**Transformers: Attention is all you need — Layer Normalization**

[[Shravan Kumar](https://medium.com/@shravankoninti?source=post_page-----1435248866d6--------------------------------)](https://medium.com/@shravankoninti?source=post_page-----1435248866d6--------------------------------)

[Shravan Kumar](https://medium.com/@shravankoninti?source=post_page-----1435248866d6--------------------------------)

·

Follow

6 min read

·

Nov 17, 2023

5

***Please refer to below blogs before reading this:***

[**Introduction to Transformer Architecture**](https://medium.com/@shravankoninti/introduction-to-transformer-architecture-26af18a425f0)

[**Transfomers: Attention is all you need — Overview on Self-attention**](https://medium.com/@shravankoninti/transformers-attention-is-all-you-need-overview-on-self-attention-2dc325dc4fcd)

[**Transformers: Attention is all you need — Overview on Multi-headed attention**](https://medium.com/@shravankoninti/transformers-attention-is-all-you-need-overview-on-multi-headed-attention-379eb8d095dc)

[**Transformers: Attention is all you need — Teacher Forcing and Masked attention**](https://medium.com/@shravankoninti/transformers-attention-is-all-you-need-teacher-forcing-and-masked-attention-9de9b3e71e9e)

[**Transformers: Attention is all you need — Zooming into Decoder Layer**](https://medium.com/@shravankoninti/transformers-attention-is-all-you-need-zooming-into-decoder-layer-3c5818fb9cb8)

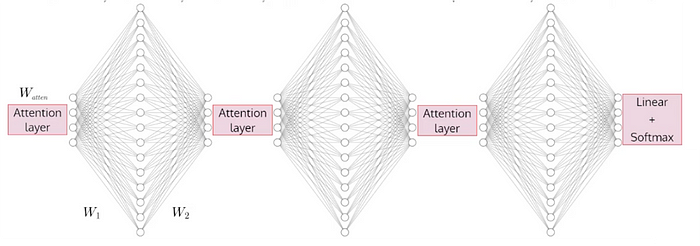
[**Transformers: Attention is all you need — Positional Encoding**](https://medium.com/@shravankoninti/transformers-attention-is-all-you-need-positional-encoding-485dcc1019fe)

There are two major concepts which we are going to discuss here are

1. Residual connections
2. Layer Normalization

Let us focus on Residual connections — in the transformer architecture when you have one-layer — you have the attention layer inside and as well as the feed forward layer. If we have decoder layer, then you have both the attention, self-attention, the cross attention and feed forward layer.

Let us look at the encoder layer — if we are looking at encoder, we have attention layer, feed forward layer and the next encoder block will contain again attention layer and feed forward layer.



Ok now we have two hidden layers and attention layer in every encoder layer and 2 attention layers and 2 hidden layers in every decoder layer. Then, the network is deep with 42 layers (6 encoder — 6\*3 layers+ 6 decoder — 6\*4 layers). So now this 42 layers are heavy and training these 42 layers and how will the gradients flow across the network is the one we need to think of? **We will use Residual connections!**

Also we need to think of speed up the training process? **Normalization is the way to go!**

In the transformer architecture we have **layer normalization** which is similar to **batch normalization** but with some variation.

**Batch Normalization:**

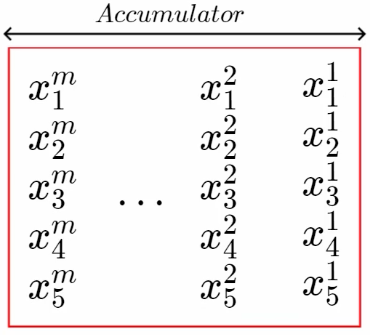
In the context of transformers; the evolution of batch normalization is called as layer normalization.

xij —denotes the activation of ith neuron for the jth training sample and this training sample is coming from a sample of batch training sample

j — training sample location

l — layer number

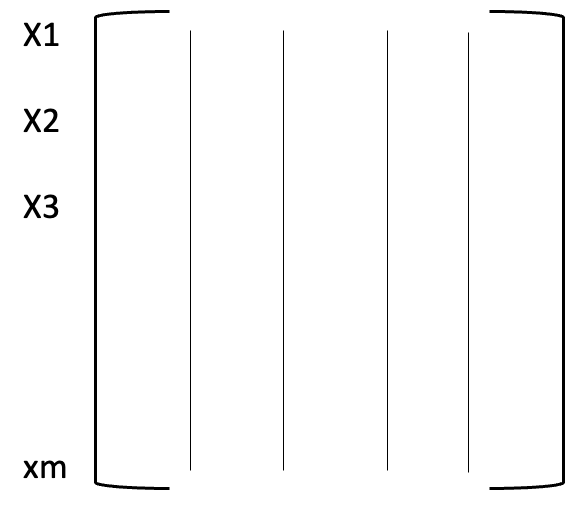
Now lets associate an accumulator with lth layer that stores the activations of batch inputs



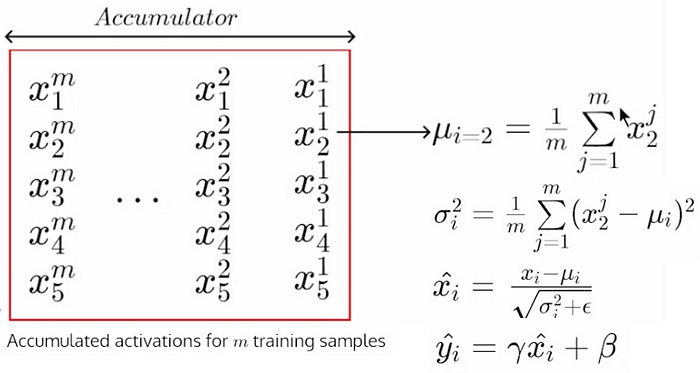
Why do we need batch normalization? If you think of deep neural network, it would be of anything with transformer or without transformer or RNN or FFNN. if we have an input x (original) and we have a series of layers (hidden) shown below.



At each layer we could see different inputs are available. Here x is the original input followed by h1, h2 etc., in the machine learning world — we have input always standardized with 0 mean and unit variance. Let’s standardize for the original input x which can be represented in this format

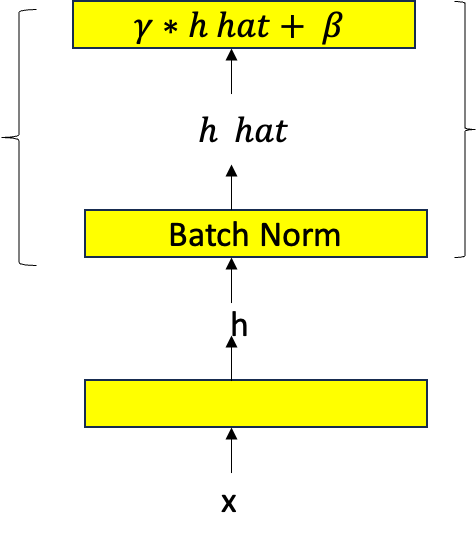


For each column (neuron number or feature)and data row (sample number) — we calculate the mean and variance values and standardize. This is applied across all layers and hence it is expected that we have convergence faster.



As you can see above for 2nd feature or 2nd neruron we have mean, variance and we have all the values as standardize values where we pass these values to next layers. When we are back propograting we need to have differential functions where we need to be able to compute derivates. Here xi = wx+b and all other equations that form in between each layer are differentiable. Here ‘m’ represents the batch and all calculations are done at this batch level only.

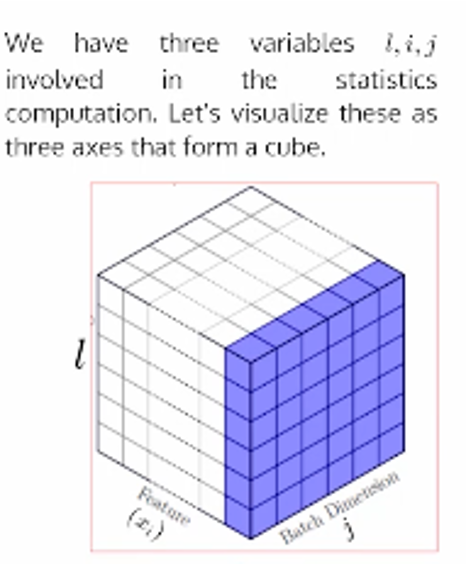
The equation yi hat — is taken into account to allow back to non-normalization status as and when it is required. Hence we could see gamma and beta are the parameters which are used to learn. So finally we should be able to see the output as non-normalized.



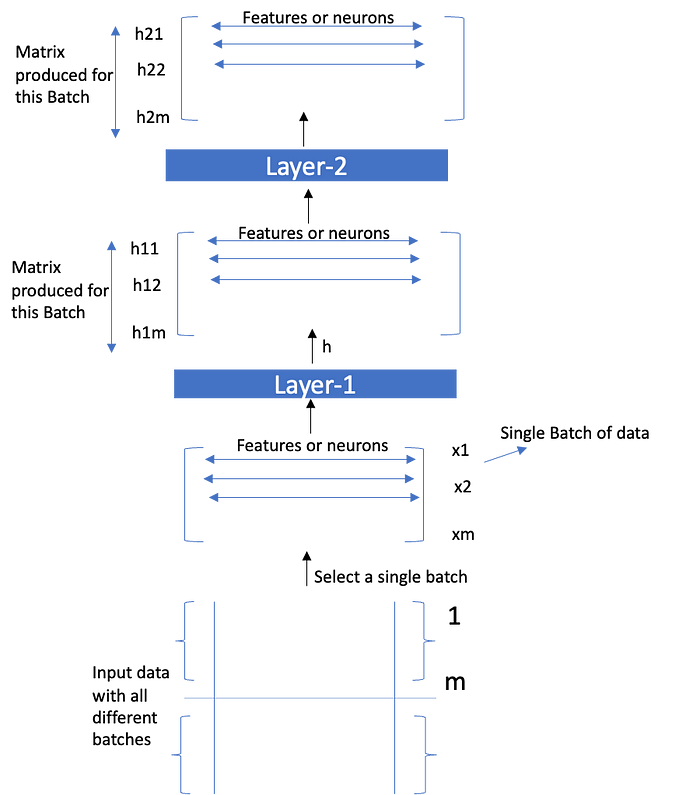
Instead of using ‘h’ originally — we will use h *tilda* which comes out after using transformations.

* Aggregating along the batch
* Computing the mean and variance
* Standardizing
* leaving the door open for non-standardizing if required

let’s look at below 3D chart where we have i — feature, j- batch, l -layer for all the batch normalization calculations.



This is how the input data flows for the **batch normalization**with all the matrices created for each of the layer outputs.



Now with this introduciton above on Batch Normalization — let’s move onto the L**ayer Normalization**.

**Layer Normalization**

***Can we apply batch normalization to transformers?***

Of course, yes. however, there are some limitations to Batch Normalization (BN).

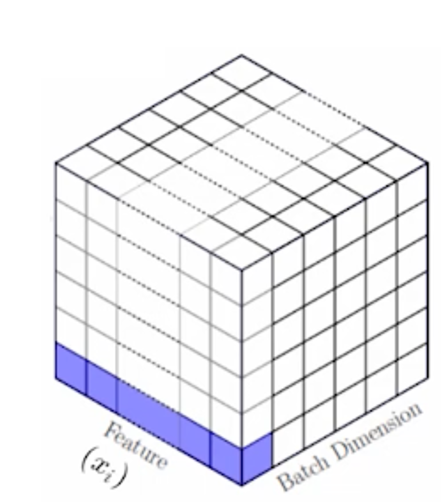
The accuracy of estimation of mean and variance depends on the size of m. So using a smaller size of m results in high error. Becuase of this, we can’t use a batch size of 1 at all (i.e., it won’t make any difference with mean = xi and standard deviation = 0). Other than this limitation, it was also empirically found that the naive use of BN leads to performance degradation in NLP tasks. There was also a systematic study that validated the statement and proposed a new normalization technique (by modifying BN) called **powerNorm**.

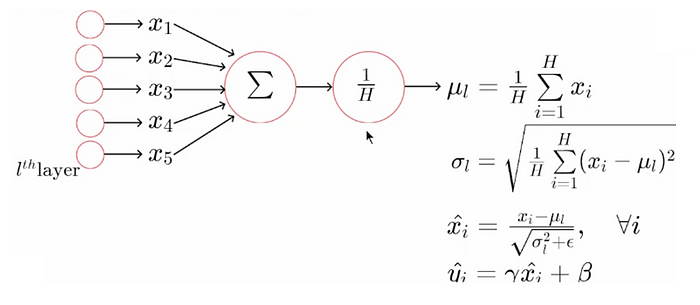
Fortunately, we have another simple normalization technique called Layer Normalization that works well.

*Layer Normalization lth layer:*

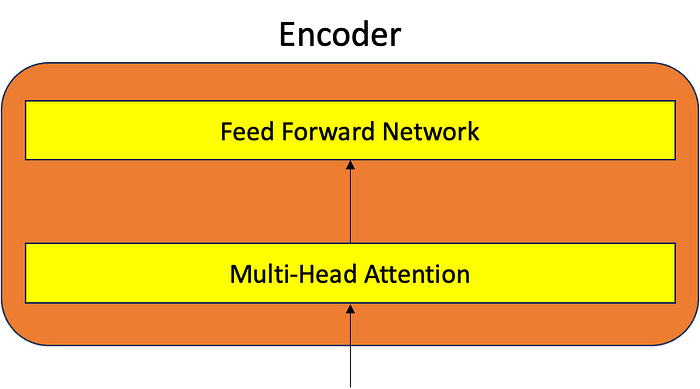
The computation is simple. Take the average across outputs of hidden units in the layer. Therefore, the normalization is independent of #of samples in a batch.

This allows us to work with a batch size of 1 (if needed as in the case of RNN) then number of neurons or features is typically high which is designated as dmodel ( 512 / 256 entries). This standardization is done across all the outputs ie., layer instead of a batch.

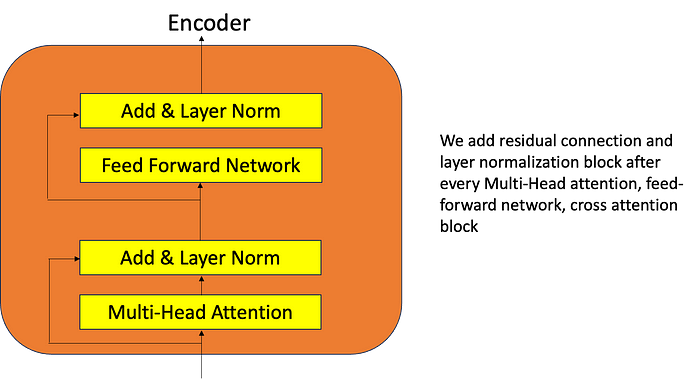




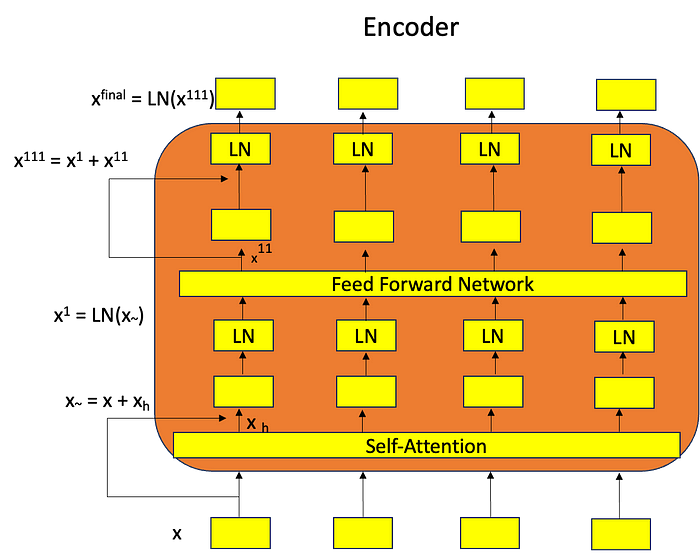
Till now we have assumed that we had Encoder like this —



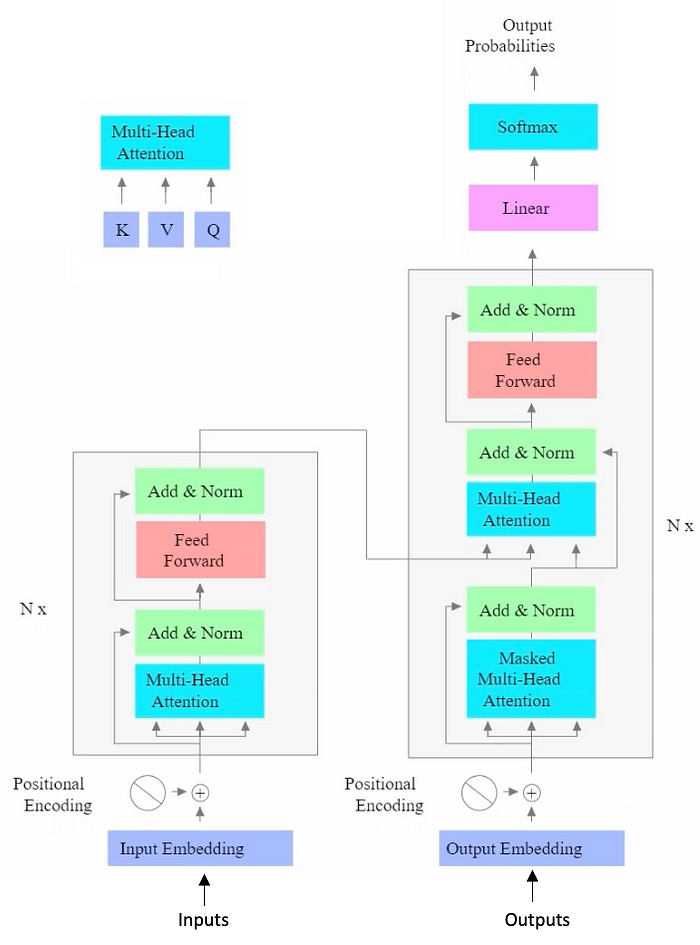
But after adding the above concepts on adding residual blocks and layer normalization. The encoder will look like this.



This is how the calculations are done in the above encoder and all the outputs represented as shown below basically explaining after every attention layer we add LN and after every Feed Forward Network we again add LN.



Here is the complete Transformer architecture is shown below.



**Some Key Components:**

* Multi head attention — which has softmax formula with (Qtk)\*V
* Feed Forward Network — simple one hidden layer network
* Masked Multi-Head Attention — why we had to masking is to do teacher forcing as well as because in practice we do not have access to remaining tokens — so we need to mask them out as it is not legal to look at them. So we did this masking and we implemented using a matrix which has 0s and -infinity.
* Feed Forward Network again — we wanted to predict the distribution over the vocabulary thats when we need the linear and softmax layer.
* To improve or stabilize the trianings with convergence to make it fast — we added residual connections and layer normalization.

**Please do clap**👏**or comment if you find it helpful ❤️🙏**

**References:**

Introduction to Large Language Models — **Instructor**: [Mitesh M. Khapra](http://www.cse.iitm.ac.in/~miteshk/)

[Large Language Models](https://medium.com/tag/large-language-models?source=post_page-----1435248866d6---------------large_language_models-----------------)

[Deep Learning](https://medium.com/tag/deep-learning?source=post_page-----1435248866d6---------------deep_learning-----------------)

[Naturallanguageprocessing](https://medium.com/tag/naturallanguageprocessing?source=post_page-----1435248866d6---------------naturallanguageprocessing-----------------)

[Transformers](https://medium.com/tag/transformers?source=post_page-----1435248866d6---------------transformers-----------------)

[Attention](https://medium.com/tag/attention?source=post_page-----1435248866d6---------------attention-----------------)