# Attention Is All You Need: The Core Idea of the Transformer

Zain ul Abideen · Follow
6 min read · Jun 26, 2023

238

An overview of the Transformer model and its key components.

## Introduction

In this blog post, I will be discussing the most revolutionary paper of this century “Attention Is All You Need” by (Vaswani et al.). First I will cover the self-attention mechanism and then move towards the architectural details of the Transformer. In the previous blog post From Seq2Seq to Attention: Revolutionizing Sequence Modeling, I discussed the origin of attention mechanism and Bahdanau attention. In this blog, I will be building upon the previous information. So if you haven’t checked out the previous post, go check it out. Bahdanau attention model uses 2 RNNs and an attention mechanism to assign weights to the encoder’s hidden states. In the “Attention is all you need” paper, the authors have gotten rid of all the RNNs.

## Self-Attention Mechanism

Self-attention mechanism enables the model to capture dependencies between different positions within a sequence by attending to all positions simultaneously. In the previous blog, we discussed the use of query and key-value pairs to calculate attention scores. The attention scores determine the importance or relevance of each key-value pair to the given query. Self-attention extends this mechanism to operate within a single sequence without requiring external inputs.

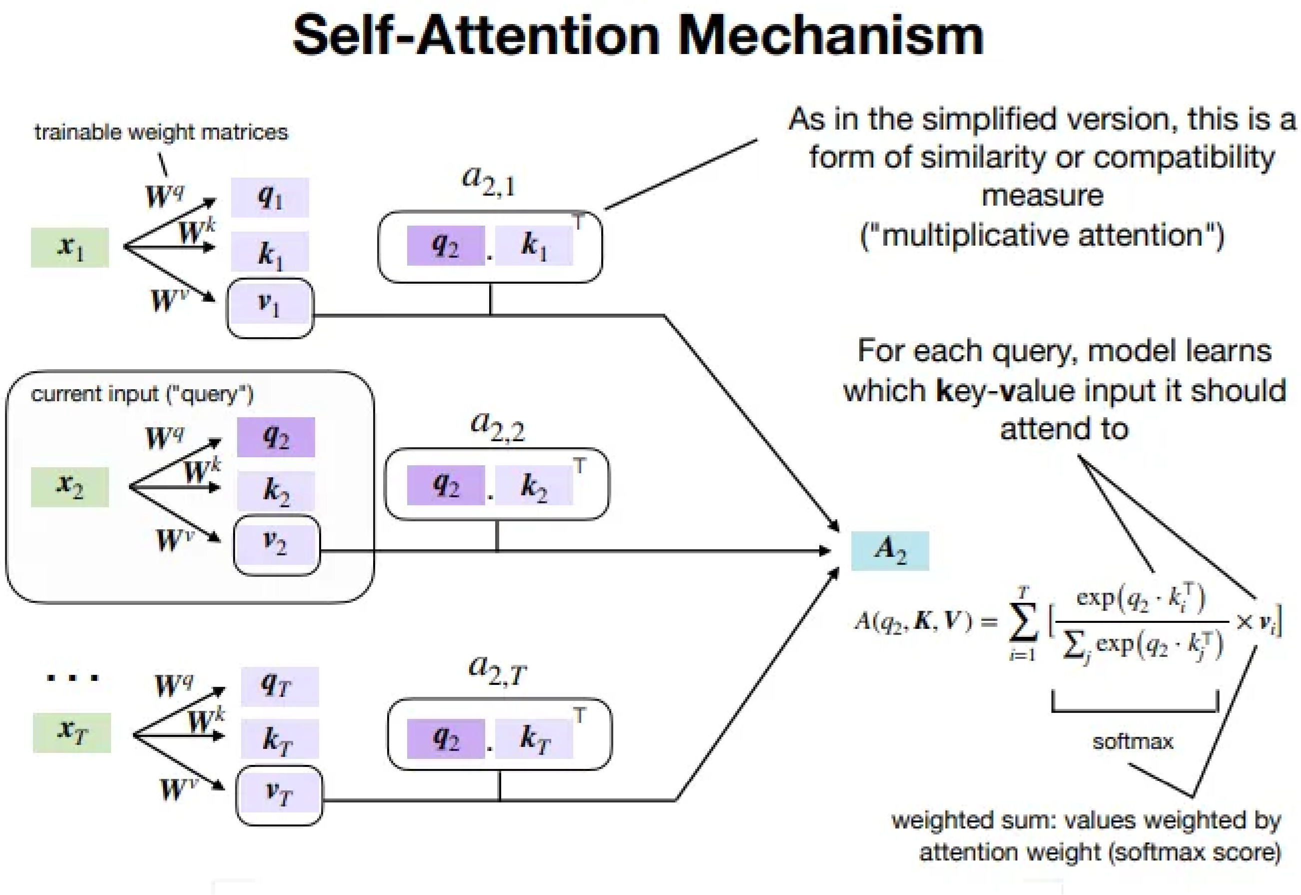

Self-Attention Mechanism
trainable weight matrices
$W^q$ $W^k$ $W^v$
$x_1$ $q_1$ $k_1$ $v_1$
$a_{2,1}$
$q_2 \cdot k_1^\top$
As in the simplified version, this is a form of similarity or compatibility measure ("multiplicative attention")
current input ("query")
$x_2$ $q_2$ $k_2$ $v_2$
$a_{2,2}$
$q_2 \cdot k_2^\top$
For each query, model learns which key-value input it should attend to
$A_2$
$A(q_2, K, V) = \sum_{i=1}^{T} [\frac{\exp(q_2 \cdot k_i^\top)}{\sum_j \exp(q_2 \cdot k_j^\top)} \times v_i]$
softmax
...
$x_T$ $q_T$ $k_T$ $v_T$
$a_{2,T}$
$q_2 \cdot k_T^\top$
weighted sum: values weighted by attention weight (softmax score)

Image source: Intro to Deep Learning by Sebastian Raschka

In the above figure, you can view the self-attention mechanism. Let me explain the figure from left to right. First of all, we have an input ***x***. We multiply this input with the trainable weight matrices (***Wq, Wk, Wv***). As an output, we get query, key, and value matrices. We use query and key matrices to find their similarity. The above figure has only taken the dot product, but in transformer architecture we also scale it. The output of this dot product is attention weight (***a***). In the same way, we will calculate attention weights for all the inputs ***x(t).*** When all the attention weights have been calculated, a softmax function is applied to normalize the dot products, producing attention weights that sum up to one. The attention weights obtained from the softmax operation are used to compute a weighted sum of the value vectors. This weighted sum represents the self-attended representation for each position in the input sequence. The strength of self-attention lies in its ability to model both local and global dependencies within a sequence. It captures contextual information from the entire sequence, providing a more comprehensive understanding of the relationships between different positions.

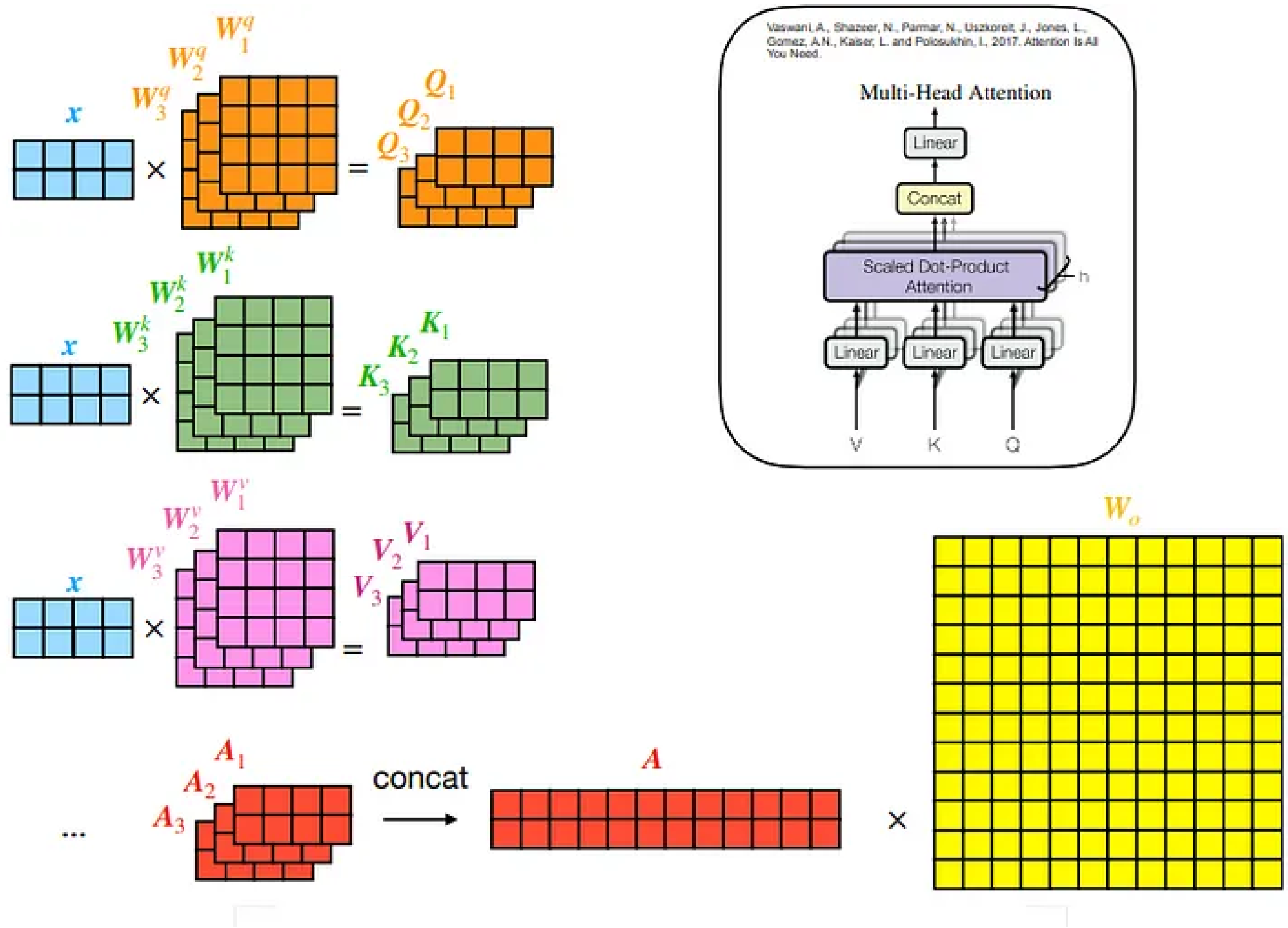
## Scaled Dot Product

As mentioned in the above paragraph, we don't only use dot product to find relevance. But we scale it as well by a factor of the square root of key dimension (***dk***). This helps in making sure that the dot-products between query and key don't grow too large for large ***dk.*** If the dot product becomes too large then the softmax output will be very small. To avoid this, we scale the dot product.

$$A(Q, K, V) = \text{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right)V$$

## Multi-Head Attention

Multi-Head Attention is just an addition to the self-attention mechanism. It allows the model to jointly attend to different positions and learn multiple representations of the input sequence simultaneously. By performing multiple sets of attention computations in parallel, multi-head attention captures diverse aspects of the input sequence and enhances the model's ability to capture complex dependencies. Each attention head has different query, key and value matrices.



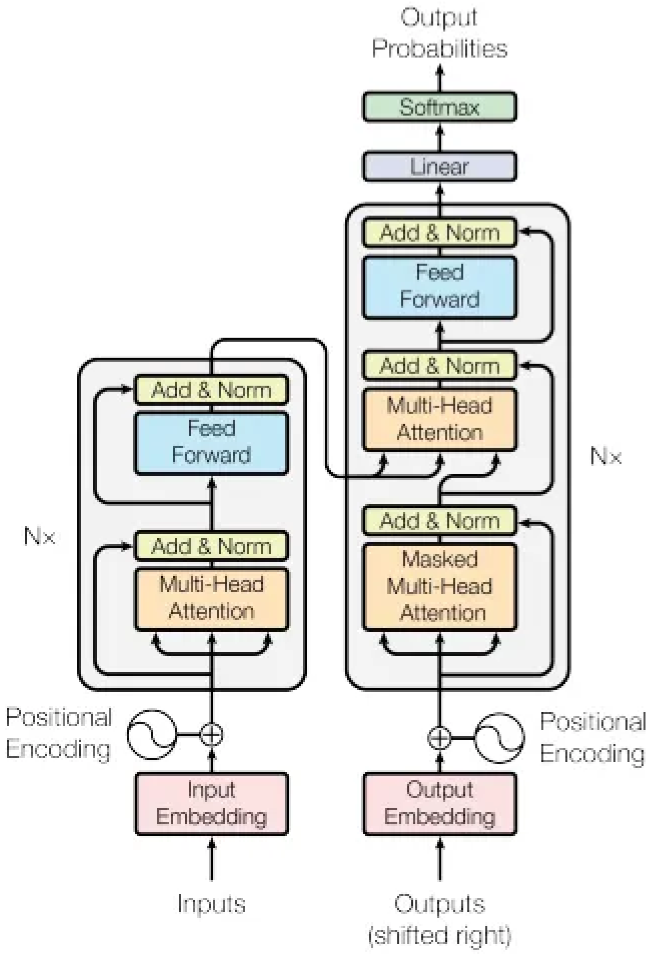
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Mult-Head Attention

The outputs from different attention heads are combined and transformed through linear projections, resulting in a final representation that integrates information from multiple perspectives.

## Transformer Architecture

The Transformer architecture, introduced in the "Attention Is All You Need" paper, consists of several key components that work together to enable effective sequence modeling. The major components are encoder, decoder, positional encoding, residual connections, layer normalization, multi-head attention block, masked multi-head attention block, and feed-forward network.



Transformer Architecture

We have already discussed the multi-head attention block. **Masked multi-head attention** is the same as that but with one change in it. We mask subsequent sequence elements. i.e., only allow to attend positions up to and including the current position. This is achieved by setting softmax values for those to negative infinity.

**Encoder**: The left part in the transformer architecture is the encoder part. It consists of one multi-head attention block, one feed-forward network, and multiple residual connections and layer normalization. It takes embeddings of the input sequence along with positional encodings added to it. In the original paper, they used 6 encoders.

**Decoder**: The right part in the transformer architecture is the decoder part. It consists of one masked multi-head attention block, one simple multi-head attention block, one feed-forward network, and multiple residual connections and layer normalization. It takes embeddings of the output sequence along with positional encodings added to it. In the original paper, they used 6 decoders.

**Residual connections and Layer normalization:** Residual connections, also known as skip connections, are direct connections that bypass one or more layers in a neural network. In the context of the Transformer architecture, residual connections are used to connect the output of a sub-layer to its input, allowing the original input to flow through the layer unchangedLayer normalization is a technique used to normalize the activations within a layer of a neural network. It aims to improve the training stability and generalization by reducing the internal covariate shift, which refers to the change in the distribution of activations as the network learns. Layer normalization is applied independently to each neuron or feature, normalizing its values across the mini-batch dimension.

**Feed-Forward Network:** In the Transformer architecture, a feed-forward network is a component that operates on each position independently and identically within each layer. It is responsible for transforming the representations of the input sequence within the self-attention mechanism and the position-wise feed-forward sub-layers. Two linear transformations are applied to the output of the self-attention mechanism with the Relu activation function after the first transformation.

**Positional Encoding:** Embeddings of input and output sequences are concatenated with positional encodings. These encodings inject information about the relative positions of elements in the sequence.

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Sine and Cosine functions for positional encoding

Learned positional embeddings and embeddings through sine and cosine functions produced nearly equal results during language tasks.

I will be covering the evaluation metrics, training methods, decoding methods at inference, and other minor details in another blog post in which I will implement Transformer architecture from scratch.

## Closing Remarks

In conclusion, the “Attention Is All You Need” paper introduced a groundbreaking architecture known as the Transformer, which revolutionized the field of sequence modeling. This architecture relies heavily on the concept of self-attention, allowing it to capture dependencies between different positions in the input sequence. The Transformer’s attention mechanism enables it to model long-range dependencies, handle variable-length inputs, and achieve state-of-the-art performance in various natural language processing tasks. The architecture introduced in this paper has been used by many language models which I will discuss in upcoming blogs. In the next blog post, I will be covering in detail the Autoregressive models like GPT, GPT-2, and GPT-3.

**Thank you for reading!**

**Follow me on LinkedIn!**

## References

1. Attention is all you need
2. Dive into Deep Learning
3. The Illustrated Transformer
4. The Narrated Transformer Language model
5. RNNs and Transformers for Sequence-to-Sequence Modeling
6. Layer Normalization
7. Deep Residual Learning

Transformers Machine Learning Deep Learning NLP Attention

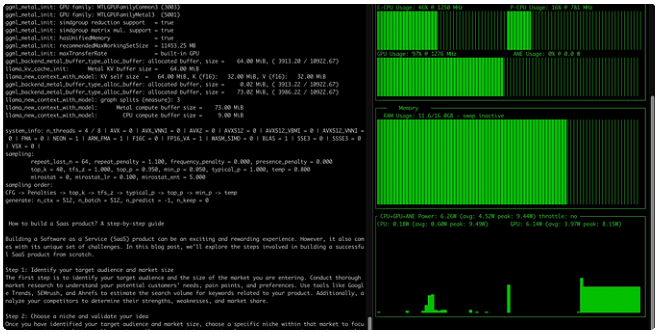
## Written by Zain ul Abideen

Follow

441 Followers

Machine Learning Engineer | I share what I learn.
https://www.linkedin.com/in/zaiinulabideen/

### More from Zain ul Abideen



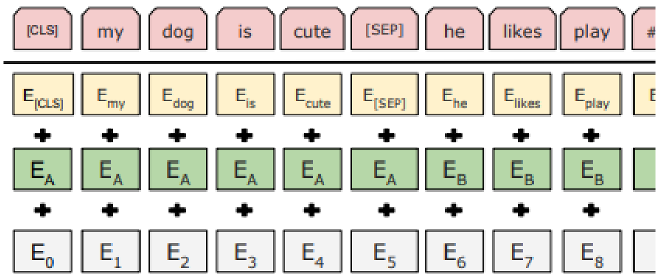
```
ggml_metal_init: GPU family: MTLGPUFamilyCommon3 (3003)
ggml_metal_init: GPU family: MTLGPUFamilyMetal3  (5001)
ggml_metal_init: simdgroup reduction support   = true
ggml_metal_init: simdgroup matrix mul. support = true
ggml_metal_init: hasUnifiedMemory              = true
ggml_metal_init: recommendedMaxWorkingSetSize  = 11453.25 MB
ggml_metal_init: maxTransferRate               = built-in GPU
ggml_backend_metal_buffer_type_alloc_buffer: allocated buffer, size =    64.00 MiB, ( 3913.20 / 10922.67)
llama_kv_cache_init:      Metal KV buffer size =    64.00 MiB
llama_new_context_with_model: KV self size  =   64.00 MiB, K (f16):   32.00 MiB, V (f16):   32.00 MiB
ggml_backend_metal_buffer_type_alloc_buffer: allocated buffer, size =     0.02 MiB, ( 3913.22 / 10922.67)
ggml_backend_metal_buffer_type_alloc_buffer: allocated buffer, size =    73.02 MiB, ( 3986.22 / 10922.67)
llama_new_context_with_model: graph splits (measure): 3
llama_new_context_with_model:      Metal compute buffer size =    73.00 MiB
llama_new_context_with_model:        CPU compute buffer size =     9.00 MiB

system_info: n_threads = 4 / 8 | AVX = 0 | AVX_VNNI = 0 | AVX2 = 0 | AVX512 = 0 | AVX512_VBMI = 0 | AVX512_VNNI =
 0 | FMA = 0 | NEON = 1 | ARM_FMA = 1 | F16C = 0 | FP16_VA = 1 | WASM_SIMD = 0 | BLAS = 1 | SSE3 = 0 | SSSE3 = 0
| VSX = 0 |
sampling:
        repeat_last_n = 64, repeat_penalty = 1.100, frequency_penalty = 0.000, presence_penalty = 0.000
        top_k = 40, tfs_z = 1.000, top_p = 0.950, min_p = 0.050, typical_p = 1.000, temp = 0.800
        mirostat = 0, mirostat_lr = 0.100, mirostat_ent = 5.000
sampling order:
CFG -> Penalties -> top_k -> tfs_z -> typical_p -> top_p -> min_p -> temp
generate: n_ctx = 512, n_batch = 512, n_predict = -1, n_keep = 0


 How to build a Saas product? A step-by-step guide

Building a Software as a Service (SaaS) product can be an exciting and rewarding experience. However, it also com
es with its unique set of challenges. In this blog post, we'll explore the steps involved in building a successfu
l SaaS product from scratch.

Step 1: Identify your target audience and market size
The first step is to identify your target audience and the size of the market you are entering. Conduct thorough
market research to understand your potential customers' needs, pain points, and preferences. Use tools like Googl
e Trends, SEMrush, and Ahrefs to estimate the search volume for keywords related to your product. Additionally, a
nalyze your competitors to determine their strengths, weaknesses, and market share.

Step 2: Choose a niche and validate your idea
Once you have identified your target audience and market size, choose a specific niche within that market to focu
```

```
E-CPU Usage: 46% @ 1250 MHz          P-CPU Usage: 16% @ 781 MHz
GPU Usage: 97% @ 1276 MHz            ANE Usage: 0% @ 0.0 W
   Memory
 RAM Usage: 11.6/16.0GB - swap inactive
 CPU+GPU+ANE Power: 6.26W (avg: 4.52W peak: 9.44W) throttle: no
CPU: 0.18W (avg: 0.60W peak: 9.49W)       GPU: 6.14W (avg: 3.97W peak: 8.15W)
```

Zain ul Abideen

## Apple MLX vs Llama.cpp vs Hugging Face Candle Rust for…

Mistral-7B and Phi-2 to experiment fastest inference/generation speed across libraries.



Zain ul Abideen

## A Comparative Analysis of LLMs like BERT, BART, and T5

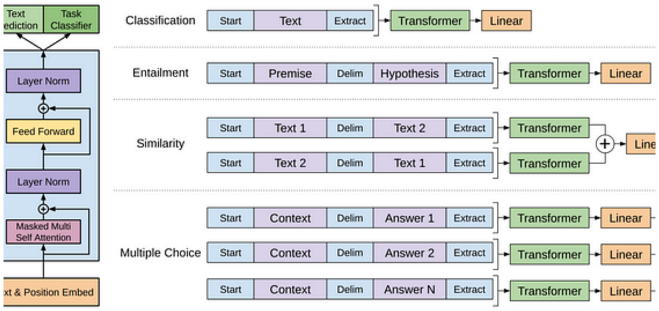
Exploring Language Models

6 min read · Jan 31, 2024

130 2

6 min read · Jun 26, 2023

47 1



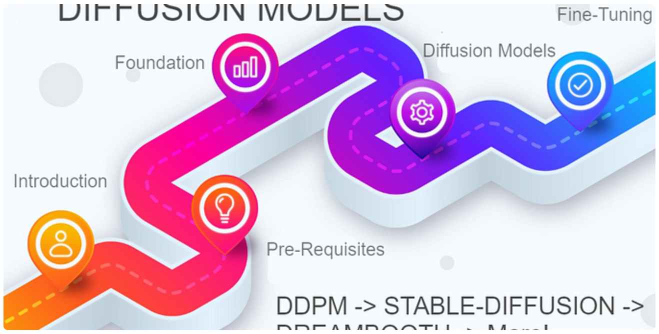
Zain ul Abideen

## Autoregressive Models for Natural Language Processing

The Evolution of GPT: From GPT to GPT-2 to GPT-3

7 min read · Jun 26, 2023

42



Zain ul Abide... in Artificial Intelligence in Plain En...

## Complete Roadmap For Learning Diffusion Models (Prereqs, DDPM...

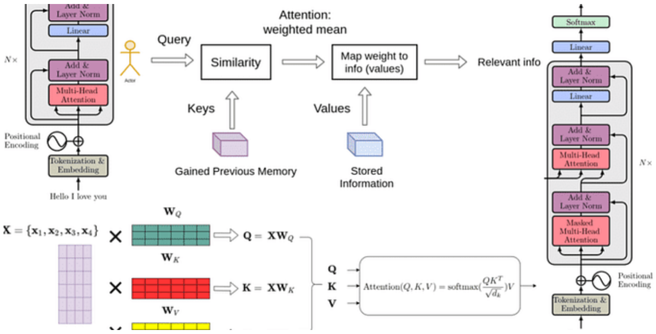
Everything you need to get started with Diffusion Models!

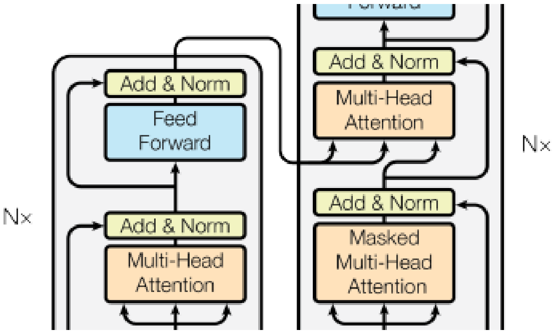
16 min read · Sep 26, 2023

74

See all from Zain ul Abideen

## Recommended from Medium





Amanatullah

### Transformer Architecture explained

Transformers are a new development in machine learning that have been making a lo…

10 min read · Sep 1, 2023

304 2

Dr. Ernesto Lee

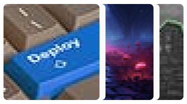
### An Intuitive Explanation of ‘Attention Is All You Need’: The…

This is the technology that makes ChatGPT works!

9 min read · Oct 13, 2023

28

## Lists



### Predictive Modeling w/ Python

20 stories · 901 saves

### Natural Language Processing

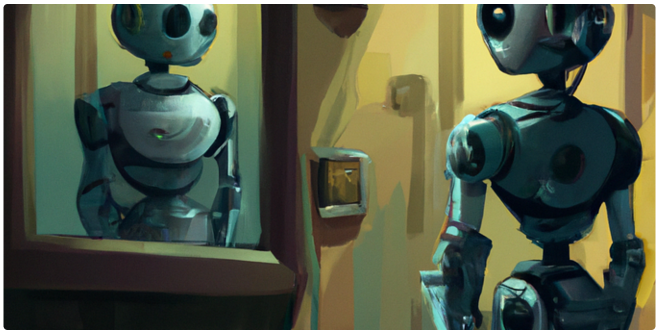
1196 stories · 668 saves

### Practical Guides to Machine Learning

10 stories · 1053 saves

### The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 307 saves



Thomas van Dongen in Towards Data Science

### Demystifying efficient self-attention

A practical overview

20 min read · Nov 7, 2022

623 2

Fareed Khan in Level Up Coding

### Solving Transformer by Hand: A Step-by-Step Math Example

Performing numerous matrix multiplications to solve the encoder and decoder parts of th...

13 min read · Dec 18, 2023

1.7K 25

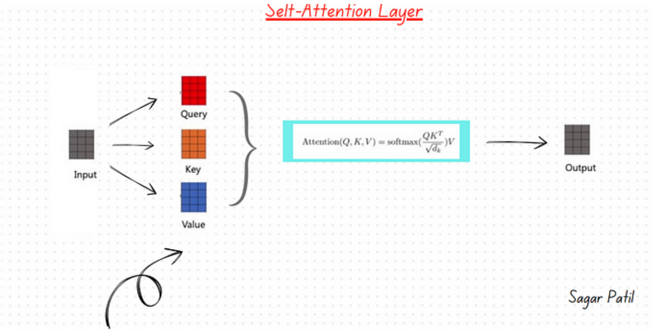
Alejandro Ito Aramendia

### Attention Is All You Need : A Complete Guide to Transformers

Everything You Need to Know

11 min read · Jan 1, 2024

265 5



Sagar Patil

### Attention Mechanism in the Transformers

In the world of natural language processing and machine learning, few innovations have...

4 min read · Sep 25, 2023

52

See more recommendations