Transformer positional embeddings

Souvik Mandal · Follow

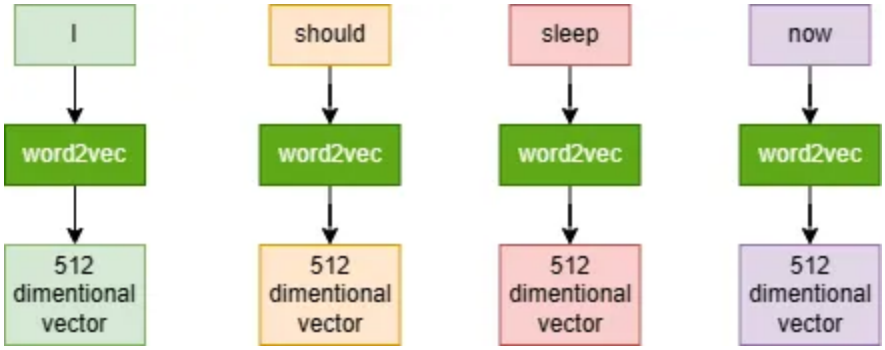
5 min read · Jun 19, 2023

Word ordering often determines the meaning of a sentence. How to utilize the position information of a word sequence is solved by positional embeddings. This blog will be on how positional embeddings is selected, why is it needed and the implementation to generate it.

This is part of a series of blogs I am writing to understand Transformer architecture from the Attention is all you need paper [1]. I will update this section with the new blogs.

1. Demystifying the Attention Logic of Transformers: Unraveling the Intuition and Implementation by visualization
2. **Transformer positional embeddings** (This blog)
3. Attention is All You Need: Understanding Transformer Decoder and putting all pieces together.

First, we create feature vectors from the input sentence by using word2vec (or another tokenizer). Let's assume we are using a word2vec which converts a word to a 512-dimensional vector.



Create vectorize representation of the words. This example we are using word2vec, but you can try any one of the tokenizers.

But we also need to add some information to let the model know the position of the word. **This is called positional encodings. With the help of positional encodings, the transformer cannot process part of sentences partially without considering about the sequential nature of a sentence.**

Now, one easy way can be is just to assign 1 to the first word, 2 to second, and so on. But in this approach, **the model during inference might get a sentence that is longer than any it saw during training.** Also, for a longer sentence, there will be large values to add which takes more memory.

We can take **a range then like add 0 for first work and 1 for last**, anything in between we split the range [0,1] and get the values. For example, for a 3-word sentence we can do 0 for the first word, 0.5 for the second, and 1 for the third; for a 4-word sentence, it would be 0,0.33, 0.66, 1 respectively. The problem with this is that the position difference delta is not constant. In the first example, it was 0.5 but in the second case, it was 0.33.

Sinusoidal Positional Encodings

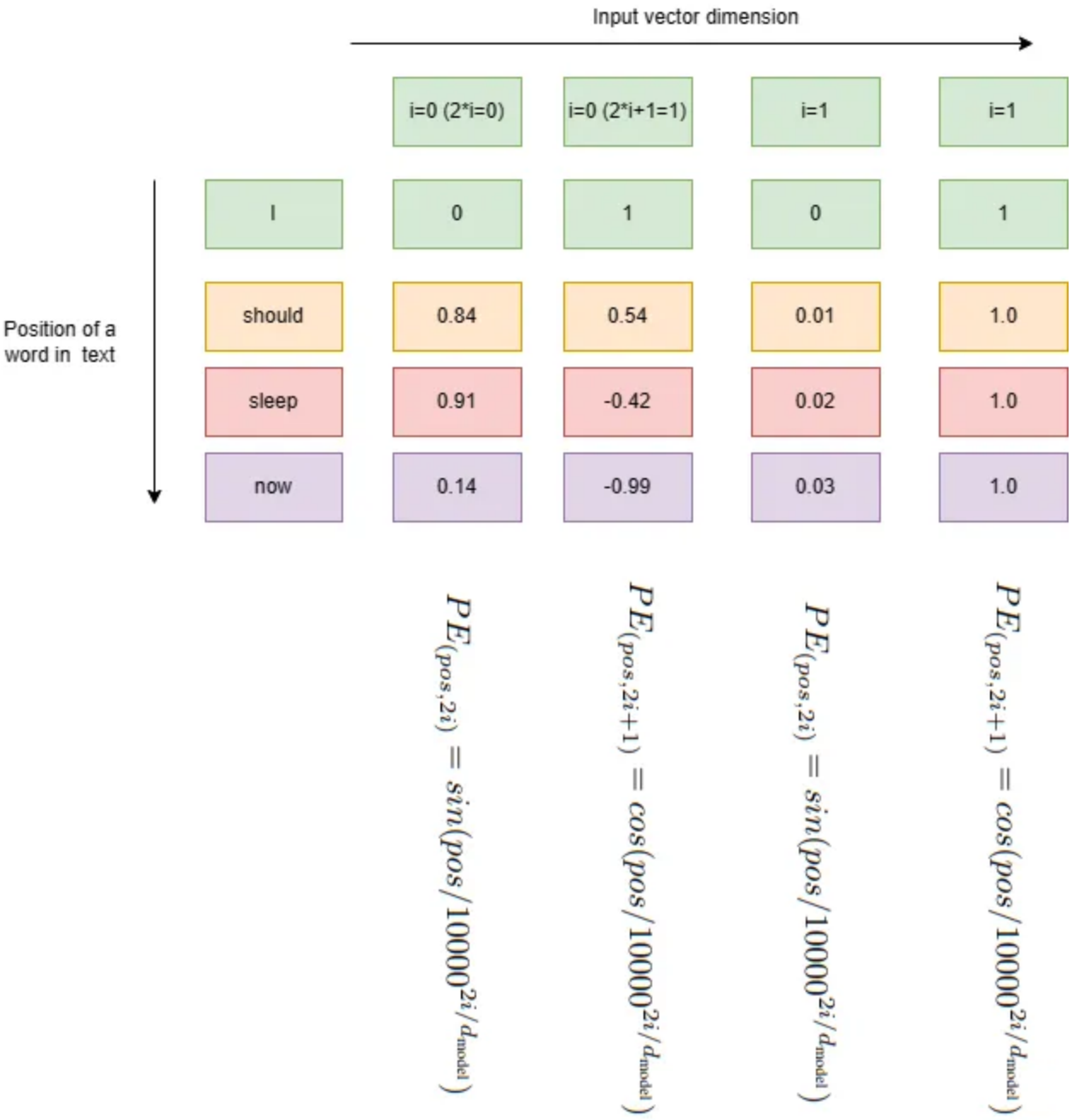
In Attention is all you need paper [1]; the authors have used sine and cosine functions (sinusoidal functions) to generate the positional encodings. The functions are defined as below.

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

Positional encoding used in the paper [1]

pos: pos is the position of the word in the text. In the example above, when generating the positional encodings, `I` will have `pos=0` , `should` will have `pos=1` , `sleep` will have `pos=2` and so on. The `d_model` parameter is the input vector dimension (dimension of the output of the tokenizer). For our example it is 512. `i` is the index of the positional encodings.

Let's understand this with a simple example. Let's assume the input vector dimension is 4 (in our old example it is 512). So, for any position of a text (`pos`) we need to generate 4 values, out of those 4 values, the even ones we generate with the first equation, and the odd ones we generate with the second equation.



Positional encodings computation. In the image I have shown the input vector size as 4 so we have 4 values for each word. In the original example since the input vector dimension is 512, positional encodings length should also be 512.

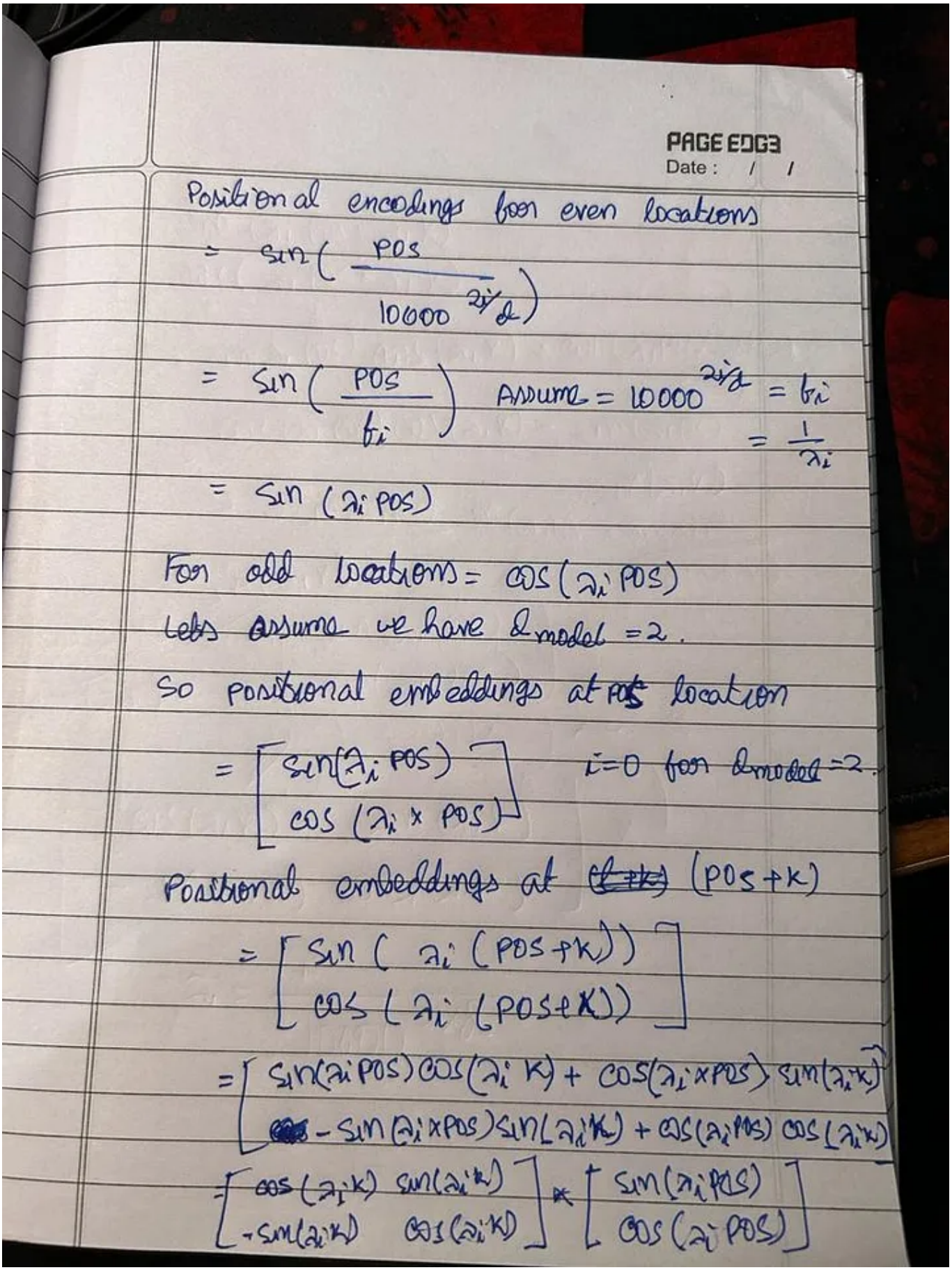
```
import numpy as np
import matplotlib.pyplot as plt

def get_sinusoidal_embedding(position, d_model):
    """
    position: Position/index of the word in the text.
    d_model: input vector dimension.
    """
```

```
    embedding = np.zeros(d_model)
    for i in range(d_model):
        if i % 2 == 0: # even indices
            embedding[i] = np.sin(position / (10000 ** (i / d_model))) # assume
        else: # odd indices
            embedding[i] = np.cos(position / (10000 ** ((i - 1) / d_model)))
    return embedding
```

Note: The above code is not generic or optimal, it's just to understand the concept.

Now, another question that should come to mind is how this makes sure that the position difference delta is linearly scaling. We can prove that as below.



PAGE EDGE
Date : / /

Positional encodings for even locations

$$= \sin\left(\frac{pos}{10000^{2i/d}}\right)$$

$$= \sin\left(\frac{pos}{f_i}\right) \quad \text{Assume} = 10000^{2i/d} = f_i = \frac{1}{\lambda_i}$$

$$= \sin(\lambda_i pos)$$

For odd locations $= \cos(\lambda_i pos)$

Lets assume we have $d_{model} = 2$.

So positional embeddings at pos location

$$= \begin{bmatrix} \sin(\lambda_i pos) \\ \cos(\lambda_i \times pos) \end{bmatrix} \quad i=0 \text{ for } d_{model}=2.$$

Positional embeddings at (pos+k)

$$= \begin{bmatrix} \sin(\lambda_i (pos+k)) \\ \cos(\lambda_i (pos+k)) \end{bmatrix}$$

$$= \begin{bmatrix} \sin(\lambda_i pos)\cos(\lambda_i k) + \cos(\lambda_i \times pos)\sin(\lambda_i k) \\ -\sin(\lambda_i \times pos)\sin(\lambda_i k) + \cos(\lambda_i pos)\cos(\lambda_i k) \end{bmatrix}$$

$$= \begin{bmatrix} \cos(\lambda_i k) & \sin(\lambda_i k) \\ -\sin(\lambda_i k) & \cos(\lambda_i k) \end{bmatrix} * \begin{bmatrix} \sin(\lambda_i pos) \\ \cos(\lambda_i pos) \end{bmatrix}$$

The multiplier metric does not depend on the position `pos`. So, `pos+k` is a linear function of `pos`

Code

Let's first define the functions as mentioned before which returns positional encodings given a index position of a word and input vector dimension (d_model).

```
import numpy as np
import matplotlib.pyplot as plt

def get_sinusoidal_embedding(position, d_model):
    """
    position: Position/index of the word in the text.
    d_model: input vector dimension.
    """
    embedding = np.zeros(d_model)
    for i in range(d_model):
        if i % 2 == 0: # even indices
            embedding[i] = np.sin(position / (10000 ** (i / d_model))) # assume
        else: # odd indices
            embedding[i] = np.cos(position / (10000 ** ((i - 1) / d_model)))
    return embedding
```

Generate positional encodings for all the words in the sequence/text.

```
def generate_positional_embeddings(seq_length, d_model):
    """
    seq_length: number of words in the text.
    d_model: input vector dimension.
    """
    embeddings = np.zeros((seq_length, d_model))
    for pos in range(seq_length):
```

```
            embeddings[pos] = get_sinusoidal_embedding(pos, d_model)
        return embeddings
```

Visualize the positional encodings.

```
def plot_positional_embeddings(embeddings):
    seq_length, d_model = embeddings.shape
    plt.figure(figsize=(20, 8))
    plt.imshow(embeddings, cmap='viridis', aspect='auto')
    plt.colorbar()
    plt.title('Sinusoidal Positional Embeddings')
    plt.xlabel('Dimension')
    plt.ylabel('Position')
    plt.xticks(np.arange(d_model), rotation=90)
    plt.yticks(np.arange(seq_length))
    plt.show()

# Example usage
seq_length = 50
d_model = 128

embeddings = generate_positional_embeddings(seq_length, d_model)
plot_positional_embeddings(embeddings)
```

The author of the attention is all you need paper have also tried the positional encodings as learnable parameters. But it did not produce any better results. Also, with learnable positional embeddings we cannot predict (during inference) on a sequence which is longer than the max sequence length during training. But the sinusoidal version allow the model to extrapolate.

Resources

1. Attention is all you need.
2. Demystifying the Attention Logic of Transformers: Unraveling the Intuition and Implementation by visualization.
3. An Empirical Study of Pre-Trained Language Model Positional Encoding

Artificial Intelligence | Deep Learning | Machine Learning | Computer Vision | NLP

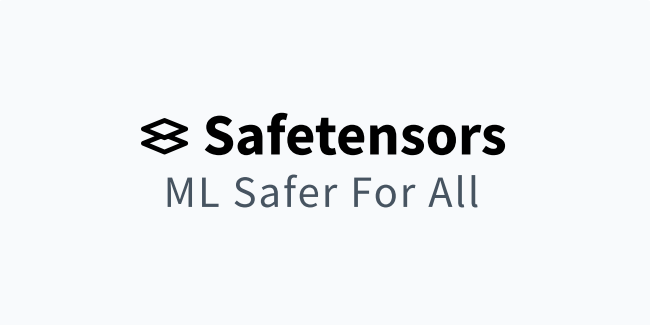
Written by Souvik Mandal

Follow

96 Followers

Senior AI Scientist @ Qure.ai, Ex Fractal, CSE IIT Indore, 20. LinkedIn: https://www.linkedin.com/in/mandalsouvik/

More from Souvik Mandal



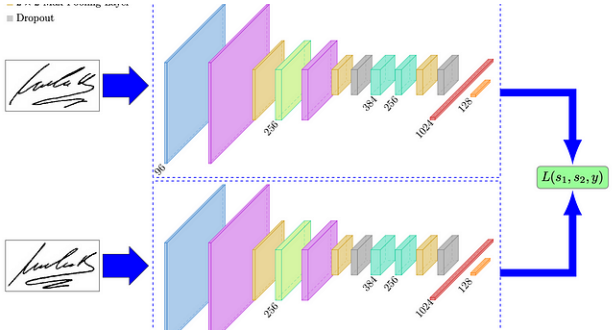
Souvik Mandal

Safetensors: a simple and safe way to store and distribute tensors

SafeTensors : a simple and safe way to store and distribute tensors. Why and how it is...

10 min read · Jul 9, 2023

59



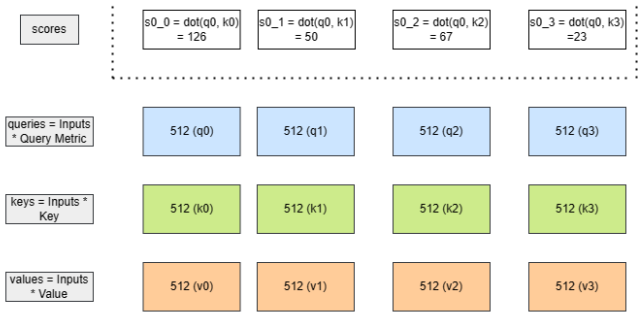
Souvik Mandal

Power of Siamese Networks and Triplet Loss: Tackling Unbalanced...

Solve unbalanced Datasets and Image Recognition Tasks: Unveiling the Potential of...

8 min read · May 18, 2023

19



Souvik Mandal in MLearning.ai

Demystifying the Attention Logic of Transformers: Unraveling the...

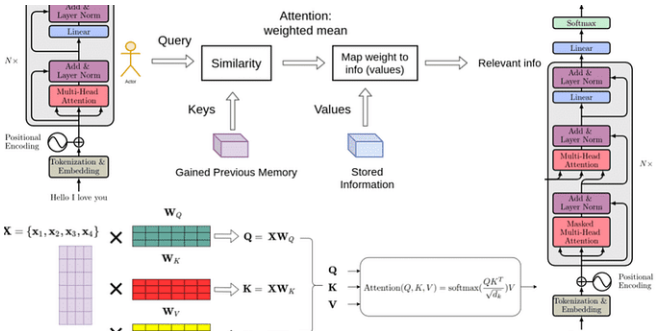
Demystifying the Attention Logic of Transformers: Unraveling the Intuition and...

8 min read · Jun 17, 2023

160 2

See all from Souvik Mandal

Recommended from Medium



Amanatullah

Transformer Architecture explained

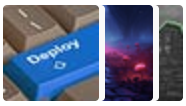
Transformers are a new development in machine learning that have been making a lo...

10 min read · Sep 1, 2023

358 2

Lists

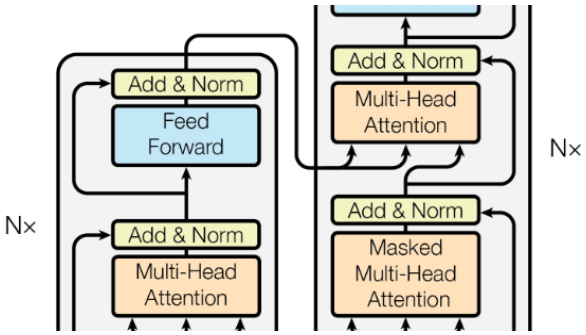
Natural Language Processing
1201 stories · 676 saves



Predictive Modeling w/ Python
20 stories · 912 saves

AI Regulation
6 stories · 318 saves

Practical Guides to Machine Learning
10 stories · 1065 saves



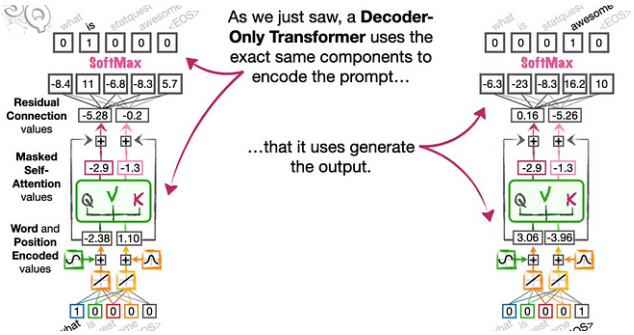
Stefan

Understanding Attention and Transformers

My notes for understanding the attention mechanism and transformer architecture...

7 min read · Nov 29, 2023

51



Witsarut Wongsim

Why ChatGPT Uses Decoder-Only

“Understanding Why ChatGPT Uses Decoder-Only Transformers: A Step-by-Step Guide by...

4 min read · Nov 28, 2023

60

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