Feed Forward Block -

→ Moves the things ahead

↳ sequence ↴

Linear layer (weights)

↓

Activation (GELU)

↓

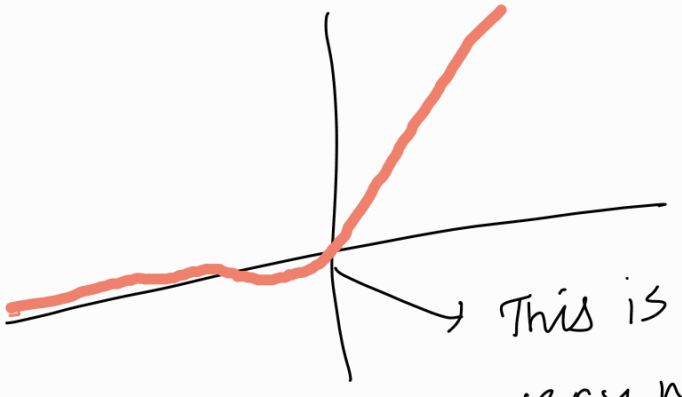
Linear layer (weights)

↓

Dropout

GELU Activation →

Gaussian Error Linear Unit



→ This is -ve for values very near to zero ↓ helps in convergence somehow?

$$= \left(0.5 \times x\right)\left(1 + \tanh\left[\sqrt{\frac{2}{\pi}}\left(x + 0.044\, x^3\right)\right]\right)$$

→ Layer Normalizations.

Shift & Scale $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

↓ &

Matrix $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

Matrix with 1

with zero

→ makes the mean zero & variance 1.

→ So let's say you have weight matrix → it makes the mean zero & var 1 for the last axis to stabilize it & we add some shift & scale to that, the shift & scale can be learned by the model during backprop.

→ Causal Attention:

Basically attention mech. which only lets the model to see the prev. tokens during train so that it can predict the next token

this is done by assigning KQV matrices.

we have each token is

query] related to other

keys token

values ↓

→ query @ keys.T

↙

this is like each word is a query & has many keys (all other tokens) to relate, it relates with them

a @ k.T → atten_scores

we get attention scores

↓

lets say we have a input of size (2, 6, 3)
↓ ↓ ↳ in_dim
batch tokens

* lets say you want op dim of 2.

final op will be (2, 6, 2)

context = 6

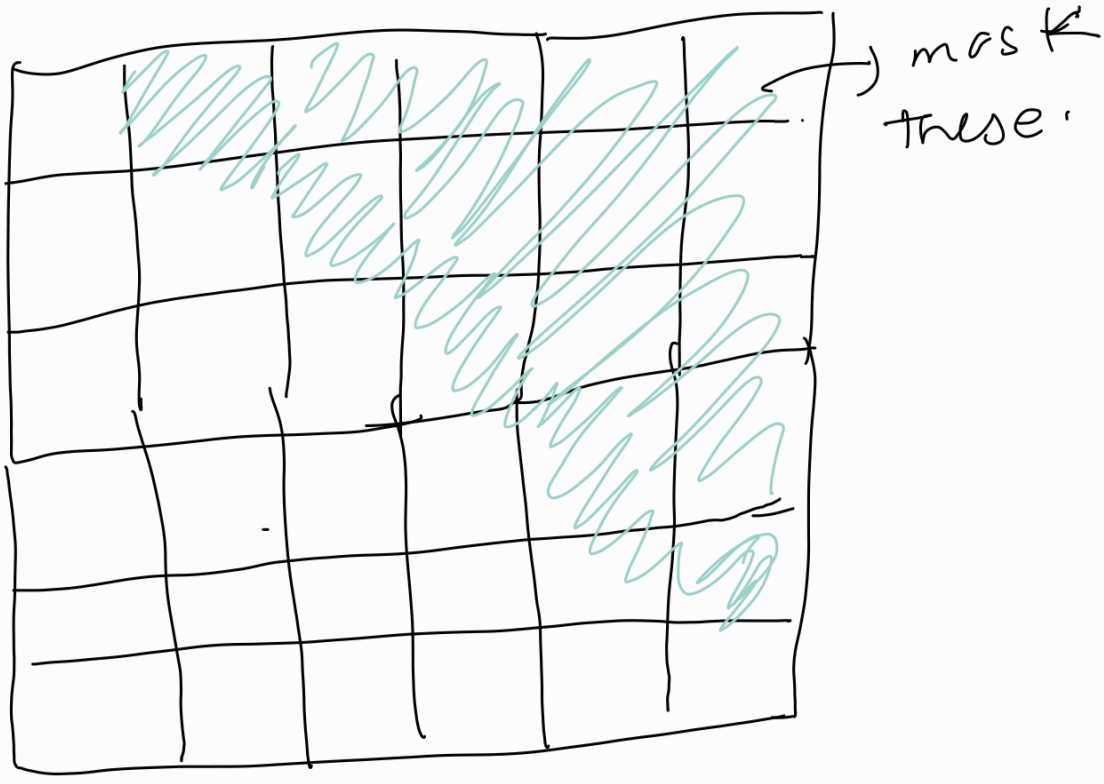
→ q shape = (2, 6, 2)
k shape = (2, 6, 2)
v shape = (2, 6, 2)

(2, 6, 2) . T
(2, 2, 6)

→ (2, 6, 2) @ (2, 2, 6)

6 × 6 → attn_scores

now we mask the future ops.
& add dropout.



mask the upper portion of the matrix tril.

→ make it as register buffer so that it can be on GPU wise weights. → Not on same device error.

register_mask_buffer('mask',
torch.tril(torch.zeros(CL,CL),
diagonal=1))

↳ apply softmax → to basically
wound up the attn scores b/w
& 1 value. 7
where their
sum = 1.

↳ its like fixing the value
b/w 2 numbers

↓

These are now attn_weights

↓

dropout random values.

↓

now we matmul attn_weights
with values. to calculate
context_vectors

cv = attn_weights @ values

(6, 6) @ (m, 6, 2)

↓ → (2, 6, 2)

↓

this aligns

(it not transpose the result

to match (Batch, Tokens, d_out).

↳ out_proj (optional)

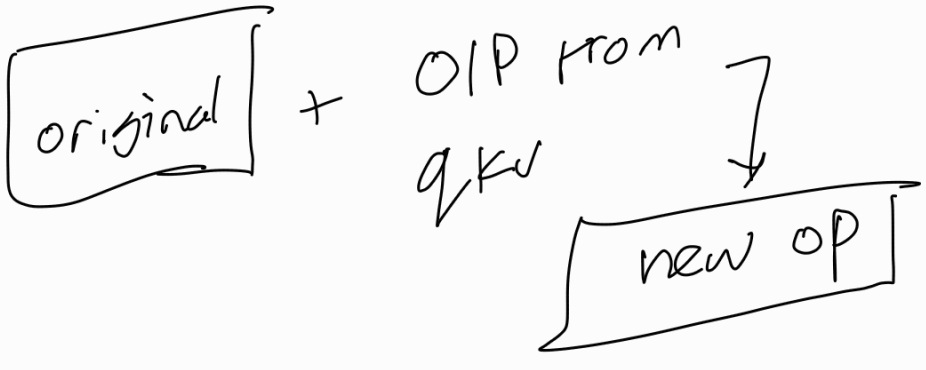
↳ embed_dim → embed_dim

↳ helps with ops.

↳ resid_drop) (optional)

Dropout

→ interpret it like this



We subtract or dropout from the O/P from qkv

So it becomes

original o/p + o/p from q K V

- Something

new OP

By dropping this something we encourage model to learn by not getting all the feedback / hints.

→ Transformer Block

i/p's

↓

Layer Norm 1

↓

attention

↓

dropout

↓

Layer Norm 2 → Feed forward

↑

dropout

↑

output

↓

This is a point where we can add shortcuts to remove problem of vanishing gradient, basically we bypass the feedback & directly give it to the FF-module.

attention block contains two Causal Attn blocks → can add multiheads there.

→ Feed forward block is like

↓

linear

↓

GeLu

↓

dropout

↓

linear.

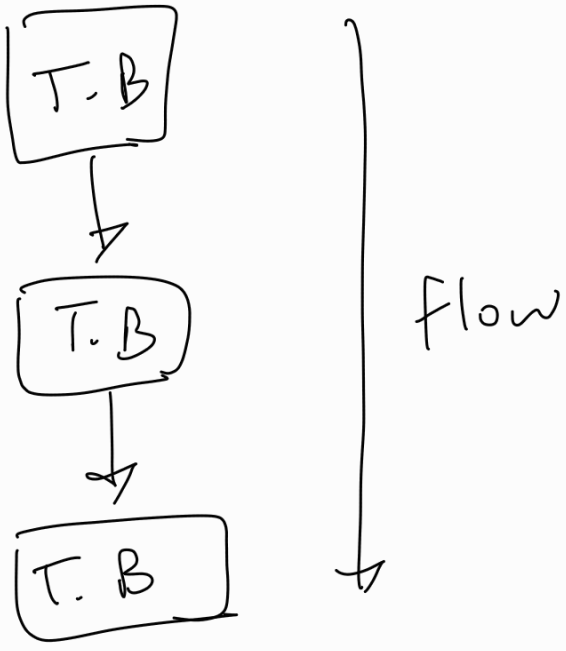
→ Putting it together.

→ word embeds

→ pos embeds

→ configs.

we make multiple transformer blocks which is layers in the model



we make a list → ModuleList

with transformer blocks &
configurations like
embeddings, layers, heads,
head_dims, dropout,
context length, etc ...

→ add final norm layer
for the op's.

→ make a layer that gives out
probab distribution of words.
(this is logits) → helps
get the next word → one with
highest probab.

→ flow

→ inputs → tokens → to device

↓

pos embeds (create)

arange → to device

↓

combine all embeddings

x = token + pos

↓

dropout

↓

pass through layer of Transformer Blocks

↓

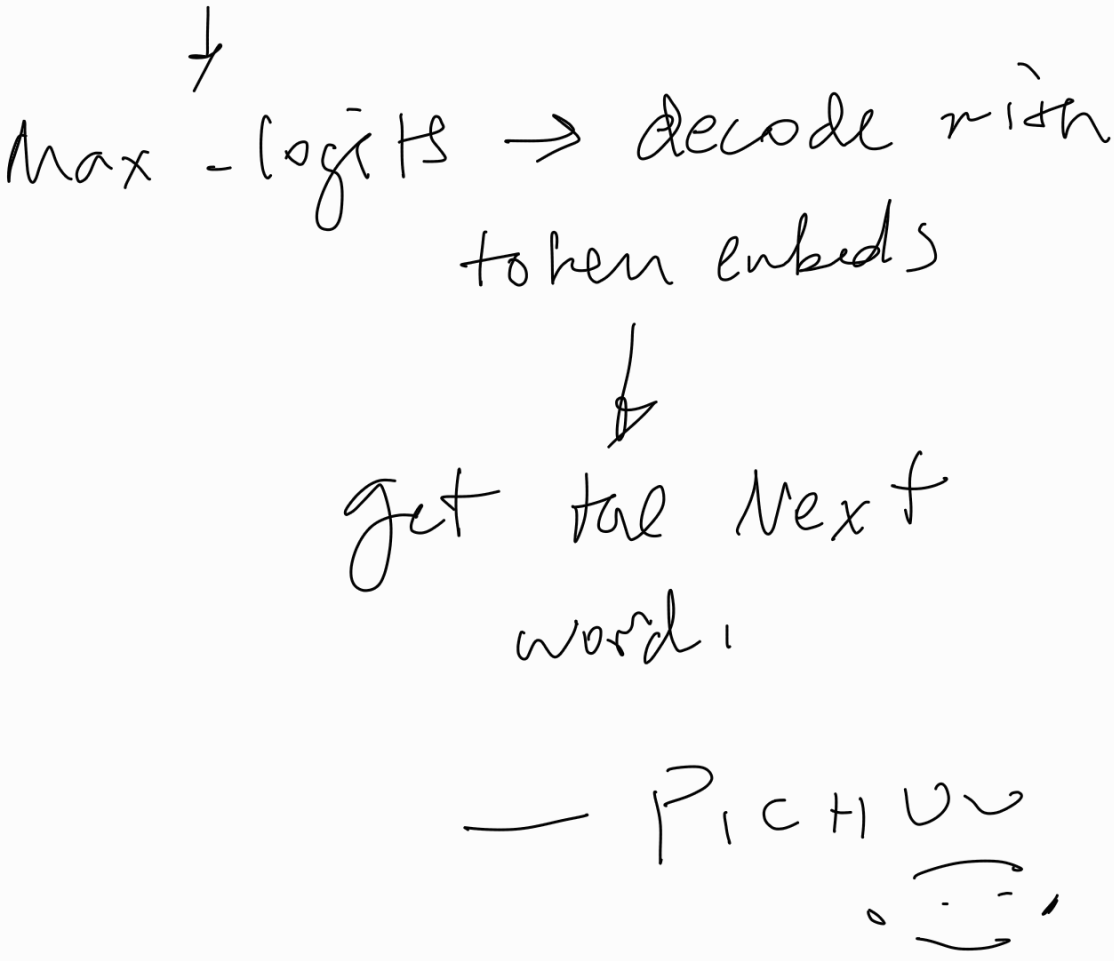
final Norm layer

↓

final Prob dist (logits)

↓

returns logits & when to get the next word → max(logits)


Max - logits → decode with
token embeds
get the Next
word.
— PICHU