



Home



My Network



Jobs



Messaging



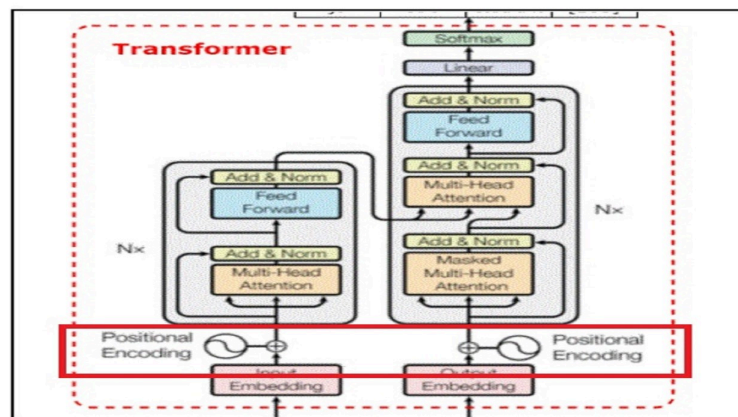
Notifications



Me



For Business



Deep Dive into the Positional Encodings of the Transformer Neural Network Architecture: With Code!



Ajay Taneja

Senior Data Engineer at Jaguar Land Rover | Ex - Rolls-Royce | Data Engineering, Data Science, Finite Eleme...



August 29, 2023

1. Introduction

This is the continuation of my series of blogs on Language Models. This blog is going to be about the positional encodings a.k.a. positional embeddings of the Transformer neural network architecture. This article is organized as follows. Firstly, I have attempted to recapitulate the flow of the discussion on language models, some important concepts that I have discussed until now in the series of articles on Language Models. Thus:

- Section 2 talks of the decomposition of Transformers into BERT and GPT and the applications of the three models.
- Section 3 provides an overview of each component/block of the Transformer architecture and its importance.
- Section 4 explains the need / reasons of having positional encodings block in the Transformer architecture.
- Section 5 explains the physical intuition of positional embeddings – which are essentially an offset to the word embeddings in the embedding space.
- Before going into the mathematical formulation of positional embeddings in section 7, section 6 discusses the requirements of the math in order to make the understanding of the formulation simpler.

- Section 8 points to my GitHub repository which provides the necessary mathematical steps to code out the positional encodings. There are two Colab notebooks – the content of both the notebooks are essentially the same but the first notebook constitutes all the math in top-to-bottom approach, the second notebook is more organized from the program architecture point of view and is split into classes and functions but performing the same calculations as the first notebook.

2. Transformer Encoder-Decoder Architecture:

In my article on “The Evolution of Language Models” discussed [here](#), I had talked about n-gram language model, Recurrent Neural Network architecture. It was discussed in detail about why Transformers scored over Recurrent Neural Networks which was followed by a discussion on the Transformer Neural Network Architecture.

It was pointed out that the Transformer architecture comprised of 2 components: Encoder and Decoder components both of which have a contextual understanding of language – if we stack the Encoder, we get the “Bidirectional Encoding Representation of Transformer (BERT)” and we stack the Decoder we get the “Generative Pre-trained Transformer (GPT)”. The article highlighted that the **Encoder only** models were good for tasks that required understanding of inputs such as: sentiment classification, named entity recognition (NER) whereas the **Decoder only** models are good for generative tasks such as text generation such as – Language Models.

Encoder – Decoder models are good for generalisation tasks such as: text summarization or translation.

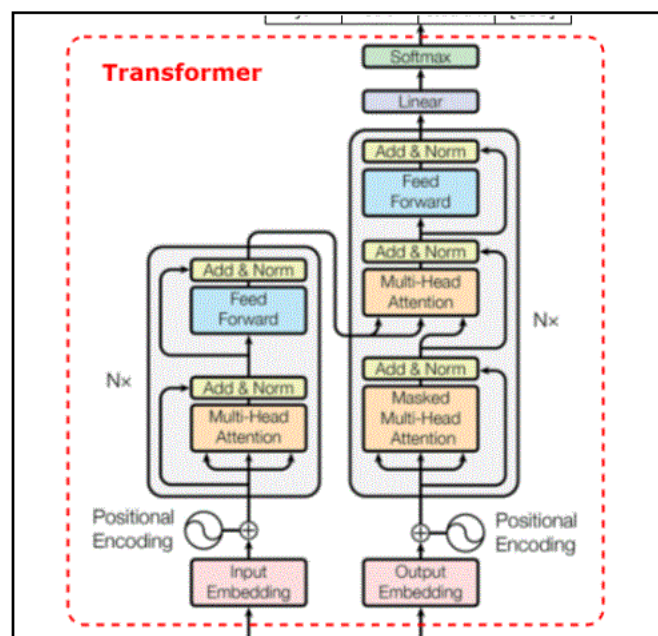


Figure: The Transformer Model Architecture

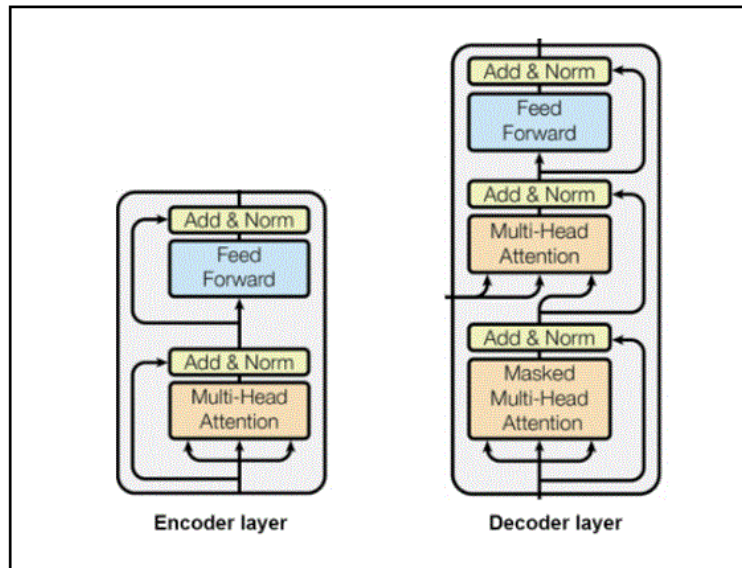


Figure: Decomposition of the Transformers into BERT and GPT

3. Transformer Components:

I had pointed out that in my article on: [Demystifying Multi-Head Attention in the Transformer Neural Network Architecture](#); that since computers do not understand words and can only understand numbers/vectors/matrices, therefore, before passing the text into the model to process, we must first tokenize the words. Once the words are tokenized, the tokens / numbers are passed to an embedding layer which is a trainable vector embedding space – a higher dimension space where each token is represented as a vector and occupies a unique location in that space. The embedding vectors “learn” to encode the meaning of the context of the individual tokens (words) in the sentence.

Looking at the sample sequence as below, it can be noticed, that each word has been mapped to a token id and each token id has been mapped to a vector. In the original Transformer paper of “Attention Is All You Need” the length of the embedding vector was 512. As an example, in the figure below, one can see that the embedding vectors corresponding to the words which are closely related are closely spaced in the embedding space.

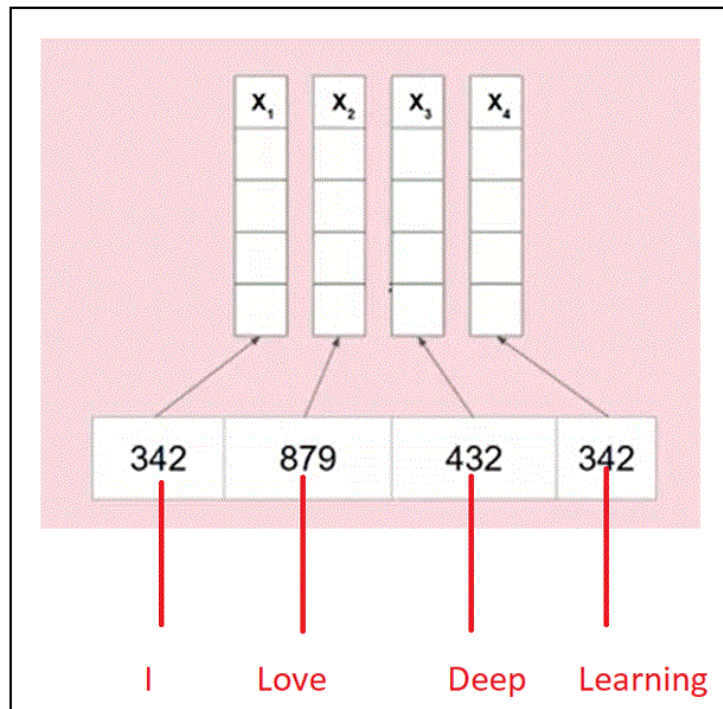


Figure: Each word mapped to a token id and each token mapped to a vector

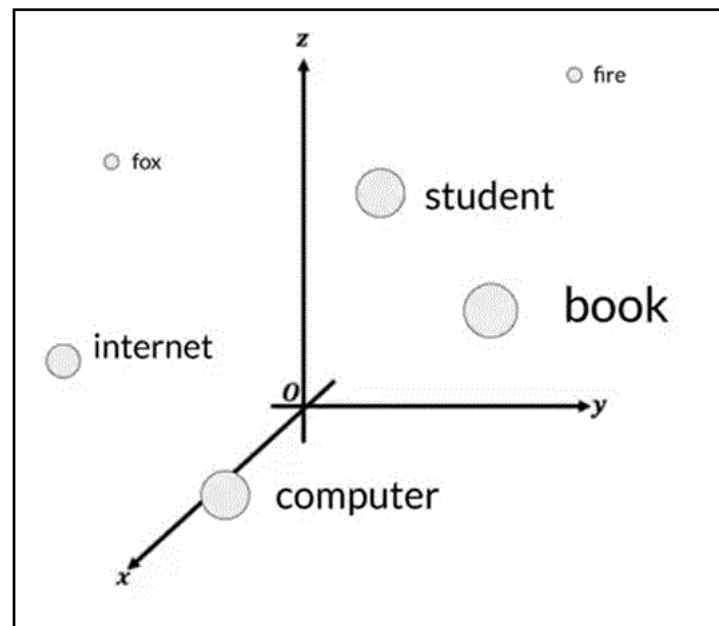


Figure: Closely related words are closely spaced in the embedding space (note: in reality the embedding is of a much higher dimension than 3 dimensional)

We then add the positional encoding to the embedding vector through which we process the information of the word order.

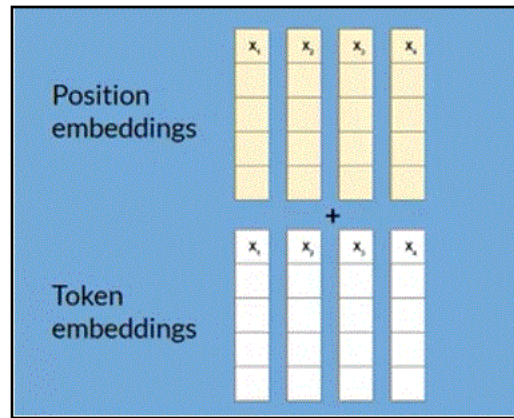


Figure: Positional encoding added to the word embedding to maintain the relevance of the word order.

After adding the positional encoding to the original word embedding – for each word – we have to generate a query, key and value vector for each word. We pass each of the vectors – comprising of the word embedding plus the positional encoding into three separate neural network units, thus, each with a trainable separate set of weights. Each of the heads are split into multiple heads to perform multi-head attention which I have discussed in sufficient detail in my blog on ,Multi-head attention: [Demystifying Multi-Head Attention in the Transformer Neural Network Architecture](#)

4 .Why do we need Positional Encoding a.k.a. Positional Embedding in the Transformer Neural Network Architecture?

It should be underscored that in the Transformer Network, the processing of the embedding vectors in the Transformer attention block to transform each vector into query, key and value vector occurs in parallel. Therefore, the order information between the words is not known anywhere. The order information has to be modelled – this information is modelled through the positional encodings.

The order information between the words is of significance in English or any other language. For example, considering the two sentences:

1. The man drove the woman to the store.
2. The woman drove the man to the store.

One can imagine, that, reversing the order information in the above sentence, we're essentially reversing the driver of the sentence – in this case man vs woman. Without the order information, transforming the embedding into contextually rich embedding is meaningless. Therefore, the positional embedding is superimposed (added) along with the word embedding for processing in the Attention block.

5. Physical intuition of the Positional Embeddings a.k.a. Positional Encodings

Positional Encodings can be looked upon as an identifier – that tells the word embeddings of the transformer of the whereabouts about the piece of word /input within a sequence of words. These embedding are then added to the initial vector representation of the input.

King					and					Queen				
0.33	0.71	0.91	0.23	0.15	0.12	0.22	0.31	0.03	0.10	0.34	0.70	0.95	0.21	0.17
+					+					+				
0.01	0.01	0.01	0.01	0.01	0.02	0.21	0.33	0.43	0.98	0.03	0.32	0.85	0.91	0.24

Figure: Adding positional embeddings to the word embeddings

This addition of the initial vector representation with the positional encodings can be physically understood as follows:

Let us say we have the words "King" and "Queen" – thus both the words share close context and let us assume that the embedding vectors representing "King" and "Queen" are represented in the three-dimensional space as below:

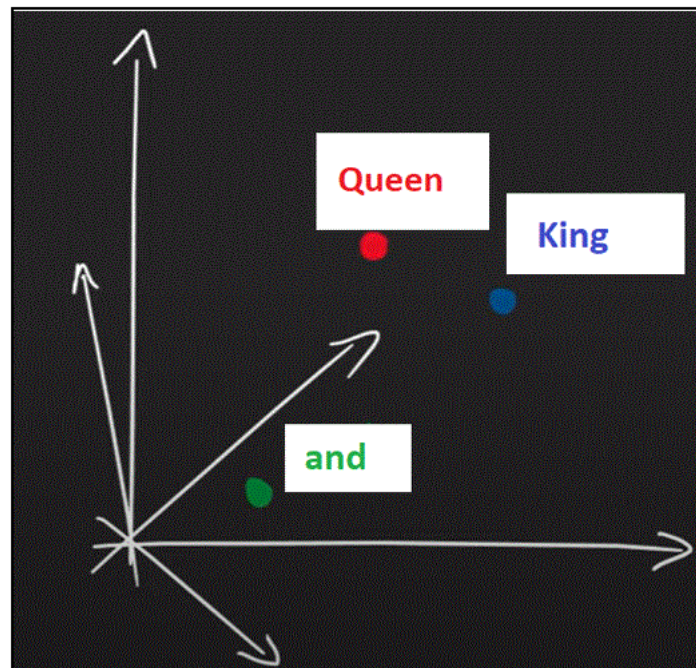


Figure: Example of a three-dimensional representation of words King and Queen in the embedding space

By identifying the order of the word in the sentence, we offset the word embedding of the word as shown in the figure below. This multi-dimensional shifting helps the transformer capture the order information.

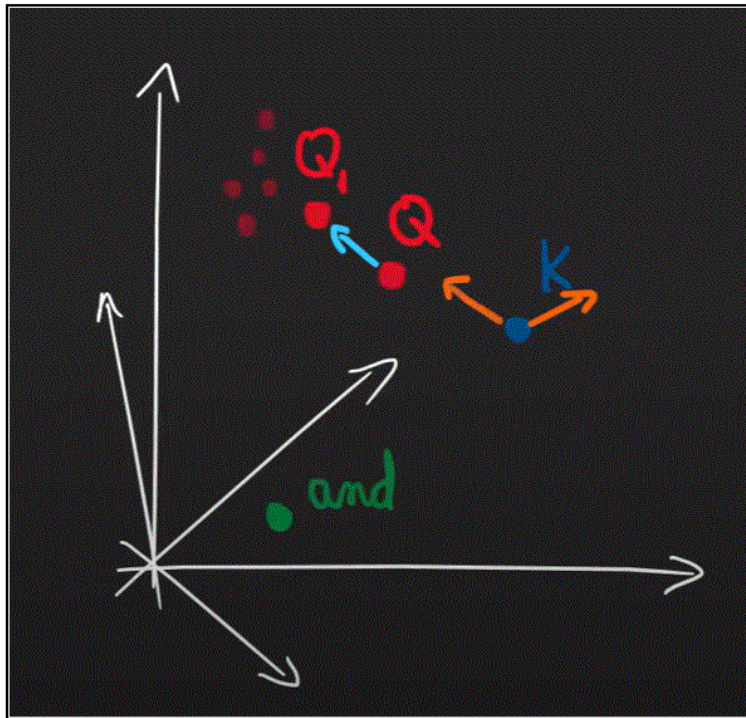


Figure: Offsetting the embedding vector corresponding to the word Queen and King to incorporate the word order

Thus, in short, positional embeddings are order or positional identifiers added to the original embedding vectors for the transformer architecture to incorporate the order of the sequence. The positional embeddings must fulfil certain requirements as discussed below which will then lead to the mathematical formulation of the positional embeddings.

6. Requirements of the Positional Embedding

Let us intuitively understand some of the requirements of the positional embeddings:

I. 1st Requirement:

Firstly, every positional embedding should have the same identifier, irrespective of the sequence length or what the input is. That is: the positional embedding for the corresponding position in a sentence must remain the same even though there might be another word in that sentence. This is illustrated through the figure below:

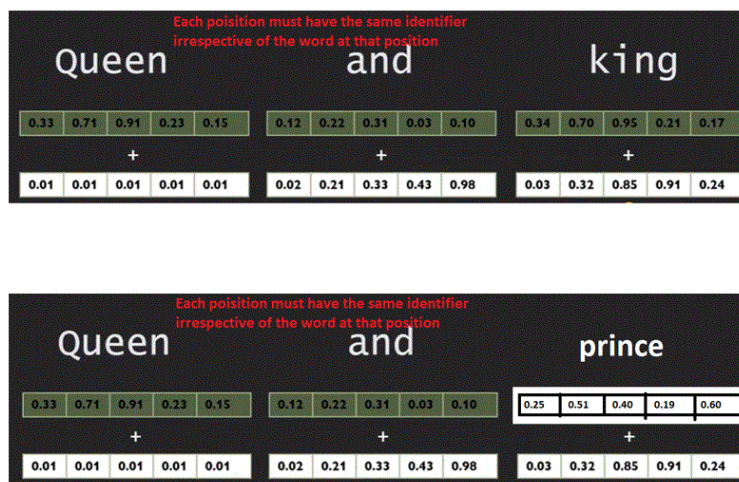


Figure: Each position has the same identifier – e.g., the position 3 has the same positional embedding irrespective of the token/word embedding at that position

II. 2nd Requirement

Since positional embeddings are offsets that get added to the original word embedding, they cannot be too large. If the positional (offset vector) embeddings are too large, then the embedding vector will get offset by a very large distance thus destroying the overall concept of the embedding space wherein the word vectors which are closely related must remain close to one another.

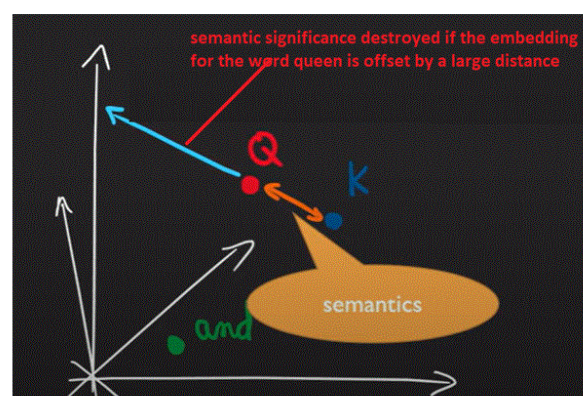
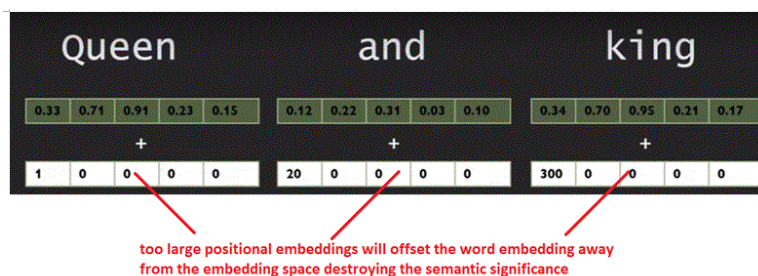


Figure: The positional embedding vector cannot offset the word embedding far away so as to destroy the semantic significance

Thus, one can imagine that if the positional encoding is represented as vectors of magnitude based on the order of the word in the sequence – this cannot be correct⁶ as that might offset the embedding vector by a huge amount as stated above. Thus, the positional embedding vectors

have to be bounded – that is: the offsets introduced to incorporate the order information must be small/bounded.

7. Formulation of Positional Embedding Vectors

From the discussion in section 6, it is clear that the function to mathematically formulate the modelling of positional embeddings in the Transformer Neural Network architecture have to be bounded in the values it can take – like the sine or the cosine functions which can take values between -1 and +1 and are periodic functions and have the upside that they are defined up to infinity – thus, emulating the fact that they are well suited to whatever be the length of a sentence (since they can take values up to infinity) and they will always have values between -1 and +1!

Another question that arises is: can we choose only sine or cosine functions?

If we choose only the sine or cosine functions as the positional embeddings, then, the outcome will repeat for different positions since these functions are periodic – to get the optimal solution satisfying the requirements explained in section 6, it is best to choose a combination of sines and cosines with different frequencies.

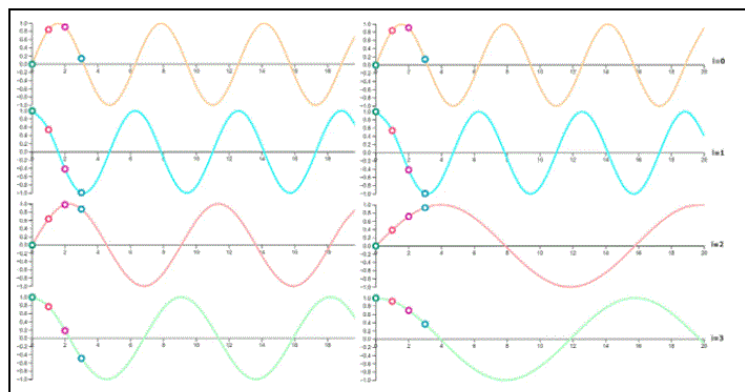


Figure: Positional embedding as a combination of sine and cosine functions

Keeping these requirements into consideration, the function is taken as:

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

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

Equations: Formulation of Positional Encodings as in "Attention Is All You Need"

Here:

- "pos" is the position.
- "i" is the dimension.

That is: each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$

- "d_model" is the vector dimension.

8. GitHub Repository - Colab notebooks with code

Positional Encoding Colab Notebooks: <https://github.com/ajaytaneja-learner/transformers-notebooks>

9. References

1. Natural Language Processing – Specialization on deeplearning.ai
2. Large Language Models – Specialization on deeplearning.ai
3. Several You-Tube Videos and medium blogs

Report this article

Comments


 30



 Like

 Comment

 Share

 Comments have been turned off on this article. You can still react or share it.

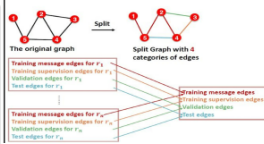


Ajay Taneja

Senior Data Engineer at Jaguar Land Rover | Ex - Rolls-Royce | Data Engineering, Data Science, Finite Element Methods Development, Stress Analysis, Fatigue and Fracture Mechanics

[+ Follow](#)

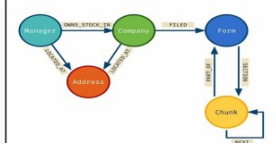
More from Ajay Taneja



The original graph is split into training, validation, and test edges. The diagram shows a graph with nodes and edges, which is then split into three sets: training message edges for r_1 , training representation edges for r_1 , validation edges for r_1 , test edges for r_1 ; training message edges for r_2 , training representation edges for r_2 , validation edges for r_2 , test edges for r_2 ; and training message edges for r_3 , training representation edges for r_3 , validation edges for r_3 , test edges for r_3 .

Heterogeneous Graphs and Relational Graph Convolutional Neural...

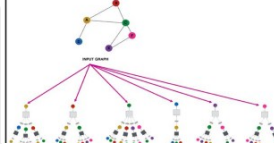
Ajay Taneja on LinkedIn



The diagram shows a flow from Documents to a Knowledge Graph, which is then used to generate a Query. The Knowledge Graph is also used to generate a Chunk, which is then used to generate a Query. The Query is then used to generate a Response.

Knowledge Graphs for RAG: Part 10 of my Graph series of blogs

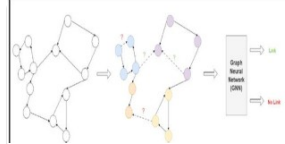
Ajay Taneja on LinkedIn



The diagram shows a hierarchical structure of nodes and edges, representing a deep learning architecture for graphs.

Deep Learning with Graphs: Part 9 of my Graph Series of blogs

Ajay Taneja on LinkedIn



The diagram shows a graph structure with nodes and edges, and a neural network layer (GNN) that processes the graph.

An Introduction to Graph Neural Networks: Part 8 of my Graph Series

Ajay Taneja on LinkedIn

[See all 108 articles](#)