CS 446/ECE 449: Machine Learning

Shenlong Wang

University of Illinois at Urbana-Champaign, 2024

Attention and Transformers

Goals of this lecture

Goals of this lecture

- Getting to know attention mechanisms

Goals of this lecture

- Getting to know attention mechanisms
- Learning about transformers

Recap: Sequence data

More flexibility regarding inputs and outputs:

Recap: Sequence data

More flexibility regarding inputs and outputs:

- Sequences of inputs

Recap: Sequence data

More flexibility regarding inputs and outputs:

- Sequences of inputs
- Sequences of outputs

Recap: Sequence data

More flexibility regarding inputs and outputs:

- Sequences of inputs
- Sequences of outputs

Length of sequences may vary

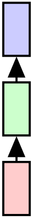
Recap: Sequence data

More flexibility regarding inputs and outputs:

- Sequences of inputs
- Sequences of outputs

Length of sequences may vary

one to one



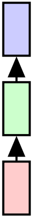
Recap: Sequence data

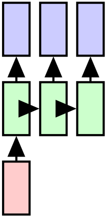
More flexibility regarding inputs and outputs:

- Sequences of inputs
- Sequences of outputs

Length of sequences may vary

one to one one to many



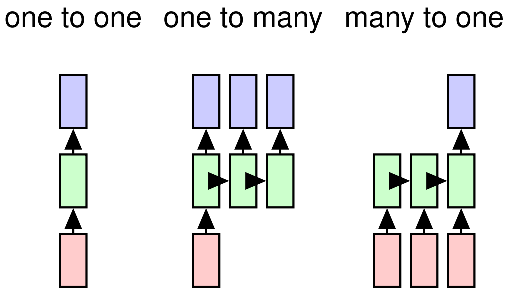


Recap: Sequence data

More flexibility regarding inputs and outputs:

- Sequences of inputs
- Sequences of outputs

Length of sequences may vary

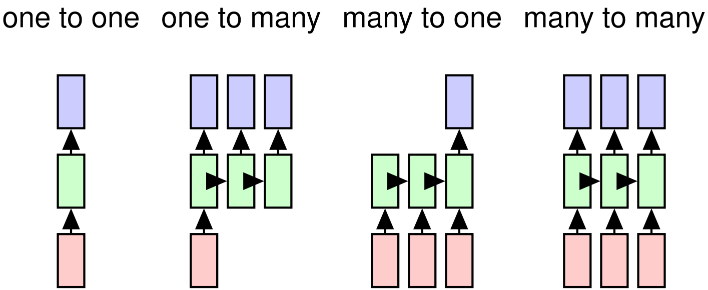


Recap: Sequence data

More flexibility regarding inputs and outputs:

- Sequences of inputs
- Sequences of outputs

Length of sequences may vary

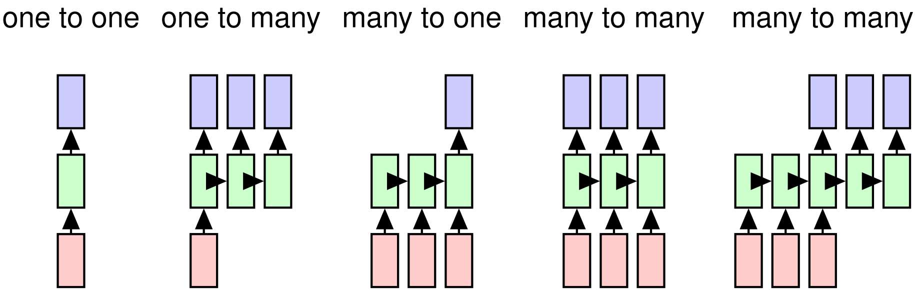


Recap: Sequence data

More flexibility regarding inputs and outputs:

- Sequences of inputs
- Sequences of outputs

Length of sequences may vary



Recap: Sequence models

Recap: Sequence models

- Recurrent Neural Net (RNNs)

Recap: Sequence models

- Recurrent Neural Net (RNNs)
- Long-short-term-memory (LSTM) unit

Recap: Sequence models

- Recurrent Neural Net (RNNs)
- Long-short-term-memory (LSTM) unit
- Gated recurrent unit (GRU)

What else:

What else:

Attention mechanisms and Transformers

What else:

Attention mechanisms and Transformers

Why:

What else:

Attention mechanisms and Transformers

Why:

- RNNs, LSTMs, GRUs are not parallelizable (process data sequentially)

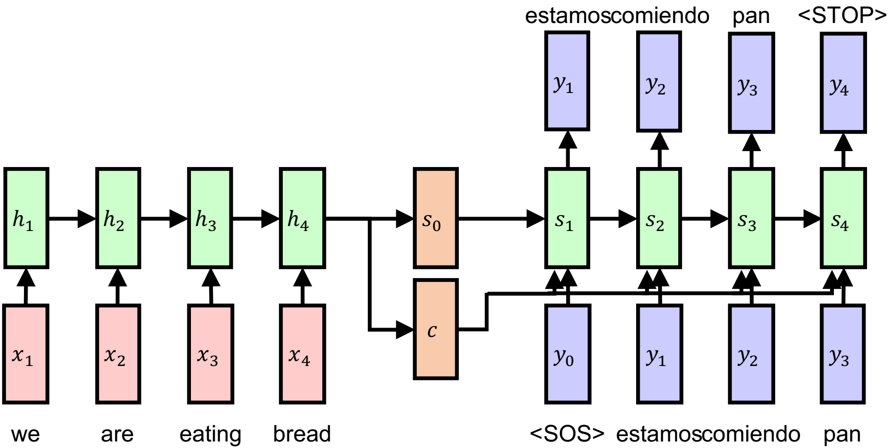
What else:

Attention mechanisms and Transformers

Why:

- RNNs, LSTMs, GRUs are not parallelizable (process data sequentially)
- Still bottleneck for very long sequences

Let's consider an example:



Recall:

- Input:

Recall:

- Input: $(x_1, x_2, \ldots)$

Recall:

- Input: $(x_1, x_2, \ldots)$
- Output:

Recall:

- Input: $(x_1, x_2, \ldots)$
- Output: $(y_1, y_2, \ldots)$

Recall:

- Input: $(x_1, x_2, \ldots)$
- Output: $(y_1, y_2, \ldots)$
- Encoder:

Recall:

- Input: $(x_1, x_2, \dots)$
- Output: $(y_1, y_2, \dots)$
- Encoder: $h_t = f(x_t, h_{t-1})$

Recall:

- Input: $(x_1, x_2, \dots)$
- Output: $(y_1, y_2, \dots)$
- Encoder: $h_t = f(x_t, h_{t-1})$
- Decoder:

Recall:

- Input: $(x_1, x_2, \dots)$
- Output: $(y_1, y_2, \dots)$
- Encoder: $h_t = f(x_t, h_{t-1})$
- Decoder: $s_t = g(y_{t-1}, s_{t-1}, c)$

Recall:

- Input: $(x_1, x_2, \dots)$
- Output: $(y_1, y_2, \dots)$
- Encoder: $h_t = f(x_t, h_{t-1})$
- Decoder: $s_t = g(y_{t-1}, s_{t-1}, c)$
- Initial input/Context:

Recall:

- Input: $(x_1, x_2, \dots)$
- Output: $(y_1, y_2, \dots)$
- Encoder: $h_t = f(x_t, h_{t-1})$
- Decoder: $s_t = g(y_{t-1}, s_{t-1}, c)$
- Initial input/Context: e.g., $s_0 = 0, c = h_T$

Problem:

Recall:

- Input: $(x_1, x_2, \ldots)$
- Output: $(y_1, y_2, \ldots)$
- Encoder: $h_t = f(x_t, h_{t-1})$
- Decoder: $s_t = g(y_{t-1}, s_{t-1}, c)$
- Initial input/Context: e.g., $s_0 = 0, c = h_T$

Problem: What if sequence very long?

Fix:

Recall:

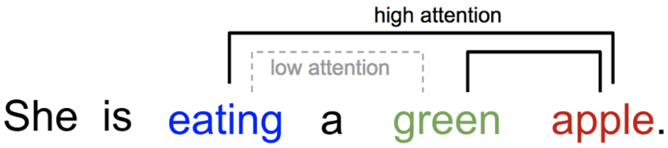
- Input: $(x_1, x_2, \ldots)$
- Output: $(y_1, y_2, \ldots)$
- Encoder: $h_t = f(x_t, h_{t-1})$
- Decoder: $s_t = g(y_{t-1}, s_{t-1}, c)$
- Initial input/Context: e.g., $s_0 = 0, c = h_T$

Problem: What if sequence very long?

Fix: Use a new context vector for every output.

What is a good context vector for each element?

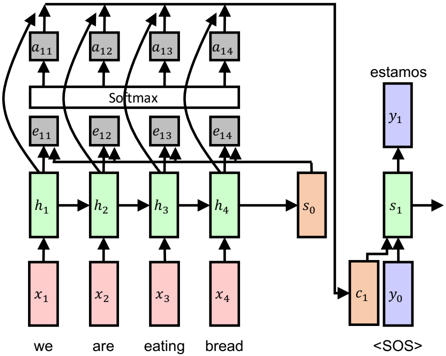
What is a good context vector for each element?



What is a good context vector for each element?

How?

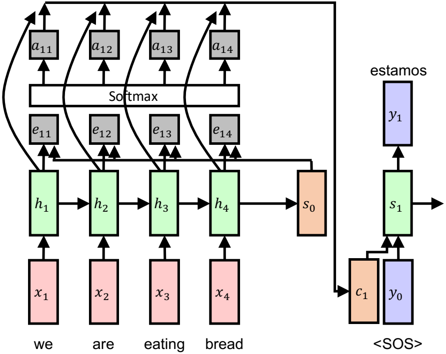
How?



a_{11}
a_{12}
a_{13}
a_{14}
Softmax
e_{11}
e_{12}
e_{13}
e_{14}
estamos
y_1
h_1
h_2
h_3
h_4
s_0
s_1
x_1
x_2
x_3
x_4
c_1
y_0
we
are
eating
bread
<SOS>

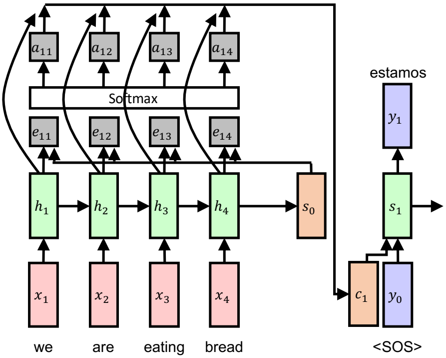
How?

- Alignment scores:
  $e_{t,i} = f_{\text{att}}(s_{t-1}, h_i) \in \mathbb{R}$
  f_{att} is an MLP



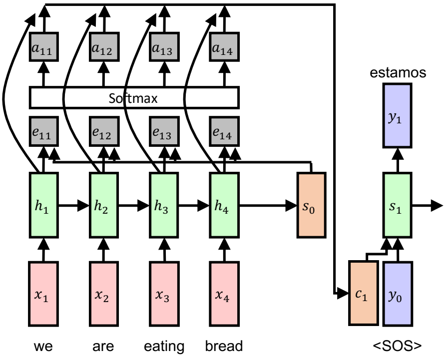
How?

- Alignment scores:
  $e_{t,i} = f_{\text{att}}(s_{t-1}, h_i) \in \mathbb{R}$
  f_{att} is an MLP
- Normalize via softmax to obtain **attention** weights $0 \leq a_{t,i} \leq 1$ $(\sum_i a_{t,i} = 1)$

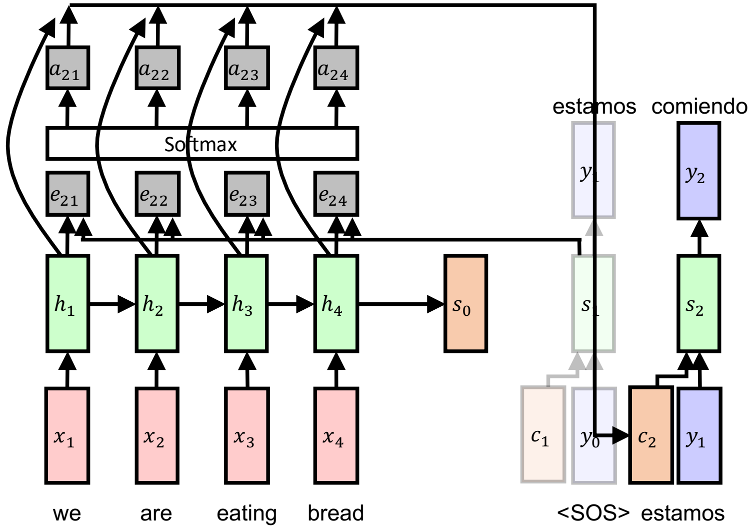


How?

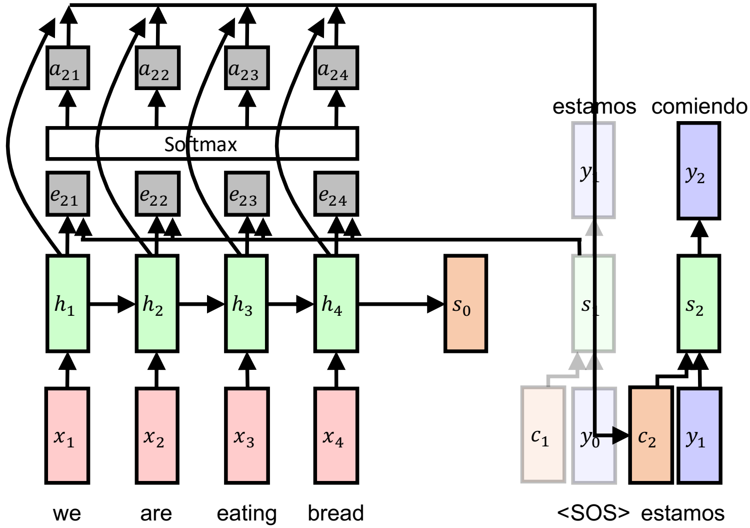
- Alignment scores:
  $e_{t,i} = f_{\text{att}}(s_{t-1}, h_i) \in \mathbb{R}$
  f_{att} is an MLP
- Normalize via softmax to obtain **attention** weights
  $0 \leq a_{t,i} \leq 1$ $(\sum_i a_{t,i} = 1)$
- Compute attended representation via linear combination:
  $c_t = \sum_i a_{t,i} h_i$



Next time step:



Next time step:



All differentiable. Don't supervise. Backprop through entire net.

Observation:

By re-computing context vectors c_t at every decoding step we avoid the bottleneck obtained when using a single vector for the input data.

At every decoding step the context vector “looks at” different parts of the input sequence. The attention weight $a_{t,i}$ determines the strength.

Example: English to French translation

Example: English to French translation

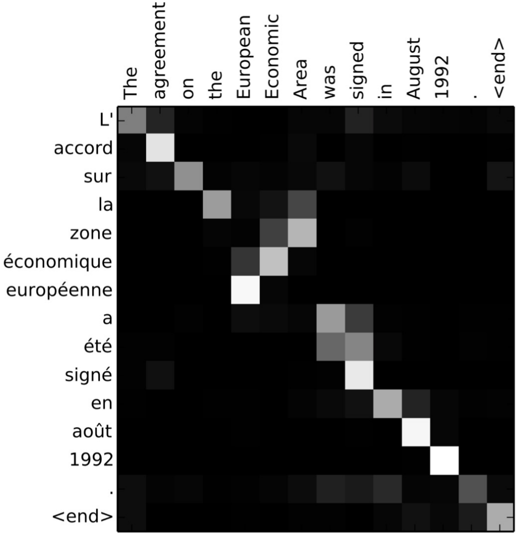
- **Input:** “The agreement on the European Economic Area was signed in August 1992.”

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.”

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."



Extension

Extension

Observe:

Decoder treats h_i as an unordered set.

Consequence:

Extension

Observe:

Decoder treats h_i as an unordered set.

Consequence:

Similar architecture can be used for any set of vectors h_i.

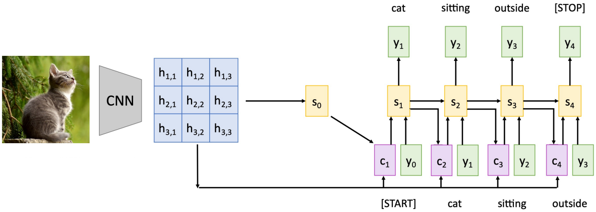
Extension

Observe:

Decoder treats h_i as an unordered set.

Consequence:

Similar architecture can be used for any set of vectors h_i.



A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Generalizing: **Towards the Attention Layer**

We started with

Inputs:

- Query vector: q
- Input vectors: X
- Similarity function: f_{att}

Computations:

- Scores: $e_i = f_{\text{att}}(q, X_i)$
- Attention: $a = \text{softmax}(e)$
- Attended representation: $y = \sum_i a_i X_i$

The Attention Layer

Inputs:

- Query vectors: $Q \in \mathbb{R}^{N_Q \times D_Q}$
- Input vectors: $X \in \mathbb{R}^{N_X \times D_X}$

The Attention Layer

Inputs:

- Query vectors: $Q \in \mathbb{R}^{N_Q \times D_Q}$
- Input vectors: $X \in \mathbb{R}^{N_X \times D_X}$
- Key matrix: $W_K \in \mathbb{R}^{D_X \times D_Q}$

The Attention Layer

Inputs:

- Query vectors: $Q \in \mathbb{R}^{N_Q \times D_Q}$
- Input vectors: $X \in \mathbb{R}^{N_X \times D_X}$
- Key matrix: $W_K \in \mathbb{R}^{D_X \times D_Q}$
- Value matrix: $W_V \in \mathbb{R}^{D_X \times D_V}$

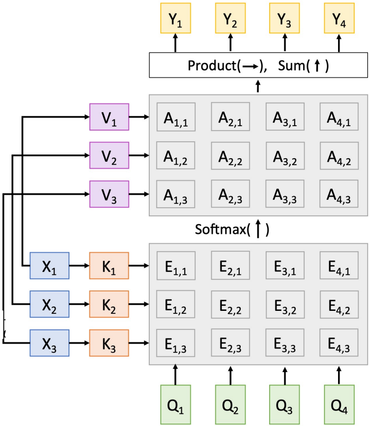
The Attention Layer

Inputs:

- Query vectors: $Q \in \mathbb{R}^{N_Q \times D_Q}$
- Input vectors: $X \in \mathbb{R}^{N_X \times D_X}$
- Key matrix: $W_K \in \mathbb{R}^{D_X \times D_Q}$
- Value matrix: $W_V \in \mathbb{R}^{D_X \times D_V}$

Computations:

- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Scores: $E = QK^T / \sqrt{D_Q}$
- Attention: $A = \text{softmax}(E, \text{dim} = 1)$
- Representation: $Y = AV$



The **Self-**Attention Layer

Inputs:

- Input vectors: $X \in \mathbb{R}^{N_X \times D_X}$

The **Self-**Attention Layer

Inputs:

- Input vectors: $X \in \mathbb{R}^{N_X \times D_X}$
- Key matrix: $W_K \in \mathbb{R}^{D_X \times D_Q}$

The **Self-**Attention Layer

Inputs:

- Input vectors: $X \in \mathbb{R}^{N_X \times D_X}$
- Key matrix: $W_K \in \mathbb{R}^{D_X \times D_Q}$
- Value matrix: $W_V \in \mathbb{R}^{D_X \times D_V}$

The **Self-**Attention Layer

Inputs:

- Input vectors: $X \in \mathbb{R}^{N_X \times D_X}$
- Key matrix: $W_K \in \mathbb{R}^{D_X \times D_Q}$
- Value matrix: $W_V \in \mathbb{R}^{D_X \times D_V}$
- Query matrix: $W_Q \in \mathbb{R}^{D_X \times D_Q}$

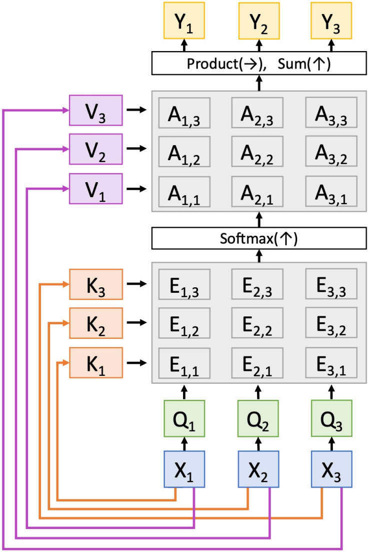
The **Self-**Attention Layer

Inputs:

- Input vectors: $X \in \mathbb{R}^{N_X \times D_X}$
- Key matrix: $W_K \in \mathbb{R}^{D_X \times D_Q}$
- Value matrix: $W_V \in \mathbb{R}^{D_X \times D_V}$
- Query matrix: $W_Q \in \mathbb{R}^{D_X \times D_Q}$

Computations:

- Query vectors $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Scores: $E = QK^T / \sqrt{D_Q}$
- Attention: $A = \text{softmax}(E, \text{dim} = 1)$
- Representation: $Y = AV$

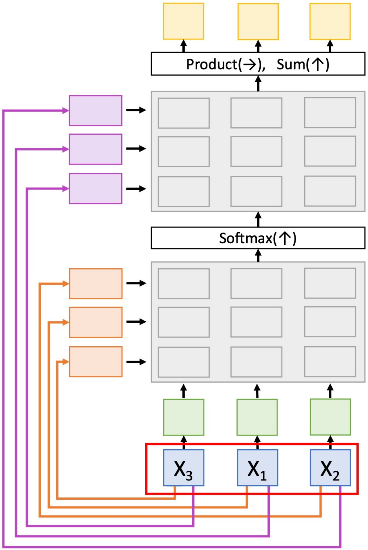


Why a **Self-**Attention Layer?

Computations:

Why a **Self-**Attention Layer? What happens when we permute the input vectors?

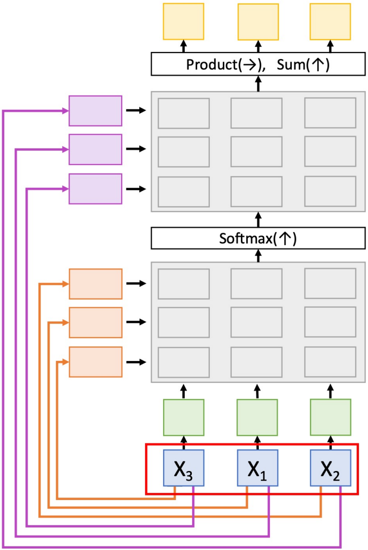
Computations:



Why a **Self**-Attention Layer? What happens when we permute the input vectors?

Computations:

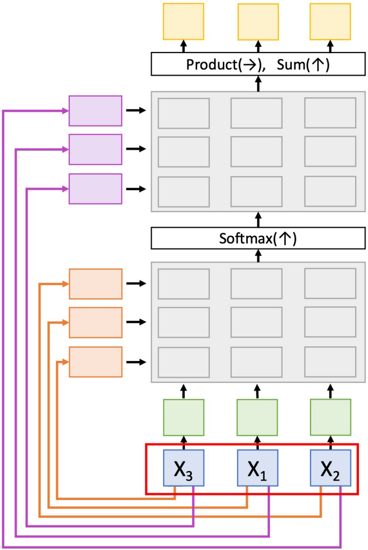
- Query vectors $Q = XW_Q$



Why a **Self-**Attention Layer? What happens when we permute the input vectors?

Computations:

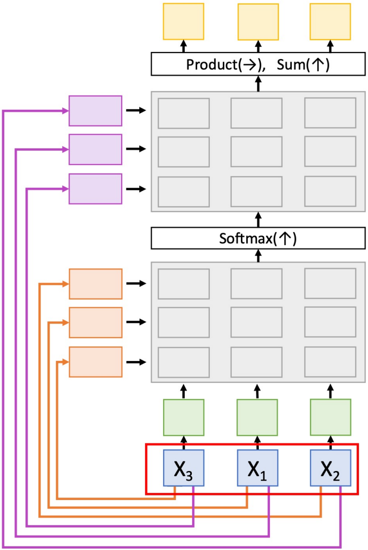
- Query vectors $Q = XW_Q$
- Key vectors: $K = XW_K$



Why a **Self-**Attention Layer? What happens when we permute the input vectors?

Computations:

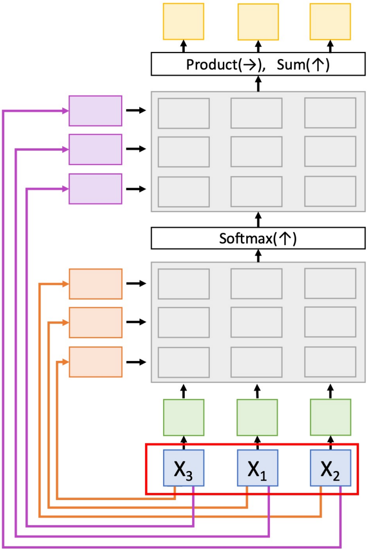
- Query vectors $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$



Why a **Self-**Attention Layer? What happens when we permute the input vectors?

Computations:

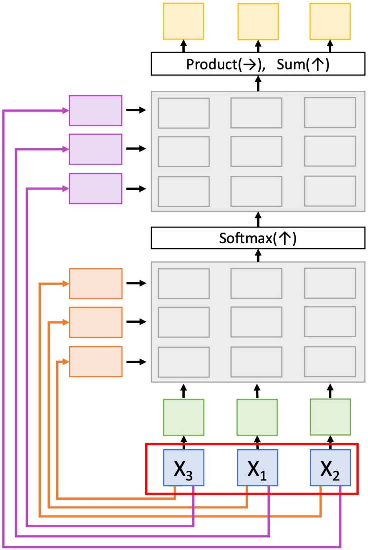
- Query vectors $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Scores: $E = QK^T / \sqrt{D_Q}$



Why a **Self-**Attention Layer? What happens when we permute the input vectors?

Computations:

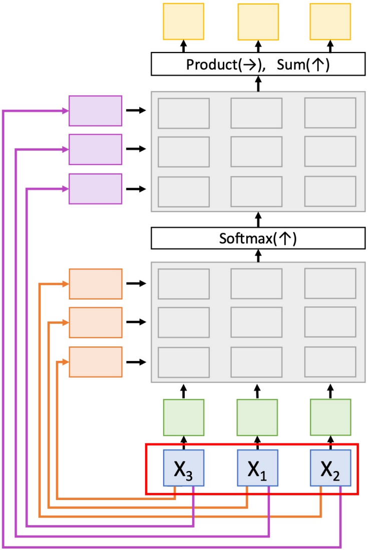
- Query vectors $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Scores: $E = QK^T / \sqrt{D_Q}$
- Attention: $A = \text{softmax}(E, \text{dim} = 1)$



Why a **Self-**Attention Layer? What happens when we permute the input vectors?

Computations:

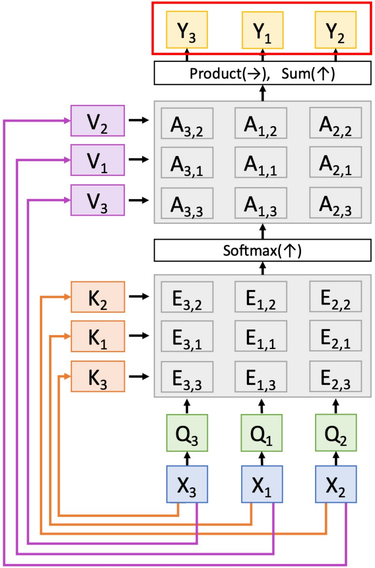
- Query vectors $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Scores: $E = QK^T / \sqrt{D_Q}$
- Attention: $A = \text{softmax}(E, \text{dim} = 1)$
- Representation: $Y = AV$



Why a **Self-**Attention Layer? What happens when we permute the input vectors?

Computations:

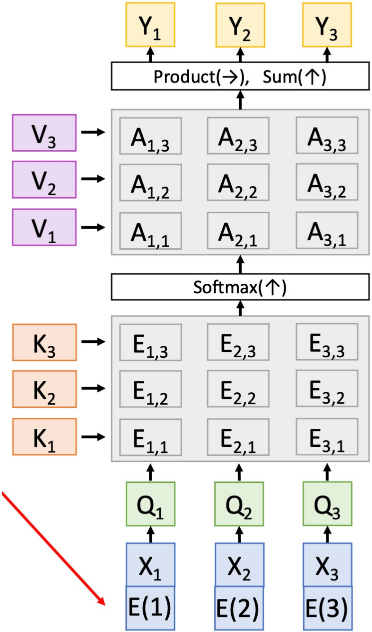
- Query vectors $Q = XW_Q$
- Key vectors: $K = XW_K$
- Value vectors: $V = XW_V$
- Scores: $E = QK^T / \sqrt{D_Q}$
- Attention: $A = \text{softmax}(E, \text{dim} = 1)$
- Representation: $Y = AV$



Self-Attention Layer:

- Permuting inputs results in permuted outputs
- Self-attention layer is **permutation equivariant** $f(s(x)) = s(f(x))$
- Self-attention layer operates on sets of vectors

However:

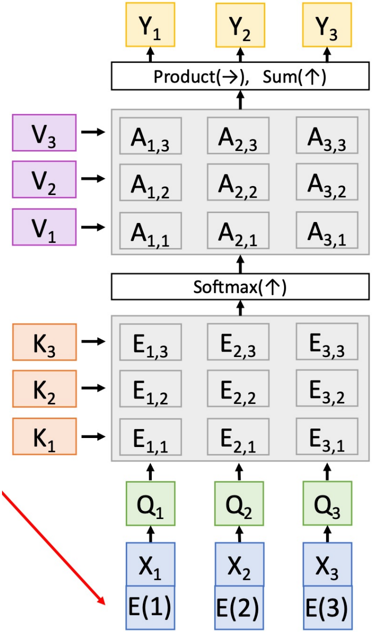


Self-Attention Layer:

- Permuting inputs results in permuted outputs
- Self-attention layer is **permutation equivariant** $f(s(x)) = s(f(x))$
- Self-attention layer operates on sets of vectors

However:

Sometimes we want to take the order into account. How?



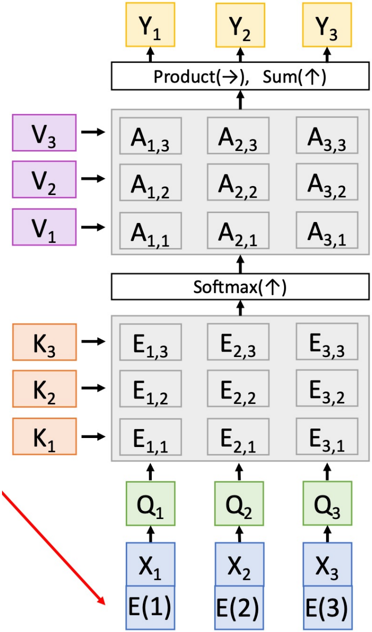
Self-Attention Layer:

- Permuting inputs results in permuted outputs
- Self-attention layer is **permutation equivariant** $f(s(x)) = s(f(x))$
- Self-attention layer operates on sets of vectors

However:

Sometimes we want to take the order into account. How?

Use positional encoding E



Problem: How to use this to encode text where we shouldn't look ahead?

Problem: How to use this to encode text where we shouldn't look ahead?

Fix: Use masking

Problem: How to use this to encode text where we shouldn't look ahead?

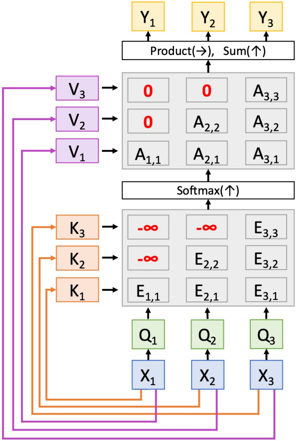
Fix: Use masking

Masked Attention

Problem: How to use this to encode text where we shouldn't look ahead?

Fix: Use masking

Masked Attention

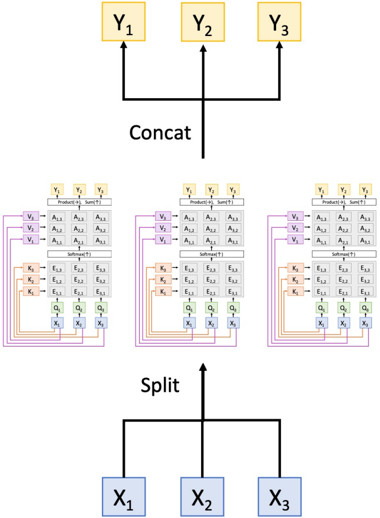


Multihead Self-Attention Layer

Use a set of independent attention heads

Multihead Self-Attention Layer

Use a set of independent attention heads



Ways to process sequence data:

Classical RNNs

(+) Reasonably good at long sequences

(-) Not parallelizable

1D Convolutions

(-) Bad at long sequences

(+) Trivially parallelizable

Self-Attention

(+) Good at long sequences

(+) Trivially parallelizable

(-) Memory intensive

Transformer

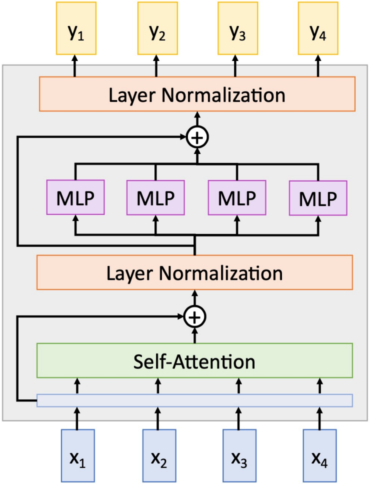
Transformer block:

- Input: set of vectors
- Output: set of vectors
- Computation:

Transformer

Transformer block:

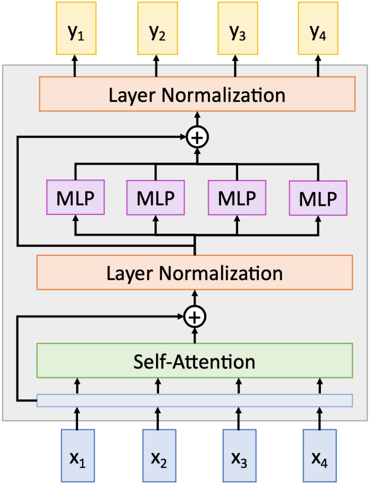
- Input: set of vectors
- Output: set of vectors
- Computation:



Transformer

Transformer block:

- Input: set of vectors
- Output: set of vectors
- Computation:
  - self-interaction is the only interaction between vectors
  - normalization and MLP operate independently
  - position of normalization may change

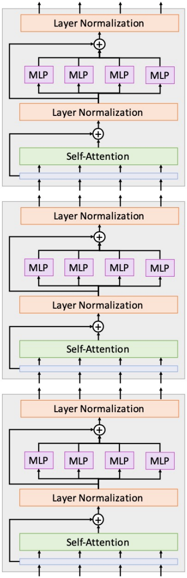


Transformer

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

Transformer

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

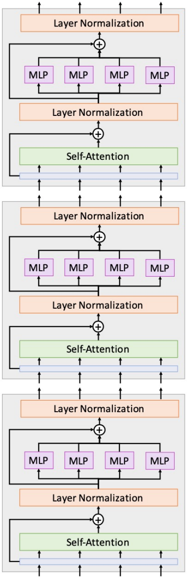


Transformer

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

Revolutionized (?) natural language processing:

- Download a lot of text
- Train a giant transformer model
- Fine-tune on desired task with little data

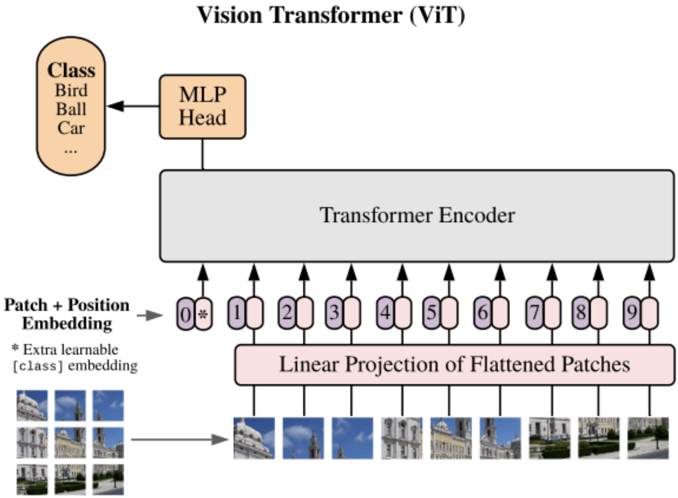


Scaling Natural Language Processing:

Model	Layers	Width	Heads	Params	Data	Training
Transformer-Base	12	512	8	65M		8x P100 (12 hours)
Transformer-Large	12	1024	16	213M		8x P100 (3.5 days)
BERT-Base	12	768	12	110M	13 GB	
BERT-Large	24	1024	16	340M	13 GB	
XLNet-Large	24	1024	16	~340M	126 GB	512x TPU-v3 (2.5 days)
RoBERTa	24	1024	16	355M	160 GB	1024x V100 GPU (1 day)
GPT-2	48	1600	?	1.5B	40 GB	
Megatron-LM	72	3072	32	8.3B	174 GB	512x V100 GPU (9 days)
Turing-NLG	78	4256	28	17B	?	256x V100 GPU
GPT-3	96	12288	96	175B	694GB	?

Plenty of website demos: ChatGPT, GPT4, Llama2, etc.

Improving Computer Vision:



VIT, MaskFormer, MaskGIT, etc.

Quiz:

Quiz:

- What is attention?

Quiz:

- What is attention?
- Properties of self-attention?

Quiz:

- What is attention?
- Properties of self-attention?
- What is a transformer?

Important topics of this lecture

Important topics of this lecture

- Getting to know different attention mechanisms

Important topics of this lecture

- Getting to know different attention mechanisms
- Understanding the transformer layer