Deep Learning

1. **What is deep learning**
2. **What is perceptron**
3. **What is ANN**
4. **Differnce between ANN vs Biological NN**
5. **Input layer, Output Layer**
6. **What is flatten layers**
7. **What is Dense Layers**
8. **What is Hyperparameters and Parameters**
9. **Activation function**
10. **Vanishing/exploding Gradient**
11. **Sigmoid function**
12. **Relu**
13. **Dying Relu**
14. **Leaky Relu(max(αz, z))**
15. **parametric leaky ReLU (PReLU)**
16. **Batch Normalization**
17. **Should we train large dnn from scratch?**
18. **Transfer learning**
19. **What is optimizer**
20. **Momentum optimization**
21. **Nesterov accelerated gradient**
22. **AdaGrad**
23. **RMSProp**
24. **Adam and nadam**
25. **Optimal Learning Rate [start with high learning rate and then reduce it]**
26. **Softmax**
27. **What is RNN**
28. **What is LSTM**
29. **Avoid Overfitting through Regularization (l1 and l2 regularization)**
30. **Dropout, Monte Carlo Dropout**
31. **Max norm regularization**
32. **What is CNN [convulation+pooling+convulation+pooling+ fully connected]**
33. **Filter, feature map**
34. **What is Pooling layer, do it have weights?[Relu is used]**
35. **LeNet-5 (i C P c p c f f)(activation function tanh)**
36. **AlexNet (i c p c p ccc fff)(activatin function relu)**

37. **GoogleNet (have dropout layer)**
38. **VGGNet**
39. **ResNet (152 layers), resNet-34**
40. **SENet**
41. **Object Detection - YOLO**
42. **What is transformers**
43. **What is LLM**
44. **About t5, gpt-3, bert**

It is generally not a good idea to train a very large DNN from scratch: instead, you should always try to find an existing neural network that accomplishes a similar task to the one you are trying to tackle (we will discuss how to find them in Chapter 14), then just reuse the lower layers of this network: this is called transfer learning. It will not only speed up training considerably, but will also require much less training data.

1. Deep Learning:

Deep learning is a subset of machine learning and a branch of artificial intelligence (AI) that focuses on training artificial neural networks with multiple layers (deep neural networks) to learn patterns, features, and representations directly from data. It uses large datasets and high computational power to solve complex problems that are difficult to address with traditional machine learning methods.

2. What is a Perceptron?

A perceptron is one of the simplest types of artificial neural networks, introduced by **Frank Rosenblatt** in 1958. It serves as the foundation for more complex neural network models. The perceptron mimics a biological neuron and is primarily used for binary classification tasks.

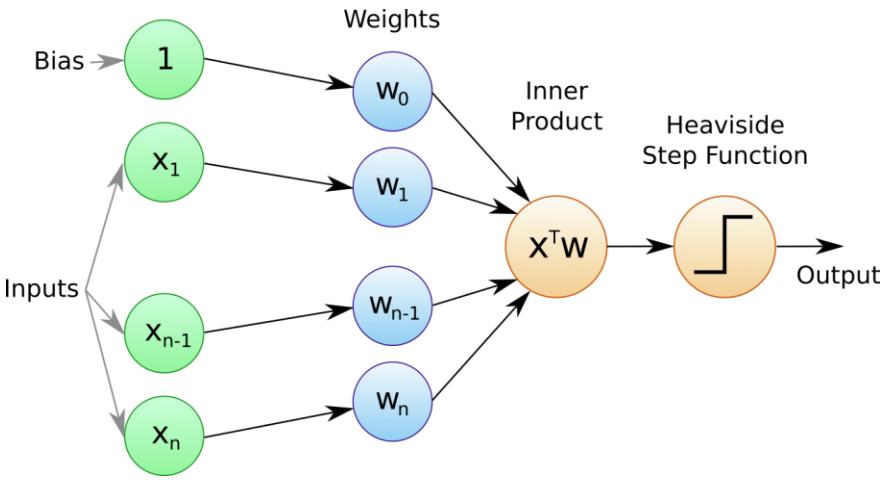
Components of a Perceptron:

1. **Inputs** ($x_1, x_2, \ldots, x_n$):
   - Features of the data (e.g., pixel values in an image or attributes in a dataset).
2. **Weights** ($w_1, w_2, \ldots, w_n$):
   - Assigned to each input to determine its importance in the prediction.
3. **Bias** (b):
   - An additional parameter to adjust the decision boundary and ensure flexibility.
4. **Weighted Sum:**
   - Computes the linear combination of inputs and weights:

$$z = \sum_{i=1}^{n} w_i \cdot x_i + b$$

5. **Activation Function (Step Function):**
   - Applies a threshold to determine the output:

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$



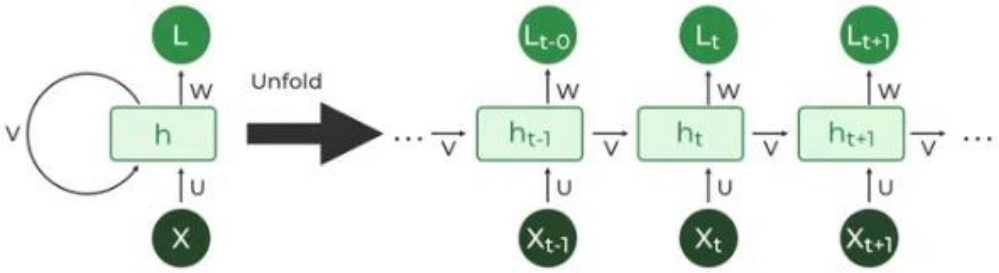
RNN:

In traditional neural networks, inputs and outputs are treated independently. However, tasks like predicting the next word in a sentence require information from previous words

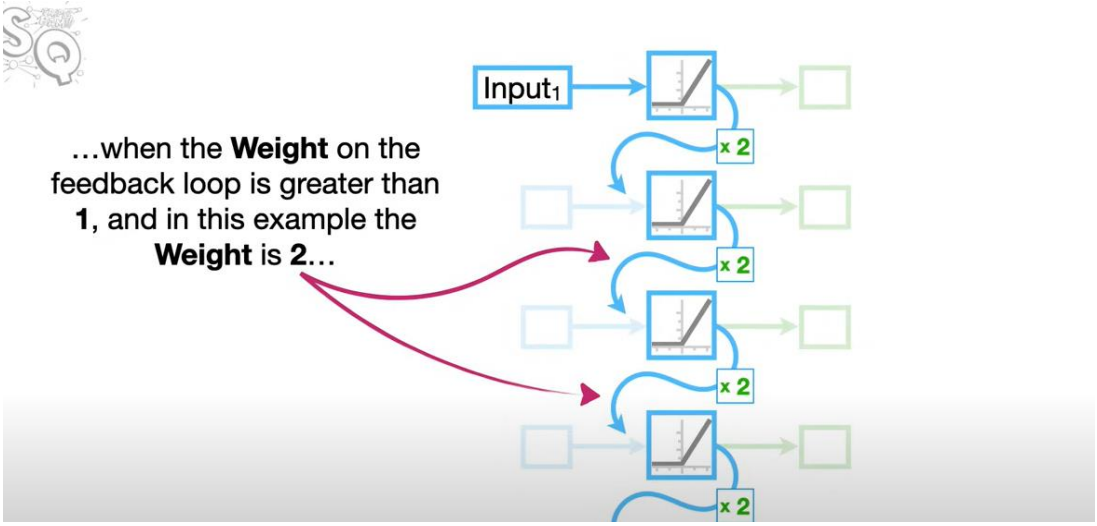
to make accurate predictions. To address this limitation, Recurrent Neural Networks (RNNs) were developed.

Recurrent Neural Networks introduce a mechanism where the output from one step is fed back as input to the next, allowing them to retain information from previous inputs. This design makes RNNs well-suited for tasks where context from earlier steps is essential, such as predicting the next word in a sentence.

The defining feature of RNNs is their hidden state—also called the memory state—which preserves essential information from previous inputs in the sequence. By using the same parameters across all steps, RNNs perform consistently across inputs, reducing parameter complexity compared to traditional neural networks. This capability makes RNNs highly effective for sequential tasks.



Basic RNN are hard to train because it can cause vanishing/explode gradient



Since we take the previous input and multiply it by weight. If we have weight greater than 1 i.e 2 then for 50 word, the grade will be Input X 2^50 it will explode.

If it is 0.5 then Input X 0.5^50 tends to zero so the gradient will vanish. RNN suffers from this

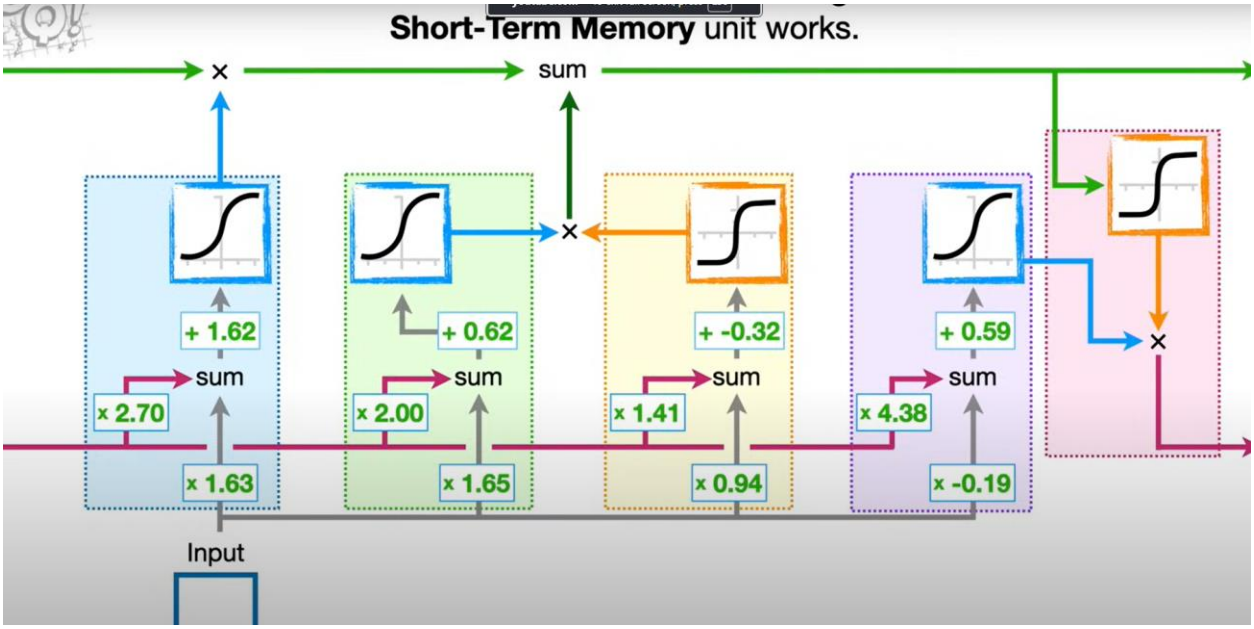
LSTM:

It has 2 path. One is long term memory another is short term memory.

LSTM uses tanh activation function and sigmoid activation function

Sigmoid activation funcition gives value between 0 and 1

Tanh activation function gives output between –1 to 1



LSTM has many stages

1stage: We get Input(suppose 1) we have a short term memory (pink line as 1) and long term memory as pink line with 2 value.

Input multiplied by 1.63 weight and short term memory multiplied by 2.70 weight and then summed then 1.62 bias is added then gone to sigmoid activation function. The output is then multipled by long term memory. (FORGET GATE)

% of long term memory to remember

2nd stage: % Potential memory to remember. Two cell

Potential long term memory uses tanh

Both 2nd and 3rd stage is multiplied and then summed with previous to have new long term memory

2nd stage is called input date

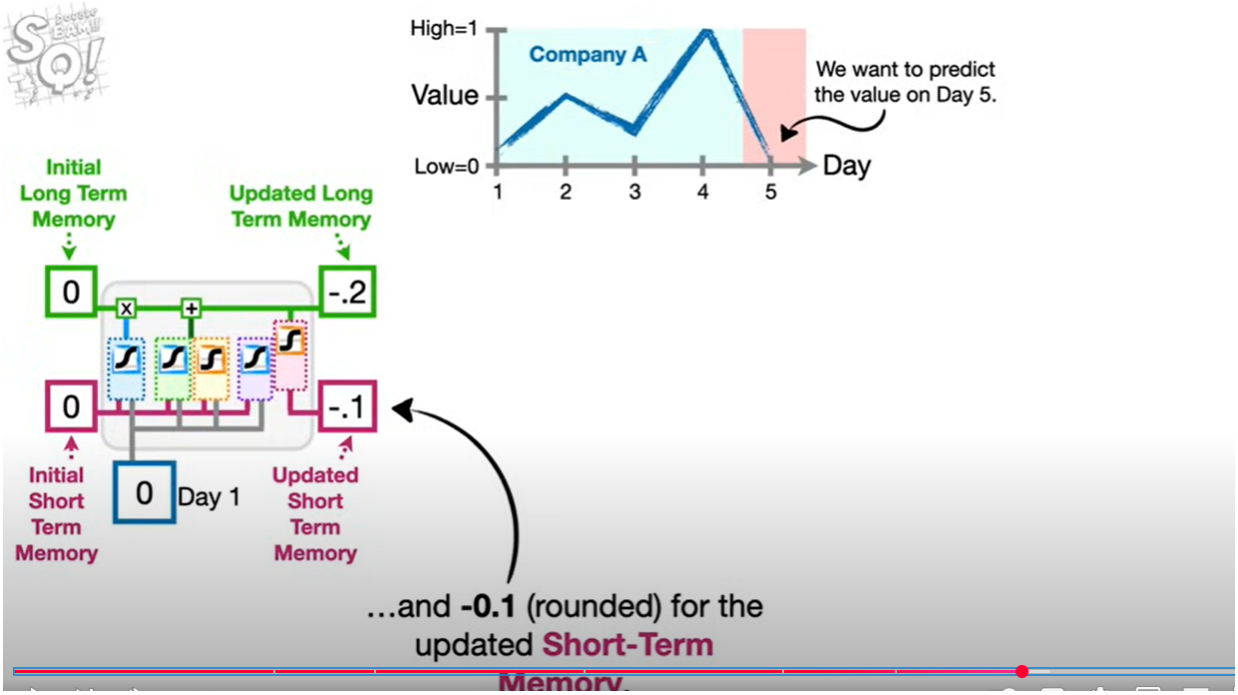
3rd stage called output gate.

Two cell violet and pink

Violet(%potential short term memory to remember) uses sigmoid

Pink(potential short term memoty): uses tanh

5th cell get input from long term memory and then go through tanh activation to get a result which is multipled by cell 4 or short term memory to remember then we get a output or prediction.



42. Transformer:

In Rnn and lstm, they are slow as they take sequential inputs. Lstm is more slower than rnn but gives better result. They are sequential. They rely on the previous input. Such method doesnt use today's gpu well as they are designed for parallel execution

So in 2017 google published attention is all you need to propose a new architecture known as transformers. They completely removed the sequential thing and introduce attention layer. Attenton layer have multi head attention layer.

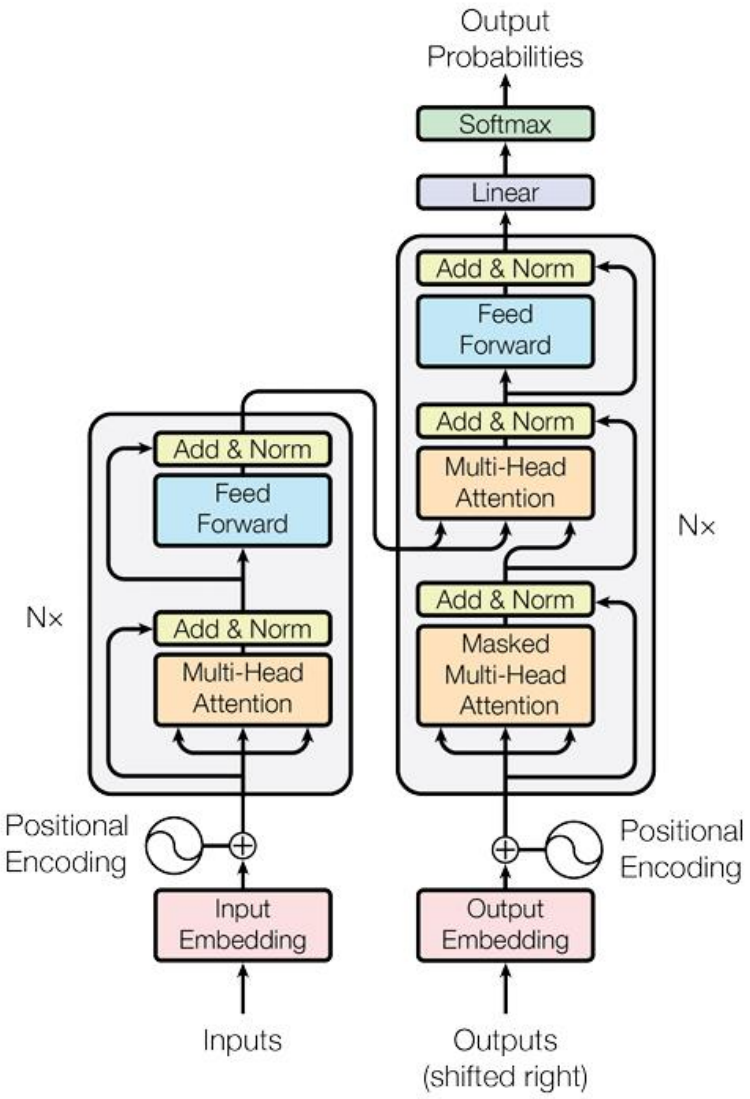


Figure 1: The Transformer - model architecture.

They use encoder decoder architecture like rnn.

Lets take a sentence **The big red dog**

In RNN encoder. The words are given in sequentially to form a embedding vectors which depends ton the previous word embedding

Transformers take out the add simultaneously

Input Embedding

Embedding – form vector of word in embedding space where similar words are close

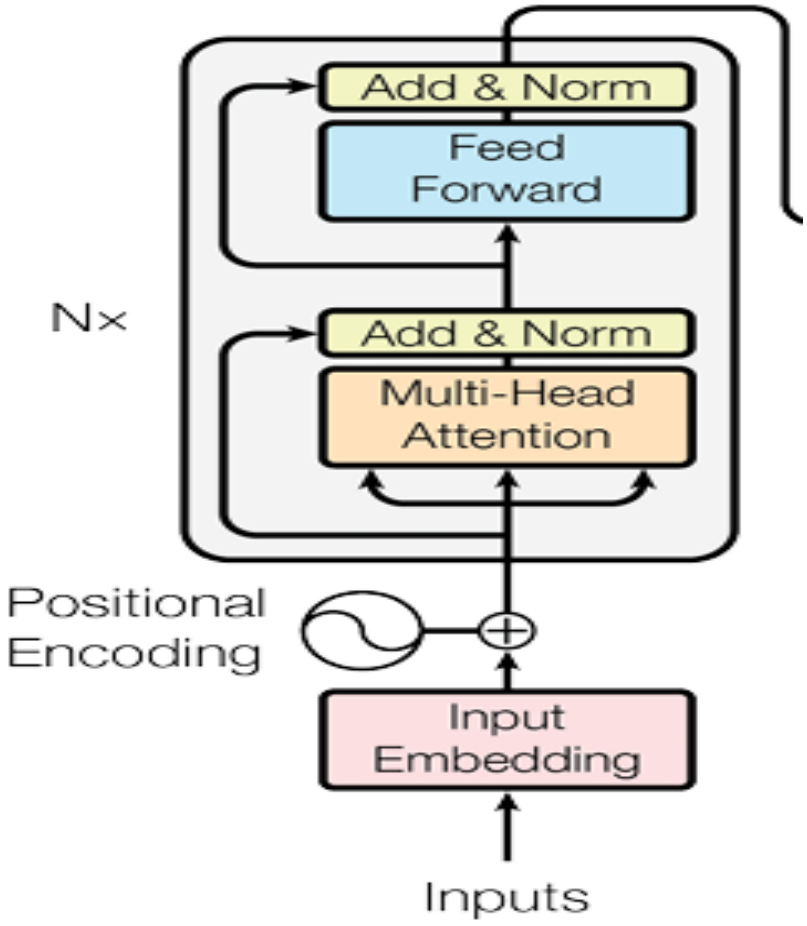
Positional encoder: same word in different sentences may have different meanings

Vector that gives context based on position of word in sentence

Aj dog is cutie

Aj looks like dog

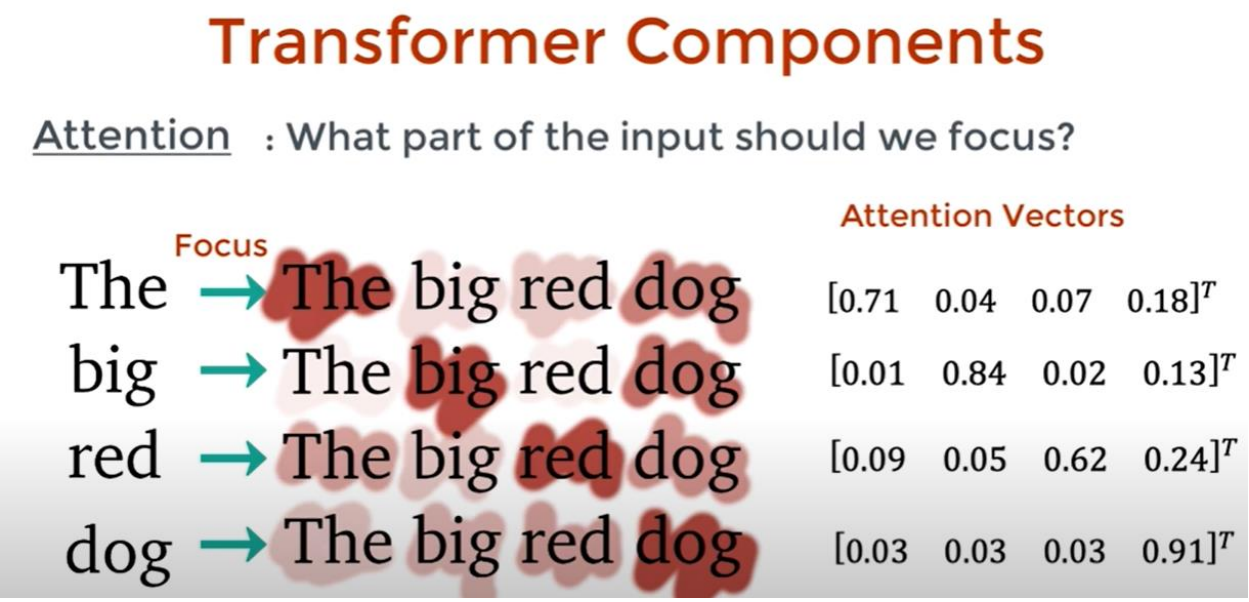
Embedding of dog+ vector encoding of position in sentence = embedding of dog (with context)



This goes into a multi head attention layer

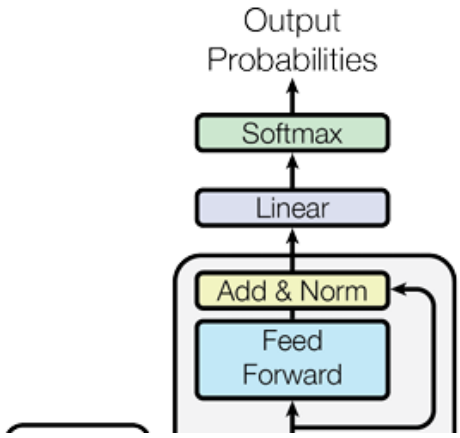
Attention: what part of input should we focus

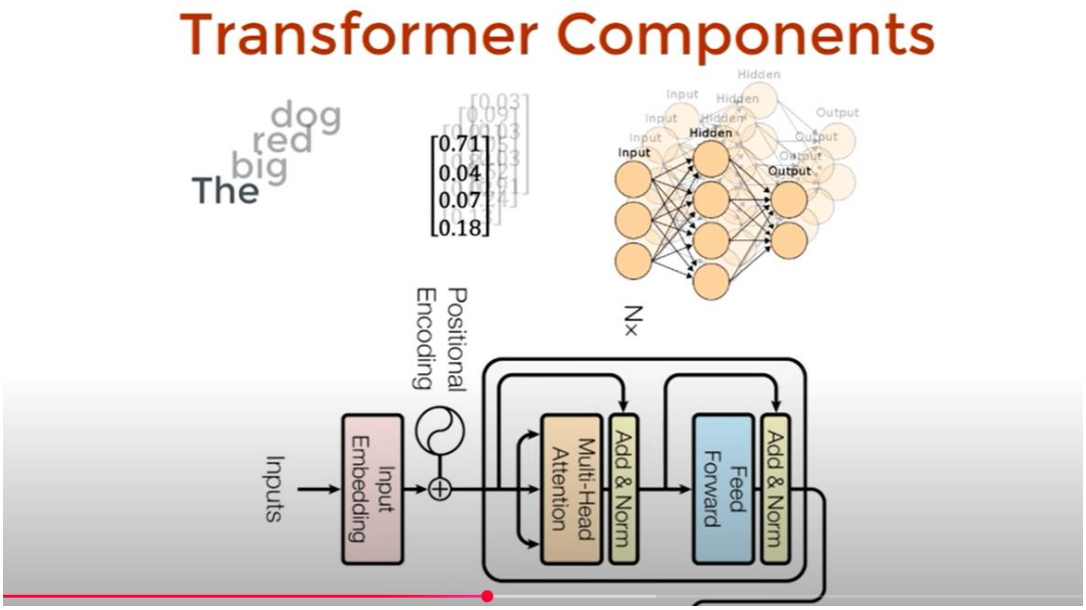
How relevant is i th word with other words



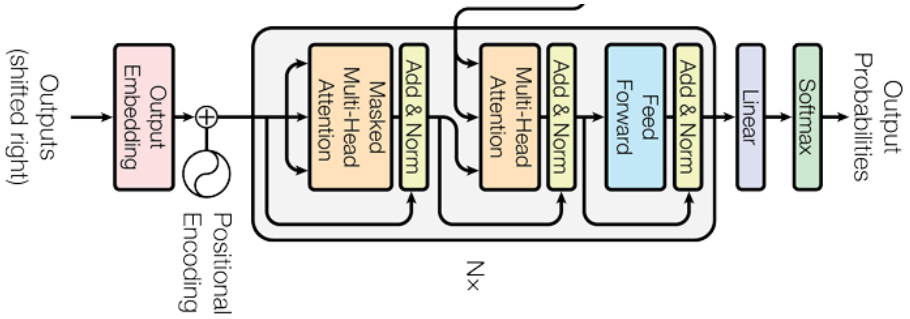
It creates attention vectors after that.

Other thing here is feed forward network





All the attention vectors are gone into feed forward network

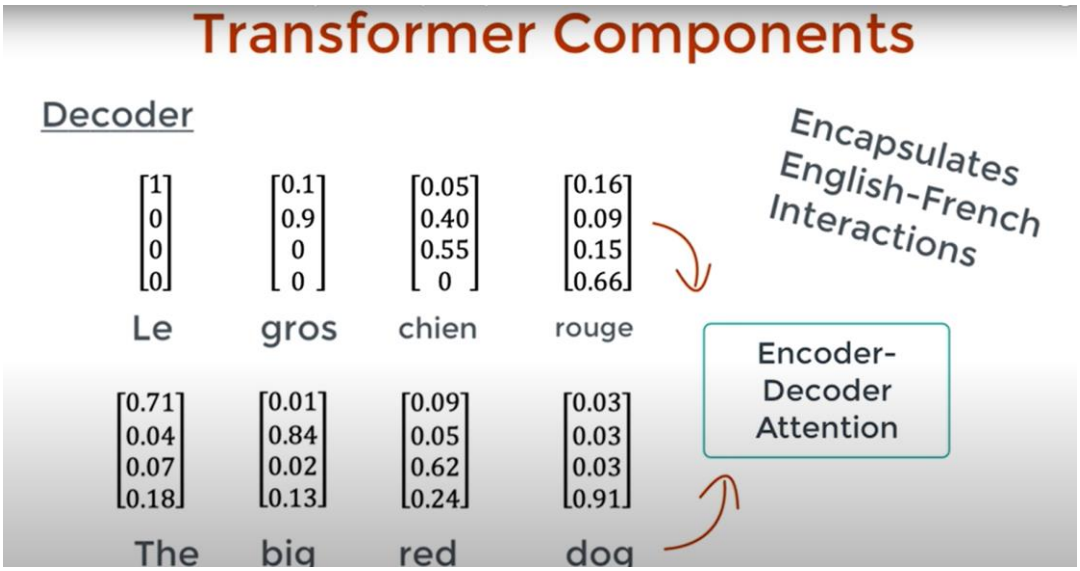


In Decoder, We give the output suppose french language into it

We add Positional encoding. And get positional embedding

image represent the attention block how much each word related to each other

This is where english to french conversion happens



After this it goes to Linear layer whichis another feed forward network

Then softmax changes into probability distribution interpretable to humans. This is the reason to predict next word based on the input which also called generative model.