Advanced Machine Learning

Bilel GUETARNI, PhD

bilel.guetarni@junia.com

JUNIA ISEN

Summary

Introduction

Recurrent Neural Network

Transformers

i. Input embedding

ii. Self-attention mechanism

iii. Multi-head attention

iv. Feed Forward Network

v. Residual connection and Layer Normalization

vi. Positional encoding

Lab session

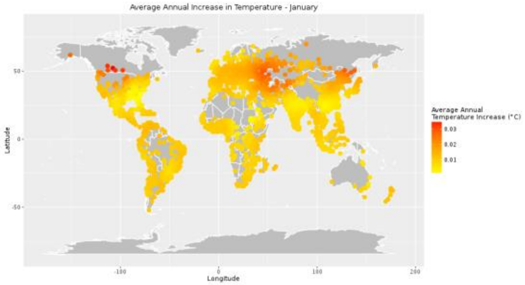
Introduction

Different data structures

Introduction

Different data structures

Images, categorical, map, ...



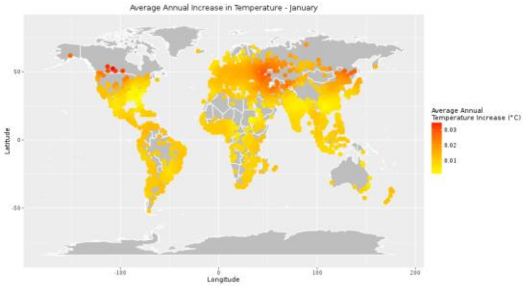
# age	A sex	# bmi	# children	✓ smoker
19	female	27.9	0	yes
18	male	33.77	1	no
28	male	33	3	no

kaggle WIKIPEDIA

Introduction

Different data structures

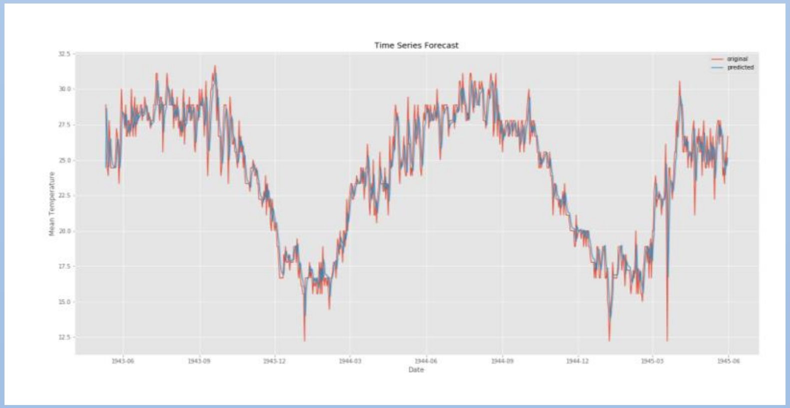
Images, categorical, map, ...

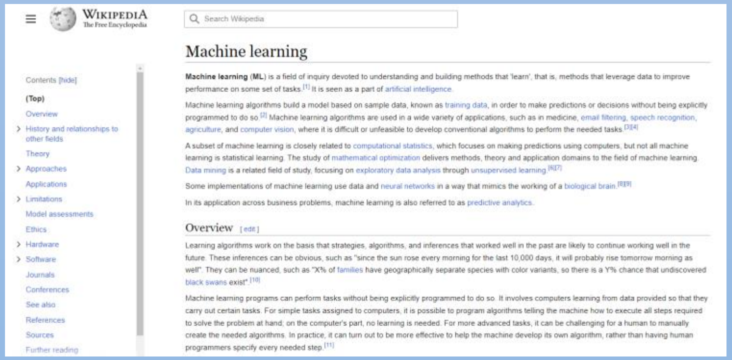


# age	A sex	# bmi	# children	✓ smoker
19	female	27.9	0	yes
18	male	33.77	1	no
28	male	33	3	no

Sequential data

Time series, text, ...





WIKIPEDIA The Free Encyclopedia

Search Wikipedia

Contents [hide]
(Top)
Overview
History and relationships to other fields
Theory
Approaches
Applications
Limitations
Model assessments
Ethics
Hardware
Software
Journals
Conferences
See also
References
Sources
Further reading

Machine learning

Machine learning (**ML**) is a field of inquiry devoted to understanding and building methods that 'learn', that is, methods that leverage data to improve performance on some set of tasks.[1] It is seen as a part of artificial intelligence.

Machine learning algorithms build a model based on sample data, known as training data, in order to make predictions or decisions without being explicitly programmed to do so.[2] Machine learning algorithms are used in a wide variety of applications, such as in medicine, email filtering, speech recognition, agriculture, and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.[3][4]

A subset of machine learning is closely related to computational statistics, which focuses on making predictions using computers, but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to the field of machine learning. Data mining is a related field of study, focusing on exploratory data analysis through unsupervised learning.[6][7]

Some implementations of machine learning use data and neural networks in a way that mimics the working of a biological brain.[8][9]

In its application across business problems, machine learning is also referred to as predictive analytics.

Overview [edit]

Learning algorithms work on the basis that strategies, algorithms, and inferences that worked well in the past are likely to continue working well in the future. These inferences can be obvious, such as "since the sun rose every morning for the last 10,000 days, it will probably rise tomorrow morning as well". They can be nuanced, such as "X% of families have geographically separate species with color variants, so there is a Y% chance that undiscovered black swans exist".[10]

Machine learning programs can perform tasks without being explicitly programmed to do so. It involves computers learning from data provided so that they carry out certain tasks. For simple tasks assigned to computers, it is possible to program algorithms telling the machine how to execute all steps required to solve the problem at hand; on the computer's part, no learning is needed. For more advanced tasks, it can be challenging for a human to manually create the needed algorithms. In practice, it can turn out to be more effective to help the machine develop its own algorithm, rather than having human programmers specify every needed step.[11]

kaggle WIKIPEDIA

Notations and symbols

x : a scalar or vector

$\{x_1, x_2, \ldots, x_n\}$: a sequence of n vectors

$X = (x_1, x_2, \ldots, x_n)$: a matrix with vectors as its columns

A^T : the transpose of a matrix

AB : matrix product between A and B

σ : an activation function (e.g. softmax)

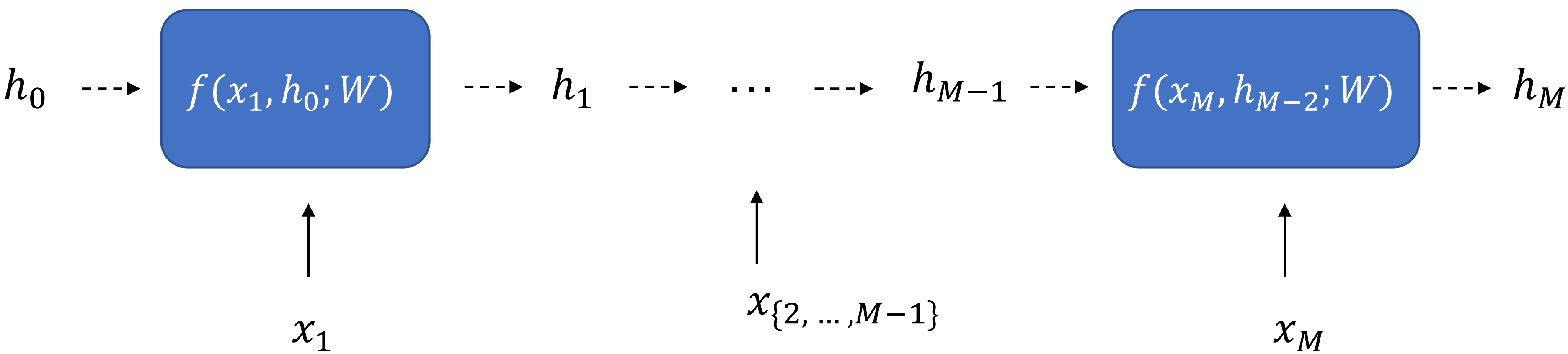
Recurrent Neural Network

A Recurrent Neural Network (RNN) is a type of neural network that iterates over a sequence (of vector) while keeping an internal memory (or state).

Recurrent Neural Network

A Recurrent Neural Network (RNN) is a type of neural network that iterates over a sequence (of vector) while keeping an internal memory (or state).

If $\{x_1, \dots, x_M\}$ is an input sequence, a RNN iterate over the sequence elements as follows:



Recurrent Neural Network

A Recurrent Neural Network (RNN) is a type of neural network that iterates over a sequence (of vector) while keeping an internal memory (or state).

Popular architectures:

- *LSTM* : Long-Short Term Memory
- *GRU* : Gated Recurrent Unit

Recurrent Neural Network

A Recurrent Neural Network (RNN) is a type of neural network that iterates over a sequence (of vector) while keeping an internal memory (or state).

Popular architectures:

- *LSTM* : Long-Short Term Memory
- *GRU* : Gated Recurrent Unit

Limitations :

- Long-range dependencies
- Vanishing / exploding gradients

Recurrent Neural Network

Limitations :

- Long-range dependencies
- **Vanishing / exploding gradients**

*“ The problem with **vanilla RNNs** is computational (or practical) in nature: when training a vanilla RNN using back-propagation, the long-term gradients which are back-propagated can "vanish" (that is, they can tend to zero) or "explode" (that is, they can tend to infinity), because of the computations involved in the **process**, which use finite-precision numbers. “*

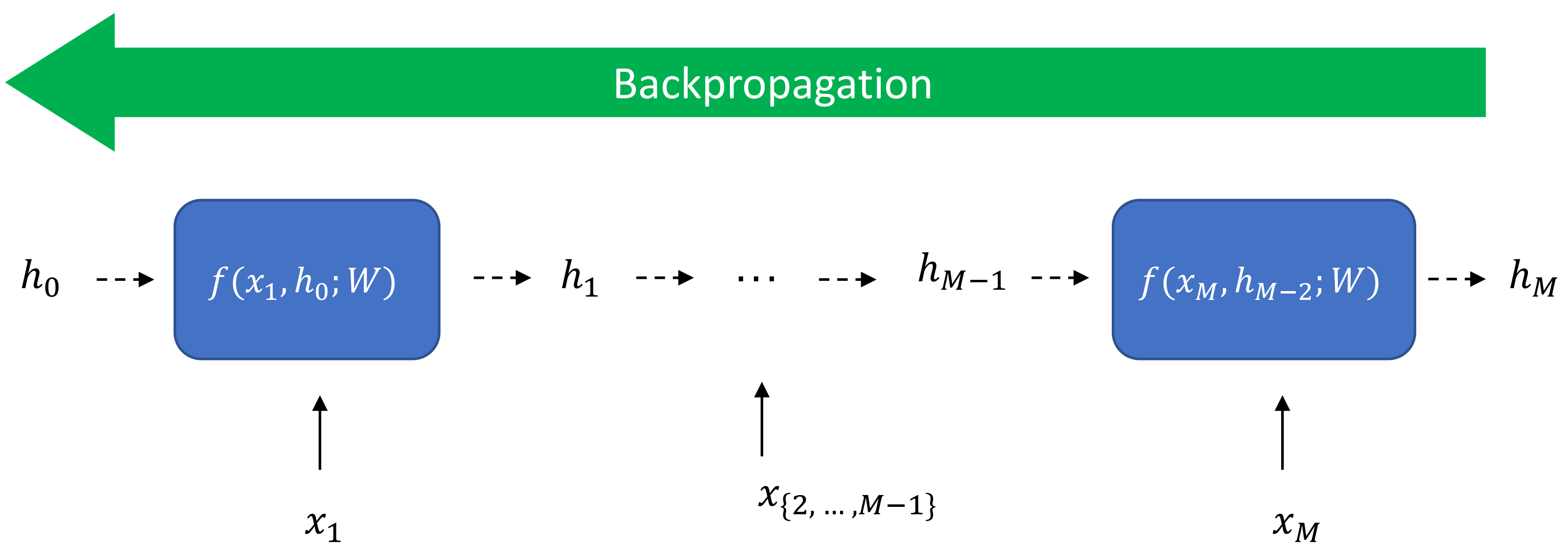
[Wikipedia - Long short-term memory]

Recurrent Neural Network

Limitations :

- Long-range dependencies
- **Vanishing / exploding gradients**

$$\frac{dL}{dW} = \sum_i \frac{dL}{df(x_i,h_{i-1};W)} * \frac{df(x_i,h_{i-1};W)}{dW}$$



Transformers [1]

Published in *Neural Information Processing Systems 2017*

Google Brain
Google Research
University of Toronto

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Transformers [1]

Architecture components

- input embedding
- position encoding
- self-attention (attention mechanism)
- feed forward (MLP)

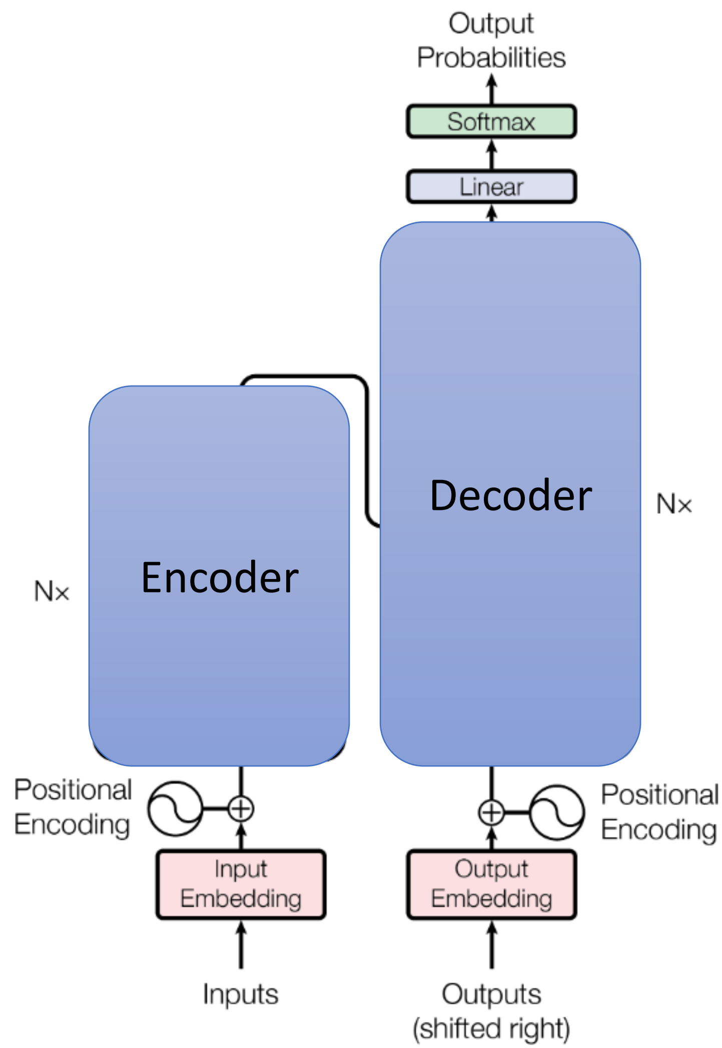


Figure 1: The Transformer - model architecture.

Transformers [1]

Architecture components

- input embedding
- position encoding
- self-attention (attention mechanism)
- feed forward (MLP)

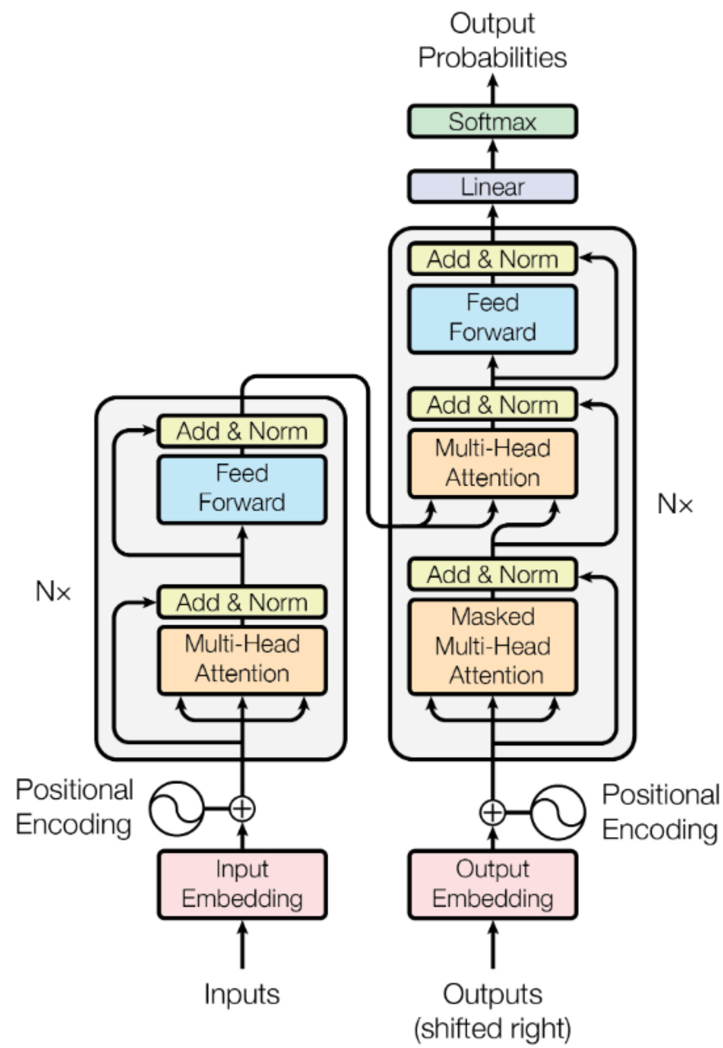


Figure 1: The Transformer - model architecture.

Transformers [1]

Architecture components

- input embedding
- position encoding
- self-attention (attention mechanism)
- feed forward (MLP)

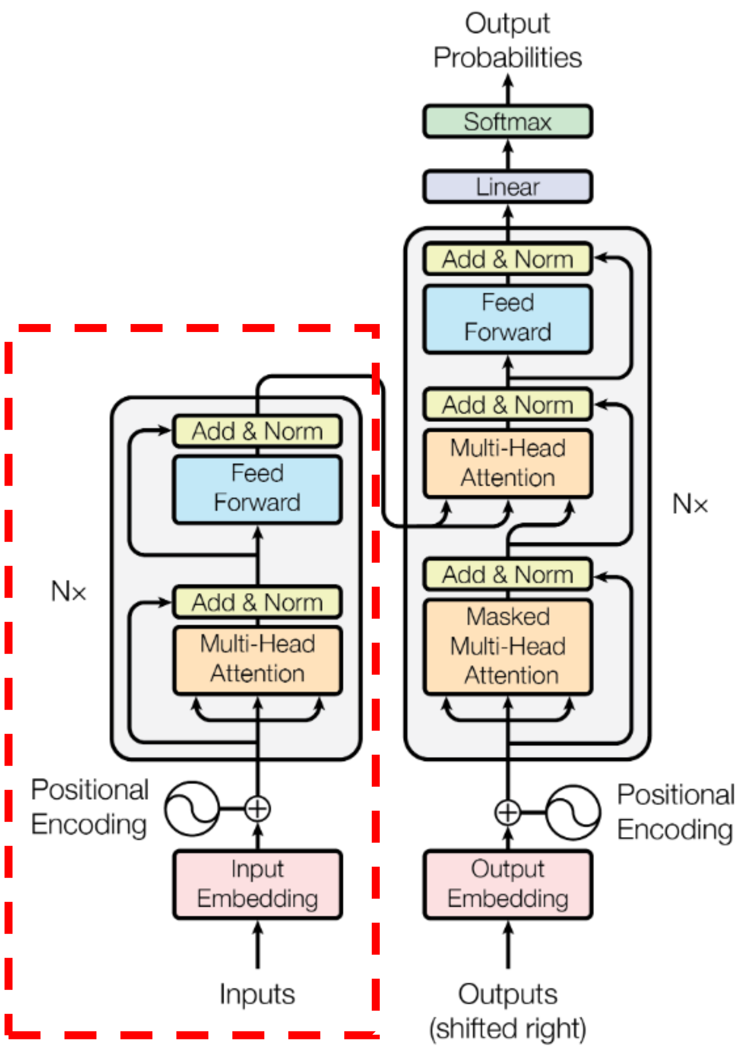
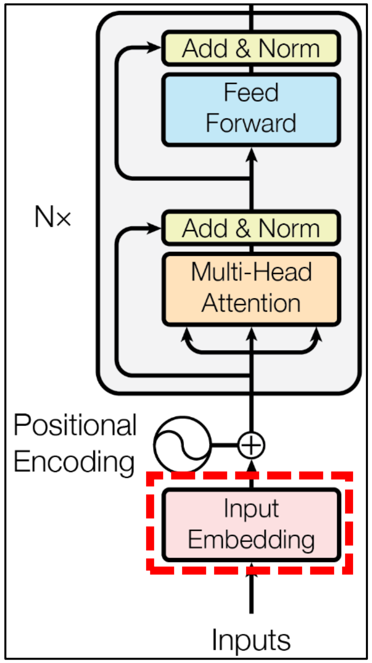


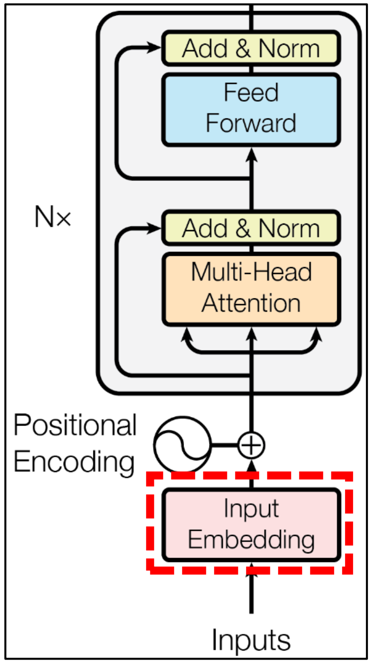
Figure 1: The Transformer - model architecture.

Input Embedding

How to feed a word (*Natural Language Processing*) to a neural network ?



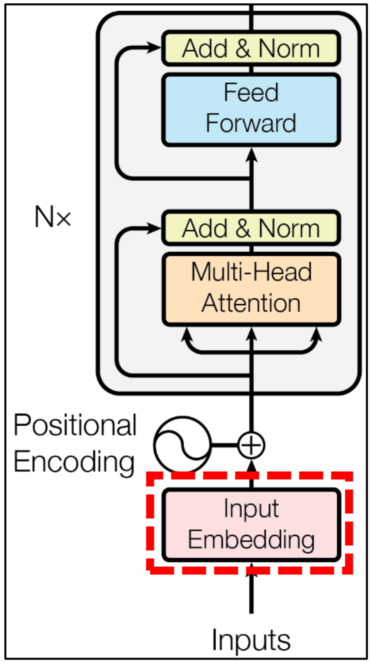
Input Embedding



How to feed a word (*Natural Language Processing*) to a neural network ?

One-hot encoding + embedding

Input Embedding



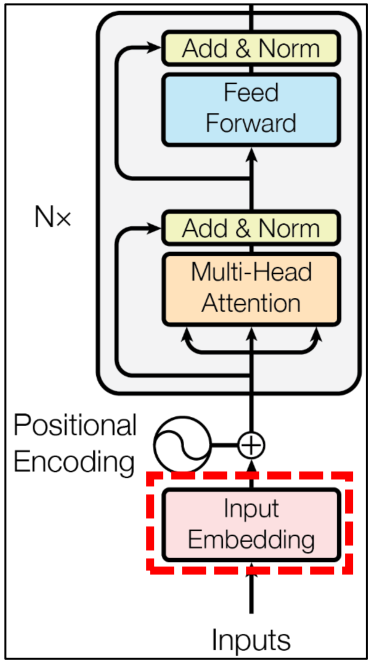
How to feed a word (*Natural Language Processing*) to a neural network ?

One-hot encoding + embedding

Vocabulary

	index		one-hot	R^C vectors
horse	→	1	→	(1 0 0 ... 0)
car	→	2	→	(0 1 0 ... 0)
bed	→	3	→	(0 0 1 ... 0)
looking	→	4	→	(0 0 0 ... 0)
...		...		
running	→	C	→	(0 0 0 ... 1)

Input Embedding



How to feed a word (*Natural Language Processing*) to a neural network ?

One-hot encoding + embedding

$x = (1 \;\; 0 \;\; 0 \;\; \dots \;\; 0)$

one-hot encoded word

embedding
(linear projection)

$$\xrightarrow{\hspace{4cm}}$$

$W^T x, \qquad W \in R^{V,C}$

$W^T x$ is the embedding vector (of dimension V) of the word encoded by x

Self-attention

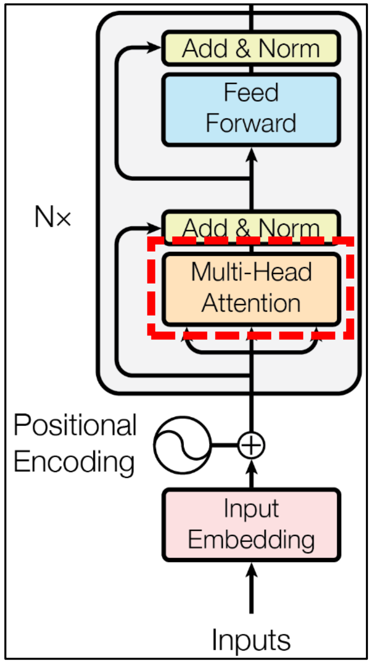
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8

encode in a new representation
that include context

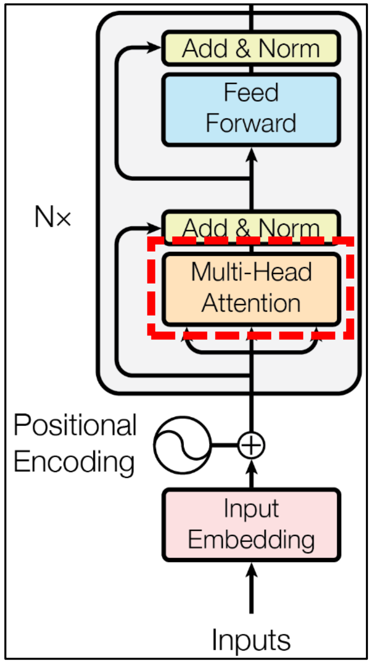
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8

embedding
(+ position encoding)

The dog is playing in the garden .



Self-attention



$x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8$

encode in a new representation that include context

$x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8$

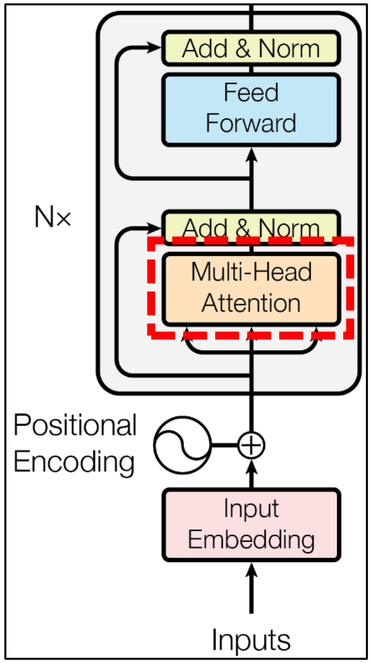
The sequence elements are not aware of one another, i.e. the representations are context-free.

embedding
(+ position encoding)

The dog is playing in the garden .

Self-attention

x_1 x_2 x_3 . . . x_N

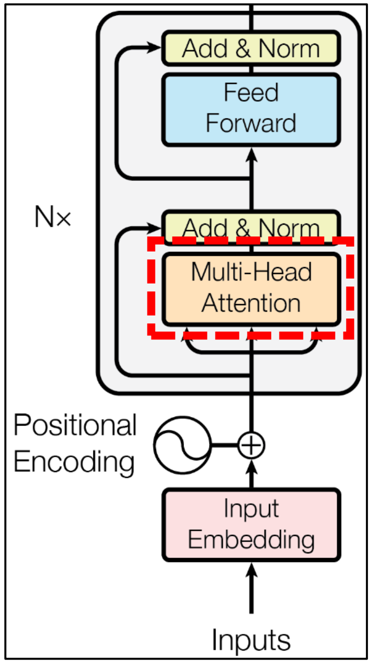


Self-attention

x_1 x_2 x_3 . . . x_N

similarity

$x_1 . x_1$ $x_1 . x_2$ $x_1 . x_3$ . . . $x_1 . x_N$



Self-attention

x_1 x_2 x_3 . . . x_N

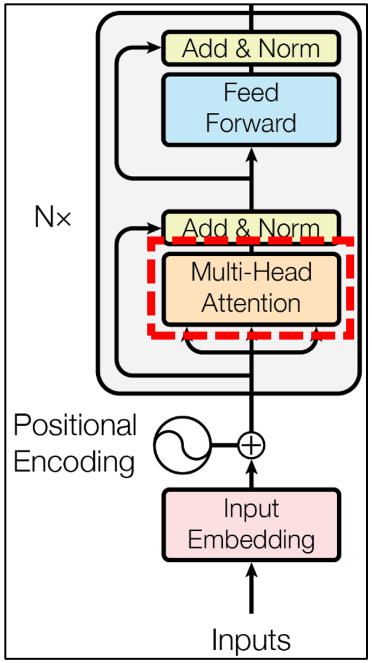
similarity

$x_1 . x_1$ $x_1 . x_2$ $x_1 . x_3$ . . . $x_1 . x_N$

softmax

normalized similarity

a_1 a_2 a_3 . . . a_N



Self-attention

x_1 x_2 x_3 . . . x_N

similarity

$x_1.x_1$	$x_1.x_2$	$x_1.x_3$	. . .	$x_1.x_N$

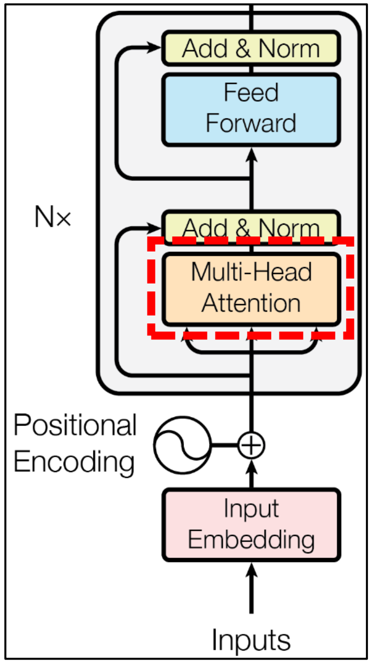
softmax

normalized similarity

a_1	a_2	a_3	. . .	a_N

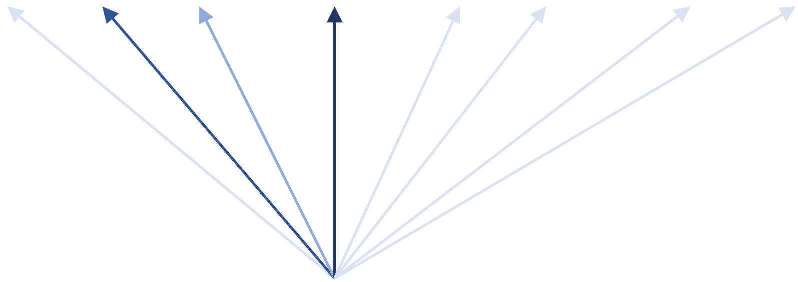
$$x_1 := \sum_i a_i x_i$$

integrate context into representation

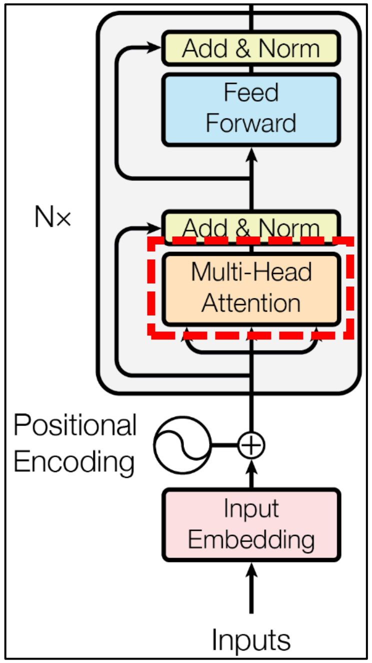


Self-attention

The dog is playing in the garden .

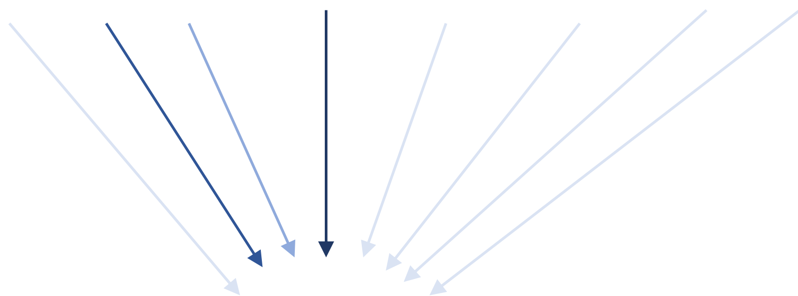


The dog is ***playing*** *in the garden .*



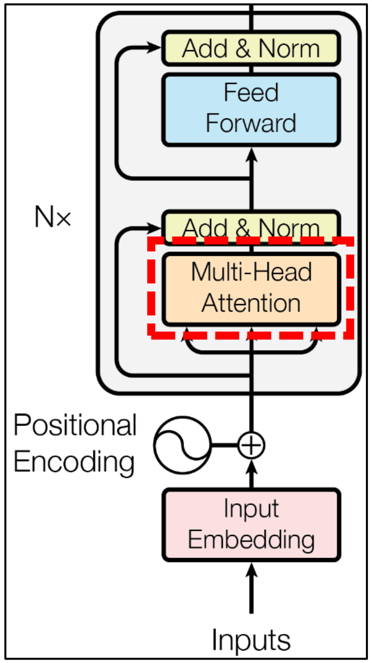
Self-attention

The dog is playing in the garden .



The dog is ***playing*** *in the garden .*

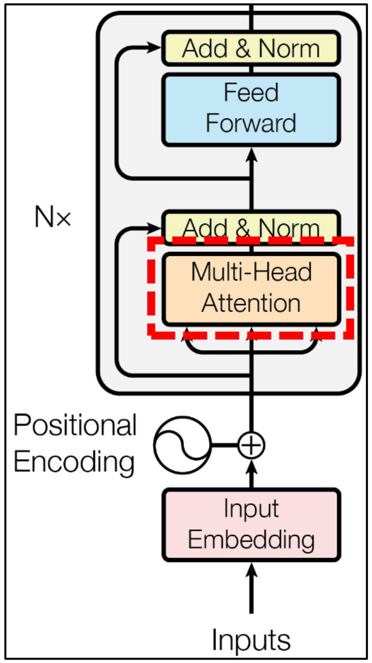
playing
dog
is
the
in
the
garden
.



Self-attention

Let's put everything in matrices !

$$X = (x_1, x_2, \dots, x_N)$$



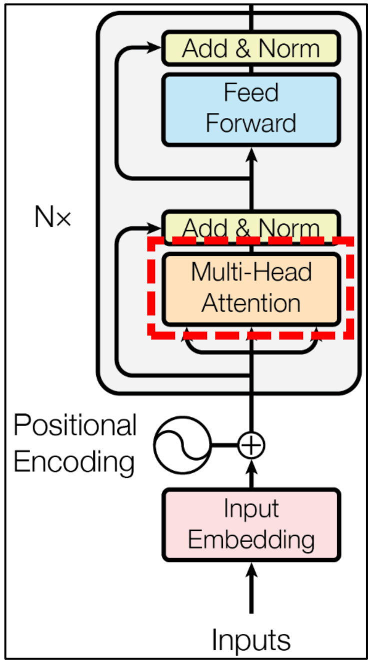
Self-attention

Let's put everything in matrices !

$$X = (x_1, x_2, \dots, x_N)$$

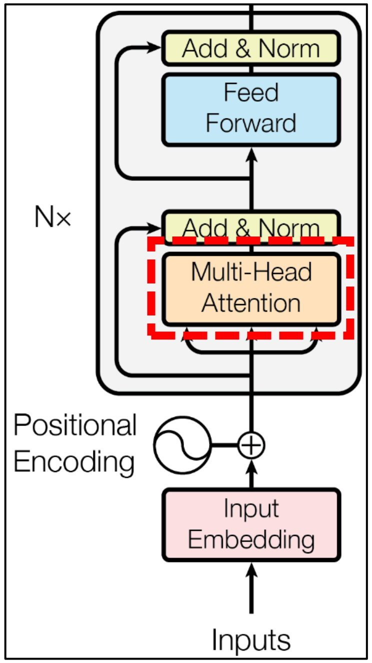
similarity

$$X^T X = \begin{pmatrix} x_1 x_1 & \cdots & x_1 x_N \\ \vdots & \ddots & \vdots \\ x_N x_1 & \cdots & x_N x_N \end{pmatrix}$$



Self-attention

Let's put everything in matrices !



$$X = (x_1, x_2, \dots, x_N)$$

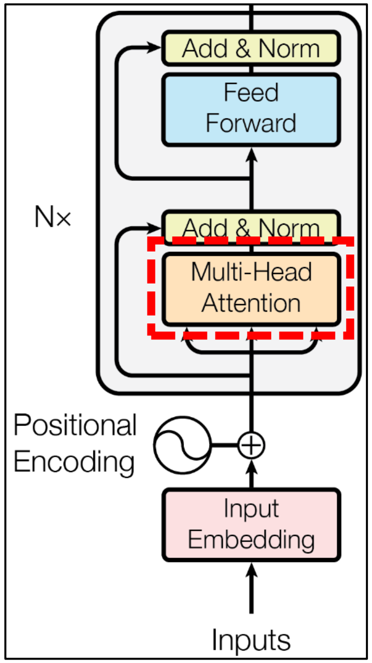
similarity

$$X^T X = \begin{pmatrix} x_1 x_1 & \cdots & x_1 x_N \\ \vdots & \ddots & \vdots \\ x_N x_1 & \cdots & x_N x_N \end{pmatrix}$$

softmax (row-wise)

$$\sigma(X^T X) = \sigma \begin{pmatrix} x_1 x_1 & \cdots & x_1 x_N \\ \vdots & \ddots & \vdots \\ x_N x_1 & \cdots & x_N x_N \end{pmatrix}$$

Self-attention



Let's put everything in matrices !

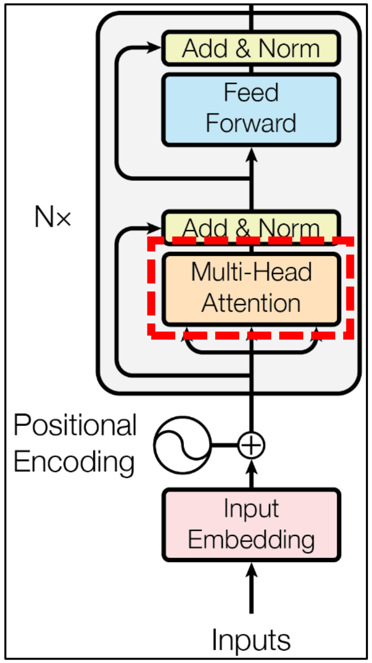
$$X := \sigma(X^T X)X$$

Add learnable weights to learn how to perform self-attention

$$X := \sigma\left(W_Q X^T \; W_K X\right) \; W_V X$$

Self-attention

Let's put everything in matrices !



$$X \; := \sigma(X^T X)X$$

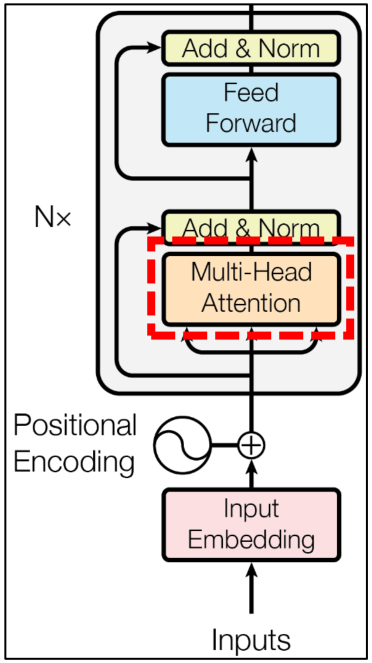
Add learnable weights to learn how to perform self-attention

$$X \; := \sigma\left(\frac{W_Q X^T \; W_K X}{\sqrt{d_{model}}}\right) W_V X$$

scale activations

Multi-Head Attention

Let's $\mathbf{head_i} = \boldsymbol{Attention}(\boldsymbol{X}; \boldsymbol{W_i^Q}, \boldsymbol{W_i^K}, \boldsymbol{W_i^V})$ be the self-attention mechanism (with parameters W_i^Q, W_i^K, W_i^V) previously discussed.

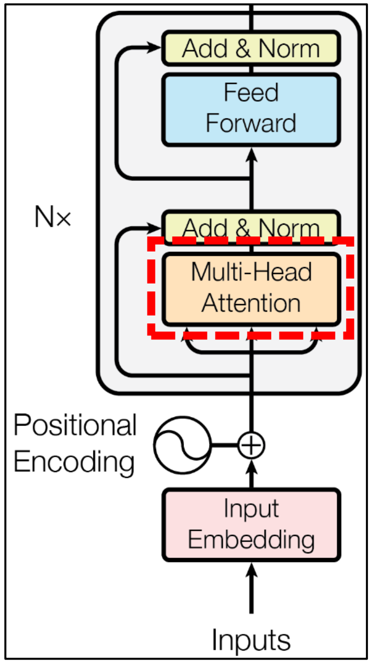


Multi-Head Attention

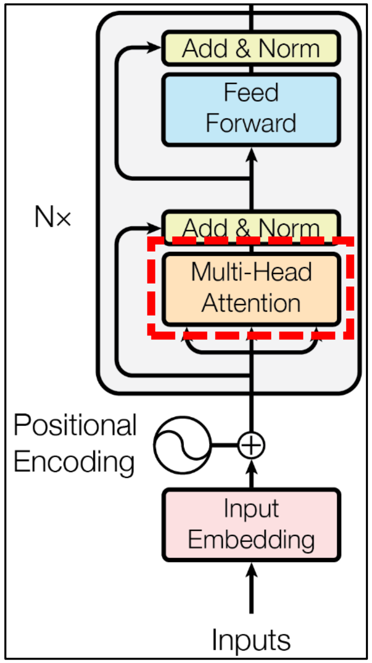
Let's $\mathbf{head_i} = \boldsymbol{Attention}(\boldsymbol{X}; \boldsymbol{W_i^Q}, \boldsymbol{W_i^K}, \boldsymbol{W_i^V})$ be the self-attention mechanism (with parameters W_i^Q, W_i^K, W_i^V) previously discussed.

We perform this operation with different sets of parameters in order to attend to different information:

$$head_1 = Attention(X; W_1^Q, W_1^K, W_1^V)$$
$$head_2 = Attention(X; W_2^Q, W_2^K, W_2^V)$$
$$head_3 = Attention(X; W_3^Q, W_3^K, W_3^V)$$
$$\ldots$$



Multi-Head Attention



Let's $\mathbf{head_i} = \boldsymbol{Attention}(\boldsymbol{X}; \boldsymbol{W_i^Q}, \boldsymbol{W_i^K}, \boldsymbol{W_i^V})$ be the self-attention mechanism (with parameters W_i^Q, W_i^K, W_i^V) previously discussed.

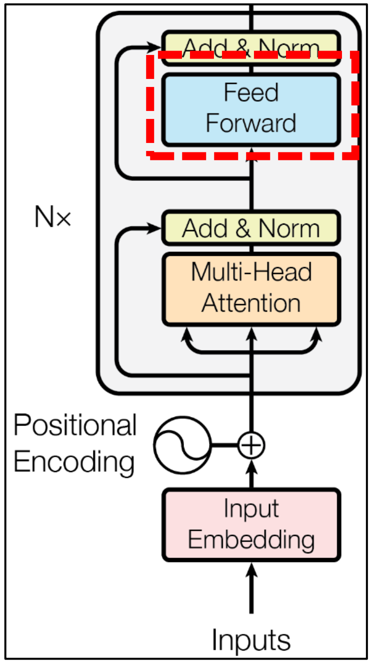
We perform this operation with different sets of parameters in order to attend to different information:

$$head_1 = Attention(X; W_1^Q, W_1^K, W_1^V)$$
$$head_2 = Attention(X; W_2^Q, W_2^K, W_2^V)$$
$$head_3 = Attention(X; W_3^Q, W_3^K, W_3^V)$$
$$\dots$$

We than concatenate (and project) to produce the output of the Multi-Head Attention:

$$MultiHead(X) = concat(head_1, \dots., head_h)W^O$$

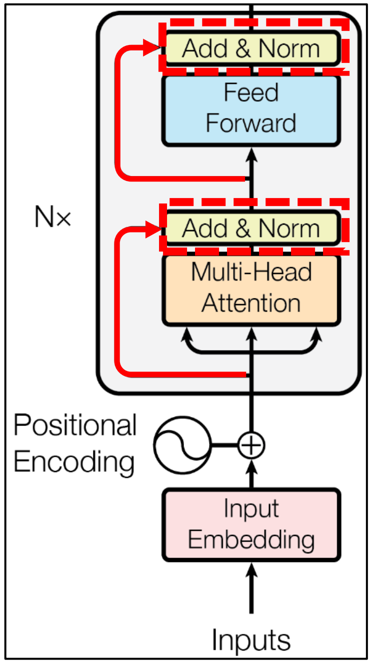
Feed Forward Network



The FFN is a simple Multi Layer Perceptron with 2 layers and a *ReLU* activation between:

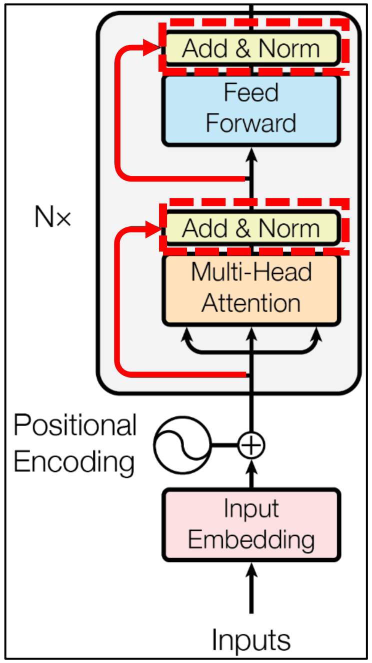
$$FFN(X) = Linear_2(ReLU(Linear_1(X)))$$

Residual connection and Layer Normalization

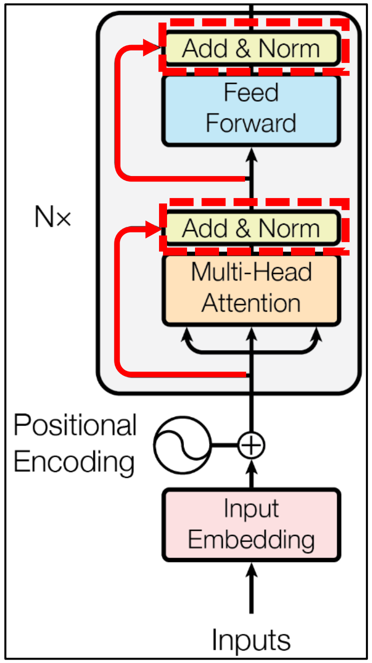


Residual connection and Layer Normalization

Residual connection [2] is a shortcut connection to let the gradient flow back through the layers with lower risks of **vanishing gradient**. If $F(X;W)$ is a layer parametrized by W that perform some operation on X (linear, convolution, ...), then a residual connection is $Y = F(X;W) + X$



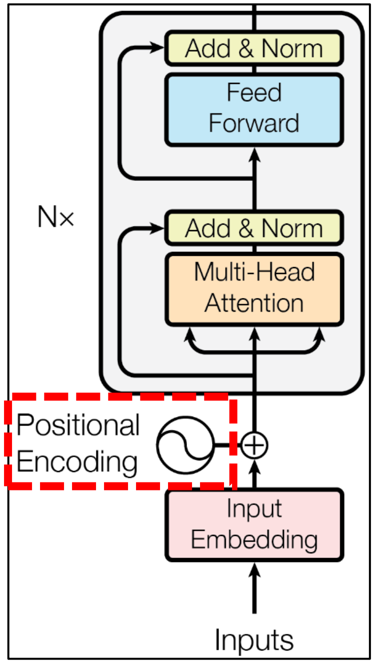
Residual connection and Layer Normalization



Residual connection [2] is a shortcut connection to let the gradient flow back through the layers with lower risks of **vanishing gradient**. If $F(X;W)$ is a layer parametrized by W that perform some operation on X (linear, convolution, ...), then a residual connection is $Y = F(X;W) + X$

Layer Normalization [3] is a learned normalization operation (similar to Batch Normalization [4]) that normalize the input across all units (i.e. neurons).

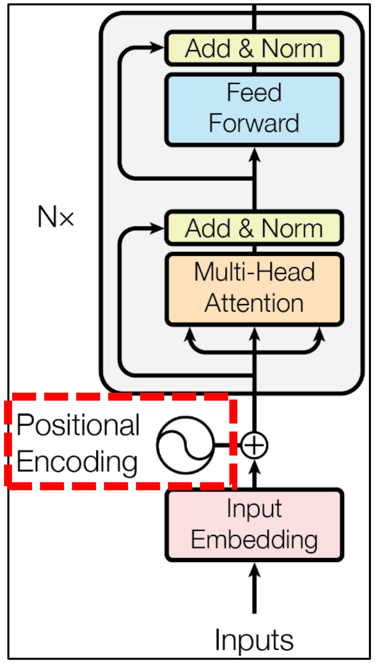
Positional Encoding



Does the relative position between two elements in a sequence matter when performing self-attention ?

In a text, two elements that are close may relate to each other. How to inform the model about their relative position ?

Positional Encoding



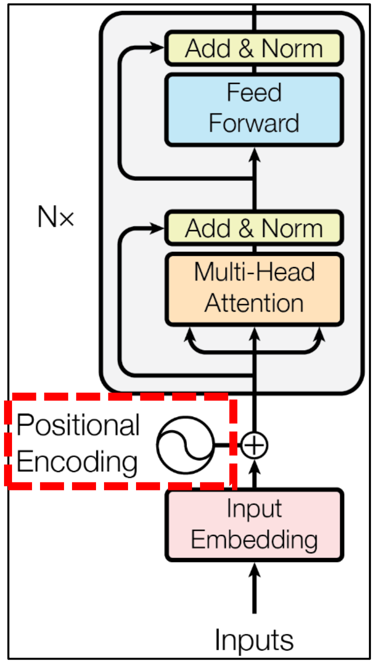
Does the relative position between two elements in a sequence matter when performing self-attention ?

In a text, two elements that are close may relate to each other.
How to inform the model about their relative position ?

$$x_i \coloneqq x_i + p_i$$

$$p_{i,\,j} = \begin{cases} \sin\left(\dfrac{i}{10000^{2j/d_{model}}}\right) & \text{, if } j \text{ is even} \\ \cos\left(\dfrac{i}{10000^{2j/d_{model}}}\right) & \text{, if } j \text{ is odd} \end{cases}$$

Positional Encoding



Does the relative position between two elements in a sequence matter when performing self-attention ?

In a text, two elements that are close may relate to each other.
How to inform the model about their relative position ?

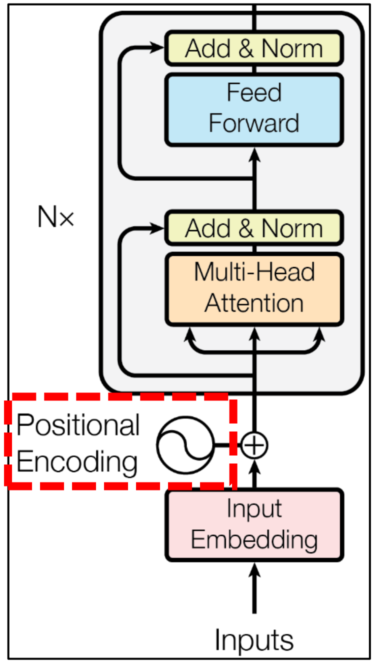
$$x_i \coloneqq x_i + p_i$$

$$p_{i,\,j} = \begin{cases} \sin(\frac{i}{10000^{2j/d_{model}}}) & \text{, if } j \text{ is even} \\ \cos(\frac{i}{10000^{2j/d_{model}}}) & \text{, if } j \text{ is odd} \end{cases}$$

$$p_1 = \begin{pmatrix} \cos(\frac{1}{10000^{2*1\,/\,d_{model}}}) \\ sin(\frac{1}{10000^{2*2\,/\,d_{model}}}) \\ \dots \\ sin(\frac{1}{10000^{2*d_{model}/\,d_{model}}}) \end{pmatrix}$$

Positional Encoding

"We chose this function because we hypothesized it would allow the model to easily learn to attend by relative positions, since for any fixed offset k, p_{i+k} can be represented as a linear function of p_i."

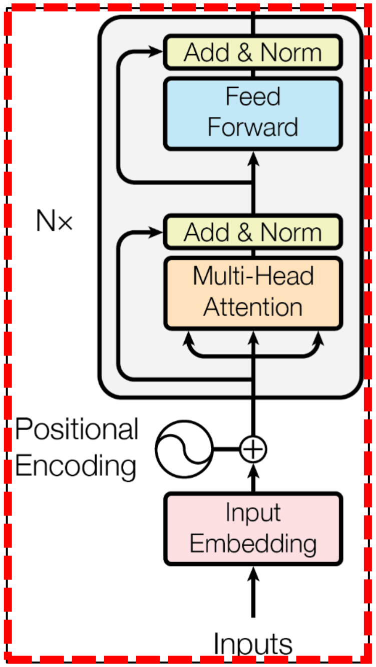


$$x_i \coloneqq x_i + p_i$$

$$p_{i,j} = \begin{cases} \sin\left(\frac{i}{10000^{2j/d_{model}}}\right) & \text{, if } j \text{ is even} \\ \cos\left(\frac{i}{10000^{2j/d_{model}}}\right) & \text{, if } j \text{ is odd} \end{cases}$$

$$p_1 = \begin{pmatrix} \cos\left(\frac{1}{10000^{2*1\,/\,d_{model}}}\right) \\ sin\left(\frac{1}{10000^{2*2\,/\,d_{model}}}\right) \\ \cdots \\ sin\left(\frac{1}{10000^{2*d_{model}/\,d_{model}}}\right) \end{pmatrix}$$

Encoder summary



Let $X = \{x_1, ..., x_M\}$ be the input sequence. The encoder proceeds as follow:

1. $X^0 := Embedding(X) + Pos(X)$
2. For l in 1 … N :
   - $Z^l := LayerNorm(MultiHead(X^{l-1}) + X^{l-1})$
   - $X^l := LayerNorm(FFN(Z^l) + Z^l)$

The output is denoted as X^N

Transformers [1]

How to perform a **classification** (or regression) task on a sequence ?

Transformers [1]

How to perform a **classification** (or regression) task on a sequence ?

Straightforward solution: Global Average Pooling (after encoder)

$$x = \frac{1}{M}\sum_i x_i \quad \text{for} \quad x_i \in X$$

Transformers [1]

How to perform a **classification** (or regression) task on a sequence ?

Straightforward solution: Global Average Pooling (after encoder)

$$x = \frac{1}{M}\sum_i x_i \quad \text{for} \quad x_i \in X$$

Learnable: class embedding (before encoder)

$X = \{x^*, x_1, \ldots, x_M\}$ where x^* is a learnable token that is used afterwards for the task.

Additional information

- Sequence elements are called **tokens** in NLP.
- The decoder is useful for sequence-to-sequence tasks (e.g. translation).
- Transformers are data-hungry: they need a considerable amount of training data to perform well, but they will usually perform better than other architectures.

References

[1] A. Vaswani et al., « Attention Is All You Need ». arXiv, 5 décembre 2017. doi: 10.48550/arXiv.1706.03762.

[2] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[3] Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer normalization." arXiv preprint arXiv:1607.06450 (2016).

[4] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International conference on machine learning. pmlr, 2015.

Further reading

Dive into Deep Learning by Aston Zhang, Zachary C. Lipton, Mu Li and Alexander J. Smola (2020)

Transformer: A Novel Neural Network Architecture for Language Understanding by Jakob Uszkoreit, Google (2017)

Formal Algorithms for Transformers by Mary Phuong and Marcus Hutter, DeepMind (2022)

Lab session

1. Code a Transformer Encoder model with:
   - tfm.nlp.layers.TransformerEncoderBlock
   - tf.keras.layers.Embedding
   - the positional encoding presented in the article
   - tf.keras.layers.GlobalAveragePooling1D

2. Train the model on the Reuters newswire classification dataset

   tip: don't forget to pad the sequences for batching !

3. Try different training hyperparameters (vectors dimension, number of heads, optimizer, etc.) and compare them based on a metric of choice (justify the metric used).

If enough time:

- Find the error in the Tensorflow-NLP version of the notebook.
- Visualize the loss and metric after training.