	Model	URL	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-mm	QNLI	RTE	WNLI	AX
	1	HFL iFLYTEK	MacALBERT + DKM		90.7	74.8	97.0	94.5/92.6	92.8/92.6	74.7/90.6	91.3	91.1	97.8	92.0	94.5	52.6
+	2	Alibaba DAMO NLP	StructBERT + TAPT	↗	90.6	75.3	97.3	93.9/91.9	93.2/92.7	74.8/91.0	90.9	90.7	97.4	91.2	94.5	49.1
+	3	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6	73.5	97.2	94.0/92.0	93.0/92.4	76.1/91.0	91.6	91.3	97.5	91.7	94.5	51.2
	4	ERNIE Team - Baidu	ERNIE	↗	90.4	74.4	97.5	93.5/91.4	93.0/92.6	75.2/90.9	91.4	91.0	96.6	90.9	94.5	51.7
	5	T5 Team - Google	T5	↗	90.3	71.6	97.5	92.8/90.4	93.1/92.8	75.1/90.6	92.2	91.9	96.9	92.8	94.5	53.1
	6	Microsoft D365 AI & MSR AI & GATECH	MT-DNN-SMART	↗	89.9	69.5	97.5	93.7/91.6	92.9/92.5	73.9/90.2	91.0	90.8	99.2	89.7	94.5	50.2
+	7	Zihang Dai	Funnel-Transformer (Ensemble B10-10-10H1024)	↗	89.7	70.5	97.5	93.4/91.2	92.6/92.3	75.4/90.7	91.4	91.1	95.8	90.0	94.5	51.6
+	8	ELECTRA Team	ELECTRA-Large + Standard Tricks	↗	89.4	71.7	97.1	93.1/90.7	92.9/92.5	75.6/90.8	91.3	90.8	95.8	89.8	91.8	50.7
+	9	Huawei Noah's Ark Lab	NEZHA-Large		89.1	69.9	97.3	93.3/91.0	92.4/91.9	74.2/90.6	91.0	90.7	95.7	88.7	93.2	47.9
+	10	Microsoft D365 AI & UMD	FreeLB-RoBERTa (ensemble)	↗	88.4	68.0	96.8	93.1/90.8	92.3/92.1	74.8/90.3	91.1	90.7	95.6	88.7	89.0	50.1
	11	Junjie Yang	HIRE-RoBERTa	↗	88.3	68.6	97.1	93.0/90.7	92.4/92.0	74.3/90.2	90.7	90.4	95.5	87.9	89.0	49.3
	12	Facebook AI	RoBERTa	↗	88.1	67.8	96.7	92.3/89.8	92.2/91.9	74.3/90.2	90.8	90.2	95.4	88.2	89.0	48.7
+	13	Microsoft D365 AI & MSR AI	MT-DNN-ensemble	↗	87.6	68.4	96.5	92.7/90.3	91.1/90.7	73.7/89.9	87.9	87.4	96.0	86.3	89.0	42.8
	14	GLUE Human Baselines	GLUE Human Baselines	↗	87.1	66.4	97.8	86.3/80.8	92.7/92.6	59.5/80.4	92.0	92.8	91.2	93.6	95.9	-
	15	Stanford Hazy Research	Snorkel MeTaL	↗	83.2	63.8	96.2	91.5/88.5	90.1/89.7	73.1/89.9	87.6	87.2	93.9	80.9	65.1	39.9

source: https://gluebenchmark.com/leaderboard

GLUE Benchmark



	Rank	Name	Model	URL	Score	CoLA	SST-2	MRPC	STS-B	QQP	MNLI-m	MNLI-mm	QNLI	RTE	WNLI	AX
	1	HFL iFLYTEK	MacALBERT + DKM		90.7	74.8	97.0	94.5/92.6	92.8/92.6	74.7/90.6	91.3	91.1	97.8	92.0	94.5	52.6
+	2	Alibaba DAMO NLP	StructBERT + TAPT	↗	90.6	75.3	97.3	93.9/91.9	93.2/92.7	74.8/91.0	90.9	90.7	97.4	91.2	94.5	49.1
+	3	PING-AN Omni-Sinitic	ALBERT + DAAF + NAS		90.6	73.5	97.2	94.0/92.0	93.0/92.4	76.1/91.0	91.6	91.3	97.5	91.7	94.5	51.2
	4	ERNIE Team - Baidu	ERNIE	↗	90.4	74.4	97.5	93.5/91.4	93.0/92.6	75.2/90.9	91.4	91.0	96.6	90.9	94.5	51.7
	5	T5 Team - Google	T5	↗	90.3	71.6	97.5	92.8/90.4	93.1/92.8	75.1/90.6	92.2	91.9	96.9	92.8	94.5	53.1
	6	Microsoft D365 AI & MSR AI & GATECH	MT-DNN-SMART	↗	89.9	69.5	97.5	93.7/91.6	92.9/92.5	73.9/90.2	91.0	90.8	99.2	89.7	94.5	50.2
+	7	Zihang Dai	Funnel-Transformer (Ensemble B10-10-10H1024)	↗	89.7	70.5	97.5	93.4/91.2	92.6/92.3	75.4/90.7	91.4	91.1	95.8	90.0	94.5	51.6
+	8	ELECTRA Team	ELECTRA-Large + Standard Tricks	↗	89.4	71.7	97.1	93.1/90.7	92.9/92.5	75.6/90.8	91.3	90.8	95.8	89.8	91.8	50.7
+	9	Huawei Noah's Ark Lab	NEZHA-Large		89.1	69.9	97.3	93.3/91.0	92.4/91.9	74.2/90.6	91.0	90.7	95.7	88.7	93.2	47.9
+	10	Microsoft D365 AI & UMD	FreeLB-RoBERTa (ensemble)	↗	88.4	68.0	96.8	93.1/90.8	92.3/92.1	74.8/90.3	91.1	90.7	95.6	88.7	89.0	50.1
	11	Junjie Yang	HIRE-RoBERTa	↗	88.3	68.6	97.1	93.0/90.7	92.4/92.0	74.3/90.2	90.7	90.4	95.5	87.9	89.0	49.3
	12	Facebook AI	RoBERTa	↗	88.1	67.8	96.7	92.3/89.8	92.2/91.9	74.3/90.2	90.8	90.2	95.4	88.2	89.0	48.7
+	13	Microsoft D365 AI & MSR AI	MT-DNN-ensemble	↗	87.6	68.4	96.5	92.7/90.3	91.1/90.7	73.7/89.9	87.9	87.4	96.0	86.3	89.0	42.8
	14	GLUE Human Baselines	GLUE Human Baselines	↗	87.1	66.4	97.8	86.3/80.8	92.7/92.6	59.5/80.4	92.0	92.8	91.2	93.6	95.9	-
	15	Stanford Hazy Research	Snorkel MeTaL	↗	83.2	63.8	96.2	91.5/88.5	90.1/89.7	73.1/89.9	87.6	87.2	93.9	80.9	65.1	39.9

source: https://gluebenchmark.com/leaderboard

SYSTEM PROMPT (HUMAN-WRITTEN)
In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)
The scientist named the population, after their distinctive horn, Ovid’s Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

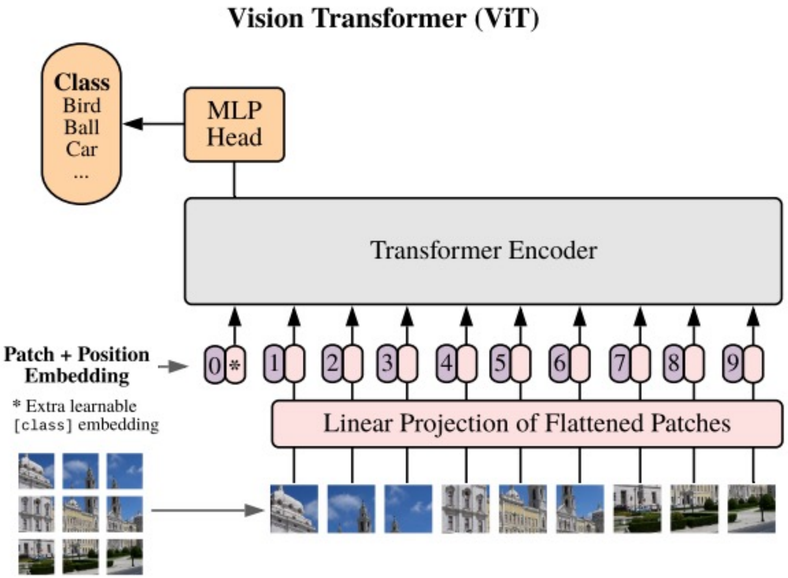
Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Pérez and the others then ventured further into the valley. “By the time we reached the top of one peak, the water looked blue, with some crystals on top,” said Pérez.

Source: OpenAI, “Better Language Models and Their Implications”
https://openai.com/blog/better-language-models/

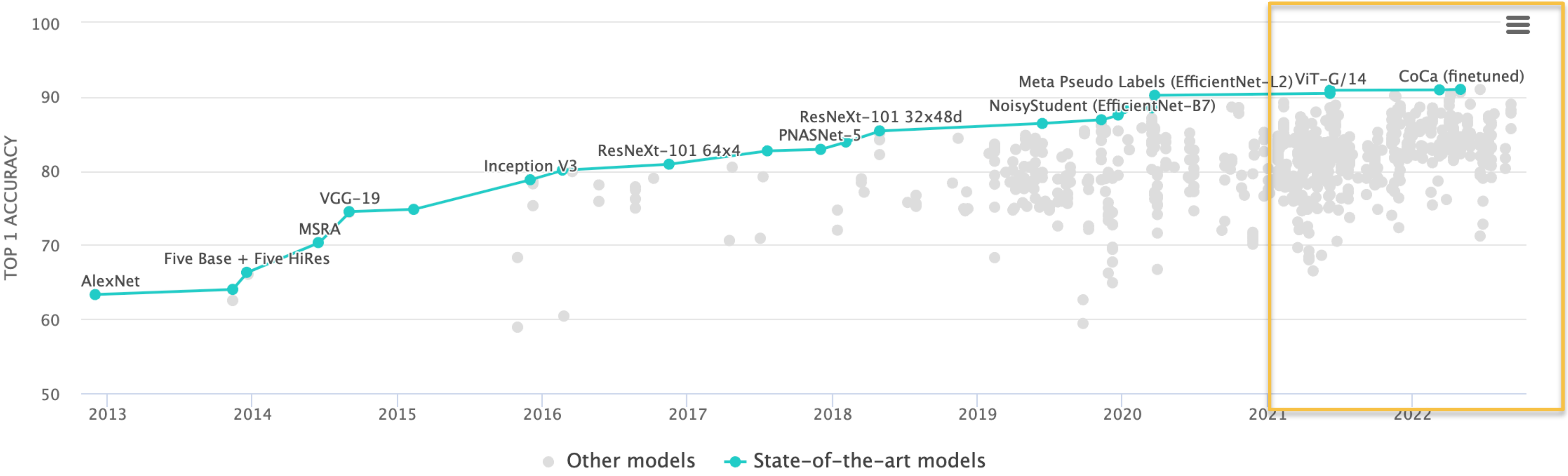
Can Attention/Transformers be used from more than text processing?

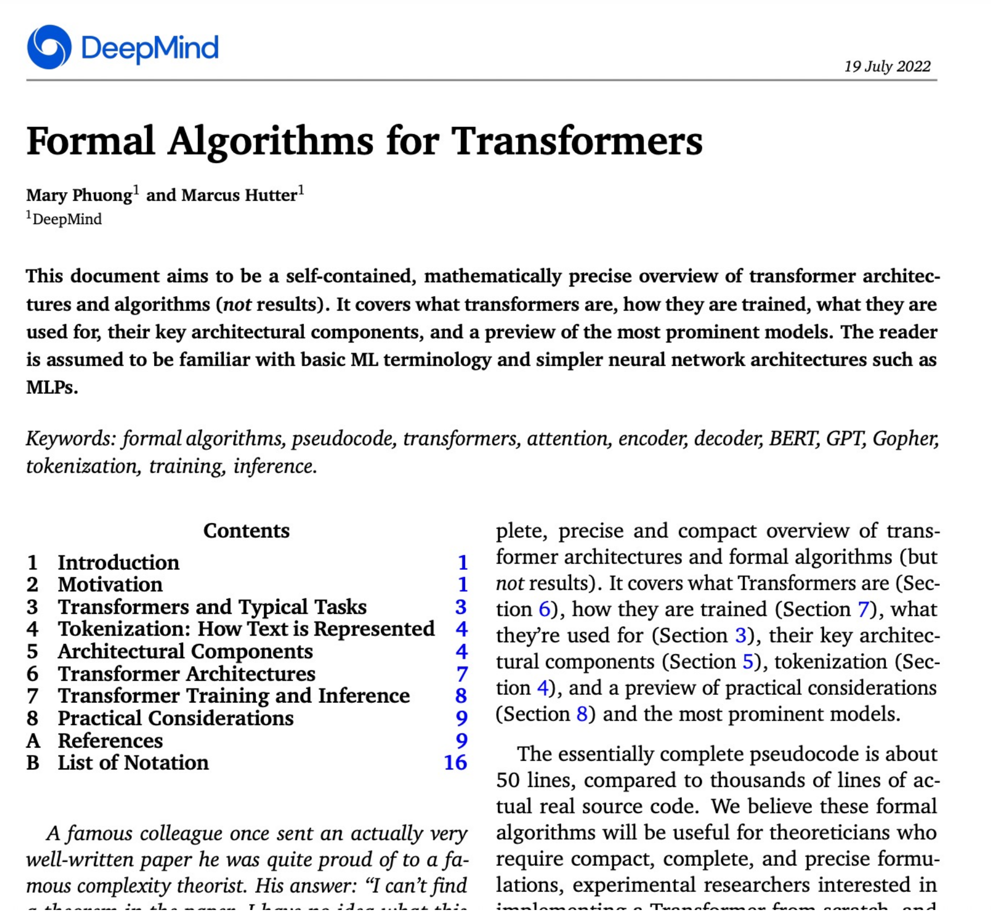
ViT: Vision Transformer



An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale
(Dosovitskiy *et al.*, 2021)

ViT: Vision Transformer





DeepMind

19 July 2022

Formal Algorithms for Transformers

Mary Phuong[1] **and Marcus Hutter**[1]
[1]DeepMind

This document aims to be a self-contained, mathematically precise overview of transformer architectures and algorithms (*not* results). It covers what transformers are, how they are trained, what they are used for, their key architectural components, and a preview of the most prominent models. The reader is assumed to be familiar with basic ML terminology and simpler neural network architectures such as MLPs.

Keywords: formal algorithms, pseudocode, transformers, attention, encoder, decoder, BERT, GPT, Gopher, tokenization, training, inference.

Contents

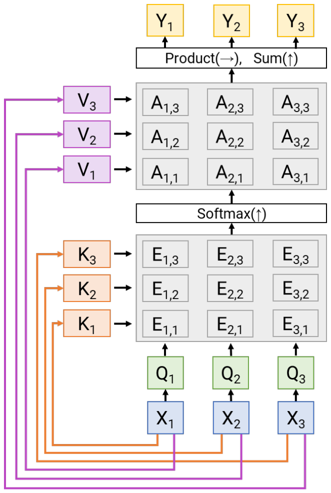
A famous colleague once sent an actually very well-written paper he was quite proud of to a famous complexity theorist. His answer: "I can't find

plete, precise and compact overview of transformer architectures and formal algorithms (but *not* results). It covers what Transformers are (Section 6), how they are trained (Section 7), what they're used for (Section 3), their key architectural components (Section 5), tokenization (Section 4), and a preview of practical considerations (Section 8) and the most prominent models.

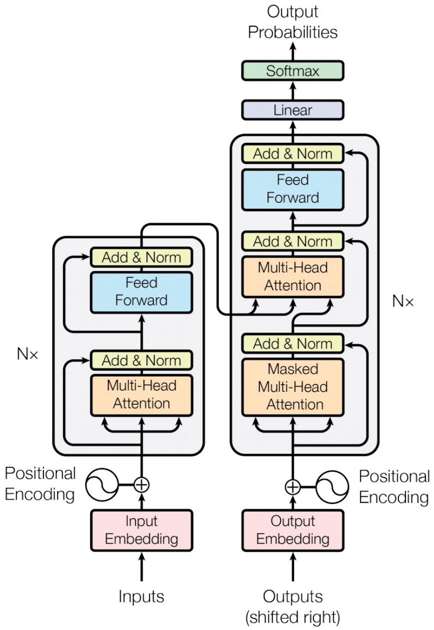
The essentially complete pseudocode is about 50 lines, compared to thousands of lines of actual real source code. We believe these formal algorithms will be useful for theoreticians who require compact, complete, and precise formulations, experimental researchers interested in

Summary

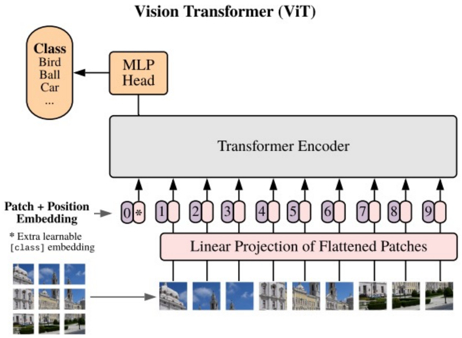
Self-Attention



Transformer Model



Beyond Language



Next time: Guest Lecture on Robotics + Language + Vision!

Dr. Fei Xia
Google Research

Embodied Reasoning Through Planning with Language and Vision Foundation Models.

Zoom only (no in-person lecture)