Topics:

- Transformers
- Vision Transformers

CS 4644-DL / 7643-A

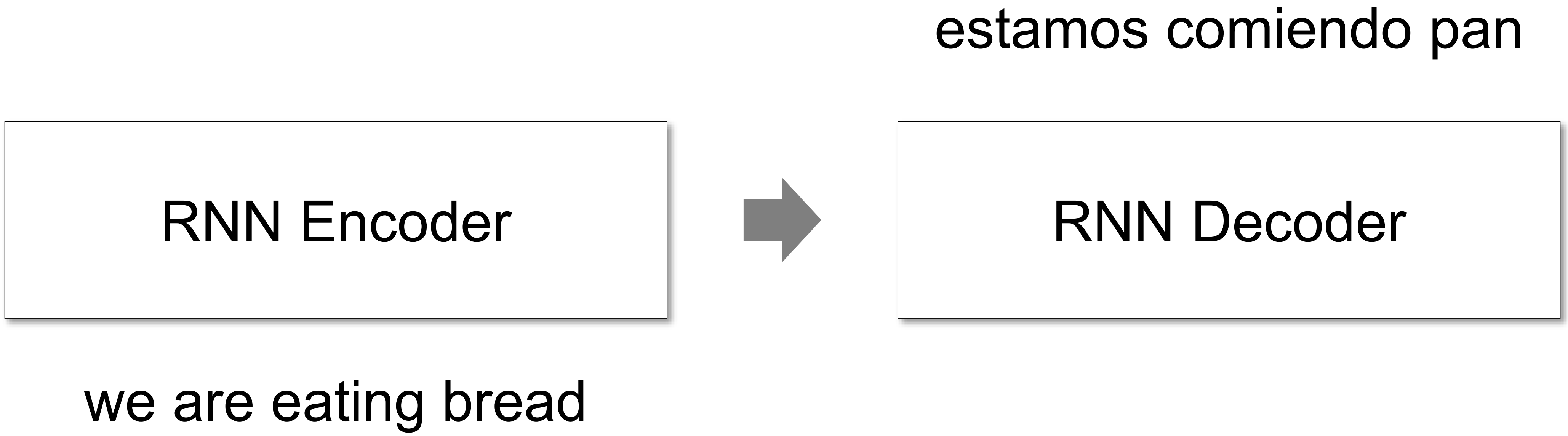
ZSOLT KIRA

- **Assignment 3 out**
  - Due **March 14th 11:59pm EST**

- Quiz **March 18th**

- Project milestone
  - Due ~~**March 14**~~ **March 20th 11:59pm EST**

Machine Translation

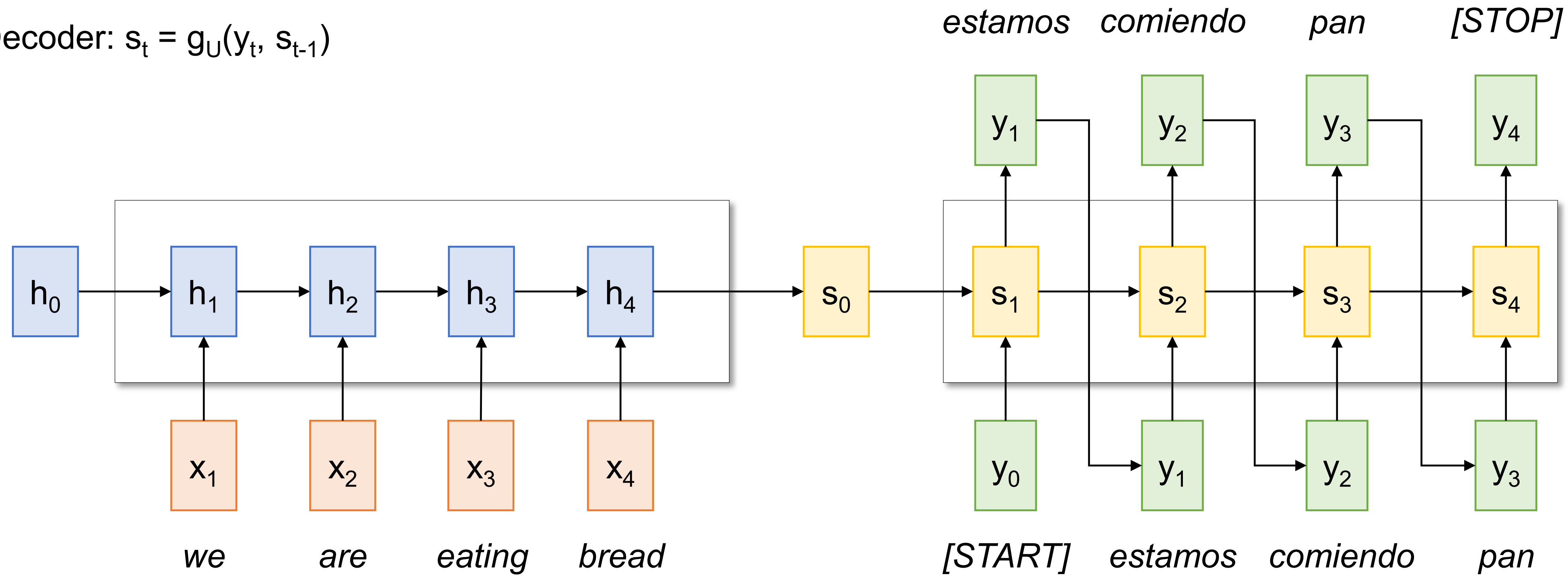


Machine Translation with RNNs

Note [START]/[STOP] words.
This can be treated as representation for entire sentence

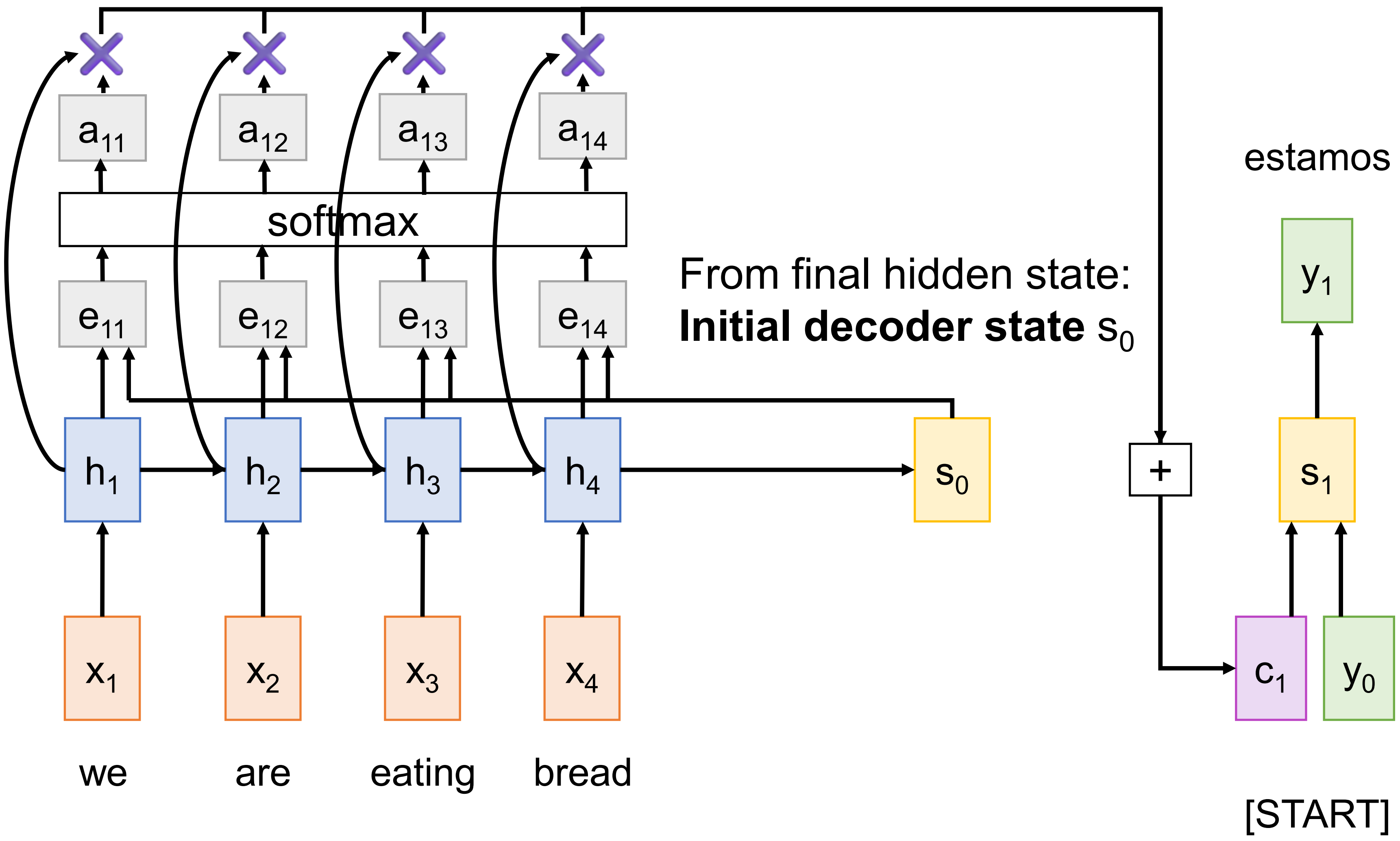
Encoder: $h_t = f_W(x_t, h_{t-1})$

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



Slide credit: Justin Johnson

Machine Translation with RNNs **and Attention**



Compute **alignment 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

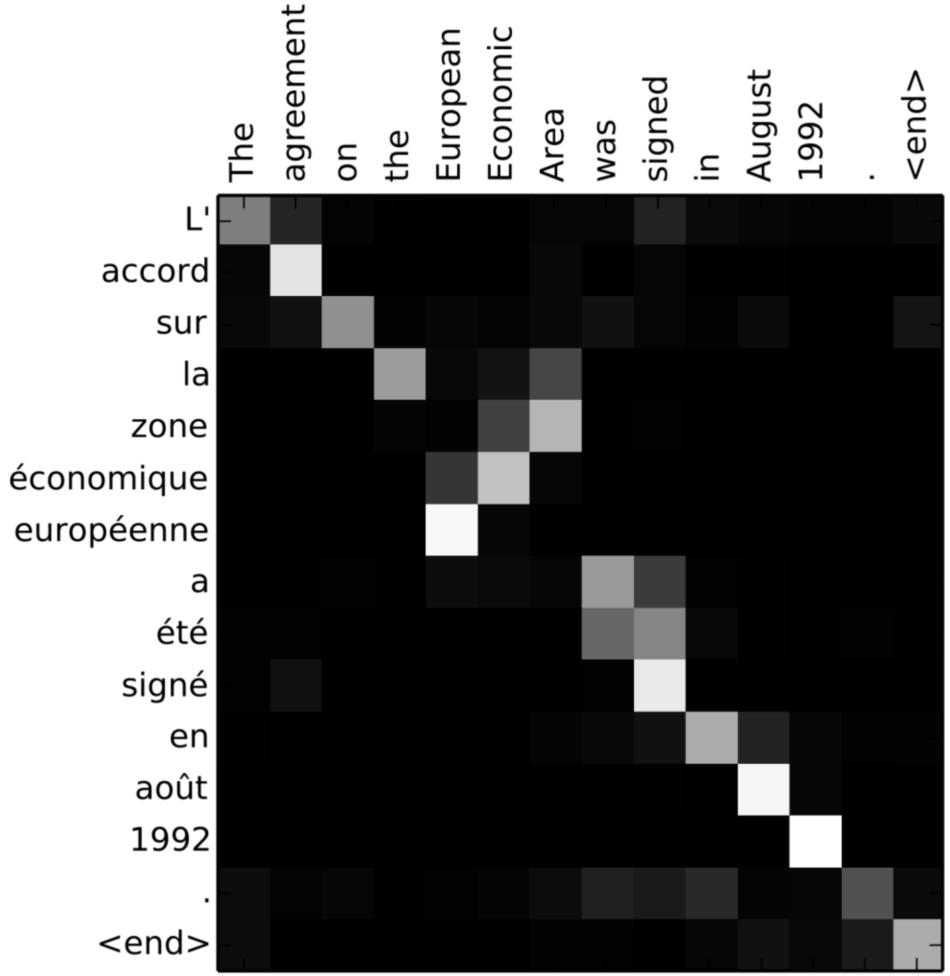
Machine Translation with RNNs **and Attention**

Example: English to French translation

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

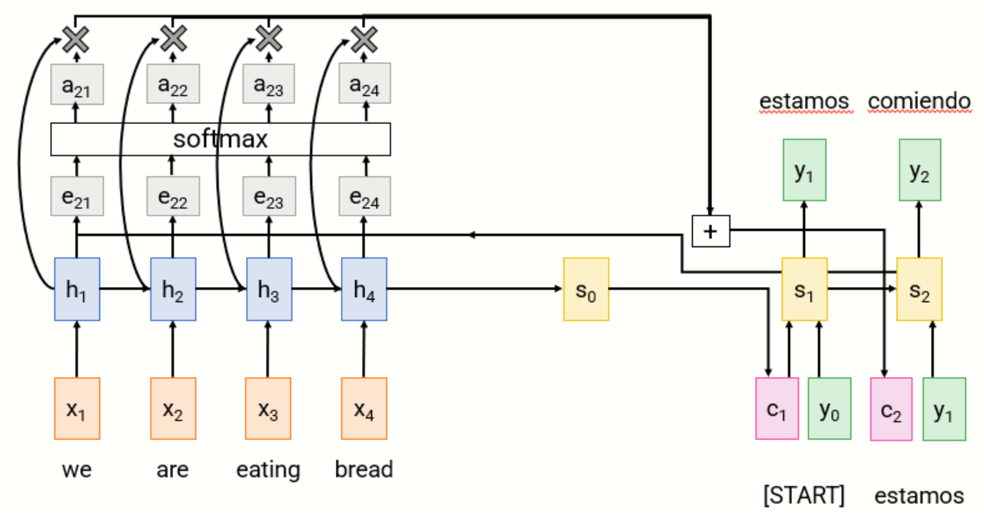
Output: “L’accord sur la zone économique européenne a été signé en août 1992.”

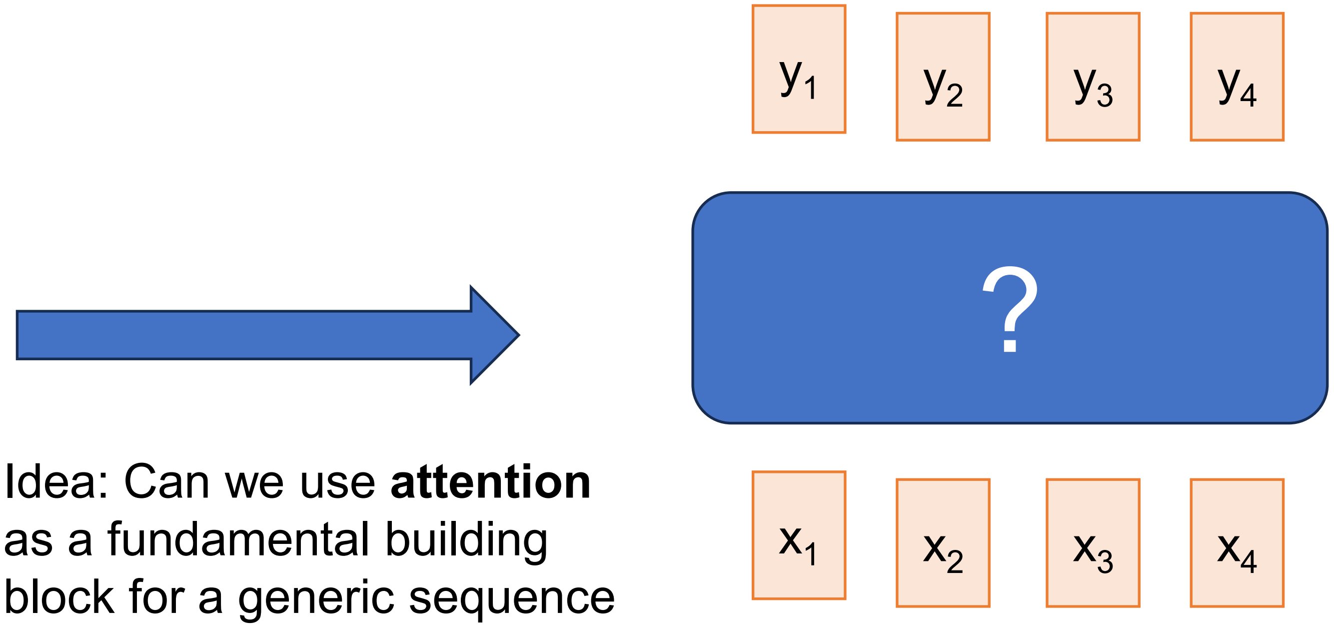
Visualize attention weights $a_{t,i}$



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

Slide credit: Justin Johnson





Idea: Can we use **attention** as a fundamental building block for a generic sequence (input) to sequence (output) layer?

Note: We just want a generic sequence-in, sequence-out model that will represent each input *contextualized* with rest of inputs, and encode meaning of entire sequence

We will progressively develop a generic mechanism using idea of attention. Don't try to map to RNN translation example!

Attention Layer

Inputs:
State vector: $\mathbf{s_i}$ (Shape: D_Q)
Hidden vectors: $\mathbf{h_i}$ (Shape: N_X x D_H)
Similarity function: f_{att}

Computation:
Similarities: e (Shape: N_X) $e_i = f_{att}(\mathbf{s_{t-1}}, \mathbf{h_i})$
Attention weights: a = softmax(e) (Shape: N_X)
Output vector: $y = \sum_i a_i \mathbf{h_i}$ (Shape: D_X)

Slide credit: Justin Johnson

Attention Layer

Inputs:
Query vectors: **Q** (Shape: N_Q x D_Q)
Input vectors: **X** (Shape: N_X x D_Q)

Make the module generic:
Sequence Input (X), Sequence Query (Q)
Output: Sequence (Weighted sum/mixture of inputs)

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

Separate concerns:
1) *Matching* (similarity) -> Key,
2) Output given weighting -> Value

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) $Y_i = \sum_j A_{i,j} V_j$

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

Slide credit: Justin Johnson

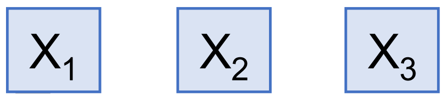
Self-Attention Layer

One query per input vector

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)

Make the module generic:
Input: Sequence (X)
Output: Sequence (Weighted sum/mixture of inputs)

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) $Y_i = \sum_j A_{i,j} V_j$



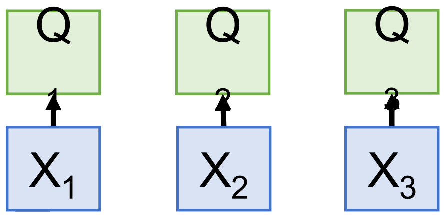
Slide credit: Justin Johnson

Self-Attention Layer

One **query** per **input vector**

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

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



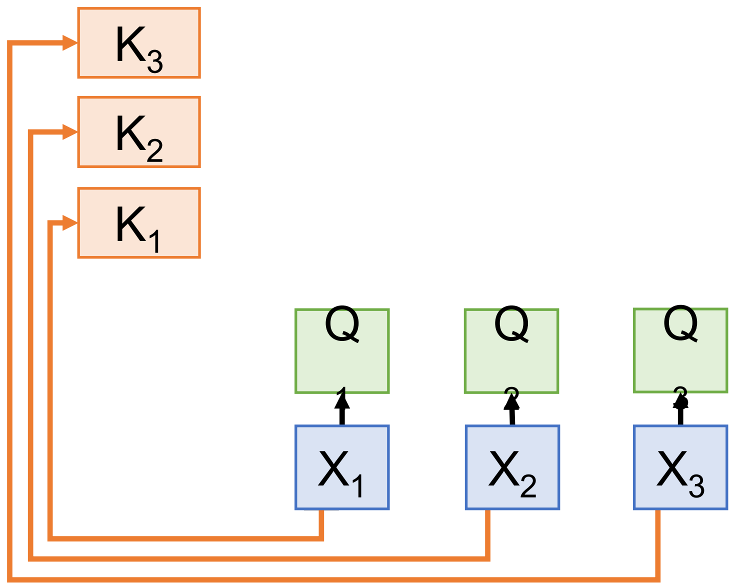
Slide credit: Justin Johnson

Self-Attention Layer

One **query** per **input vector**

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

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



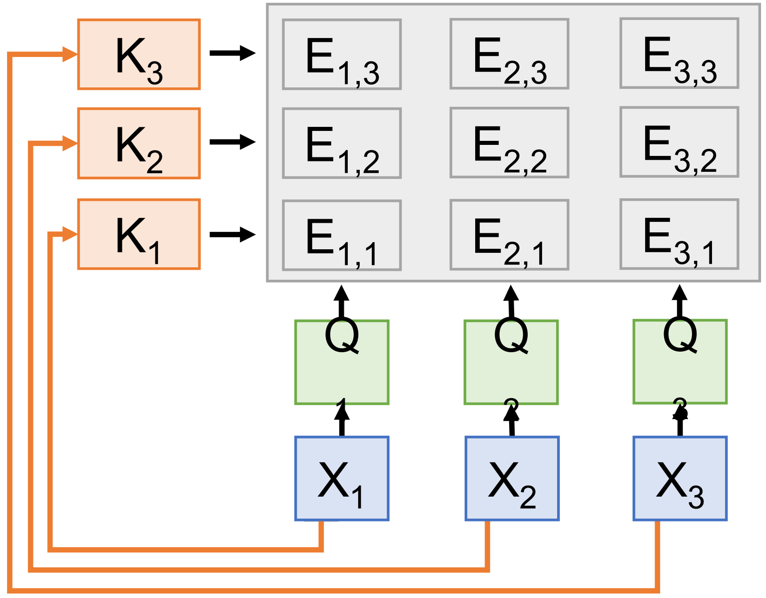
Slide credit: Justin Johnson

Self-Attention Layer

One query per input vector

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

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



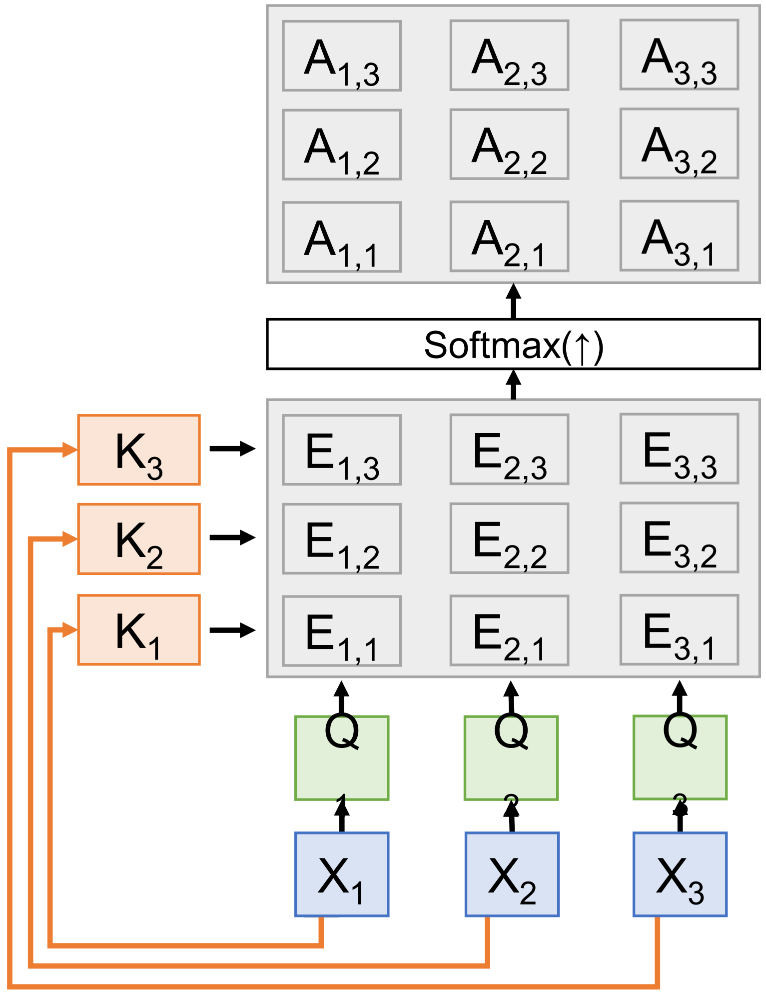
Slide credit: Justin Johnson

Self-Attention Layer

One **query** per **input vector**

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

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



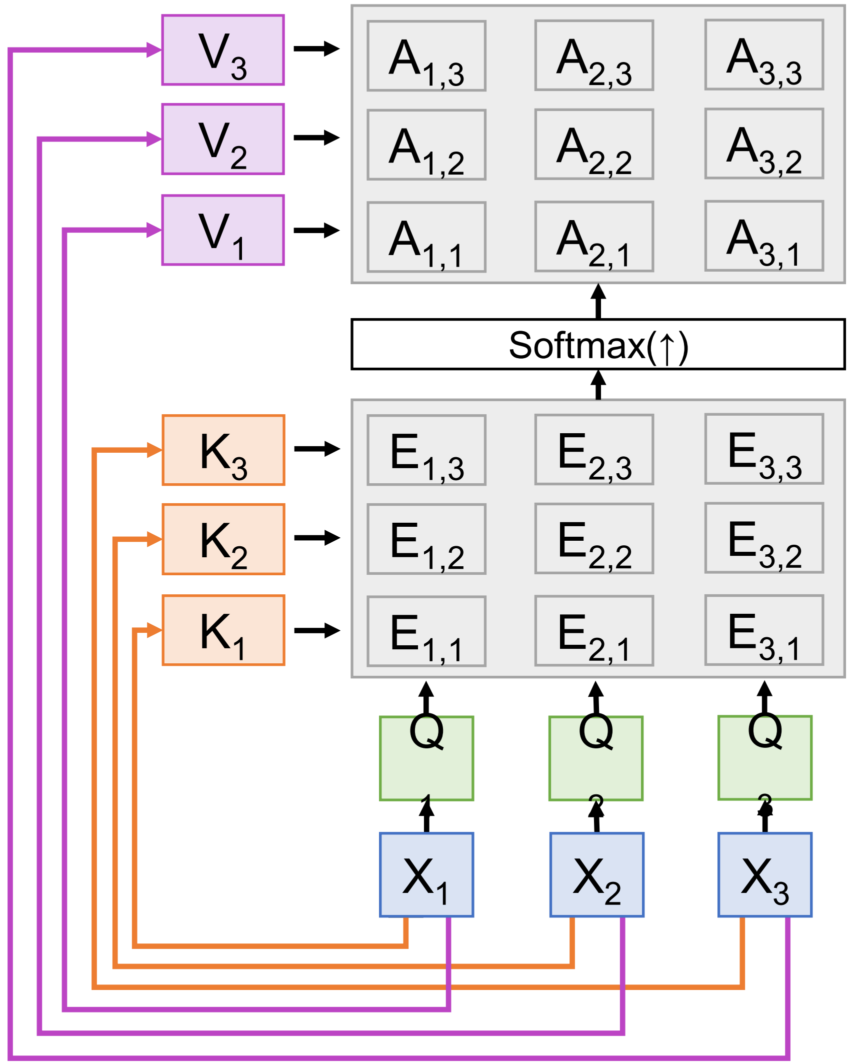
Slide credit: Justin Johnson

Self-Attention Layer

One **query** per **input vector**

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) $Y_i = \sum_j A_{i,j} V_j$



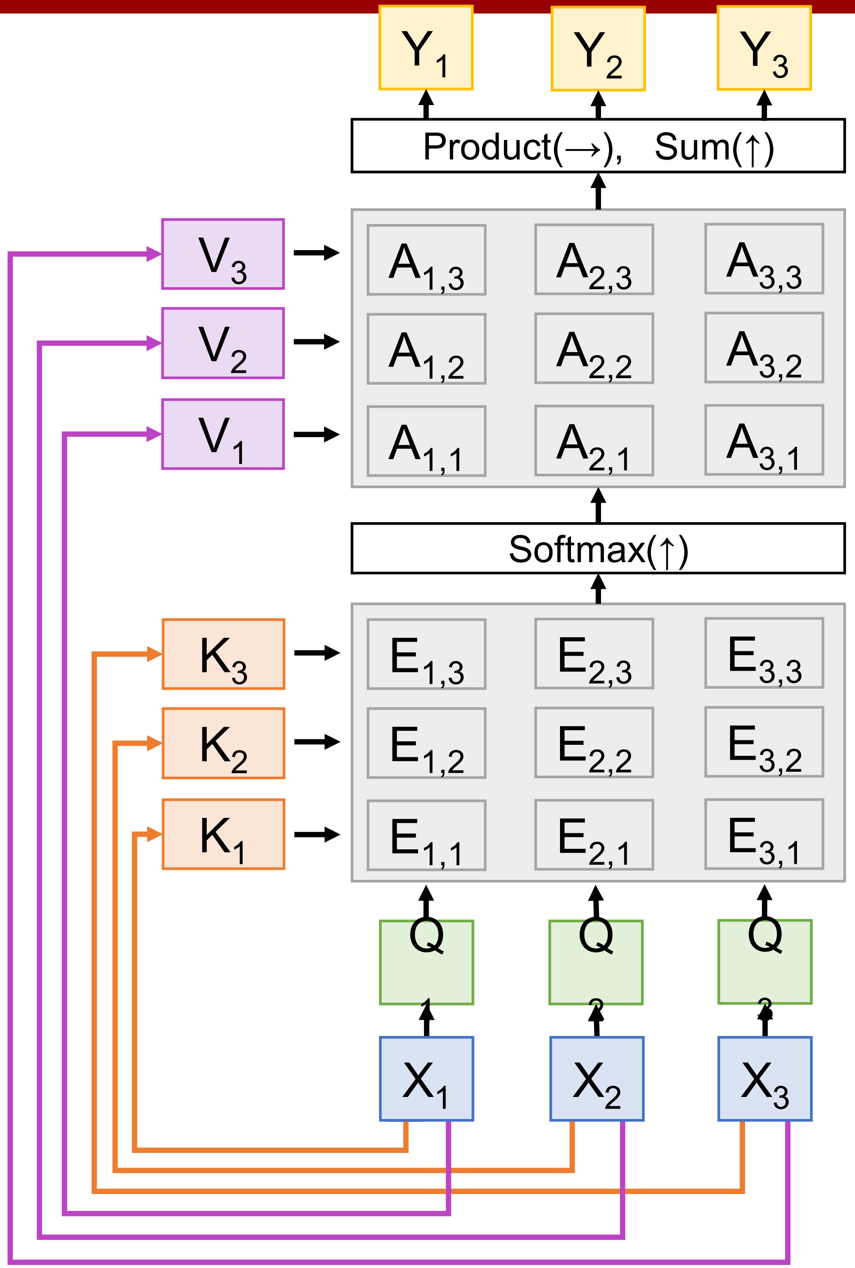
Slide credit: Justin Johnson

Self-Attention Layer

One **query** per **input vector**

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

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



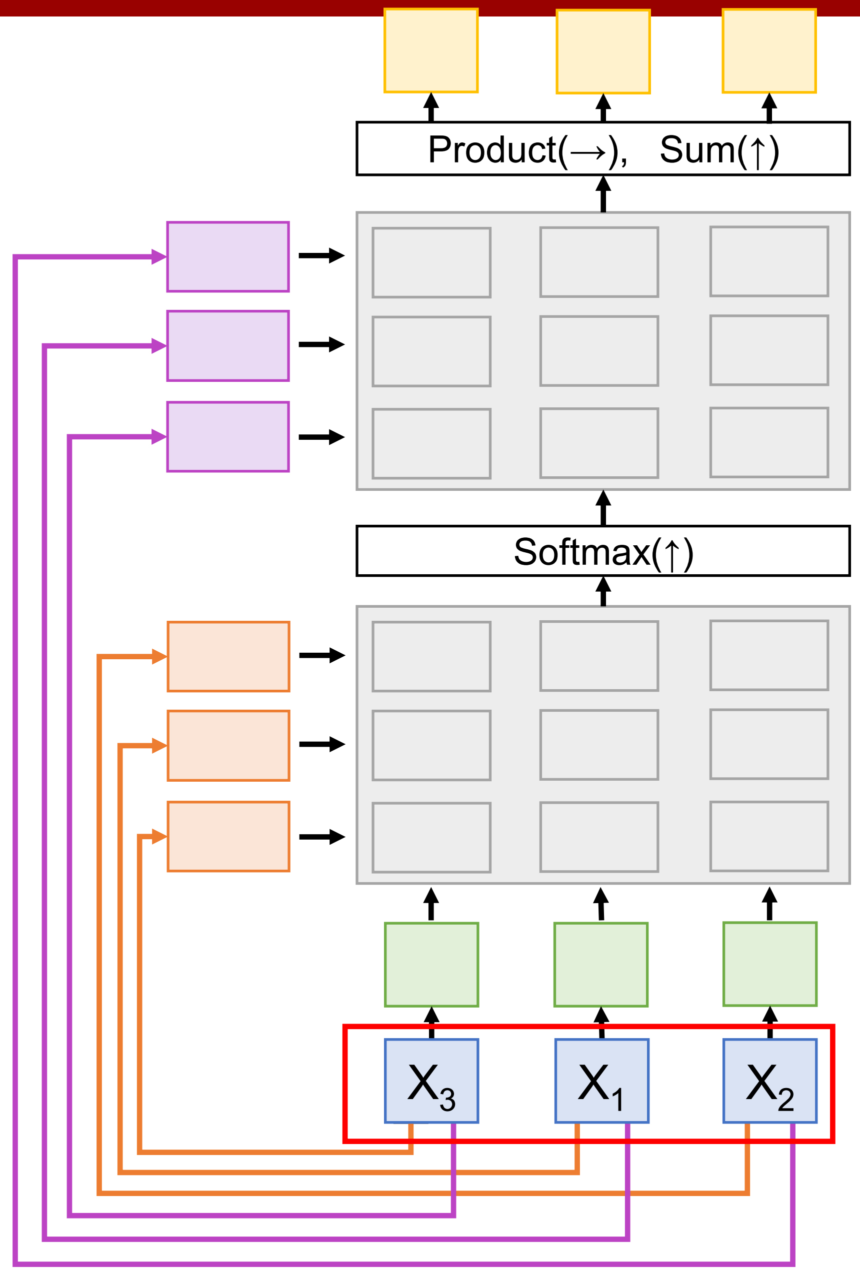
Slide credit: Justin Johnson

Self-Attention Layer

Consider **permuting** the input vectors:

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

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



Slide credit: Justin Johnson

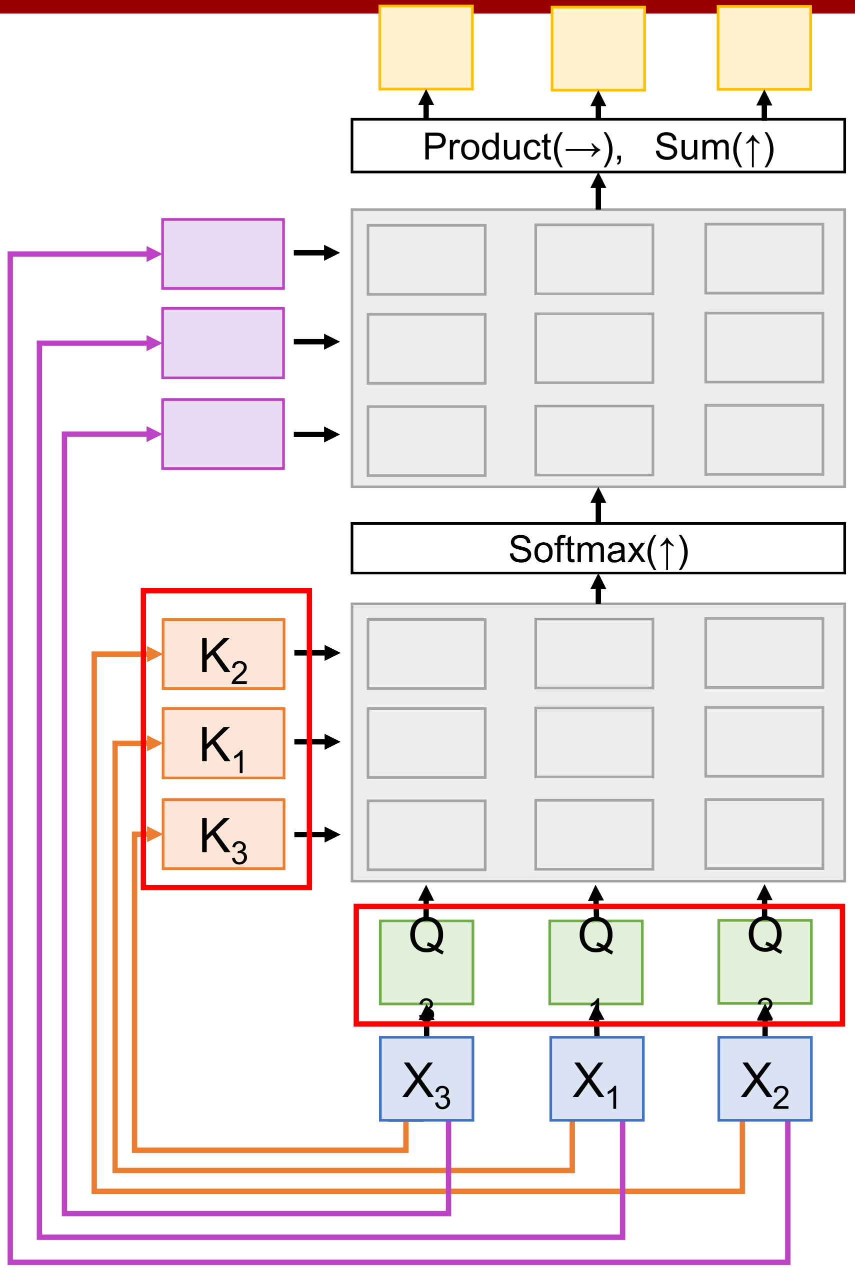
Self-Attention Layer

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:

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: $A = \text{softmax}(E, \text{dim}=1)$ (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

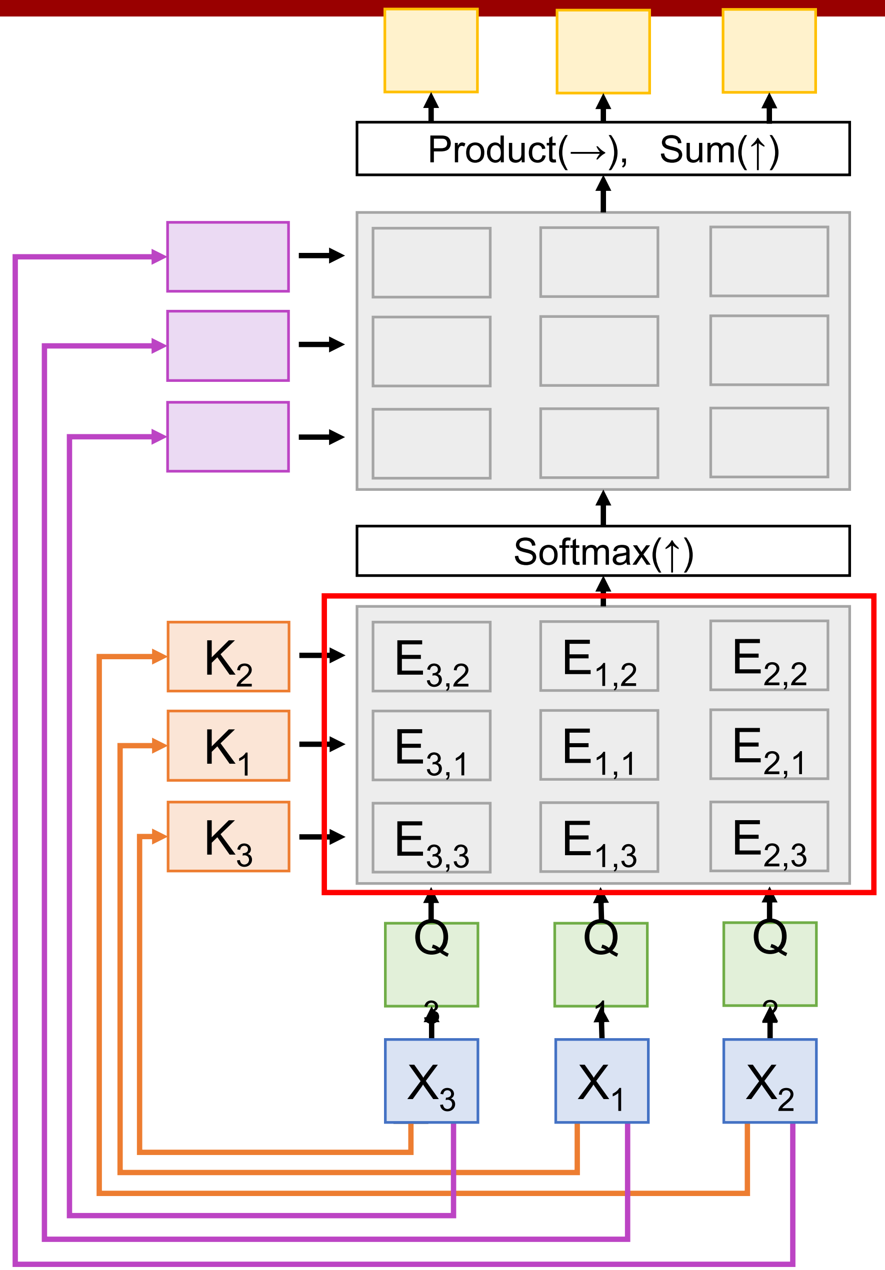
Self-Attention Layer

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)

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: $A = \text{softmax}(E, \text{dim}=1)$ (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

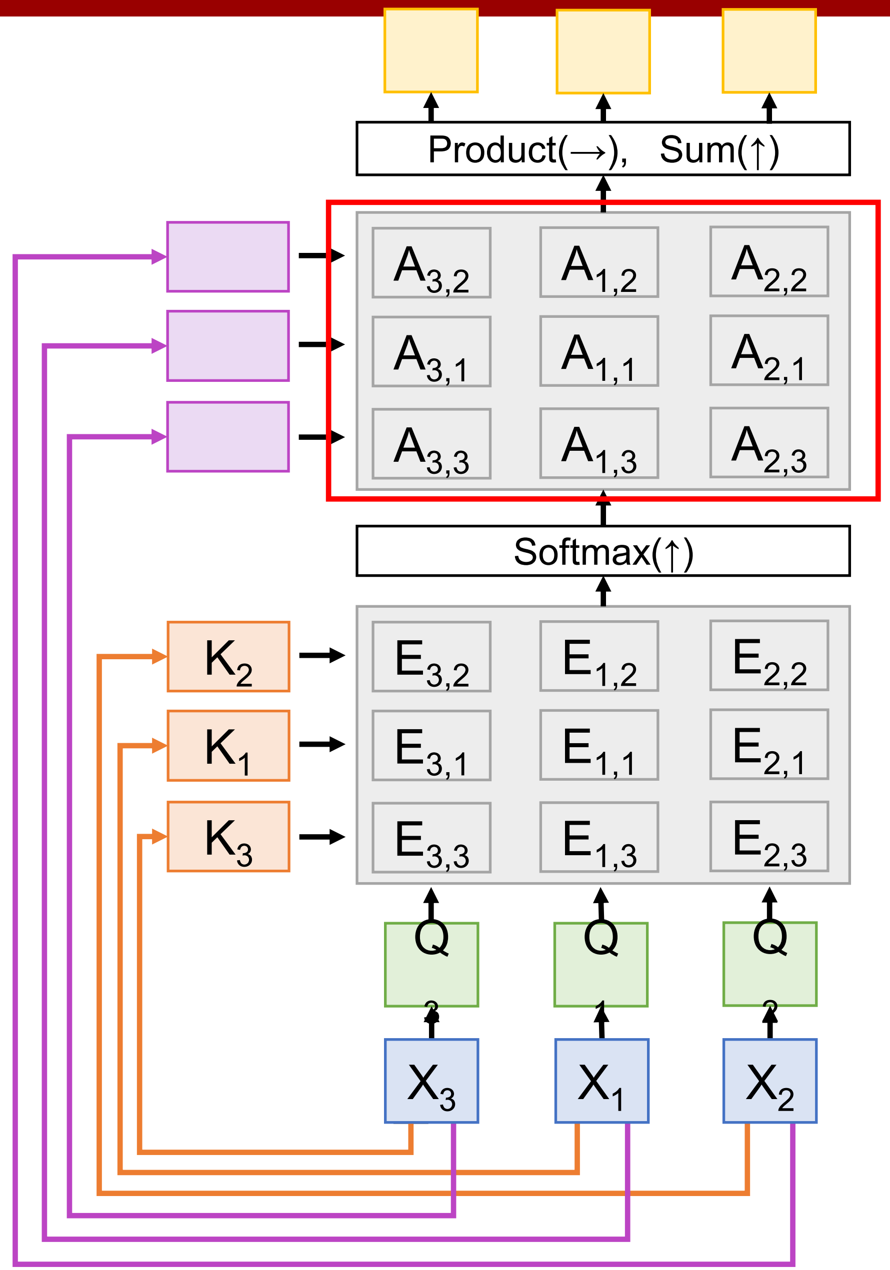
Self-Attention Layer

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)

Consider **permuting** the input vectors:

Attention weights 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) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: $Y = AV$ (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

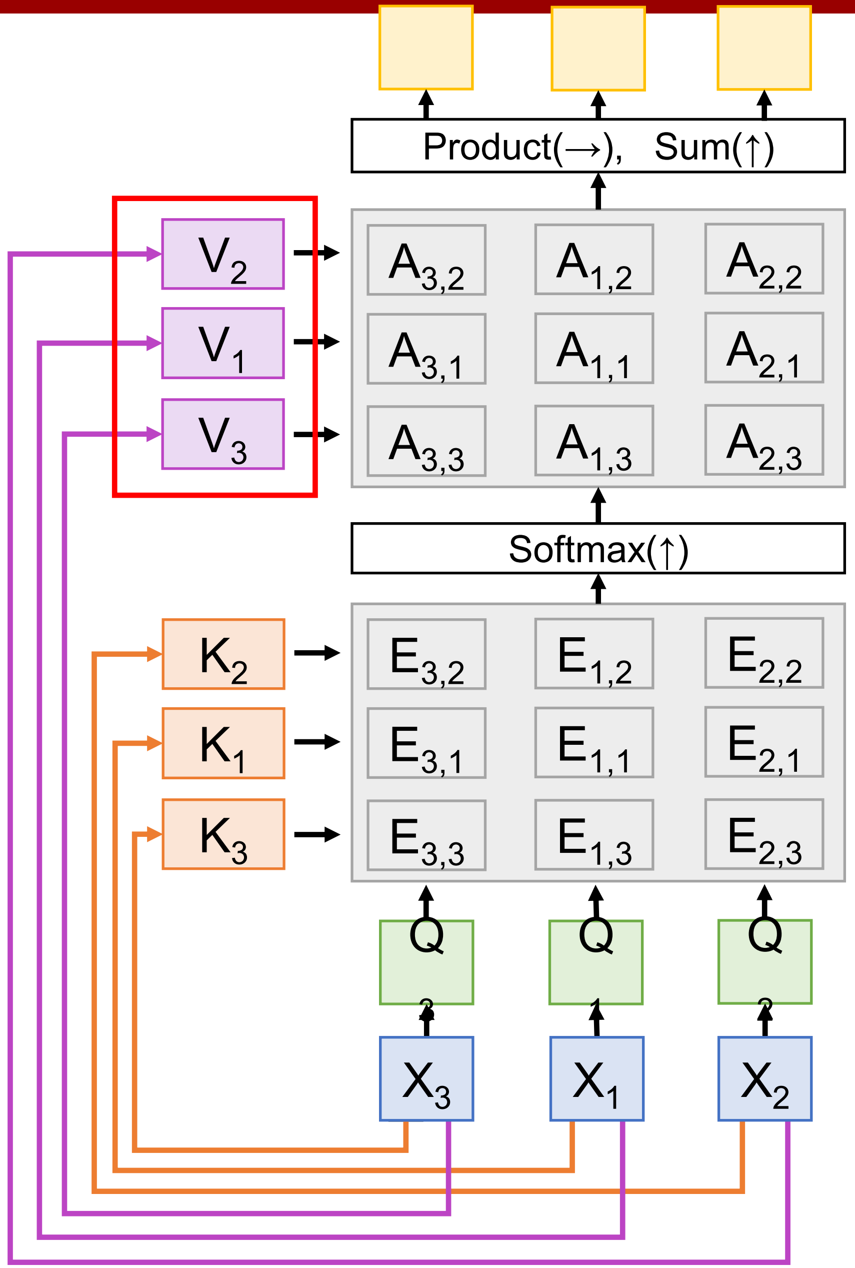
Self-Attention Layer

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 = A**V** (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

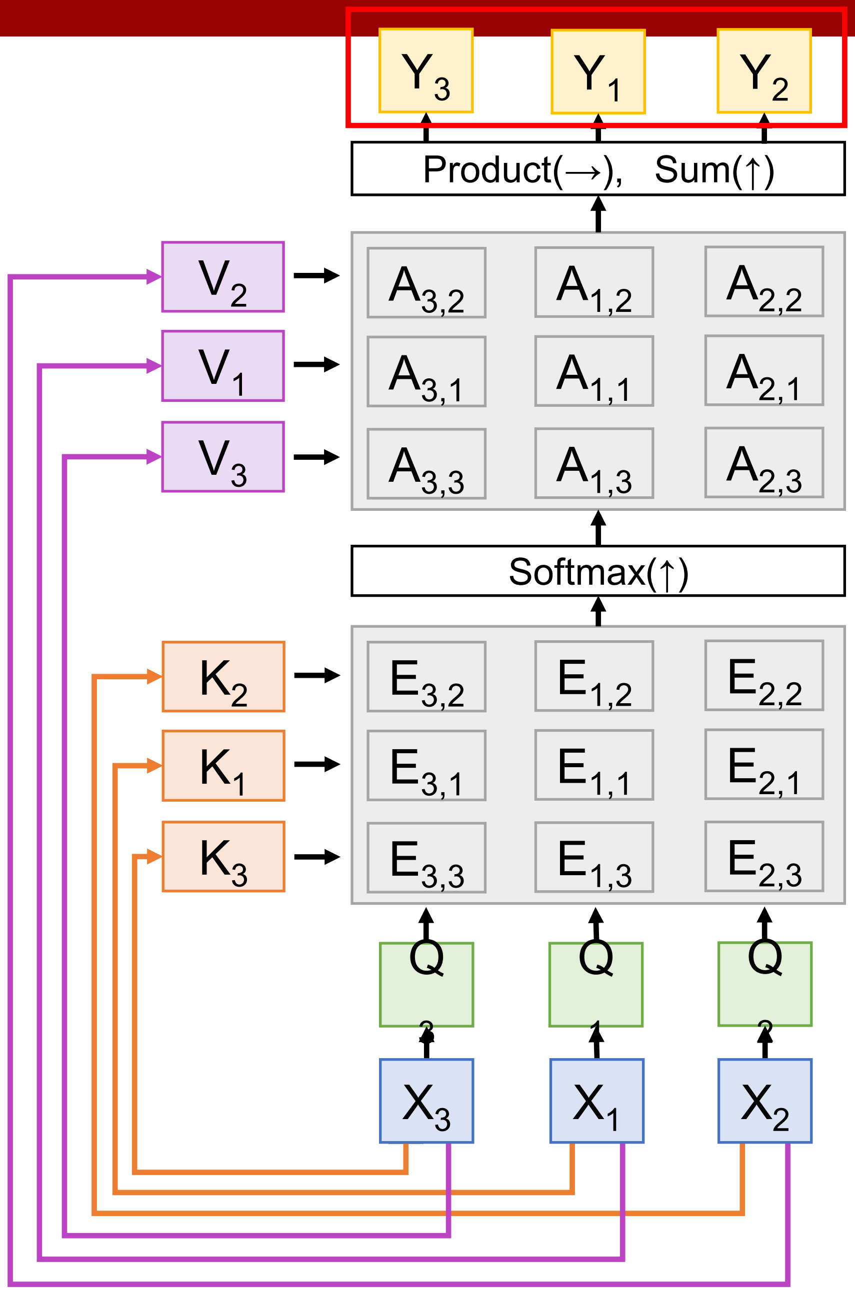
Self-Attention Layer

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:

Outputs 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) $E_{i,j} = Q_i \cdot K_j / \mathrm{sqrt}(D_Q)$
Attention weights: A = softmax(E, dim=1) (Shape: N_X x N_X)
Output vectors: Y = A**V** (Shape: N_X x D_V) $Y_i = \sum_j A_{i,j} V_j$



Slide credit: Justin Johnson

Self-Attention Layer

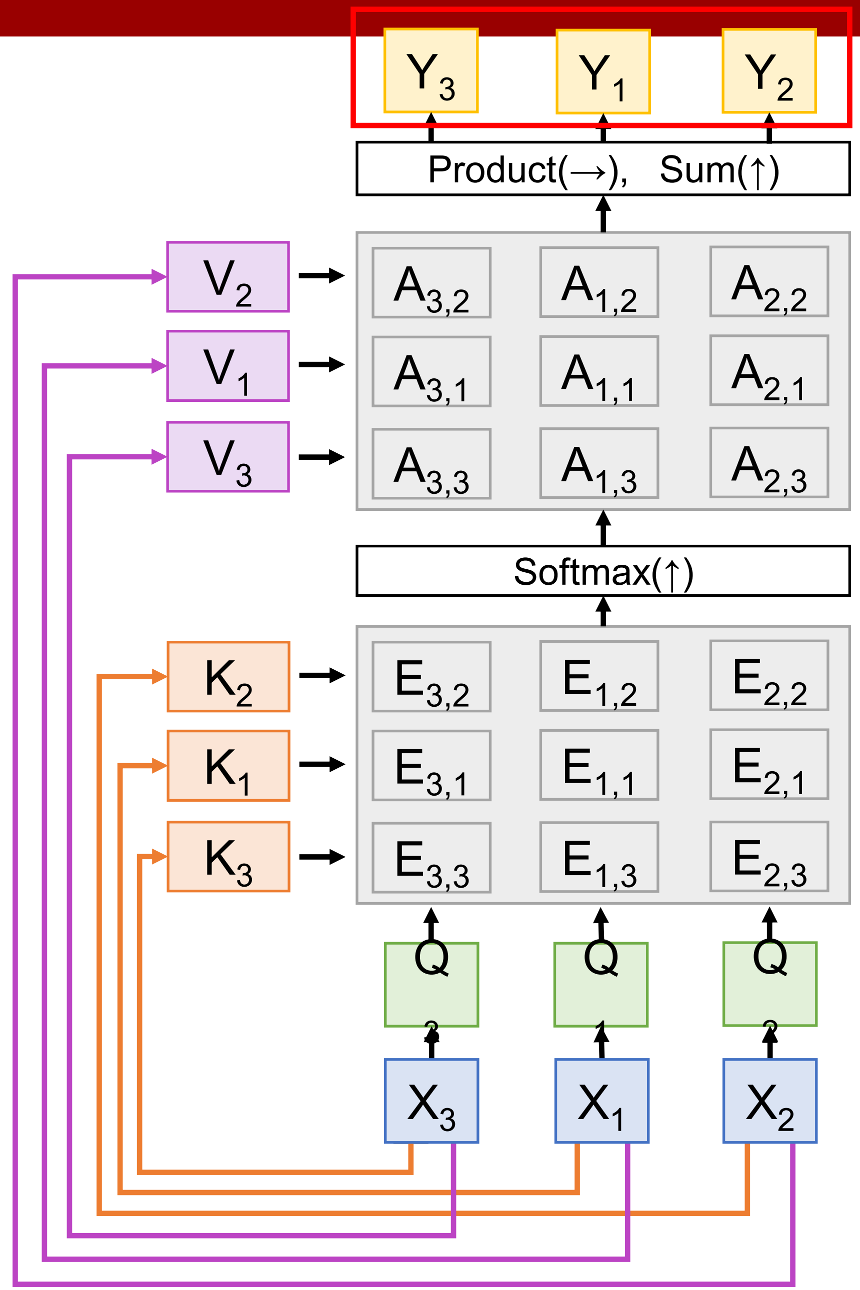
Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

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

Consider **permuting** the input vectors:

Outputs will be the same, but permuted

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



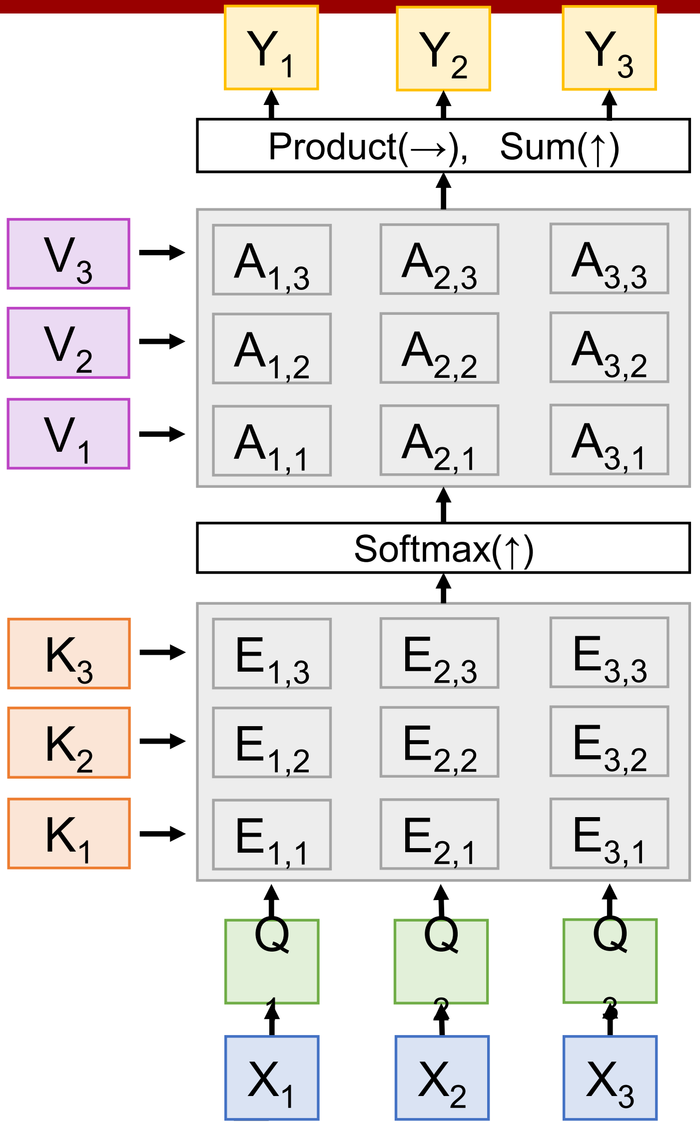
Slide credit: Justin Johnson

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!

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) $Y_i = \sum_j A_{i,j} V_j$



Self-Attention Layer

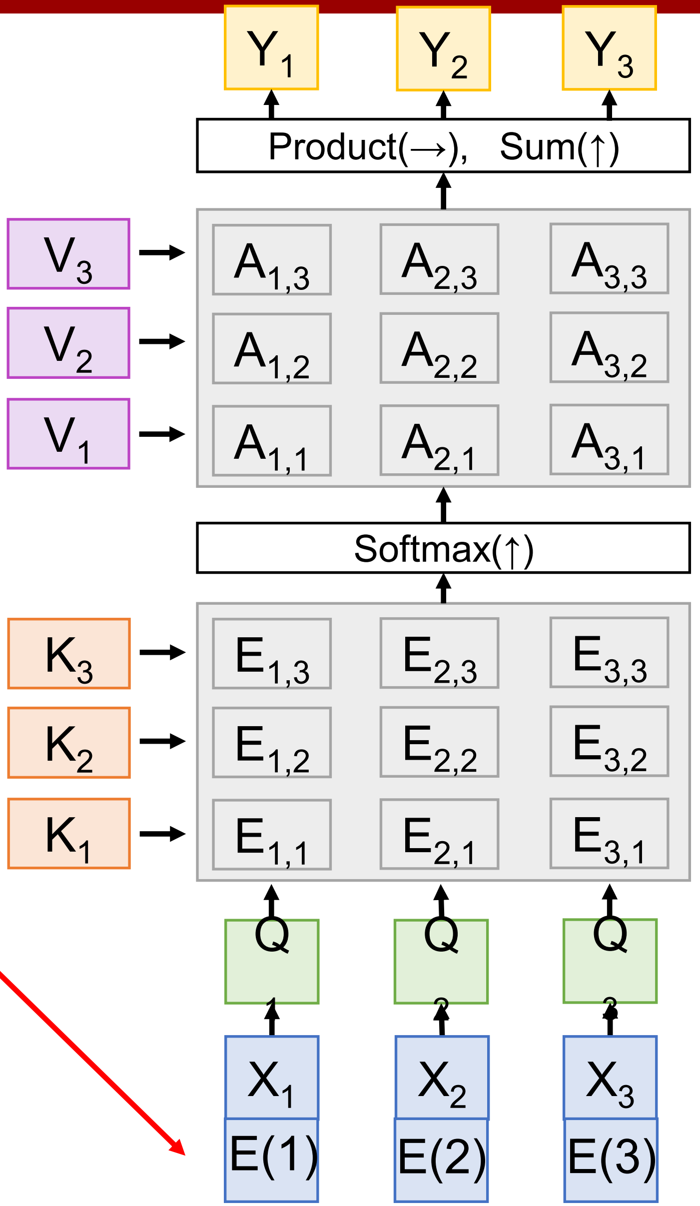
Inputs:
Input vectors: **X** (Shape: $N_X \times D_X$)
Key matrix: **W_K** (Shape: $D_X \times D_Q$)
Value matrix: W_V (Shape: $D_X \times D_V$)
Query matrix: **W_Q** (Shape: $D_X \times D_Q$)

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: $N_X \times N_X$) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

Self attention doesn't "know" the order of the vectors it is processing!

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

E can be learned lookup table, or fixed function



Slide credit: Justin Johnson

Summary

- We have made a generic sequence-in to sequence-out layer
  - This is what we want for language processing!
  - Each output is a contextualized representation of the corresponding input word
  - Vector for stop word can be treated as representation of entire sentence (e.g. project its output to classifier and add loss)

- Unlike RNNs/LSTMs, it processes all inputs (e.g. entire sentence) **at once**
  - **Highly parallelizable**
  - **-> SCALE! -> Reduction of loss -> Magic**

- Next time: Entire transformer architecture that combines this new layer with other layers/concepts we know about (fully-connected, normalization, residual/skip connections)

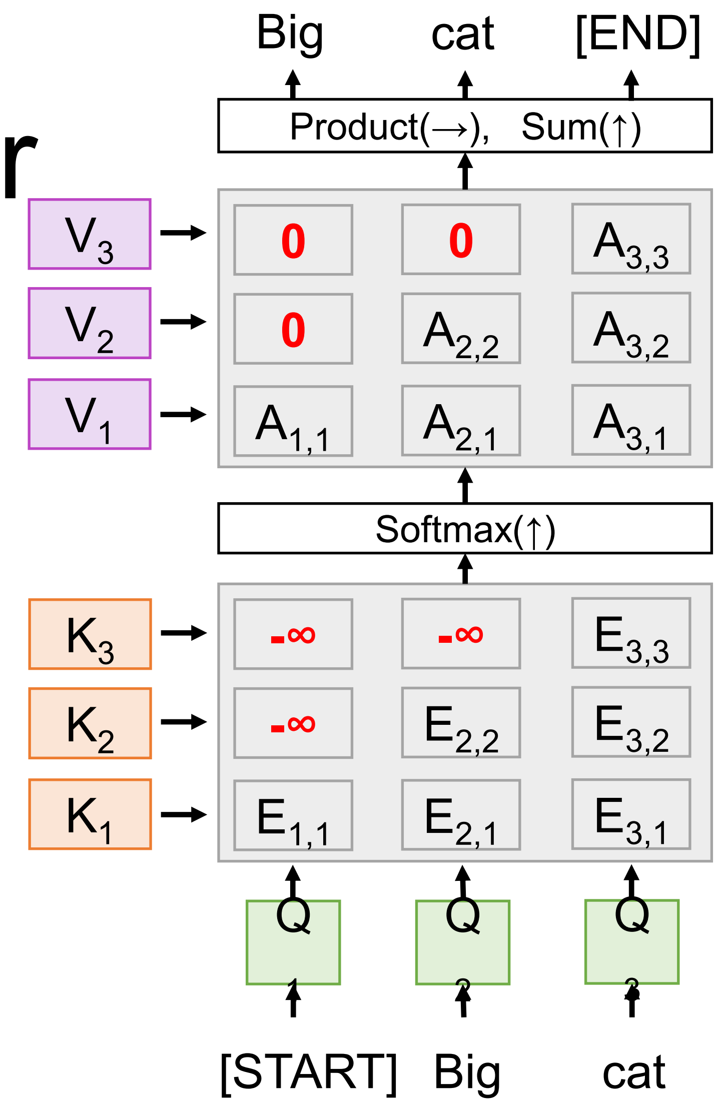
Masked Self-Attention Layer

Inputs:
Input vectors: **X** (Shape: N_X x D_X)
Key matrix: $\mathbf{W_K}$ (Shape: D_X x D_Q)
Value matrix: $\mathbf{W_V}$ (Shape: D_X x D_V)
Query matrix: $\mathbf{W_Q}$ (Shape: D_X x D_Q)

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

Don't let vectors "look ahead" in the sequence

Used for language modeling (predict next word)



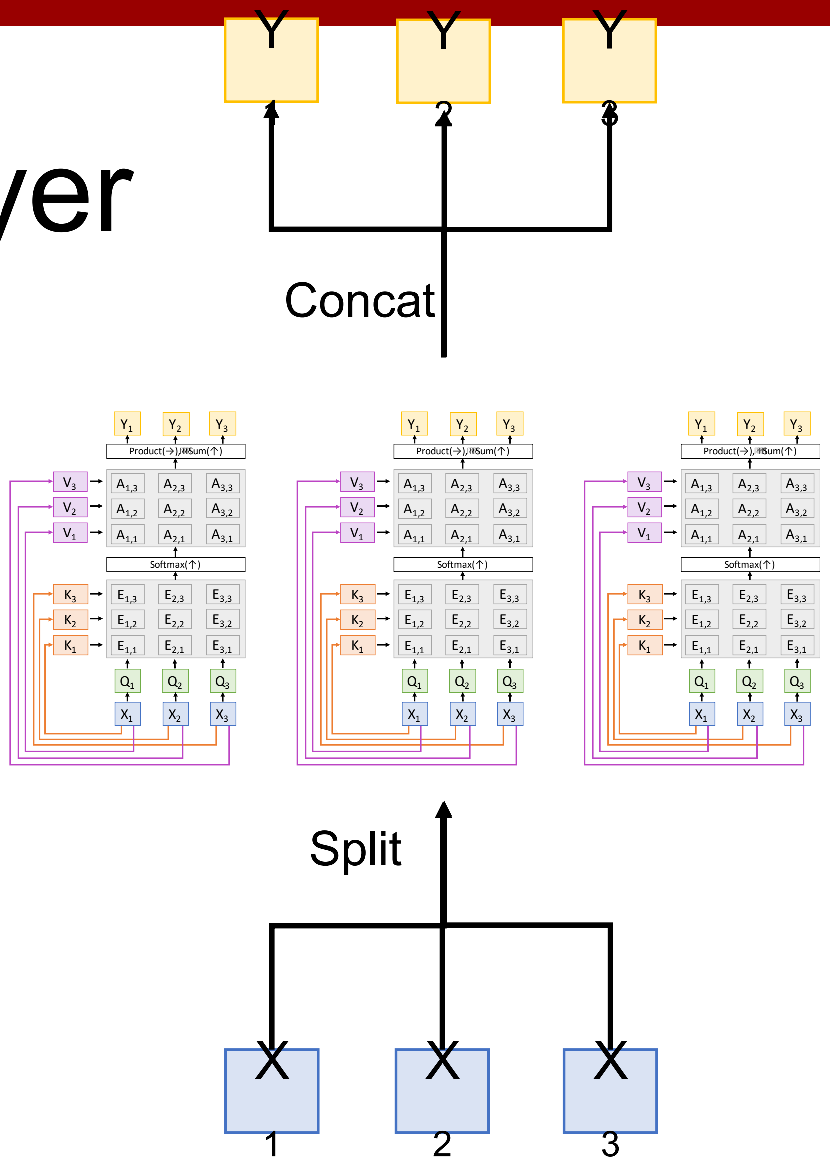
Slide credit: Justin Johnson

Multihead 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: $N_X \times N_X$) $E_{i,j} = Q_i \cdot K_j / \text{sqrt}(D_Q)$
Attention weights: $A = \text{softmax}(E, \text{dim}=1)$ (Shape: $N_X \times N_X$)
Output vectors: $Y = AV$ (Shape: $N_X \times D_V$) $Y_i = \sum_j A_{i,j} V_j$

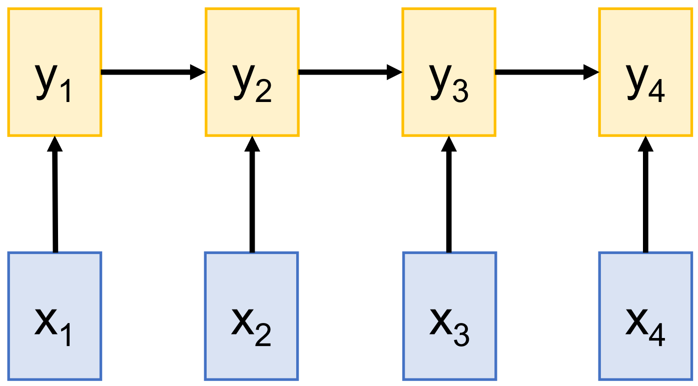
Use H independent "Attention Heads" in parallel



Slide credit: Justin Johnson

Three Ways of Processing Sequences

Recurrent Neural Network



Works on **Ordered Sequences**

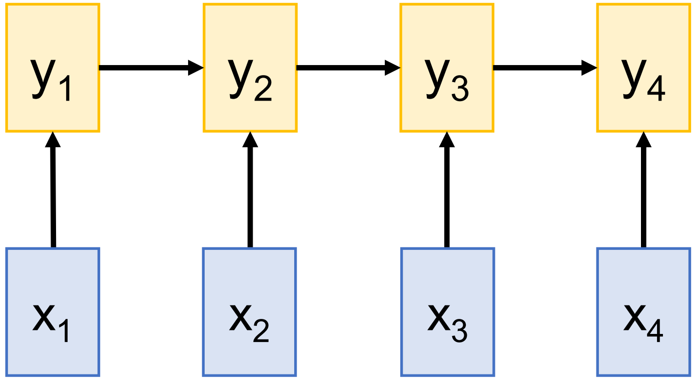
(+) Good at long sequences: After one RNN layer, h_T ”sees” the whole sequence

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

Slide credit: Justin Johnson

Three Ways of Processing Sequences

Recurrent Neural Network

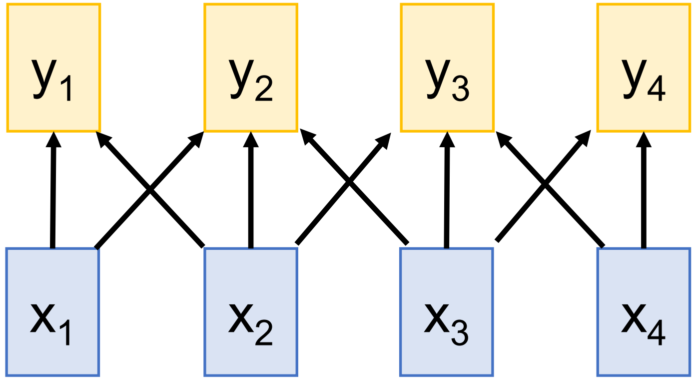


Works on **Ordered Sequences**

(+) 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**

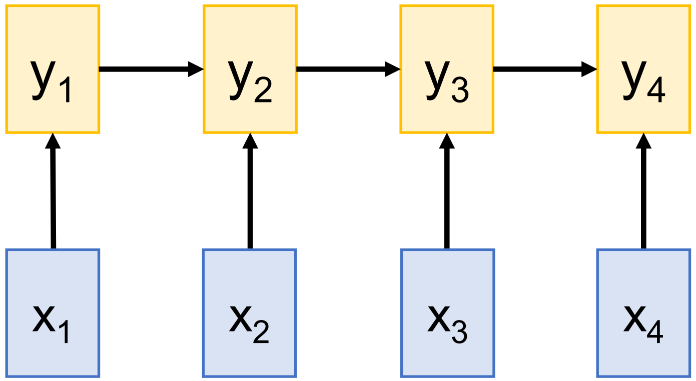
(-) Bad at long sequences: Need to stack many conv layers for outputs to "see" the whole sequence

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

Slide credit: Justin Johnson

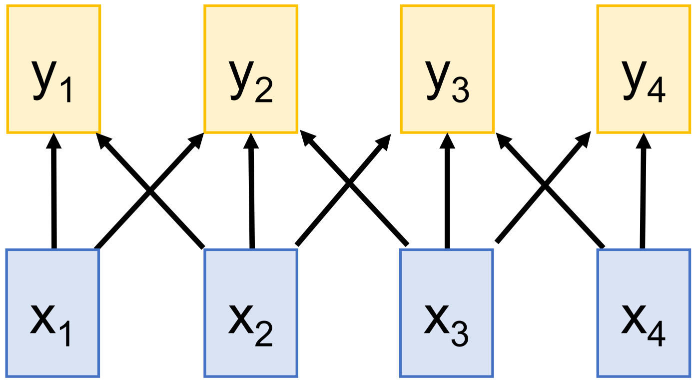
Three Ways of Processing Sequences

Recurrent Neural Network



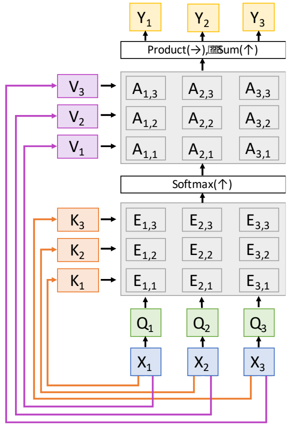
Works on **Ordered Sequences**
(+) Good at long sequences: After one RNN layer, h_T "sees" the whole sequence
(-) Not parallelizable: need to compute hidden states sequentially

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

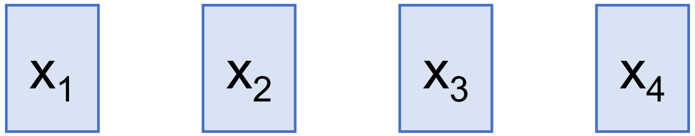
Works on **Ordered Sequences**
(+) Good at long sequences: After one RNN layer, h_T "sees" the whole sequence
(-) Not parallelizable: need to compute hidden states sequentially

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

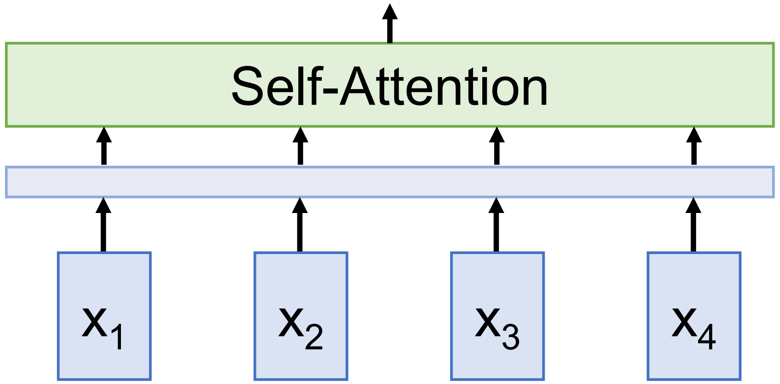


Vaswani et al, “Attention is all you need”, NeurIPS 2017

Slide credit: Justin Johnson

The Transformer

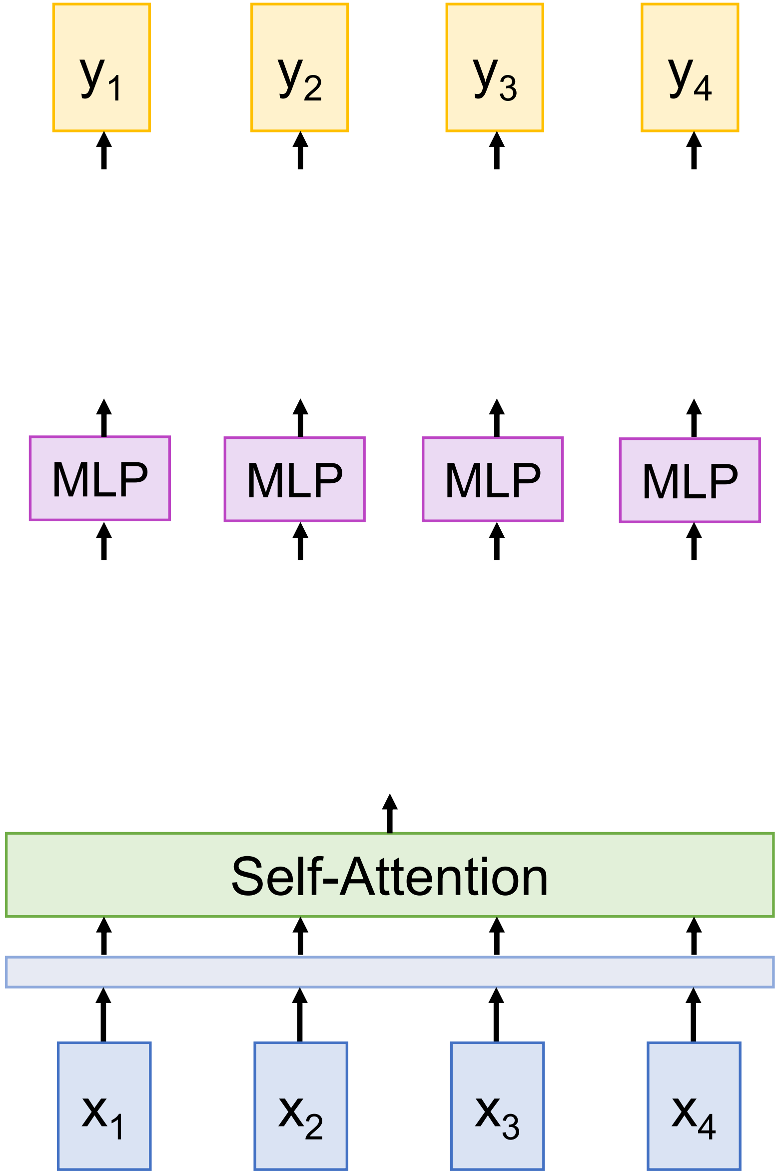
All vectors interact with each other



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

Slide credit: Justin Johnson

The Transformer



MLP independently on each vector

All vectors interact with each other

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

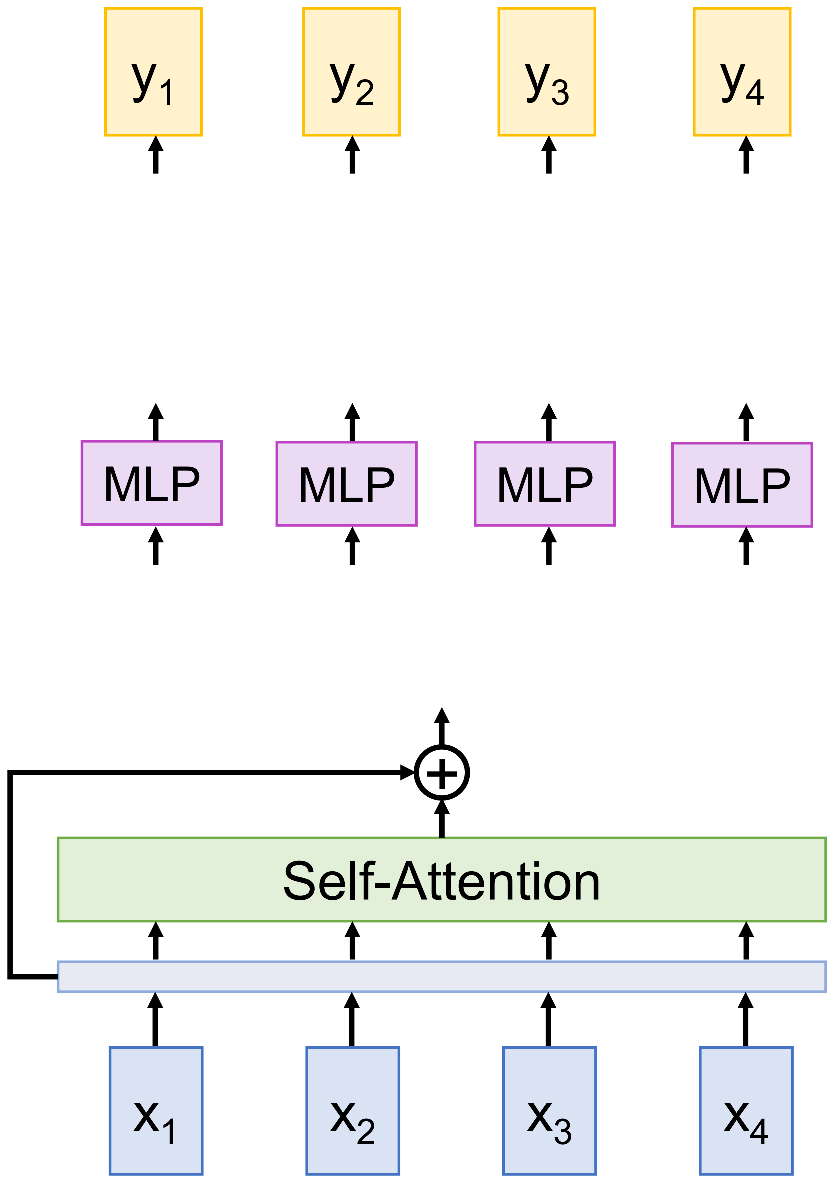
Slide credit: Justin Johnson

The Transformer

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

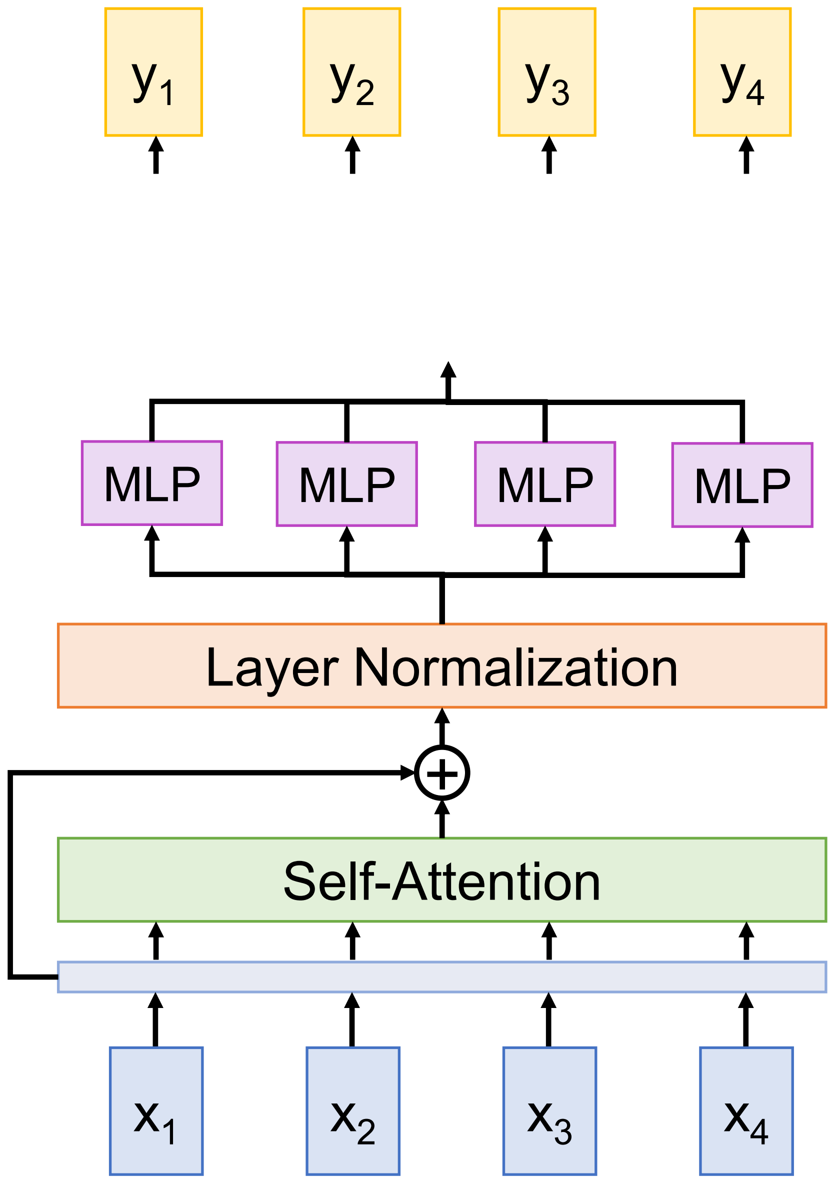
Recall **Layer Normalization**:
Given $h_1, \ldots, h_N$ (Shape: D)
scale: γ (Shape: D)
shift: β (Shape: D)
$\mu_i = (1/D)\sum_j h_{i,j}$ (scalar)
$\sigma_i = (\sum_j (h_{i,j} - \mu_i)^2)^{1/2}$ (scalar)
$z_i = (h_i - \mu_i) / \sigma_i$
$y_i = \gamma * z_i + \beta$

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

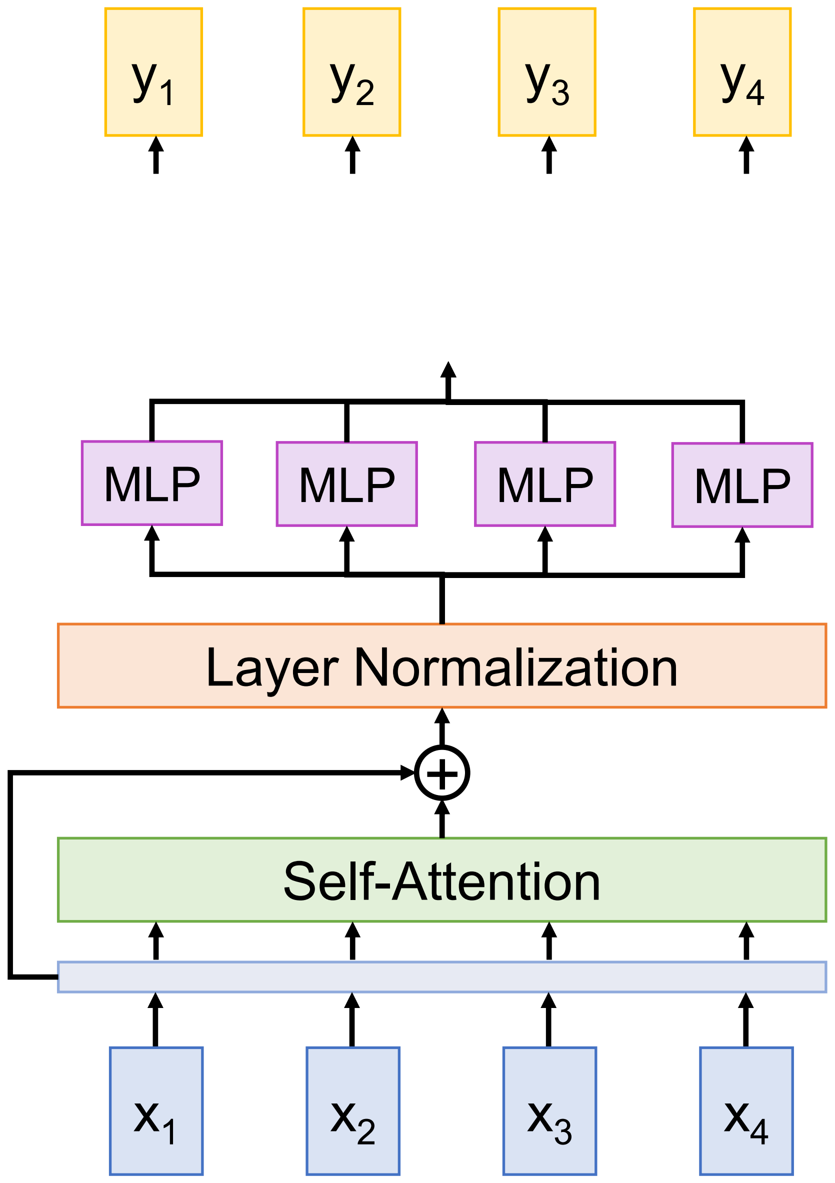
Slide credit: Justin Johnson

The Transformer

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

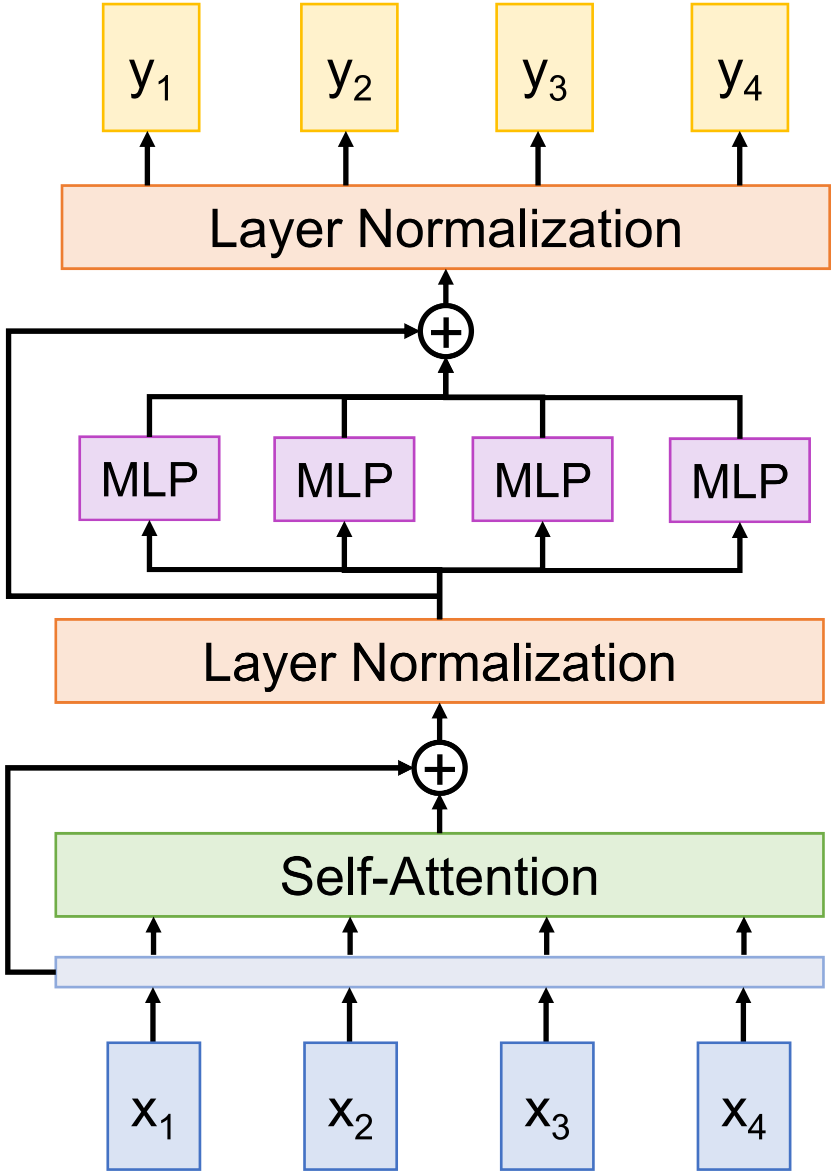
The Transformer

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

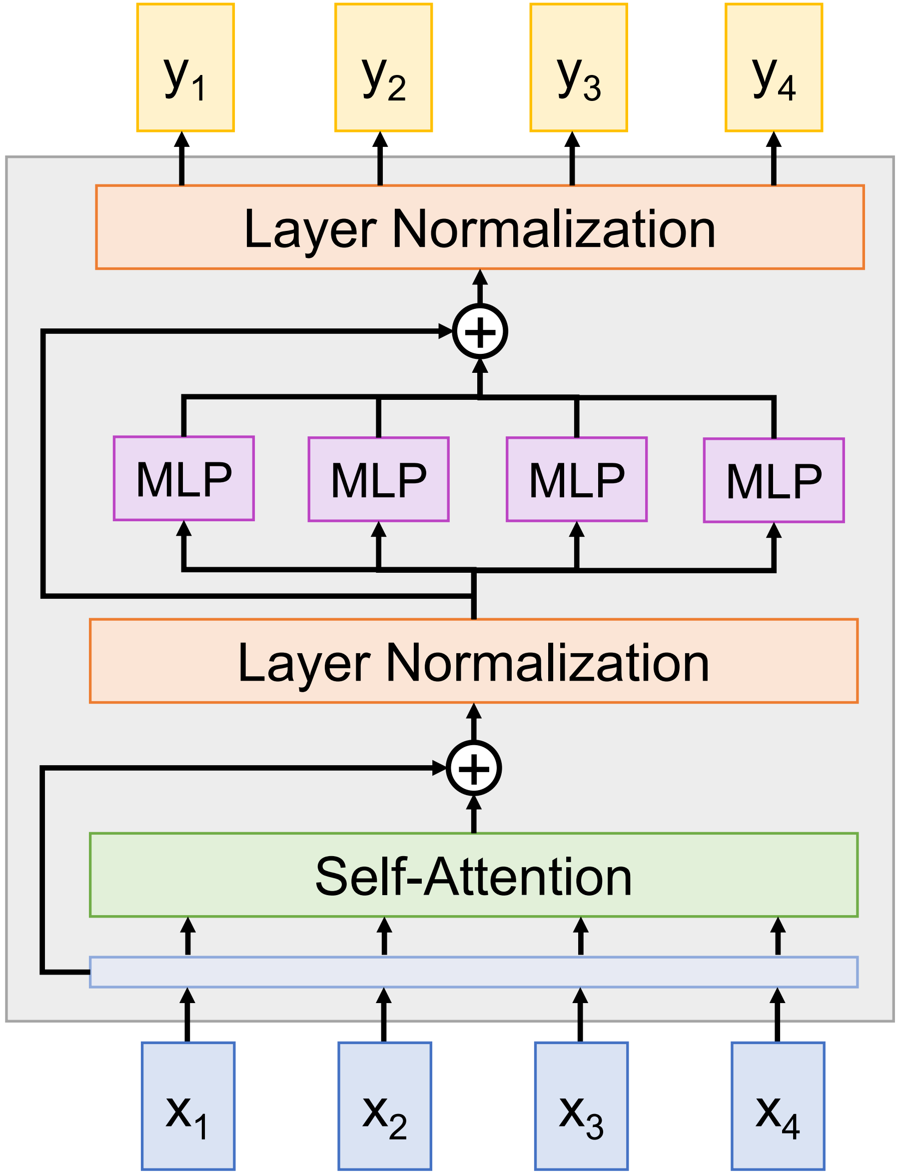
The Transformer

<u>Transformer Block</u>:
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



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

Slide credit: Justin Johnson

The Transformer

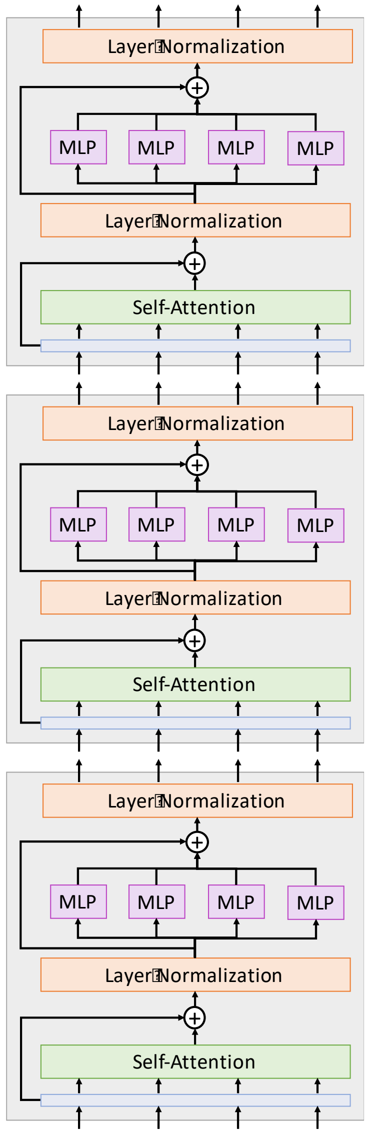
<u>Transformer Block</u>:
Input: Set of vectors x
Output: Set of vectors y

Self-attention is the only interaction between vectors!

Layer norm and MLP work independently per vector

Highly scalable, highly parallelizable

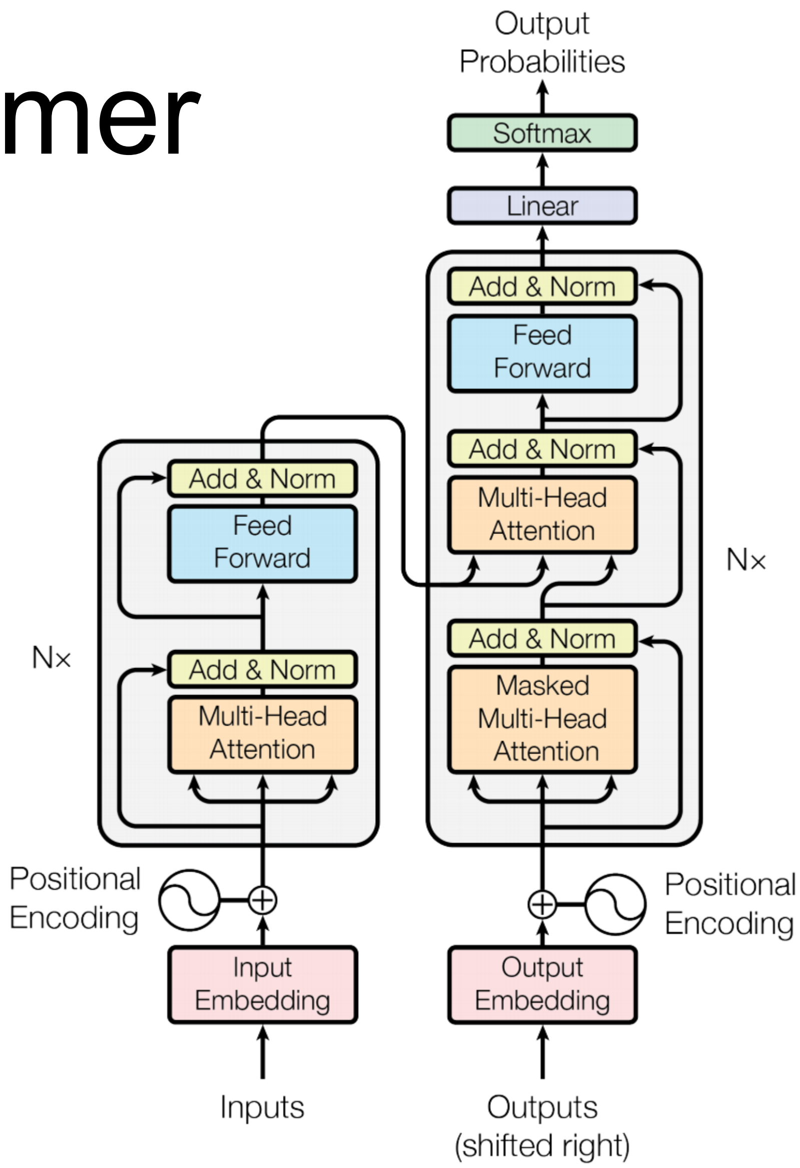
A **Transformer** is a sequence of transformer blocks



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?

ViLBERT: A Visolinguistic Transformer

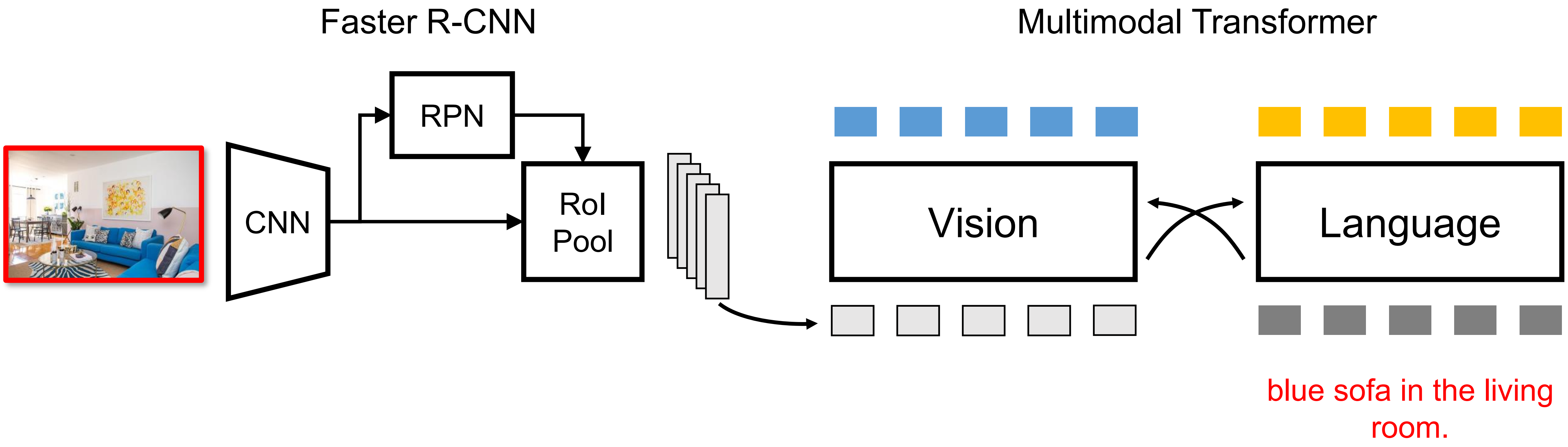
pop artist performs at the festival in a city.

a worker helps to clear the debris.

blue sofa in the living room.

Image and captions from: Sharma, Piyush, et al. "Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning." ACL. 2018.

ViLBERT: A Visolinguistic Transformer



Lu et al "Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks." *NeurIPS*. 2019.
Ren et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *NeurIPS*. 2015.

Jean Maillard

Jean Maillard is a Research Scientist on the Language And Translation Technologies Team (LATTE) at Facebook AI. His research interests within NLP include word- and sentence-level semantics, structured prediction, and low-resource languages. Prior to joining Facebook in 2019, he

Module 3 Lesson 12 (M3L12) on Dropbox
https://www.dropbox.com/sh/iviro188gq0b4vs/AADdHxX_Uy1TkpF_yvIzX0nPa?dl=0

- **Recall:** language models estimate the probability of sequences of words:

$$p(\boldsymbol{s}) = p(w_1, w_2, \ldots, w_n)$$

- ***Masked language modeling*** is a related ***pre-training task*** – an auxiliary task, different from the final task we're really interested in, but which can help us achieve better performance by finding good initial parameters for the model.

- By pre-training on masked language modeling before training on our final task, it is usually possible to obtain higher performance than by simply training on the final task.

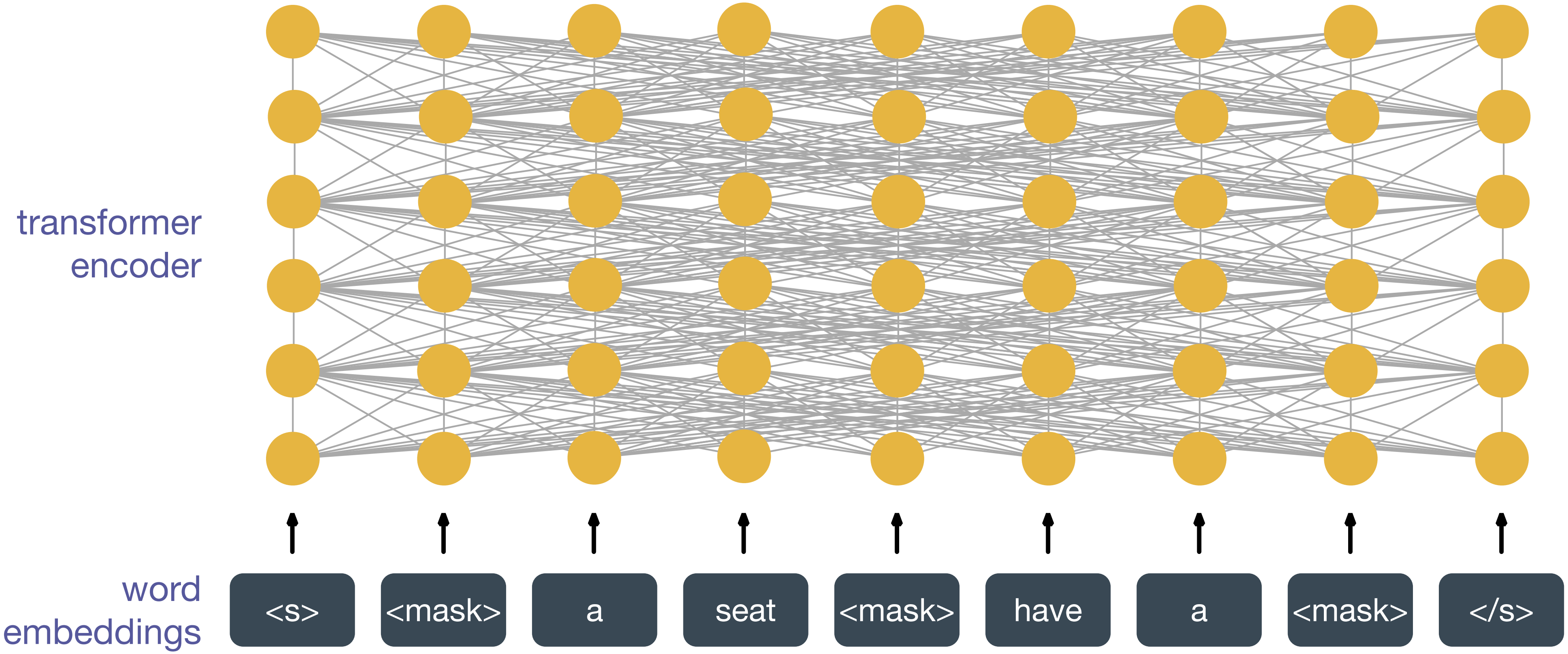
take | a | seat | , | have | a | drink

Masked Language Models

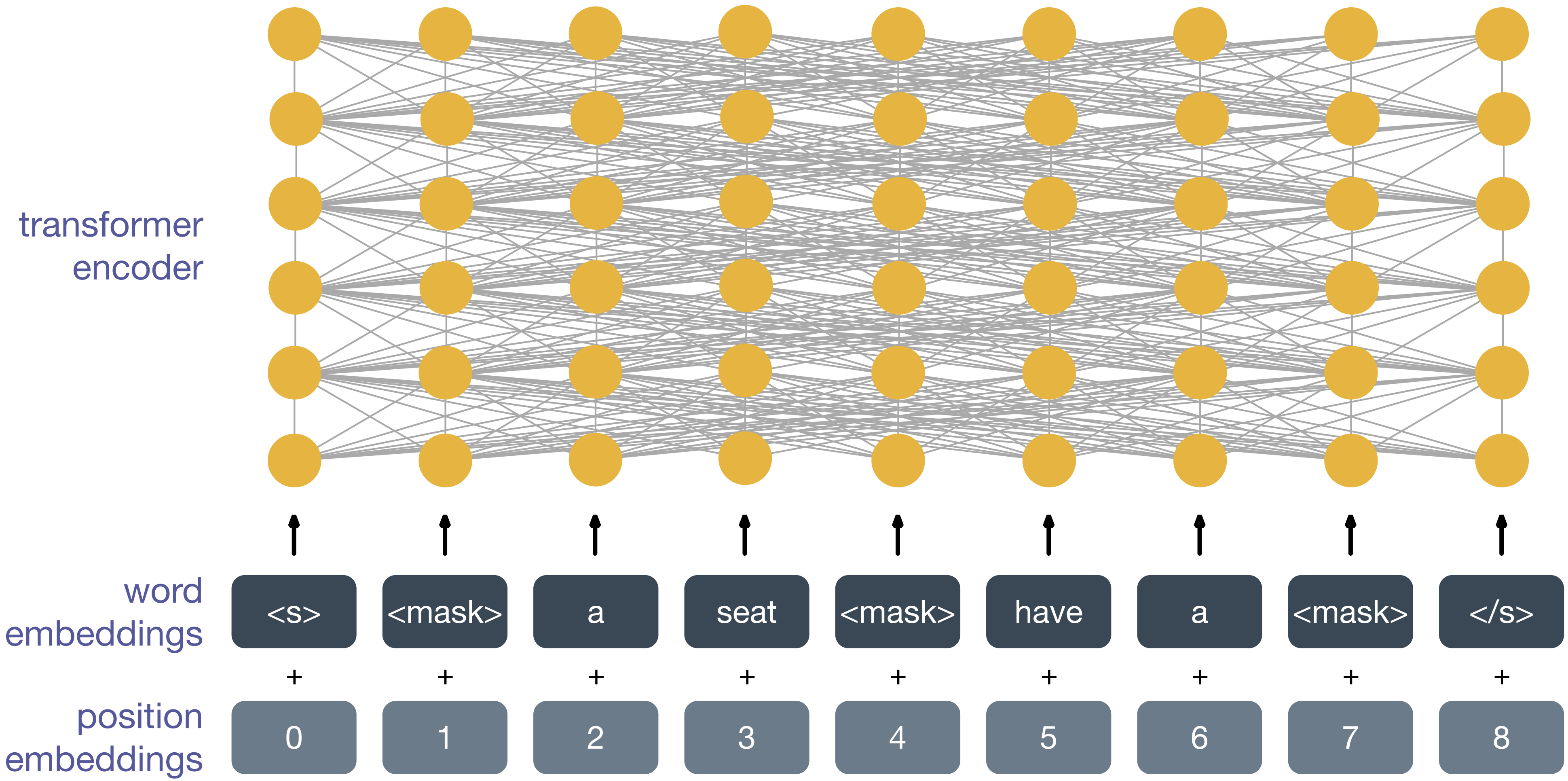
FACEBOOK AI Georgia Tech

<s> | <mask> | a | seat | <mask> | have | a | <mask> | </s>

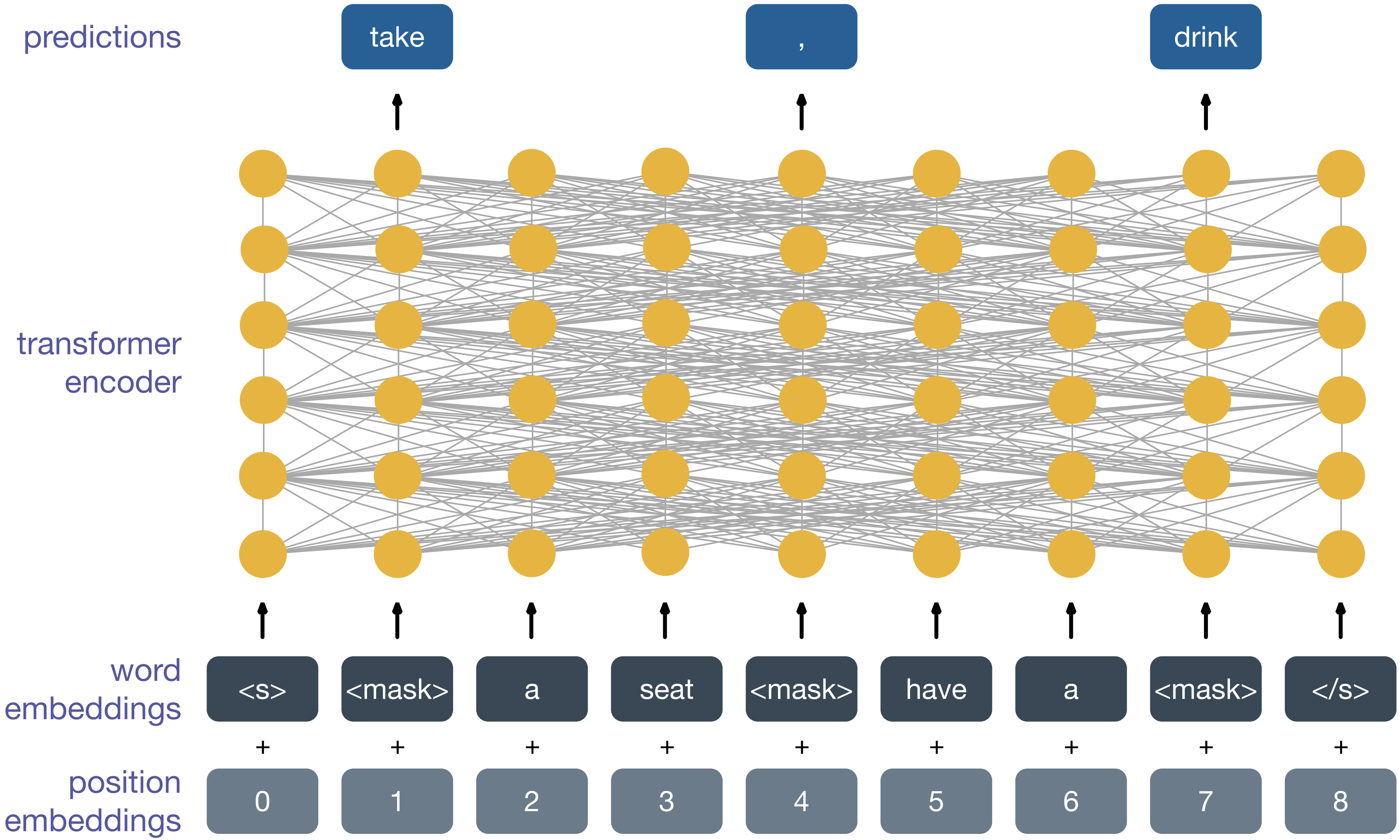
Masked Language Models



Masked Language Models

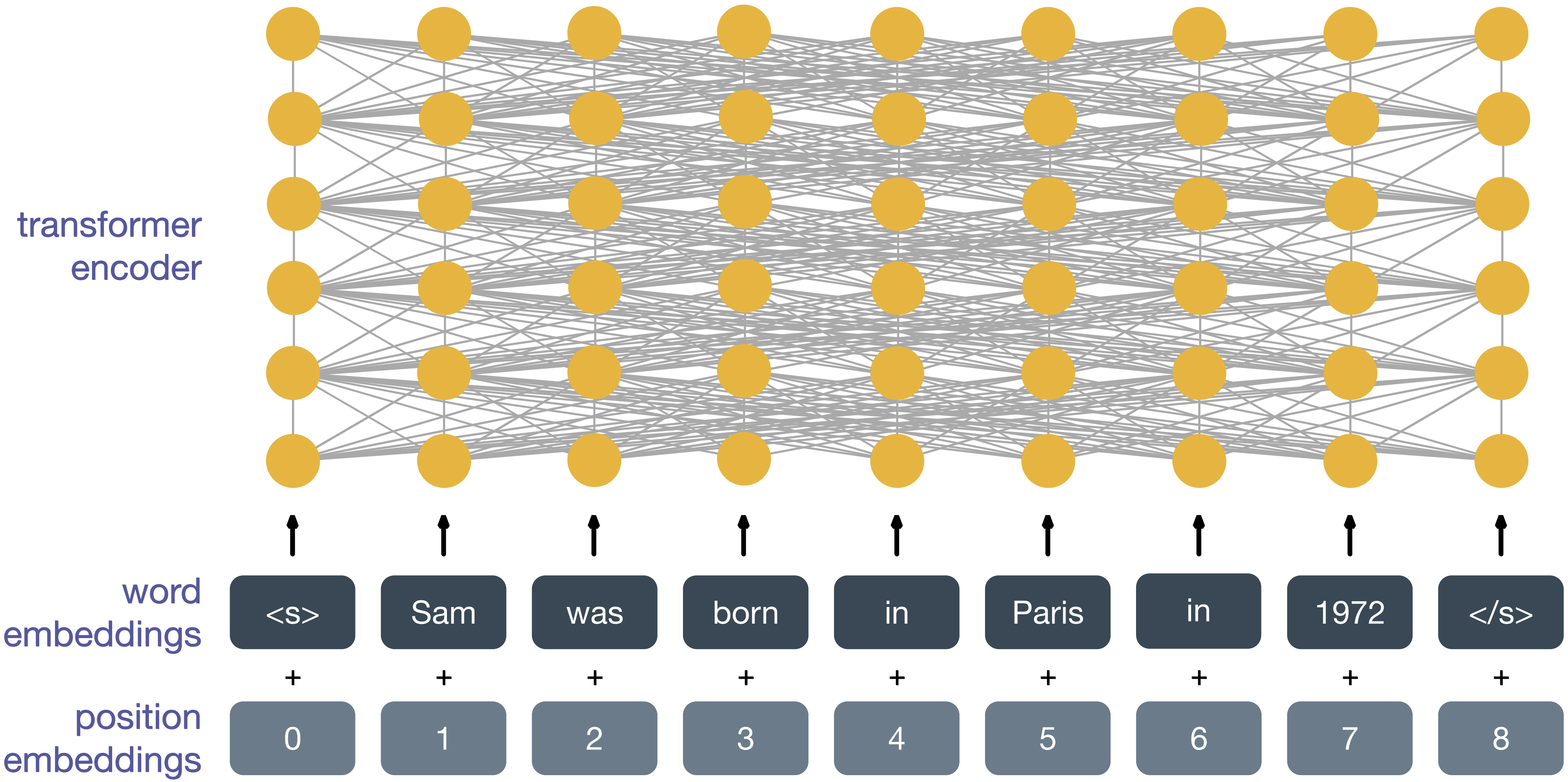


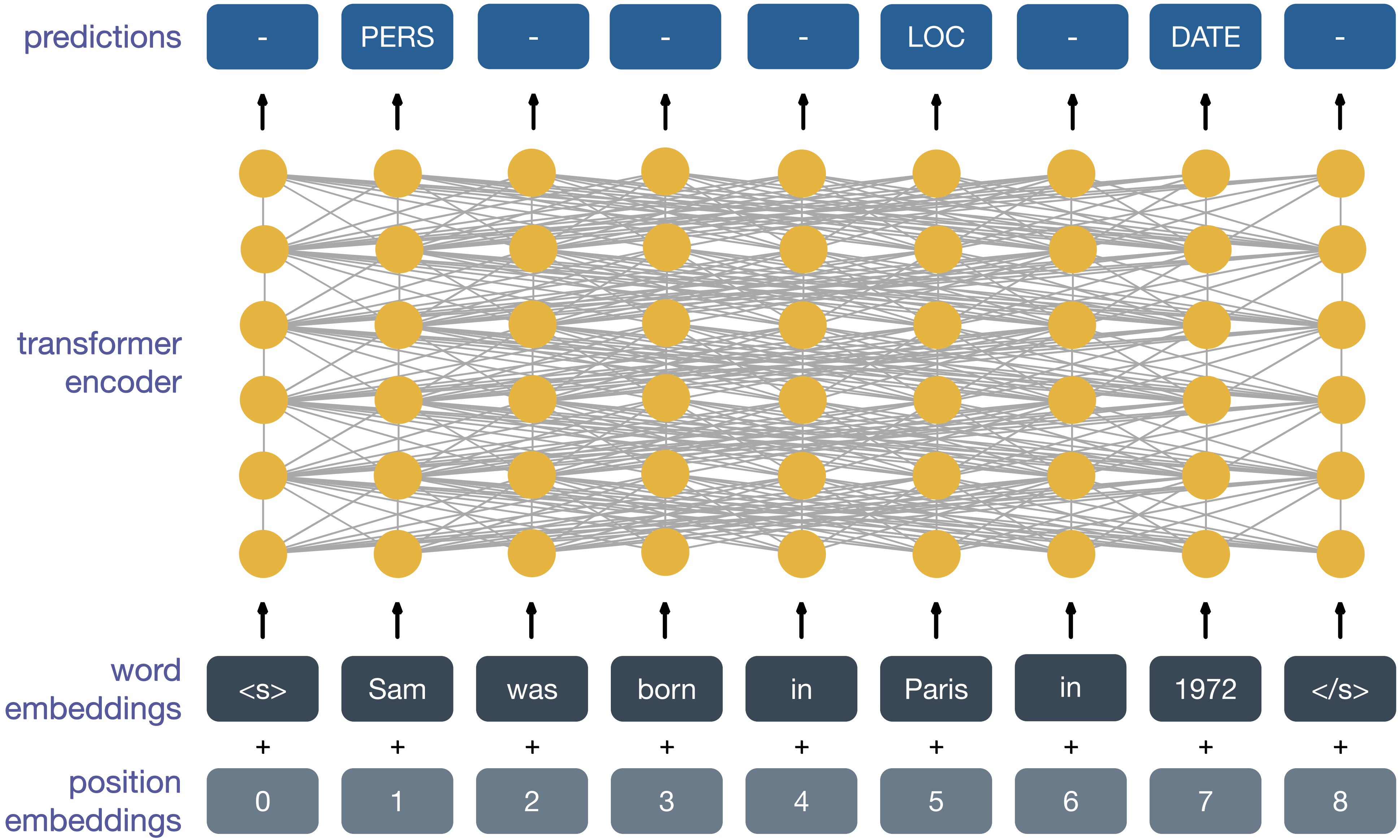
Masked Language Models



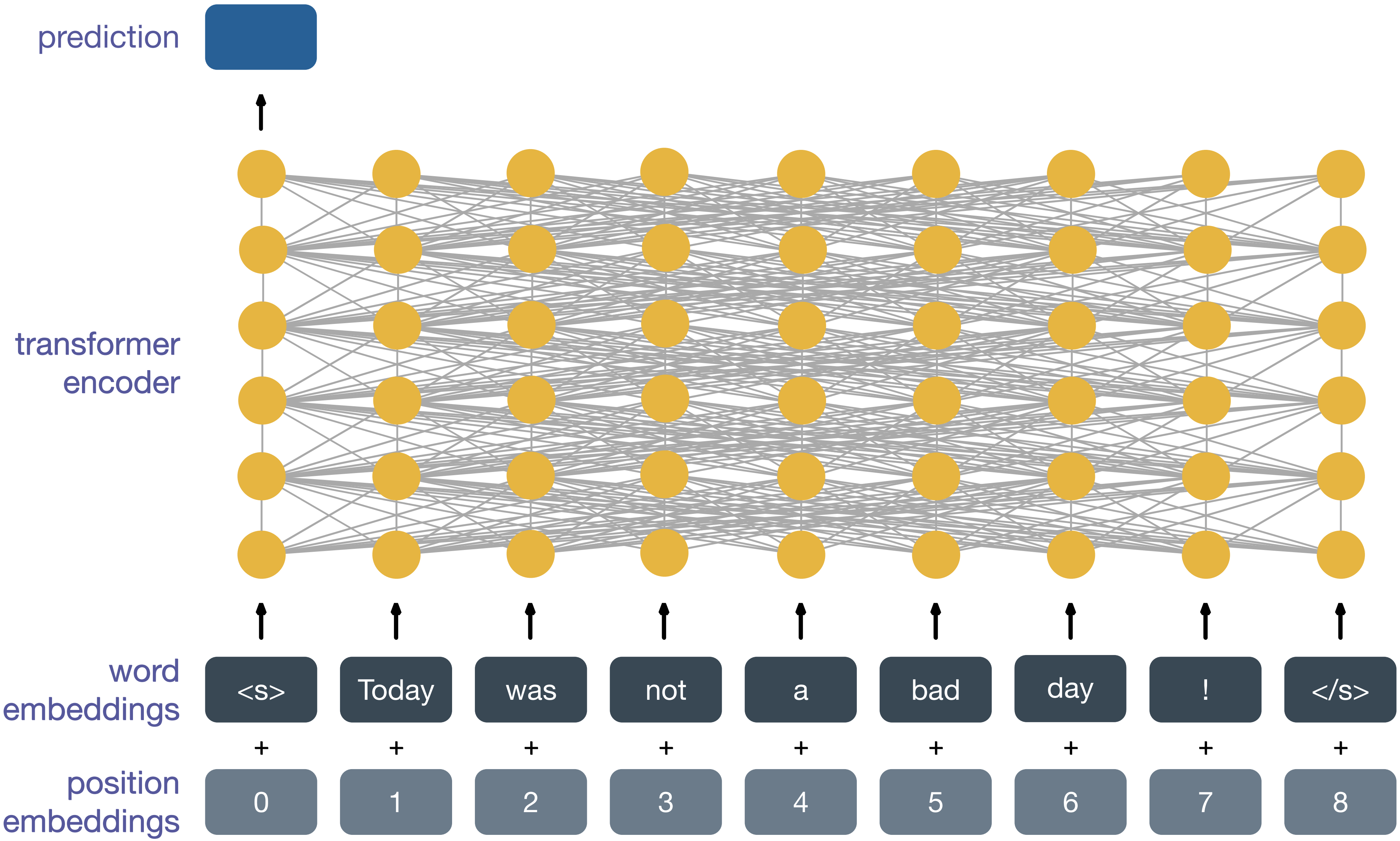
Masked Language Models

Token-level Tasks

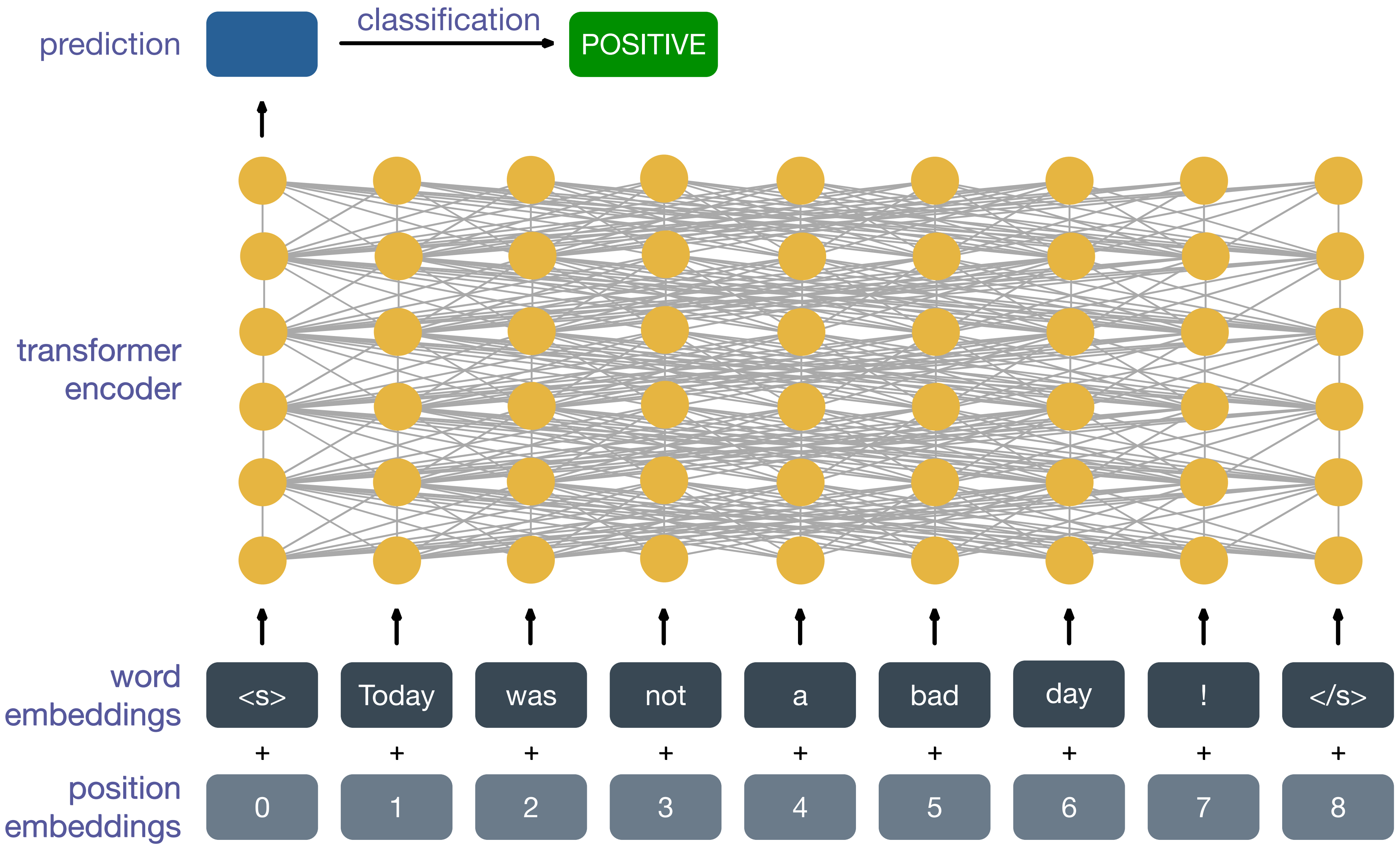




Token-level Tasks



Sentence-level Tasks



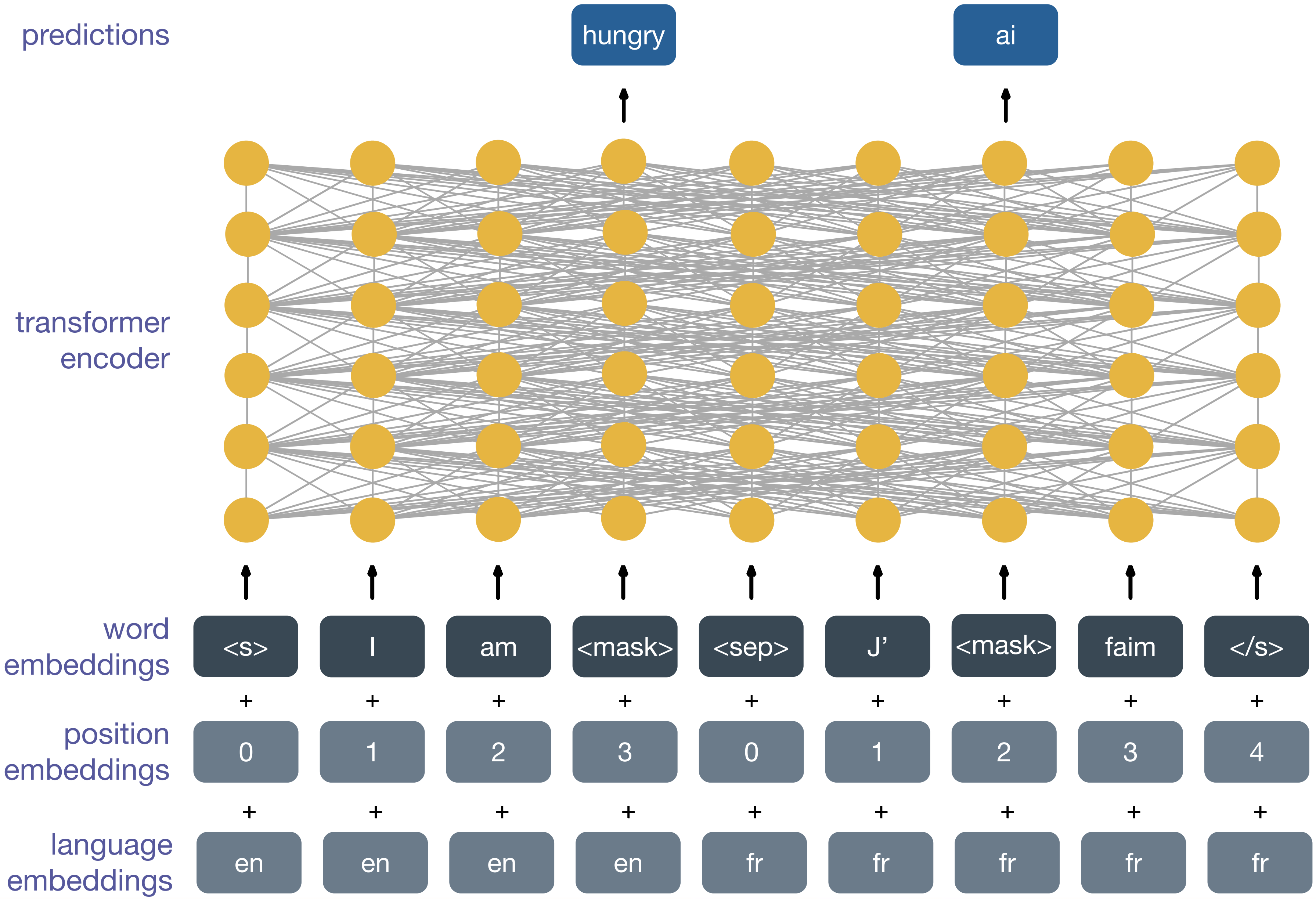
Sentence-level Tasks

I am hungry J’ ai faim

Cross-lingual Masked Language Modeling

<s> I am <mask> <sep> J’ <mask> faim </s>

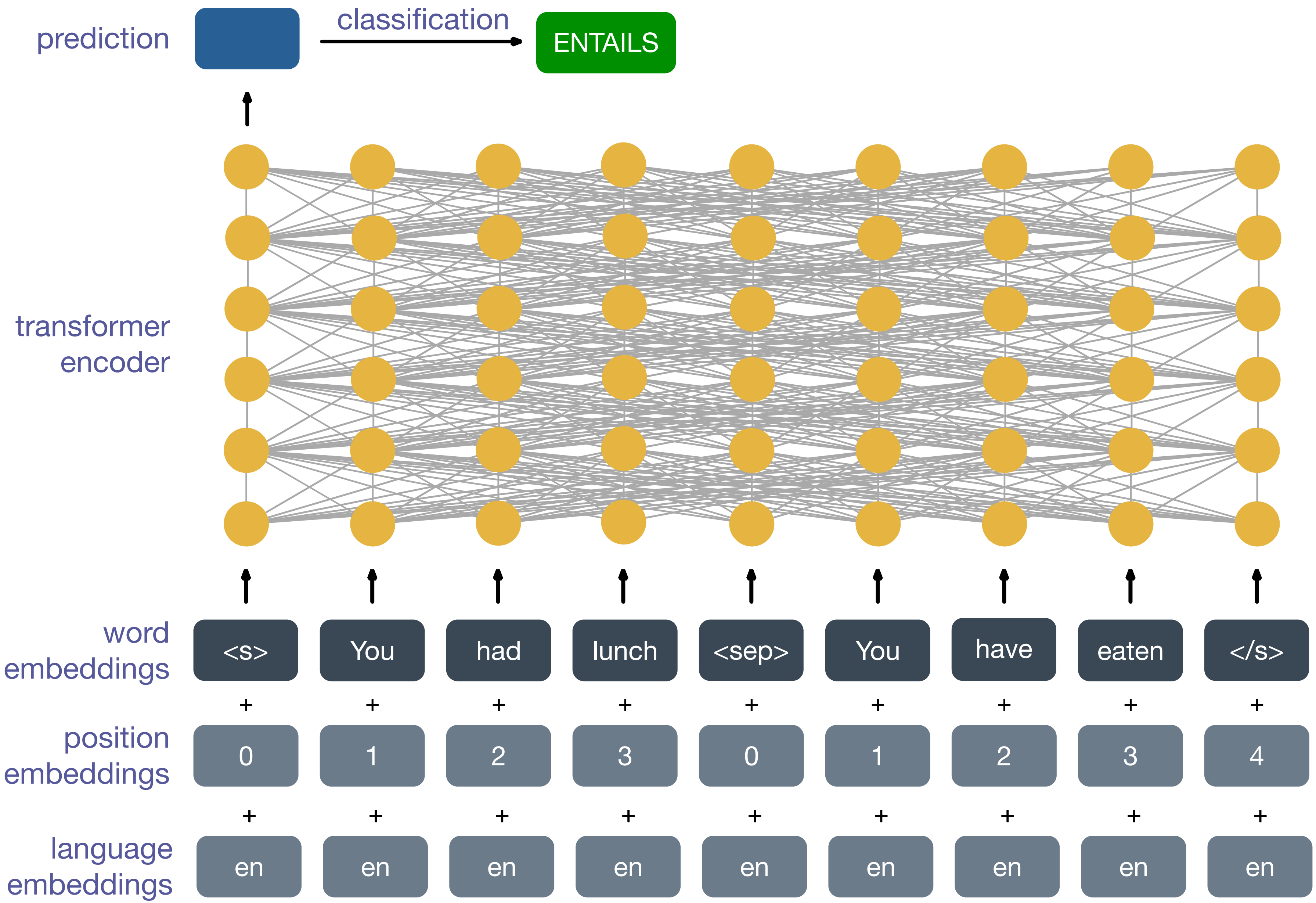
Cross-lingual Masked Language Modeling



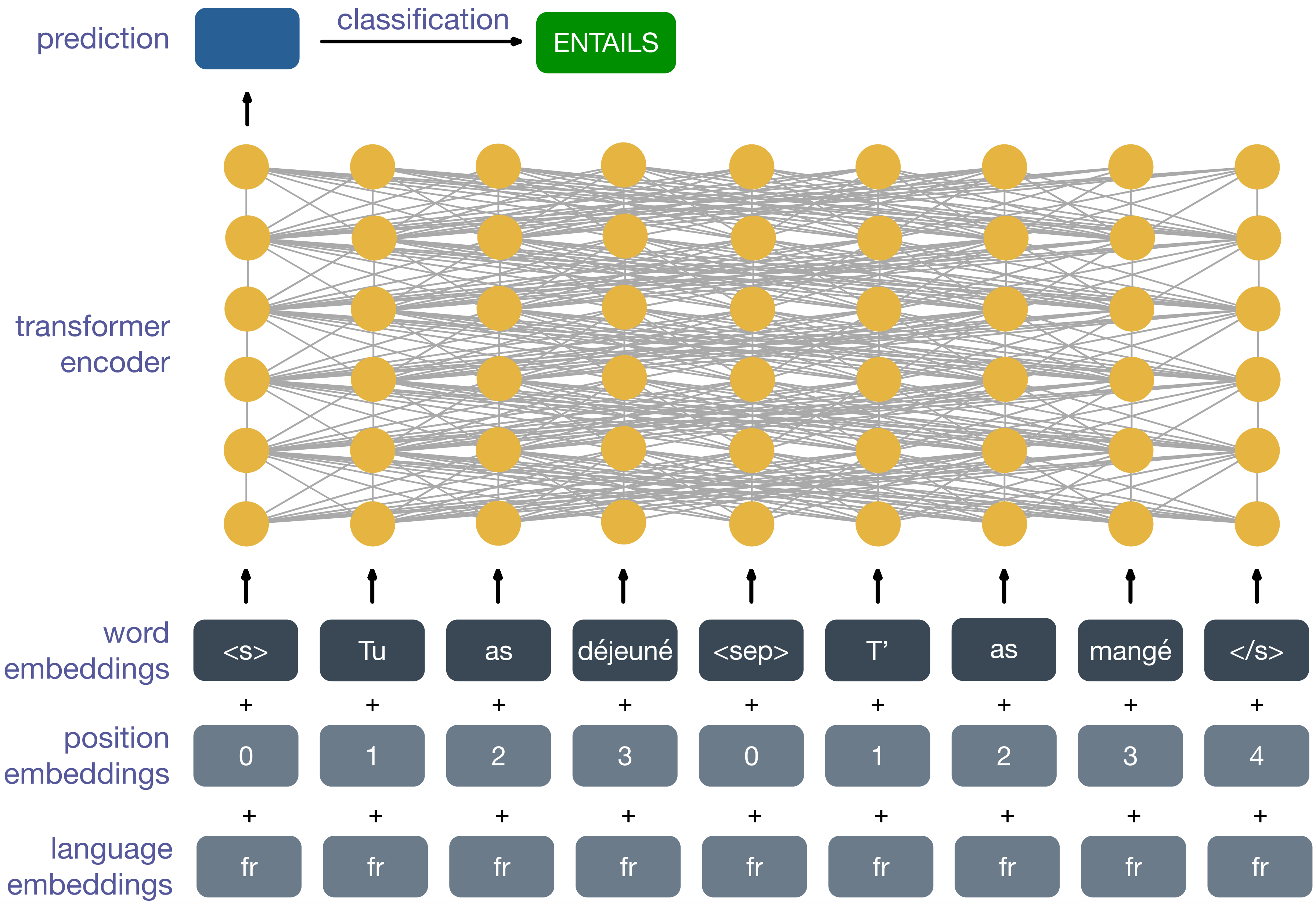
Cross-lingual Masked Language Modeling

FACEBOOK AI

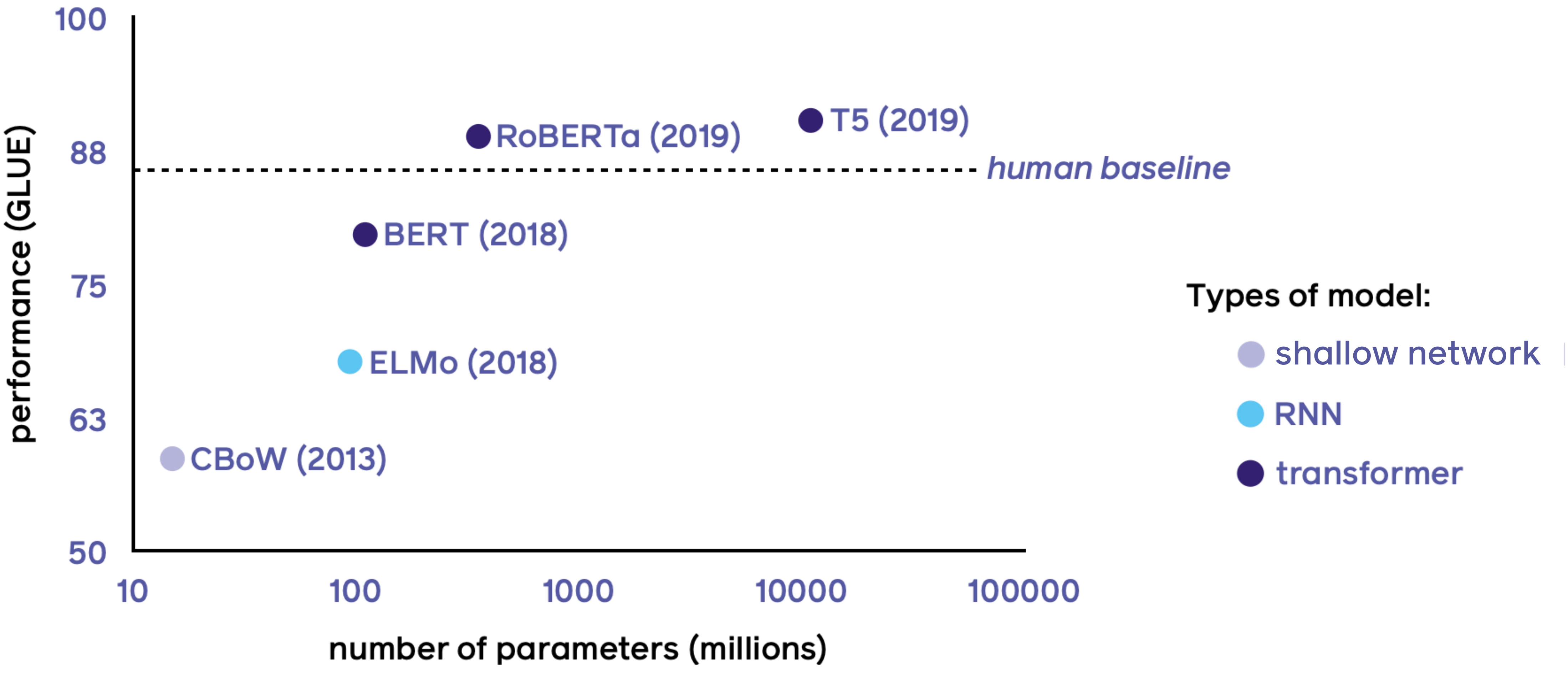
Georgia Tech



Cross-lingual Task: Natural Language Inference



Cross-lingual Task: Natural Language Inference



Model Size in Perspective

Preprint. Under review.

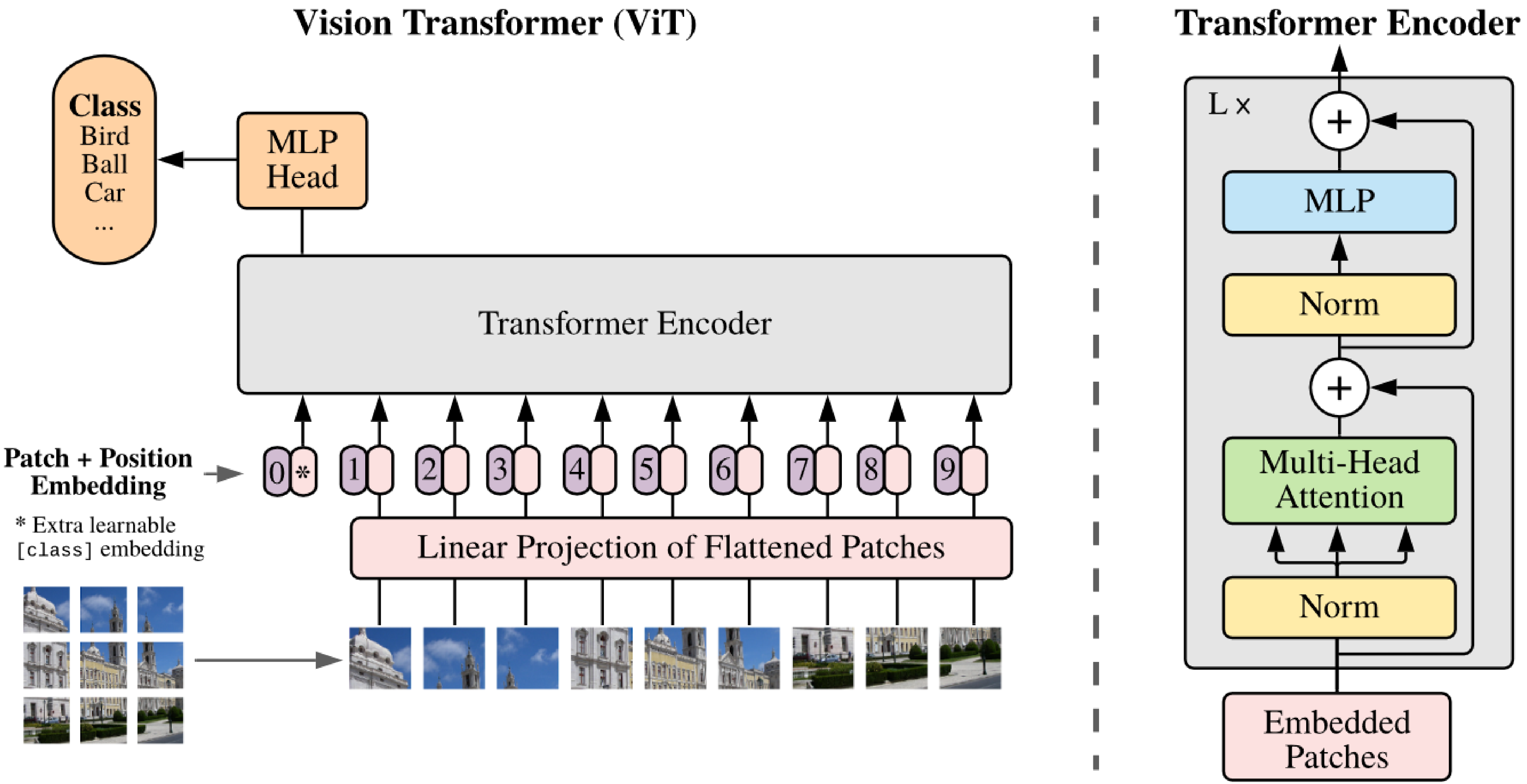
An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

Alexey Dosovitskiy[*,†], **Lucas Beyer**[*], **Alexander Kolesnikov**[*], **Dirk Weissenborn**[*], **Xiaohua Zhai**[*], **Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby**[*,†]
[*]equal technical contribution, [†]equal advising
Google Research, Brain Team
{adosovitskiy, neilhoulsby}@google.com

Abstract

While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.[1]

What About Vision?



Vision Transformer (ViT)

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21K (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** $\pm$ 0.04	87.76 $\pm$ 0.03	85.30 $\pm$ 0.02	87.54 $\pm$ 0.02	88.4/88.5*
ImageNet ReaL	**90.72** $\pm$ 0.05	90.54 $\pm$ 0.03	88.62 $\pm$ 0.05	90.54	90.55
CIFAR-10	**99.50** $\pm$ 0.06	99.42 $\pm$ 0.03	99.15 $\pm$ 0.03	99.37 $\pm$ 0.06	–
CIFAR-100	**94.55** $\pm$ 0.04	93.90 $\pm$ 0.05	93.25 $\pm$ 0.05	93.51 $\pm$ 0.08	–
Oxford-IIIT Pets	**97.56** $\pm$ 0.03	97.32 $\pm$ 0.11	94.67 $\pm$ 0.15	96.62 $\pm$ 0.23	–
Oxford Flowers-102	99.68 $\pm$ 0.02	**99.74** $\pm$ 0.00	99.61 $\pm$ 0.02	99.63 $\pm$ 0.03	–
VTAB (19 tasks)	**77.63** $\pm$ 0.23	76.28 $\pm$ 0.46	72.72 $\pm$ 0.21	76.29 $\pm$ 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

ViT Results

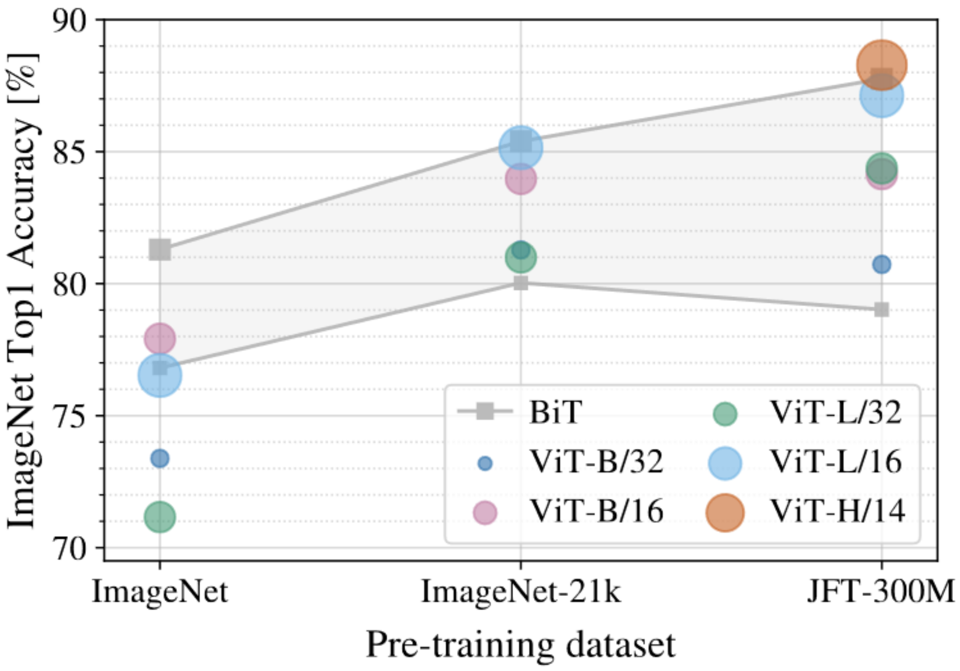


Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.

When trained on mid-sized datasets such as ImageNet, such models yield modest accuracies of a few percentage points below ResNets of comparable size. This seemingly discouraging outcome maybe expected: Transformers lack some of the inductive biases inherent to CNNs, such as translation equivariance and locality, and therefore do not generalize well when trained on insufficient amounts of data.

However, the picture changes if the models are trained on larger datasets (14M-300M images). We find that large scale training trumps inductive bias.

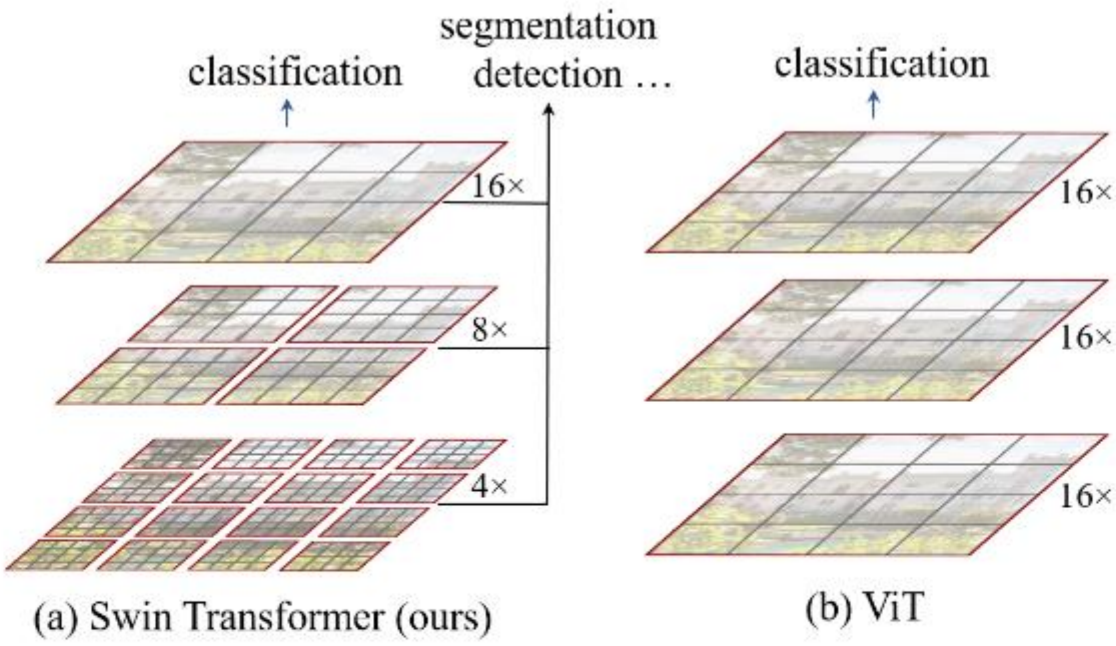
Dosovitskiy et al.

ViTs and Transfer Learning

https://paperswithcode.com/sota/image-classification-on-imagenet

Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, Baining Guo



Swin Transformers

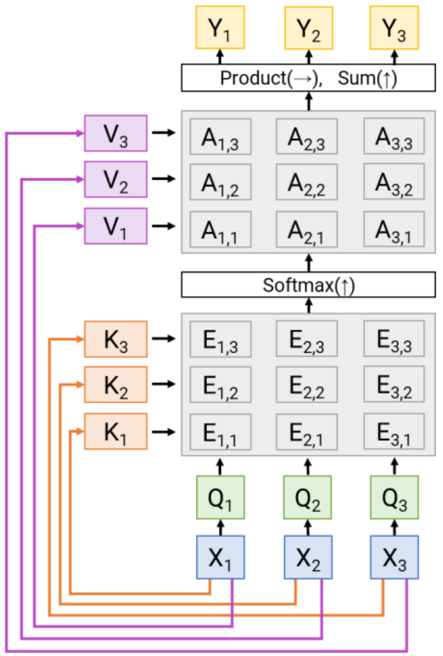
https://paperswithcode.com/sota/instance-segmentation-on-coco

Summary

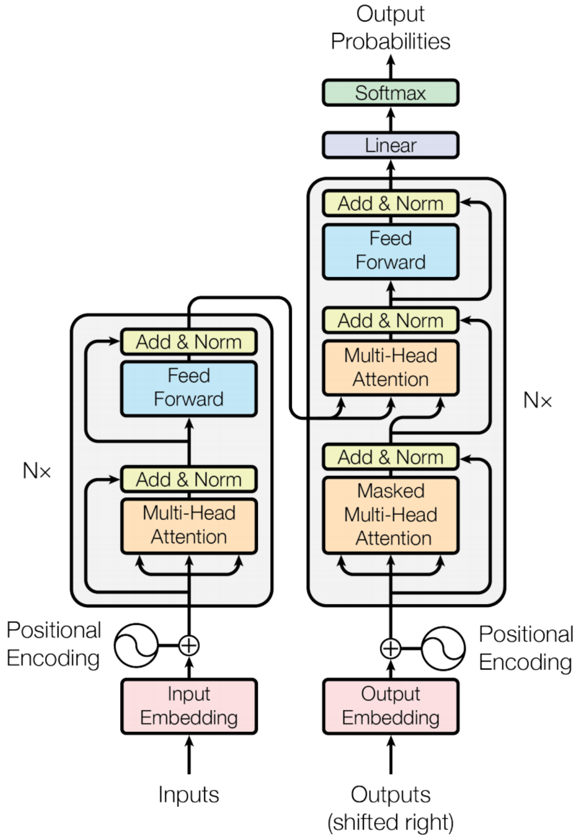
- “Attention” models outperform recurrent models and convolutional models for sequence processing. They allow long range interactions.
- These models do best with LOTS of training data
- Surprisingly, they seem to outperform convolutional networks for image processing tasks. Again, long range interactions might be more important than we realized.
- Naïve attention mechanisms have quadratic complexity with the number of input tokens, but there are often workarounds for this.

Summary

Self-Attention



Transformer Model



ViLBERT