6 layers in gpt

Self attention helps to find the important words in a sentence

input/prompt → model which is trained on a large dataset → output/response

Self attention parameters : QKV —> Query Key Value

Self attention = Softmax(QKT / sqrt(dk)) V

V: In- depth features

K: Response

Q: Question

dk : Dimensionality of the matrix

Softmax : we get the probability distribution

**Architecture of Transformer**

All the components are separate blocks which work together connecting one output to the other input to finally give the output of the transformer.

Input embedding -—->Positional Embedding —--> Self Attention -—->Feed forward —> Normalization —->Output

**Input Embedding**

→Splits the individual words into different components, ie **tokenization**.

**Position Embedding**

→Word positions are encoded in the sentence

**Self Attention**

→Used to find the relationship between different elements of the input sequence and to find in depth features of the input sequence

**Feed Forward**

→Provides in depth feature extraction

**Normalization**

→Prove solutions for errors and other issues that might occur with the model

Eg: Gradient explosion, Errors etc

—---------------------------------------------------------------------------------------------------------

**Training**

→Means you are inputting your data to a pre existing model and then make the model do well on your input data

**Fine Tuning**

Eg: The quick brown fox jumps over the lazy dog.

**Layer 1** (Input Encoding) : It tokenizes the sentence and places it as tokens.

Converts each word into a 4d matrix

**Layer 2** (Positional Encoding) :

PE(pos,2i)= sin(pos/10,000 2i/dmodel)

PE(pos,2i+1) = cos(pos/10,000 2i/dmodel)

**Layer 3** (Self Attention) : Comparing itself(Each word) with other words to get more features.

Self attention = Softmax(QKT / sqrt(dk)) V

Q=XWQ

K=XWK

V=XWV

For example, we are querying on “quick”, K compares the relevance of the words with each other and how much importance can be given to the query. In this particular example, it shows how much the input is relevant to the key.

V is the value vector which gives the aggregation score, i.e. it is an aggregator vector which is highly dependent on Q. We obtain it during training and it represents the in-depth and detailed features of the data.

**Layer 4** (Feed forward) : Max(0,x) i.e. RELU function

FFN= max (0,xW1+ b1) W2 + b2

It has MLP layers, i.e. multilayer perceptron layers.

W : weight matrices

b : biases

FFN gives the high dimensional representation which is very hard to visualize. This layer is responsible for artificial generative AI, AI with higher intelligence than humans etc etc

**Layer 5** (Normalization) : mapping the outputs to sensible values in the output domain.

Layer normalization(x) = () +

X: input(h)

: mean of input

: standard deviation

Gamma beta are learned parameters

**Layer 6** (Multihead attention) : Part of the decoder architecture. After this , we perform FFN, normalization etc

Multihead(Q,K,V) = concat(head1 , head2 ….. headi )

headi = attention(Q WiQ , K WiK , V WiV )

These heads are used to perform more feature extraction

Wi  are the weight matrices, multiplied with the query , key and value matrices to perform many attention steps.

**Layer 7** (Feedforward layer) :

Gate —> Up and down

Gate → High dimensional projection

Up → Deeper high dimensional projection

Down → Back to embedded projection

**Layer 8** (Normalization)

**Layer 9** (Output)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Summary**

* Input is tokenized and split into a matrix with all the different words in the query.
* Then a token ID is assigned to each of the tokens.
* This token ID is mapped to embedding vectors wherein each token is converted into matrices of the specified dimensions.

* We get output logits(numbers) which are the raw unnormalized predictions which are generated by a model before applying any activation function.
* Then the softmax of logits give the probabilities which are then used to generate the solution based off of token id’s. This is called token selection wherein each token is monitored and the token with the highest probability is selected and put into the output.

KV cache is a method to reduce the computational cost by storing and retrieving previously computed data thus enabling the model to not recompute previously processed data .