The Transformer Model

by **Stefania Cristina** on September 18, 2022 in **Attention**

Last Updated on January 6, 2023

We have already familiarized ourselves with the concept of self-attention as implemented by the Transformer attention mechanism for neural machine translation. We will now be shifting our focus to the details of the Transformer architecture itself to discover how self-attention can be implemented without relying on the use of recurrence and convolutions.

In this tutorial, you will discover the network architecture of the Transformer model.

After completing this tutorial, you will know:

- How the Transformer architecture implements an encoder-decoder structure without recurrence and convolutions
- How the Transformer encoder and decoder work
- How the Transformer self-attention compares to the use of recurrent and convolutional layers

Kick-start your project with my book Building Transformer Models with Attention. It provides **self-study tutorials** with **working code** to guide you into building a fully-working transformer model that can
translate sentences from one language to another...

Let's get started.

The Transformer Model

Photo by Samule Sun, some rights reserved.

Tutorial Overview

This tutorial is divided into three parts; they are:

- The Transformer Architecture
  - The Encoder
  - The Decoder
- Sum Up: The Transformer Model
- Comparison to Recurrent and Convolutional Layers

Machine Translator in 12 Days

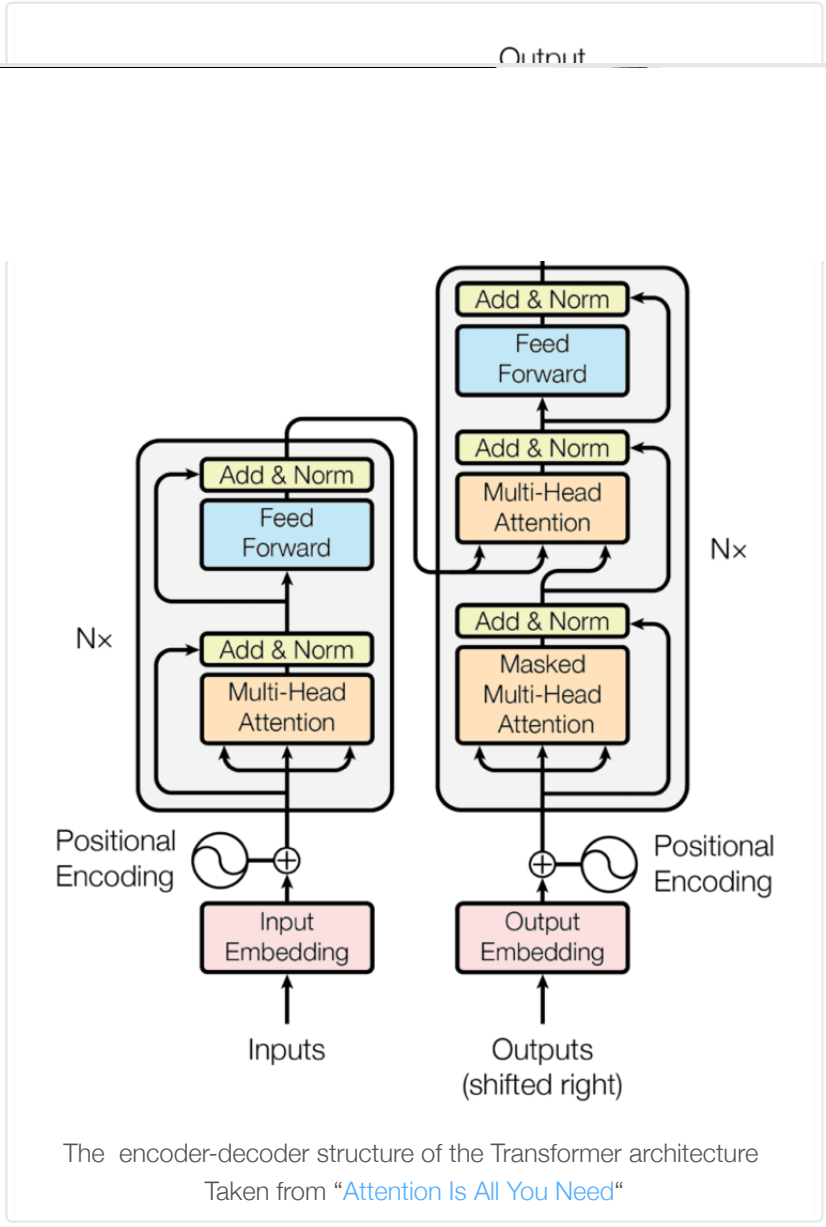
Prerequisites

For this tutorial, we assume that you are already familiar with:

- The concept of attention
- The attention mechanism
- The Transformer attention mechanism

The Transformer Architecture

The Transformer architecture follows an encoder-decoder structure but does not rely on recurrence and convolutions in order to generate an output.



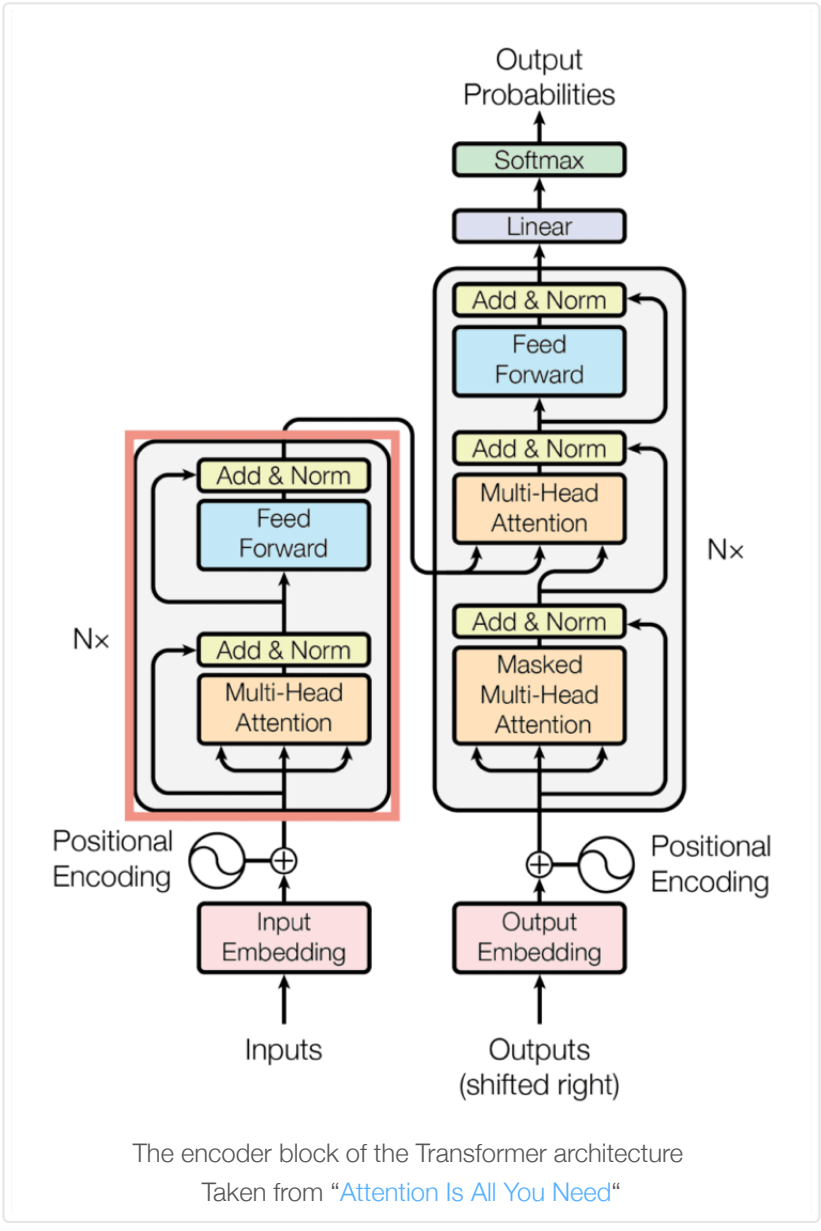
The encoder-decoder structure of the Transformer architecture
Taken from "Attention Is All You Need"

In a nutshell, the task of the encoder, on the left half of the Transformer architecture, is to map an input sequence to a sequence of continuous representations, which is then fed into a decoder.

The decoder, on the right half of the architecture, receives the output of the encoder together with the decoder output at the previous time step to generate an output sequence.

> *At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next.*
>
> – *Attention Is All You Need, 2017.*

The Encoder



The encoder block of the Transformer architecture
Taken from "Attention Is All You Need"

The encoder consists of a stack of N = 6 identical layers, where each layer is composed of two sublayers:

1. The first sublayer implements a multi-head self-attention mechanism. You have seen that the multi-head mechanism implements h heads that receive a (different) linearly projected version of the queries, keys, and values, each to produce h outputs in parallel that are then used to generate a final result.

2. The second sublayer is a fully connected feed-forward network consisting of two linear transformations with Rectified Linear Unit (ReLU) activation in between:

$$\text{FFN}(x) = \text{ReLU}(\mathbf{W}_1 x + b_1)\mathbf{W}_2 + b_2$$

The six layers of the Transformer encoder apply the same linear transformations to all the words in the input sequence, but *each* layer employs different weight ($\mathbf{W}_1, \mathbf{W}_2$) and bias ($b_1, b_2$) parameters to do so.

Furthermore, each of these two sublayers has a residual connection around it.

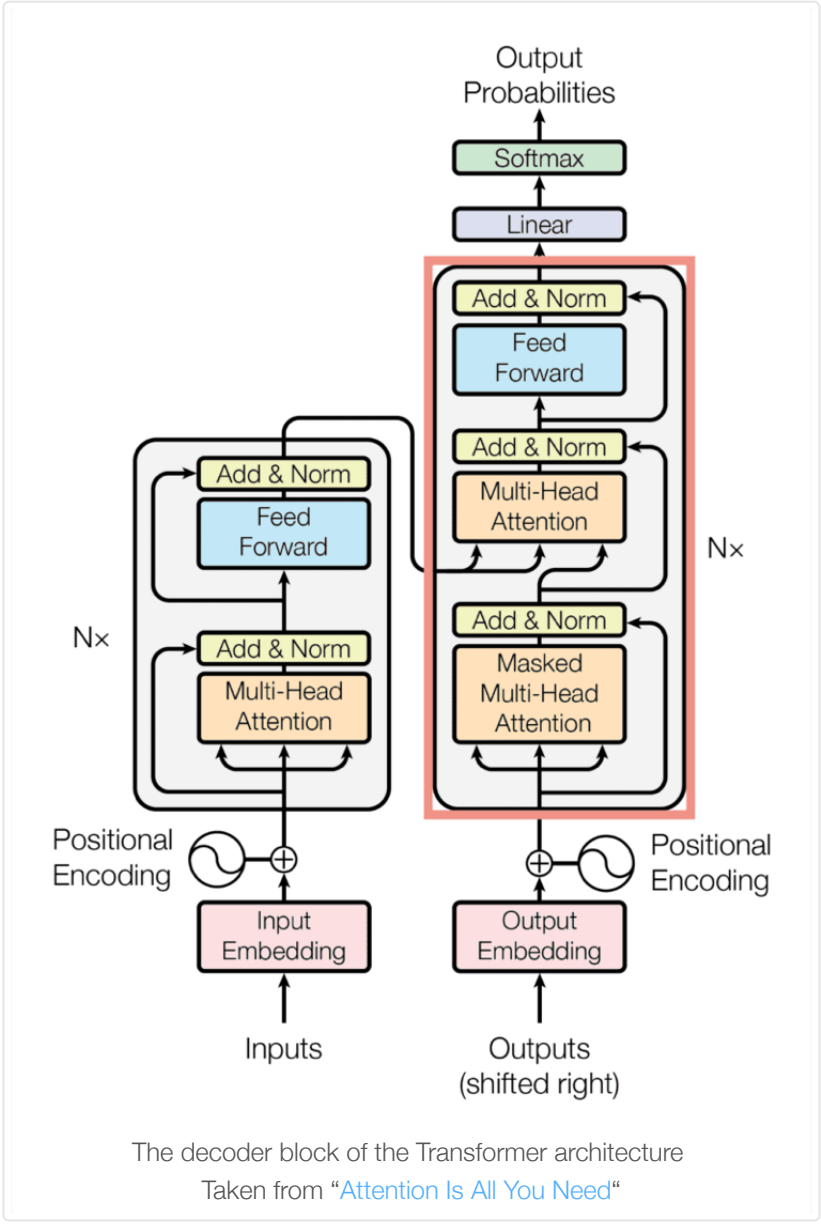
Each sublayer is also succeeded by a normalization layer, $\text{layernorm}(.)$, which normalizes the sum computed between the sublayer input, x, and the output generated by the sublayer itself, $\text{sublayer}(x)$:

$$\text{layernorm}(x + \text{sublayer}(x))$$

An important consideration to keep in mind is that the Transformer architecture cannot inherently capture any information about the relative positions of the words in the sequence since it does not make use of recurrence. This information has to be injected by introducing *positional encodings* to the input embeddings.

The positional encoding vectors are of the same dimension as the input embeddings and are generated using sine and cosine functions of different frequencies. Then, they are simply summed to the input embeddings in order to *inject* the positional information.

The Decoder



The decoder block of the Transformer architecture
Taken from "Attention Is All You Need"

The decoder shares several similarities with the encoder.

The decoder also consists of a stack of $N = 6$ identical layers that are each composed of three sublayers:

1. The first sublayer receives the previous output of the decoder stack, augments it with positional information, and implements multi-head self-attention over it. While the encoder is designed to attend to all words in the input sequence *regardless* of their position in the sequence, the decoder is modified to attend *only* to the preceding words. Hence, the prediction for a word at position i can only depend on the known outputs for the words that come before it in the sequence. In the multi-head attention mechanism (which implements multiple, single attention functions in parallel), this is achieved by introducing a mask over the values produced by the scaled multiplication of matrices $\mathbf{Q}$ and $\mathbf{K}$. This masking is implemented by suppressing the matrix values that would otherwise correspond to illegal connections:

$$\text{mask}(\mathbf{QK}^T) = \text{mask}\left(\begin{bmatrix} e_{11} & e_{12} & \dots & e_{1n} \\ e_{21} & e_{22} & \dots & e_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ e_{m1} & e_{m2} & \dots & e_{mn} \end{bmatrix}\right) = \begin{bmatrix} e_{11} & -\infty & \dots & -\infty \\ e_{21} & e_{22} & \dots & -\infty \\ \vdots & \vdots & \ddots & \vdots \\ e_{m1} & e_{m2} & \dots & e_{mn} \end{bmatrix}$$

The multi-head attention in the decoder implements several masked, single-attention functions
Taken from "Attention Is All You Need"

> *The masking makes the decoder unidirectional (unlike the bidirectional encoder).*
>
> – *Advanced Deep Learning with Python, 2019.*

2. The second layer implements a multi-head self-attention mechanism similar to the one implemented in the first sublayer of the encoder. On the decoder side, this multi-head mechanism receives the queries from the previous decoder sublayer and the keys and values from the output of the encoder. This allows the decoder to attend to all the words in the input sequence.

3. The third layer implements a fully connected feed-forward network, similar to the one implemented in the second sublayer of the encoder.

Furthermore, the three sublayers on the decoder side also have residual connections around them and are succeeded by a normalization layer.

Positional encodings are also added to the input embeddings of the decoder in the same manner as previously explained for the encoder.

Sum Up: The Transformer Model

The Transformer model runs as follows:

1. Each word forming an input sequence is transformed into a d_{model}-dimensional embedding vector.
2. Each embedding vector representing an input word is augmented by summing it (element-wise) to a positional encoding vector of the same d_{model} length, hence introducing positional information into the input.
3. The augmented embedding vectors are fed into the encoder block consisting of the two sublayers explained above. Since the encoder attends to all words in the input sequence, irrespective if they precede or succeed the word under consideration, then the Transformer encoder is *bidirectional*.
4. The decoder receives as input its own predicted output word at time-step, $t–1$.
5. The input to the decoder is also augmented by positional encoding in the same manner done on the encoder side.
6. The augmented decoder input is fed into the three sublayers comprising the decoder block explained above. Masking is applied in the first sublayer in order to stop the decoder from

attending to the succeeding words. At the second sublayer, the decoder also receives the output of the encoder, which now allows the decoder to attend to all the words in the input sequence.

7. The output of the decoder finally passes through a fully connected layer, followed by a softmax layer, to generate a prediction for the next word of the output sequence.

Comparison to Recurrent and Convolutional Layers

Vaswani et al. (2017) explain that their motivation for abandoning the use of recurrence and convolutions was based on several factors:

1. Self-attention layers were found to be faster than recurrent layers for shorter sequence lengths and can be restricted to consider only a neighborhood in the input sequence for very long sequence lengths.

2. The number of sequential operations required by a recurrent layer is based on the sequence length, whereas this number remains constant for a self-attention layer.

3. In convolutional neural networks, the kernel width directly affects the long-term dependencies that can be established between pairs of input and output positions. Tracking long-term dependencies would require using large kernels or stacks of convolutional layers that could increase the computational cost.

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

Books

- Advanced Deep Learning with Python, 2019.

Papers

- Attention Is All You Need, 2017.

Summary

In this tutorial, you discovered the network architecture of the Transformer model.

Specifically, you learned:

- How the Transformer architecture implements an encoder-decoder structure without recurrence and convolutions
- How the Transformer encoder and decoder work
- How the Transformer self-attention compares to recurrent and convolutional layers

Do you have any questions?
Ask your questions in the comments below, and I will do my best to answer.

More On This Topic

Building Transformer Models with Attention Crash…

How to Develop a CNN From Scratch for CIFAR-10 Photo…

Implementing the Transformer Decoder from Scratch in…

Inferencing the Transformer Model

Implementing the Transformer Encoder from Scratch in…

Multi-Label Classification of Satellite Photos of…

About Stefania Cristina

Stefania Cristina, PhD is a Lecturer with the Department of Systems and Control Engineering, at the University of Malta.

View all posts by Stefania Cristina →

attention, machine translation, transformer

‹ The Transformer Attention Mechanism

A Gentle Introduction to Positional Encoding in Transformer Models, Part 1 ›

19 Responses to *The Transformer Model*

Kelvin November 17, 2021 at 3:44 am #

REPLY

Hi Stefania Cristina, can you use the encoder block right away after taking convolved features from a backbone for object detection task excluding the decoder part?

Stefania Cristina November 18, 2021 at 7:47 pm #

REPLY

Hi Kelvin, in the computer vision domain it is the encoder block that is usually used, which receives several image patches in parallel, plus the corresponding positional encodings. The encoder output is then typically passed on to an MLP for classification. However, I have also encountered architectures whereby the encoder block is preceded by convolutional layers, which first extract the image features before passing them on to the encoder.

Saeid March 31, 2022 at 10:36 pm #

REPLY

Hi Stefania Cristina, Is it possible I use transformer model for time series classification? If it is ok, please guide me in coding. I'm a beginner in this field.
Thanks

James Carmichael April 1, 2022 at 9:08 am #

REPLY

Hi Saeid…You may find the following of interest:

https://towardsdatascience.com/multi-class-classification-with-transformers-6cf7b59a033a

Furkan Luleci May 13, 2022 at 4:44 am #

REPLY

In the text: "In convolutional neural networks, the kernel width directly affects the long-term dependencies that can be established between pairs of input and output positions. Tracking long-term dependencies would require the use of large kernels, or stacks of convolutional layers that could increase the computational cost"

I am not sure I get this paragraph correctly. The transformer model gets a sequential input e.g., text,

audio etc. Similarly, tr o use text audio types of input in CNNs, we use 1-D convolutions, which use single dimension kernels where the width is always 1. In this case, we only configure the height of the kernel where I mostly use 4 or 7.

From my perspective, I think that statement is not a valid comparison.

Thanks for the article

Stefania Cristina May 13, 2022 at 7:12 pm #

REPLY

Hi Furkan. The idea behind the attention mechanism is to draw *global dependencies* between the input and the output. This means that the process of predicting an output value will take into consideration the input data in its entirety (whether this is in the form of text, or an image etc). In this sense we can say that the Transformer can capture long-range dependencies, be it between words when these are distant in lenghty bodies of text, or pixels in distant parts of an image, etc.

In Vaswani et al.'s words, "A single convolutional layer with kernel width $k < n$ does not connect all pairs of input and output positions. Doing so requires a stack of $O(n/k)$ convolutional layers in the case of contiguous kernels…", where n is the sequence length. In CNNs, the receptive field depends on the size of the kernel. If one would want to capture long-range dependencies in an image by a CNN, for example, one would either require a large 2D kernel (covering a neighbourhood of k x k pixels) to widen the receptive field as much as possible, or stack long sequences of convolutional layers, both of which can be computationally costly. When dealing with text, you would be working with 1D kernels as you mention, however your kernel width (you referred to it as *height*) will still define the size of your receptive field, which is often a local neighbourhood around each token in a sequence.

Imene June 28, 2022 at 12:11 am #

REPLY

Thank you so much Sir, it's really brief and understandable!
Please, Could you explain about image classification using transformers with codes?

James Carmichael June 28, 2022 at 12:50 pm #

REPLY

Hi Imene…The following may be of interest to you:

https://paperswithcode.com/method/vision-transformer

Roman Leventov July 19, 2022 at 12:14 am #

REPLY

"The number of sequential operations required by a recurrent layer is based upon the sequence length, whereas this number remains constant for a self-attention layer. "

— This doesn't make sense, given that the Transformer model is itself limited in the input length? Transformer doesn't allow to process arbitrarily long sequence with a constant number of operations 🙂

James Carmichael July 19, 2022 at 10:35 am #

REPLY

Hi Roman…The following may be of interest to you:

https://machinelearningmastery.com/the-transformer-attention-mechanism/

Stefania Cristina July 20, 2022 at 2:41 am #

REPLY

Hi Roman, the comment that you are referring to is not as much about the length of the input sequence, as it is about *the number of operations* that are applied to the input. To quote Vaswani et al., "As noted in Table 1, a self-attention layer connects all positions with a constant number of sequentially executed operations, whereas a recurrent layer requires O(*n*) sequential operations.", where *n* is the input sequence length.

Having a look at Table 1 in Vaswani's paper, we can see that the number of sequential operations for the self-attention layer is O(1), which means that it is *not* a function of the input size, as opposed to the number of sequential operations for recurrent layers. This is because, when the self-attention layer in the Transformer architecture receives its inputs in the form of Queries, Keys and Values, it will apply a set number of sequential operations to them, namely: dot-product multiplication between the Queries and Keys, followed by scaling (and masking, which is however optional), softmax normalisation, and one final dot-product multiplication with the Values (refer to the link provided by James for more details).

Rashedur Rahman August 19, 2022 at 10:35 pm #

REPLY

Hi Stefania,

Thank you very much for the nice and easy to understand article on the transformer model. However, to use transformer as a seq2seq model, I am curious to know how to add constraints to the encoder and/or the decoder. For example, the constraints might be the expected length of the output sequence. Thanks!

James Carmichael August 20, 2022 at 7:34 am #

REPLY

Hi Rashedur…The following resource may be of interest to you:

https://machinelearningmastery.com/encoder-decoder-attention-sequence-to-sequence-prediction-keras/

Rashedur Rahman August 22, 2022 at 6:19 pm #

REPLY

Hi James,
Thanks for your reply. In fact, I have an LSTM based seq2seq model to add the constraints (e.g. length of the output sequence). Now I am interested to use transformers (TransformerEncoder and TransformerDecoder) to add the constraint. But I don't know how to do it with transformers.

Hasan October 16, 2022 at 8:31 am #

REPLY

@Stefania Cristina, I have a small doubt.I would be glad if you could answer my following questions.
@Any response from anyone would be appreciated.

Task: "Extractive/Abstractive Text Summarization using NLP with Transformer Model (Ex: T5 base model)"

1. What are the benefits of having multiple Encoder and Decoder block in Transformers although all encoder and decoder are identical by architecture?
2. As very 1st Encoder receives the input sentences in text summarization task and then processed input vectors transfer to 2nd Encoder and so on. So, what are the changes of these input vectors in 2nd to onward Encoders that comes from the 1st Encoder?
3. As per the Architecture, Last Encoder's input vector representations moves to all Decoder, then what are the calculations happen in each Decoder block as all Decoders are in stacked by nature?
4. In English to English text summary, which input is fed into the Decoder block?Is it target summary? I am bit confused. If target summary is fed into the decoder, then what calculation will made between targeted summary input and encoder input?
5.Which part of the Decoder is basically responsible to choose words/sentences that will be part of the final summary sentences?
It would be really appreciated if anyone can help me to understand these queries.
Thanks in Advance.

James Carmichael October 17, 2022 at 10:43 am #

REPLY

Hi Hasan…Please narrow your query to one question so that we may better assist you.

Diego January 22, 2023 at 11:04 pm #

REPLY

hi Stefania,

how exactly is the transformer decoder not using for loops to predict its own word based on all previous predicted words?
I see that when training, the decoder iteratively predicts a next word with the final dense layer, which is done in parralel in training but I don't understand how parrelel training works here.

Dario March 3, 2023 at 3:08 am #

REPLY

Questions:
– are the input embeddings fed one by one to the encoder?
– the identical layers that form the encoder and decoder, are they in parallel or in series (do they process each the same copy of the input, or they are sequentially stacked)?

James Carmichael March 3, 2023 at 9:25 am #

REPLY

Hi Dario…The following resource may be of interest:

https://machinelearningmastery.com/lstm-autoencoders/

Leave a Reply

Name (required)

Email (will not be published) (required)

SUBMIT COMMENT

Welcome!
I'm *Jason Brownlee* PhD
and I **help developers** get results with **machine learning**.
Read more