5th May:

What is a trasformer:
we wanna to open trasformers and see what is inside it.

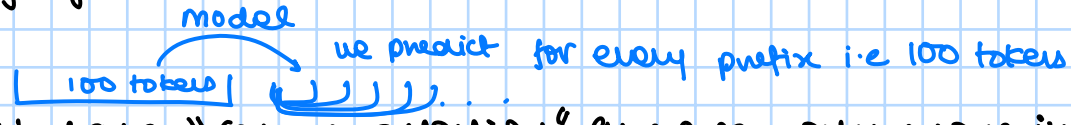
einops - library to manipulate tensor
einsum - take in a bunch of tensors and do multiplication
Easytrasformer - main class of our trasformer

```
gpt2_xl = EasyTransformer.from_pretrained("gpt2-xl")
```
↓
we load an open sourse model

Note: we also need a refernce model (here we take gpt-2-small)

Understanding inputs and outputs of a model:
Trasformers exist to model text, they generate text, model gets prob. samples
Inputs - Tokens
Outputs - logits

model traiing - give bunch of text, train it to predict next token.



we do this to have "casual attention" as we can only move information forward. (Auto-regersive)

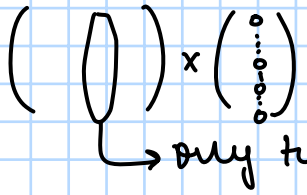
So they are like sequnce modelling engines

input: we convert language to vectors

language → integers → vectors
(one-hot encoding)

1 to 100 to 100 dim vetor
1 in k^{th} position
0 else
} we do this to make all integrs independity

dimension = Each input has its own dim
Lookup tables = matrix x one-hot encoded vetor



language ⇒ integers ← Bounded (say from 0 to 100)

Note: we can just dirrty map language to 256 ASCII but it looses structure
we want "apple" to be a token
"hfg" not

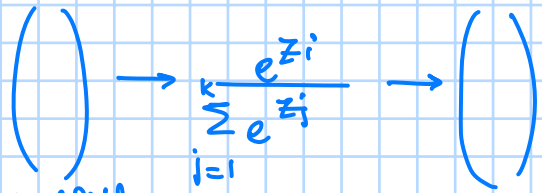
Byte-pair encodings

```
reference_gpt2.tokenizer.vocab
```
← This means sparse
(Shanahan: 48906

```
reference_gpt2.to_str_tokens(input)
```
"Jennifer" → 'Jennifer'
" jennifer" → 'j', 'enn', 'ifer' even arithmetic is bad

reference_gpt2.to_tokens("something here")
vetus tensor [[. . . .]]

logits: we want a probability distribution over next tokens.
we use softmax to convert vetor into prob. distribution
so model outputs tensor of logits (vetor of size d_{vocab} for each input)



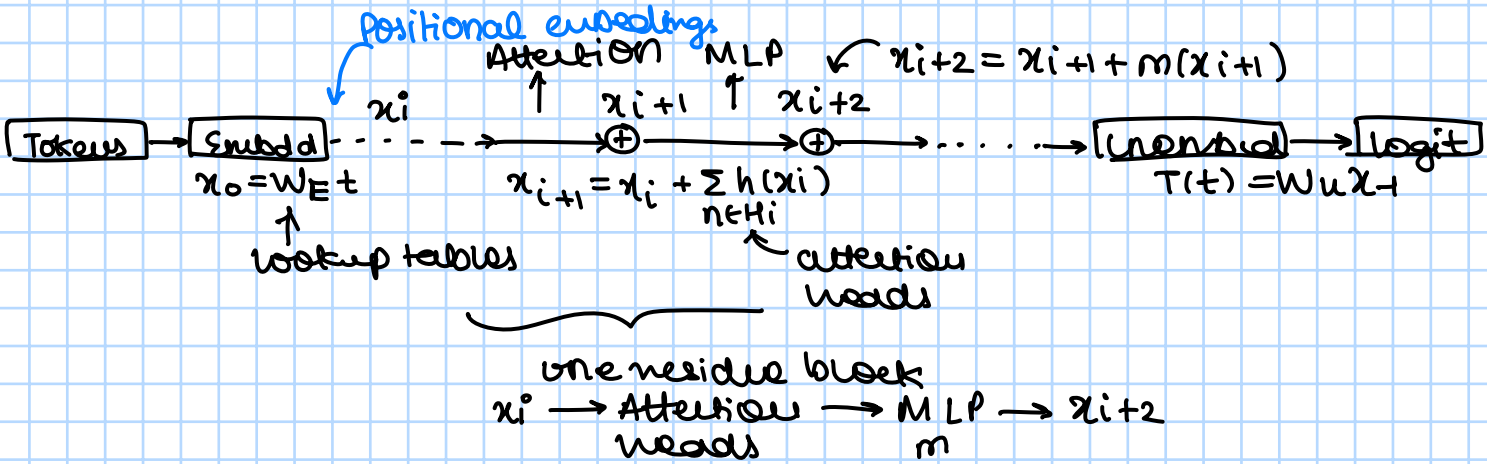
Shape of tensor
Shape = batch x size

Text → tokens → logits
logits, cache = reference_gpt2.run_with_cache(tokens)

shape = batch x size x d_vocab
↑ shape of logit

log_probs = logits.log_softmax(dim=-1)
now we get the next token:
next_token = logits[0,-1].argmax(dim=-1)
add this is input and re-run

Architecture:
Input tokens, integers → Embedding is a lookup table mapping tokens to vectors
vetors (lives in residual stream.



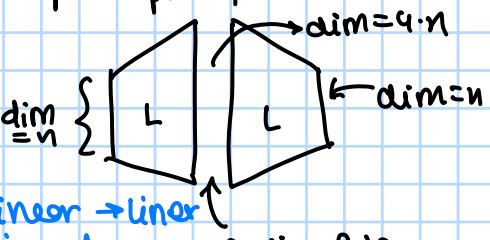
sum of all previous layers
n-layer transformer block
Attention + MLP = one-block
k-layers = k-blocks

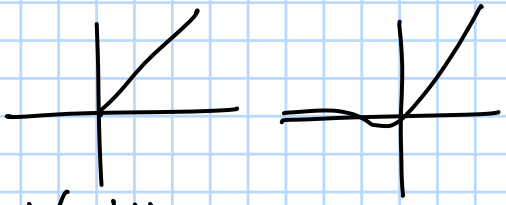
Attention:
Attention moves information from prior position in seq to current token, we do this for every token, we only look backwards and is called attention pattern → probability dist, which some tokes to get input from

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

Query, value, key

MLP: multilayer preseptron, Steadard NN





Linear → non linear → linear
can approximate any arbitary fution

activation geLU / reLU

d_mlp = 4 x d_model

Note: one attention has moved relevant info into to single position, MLP can do computation, reasoning, lookup info, ect.

Unembedding: final step of conig vetor to logits, and logit to text

layer norm: applied at each layer steal, normalisation futions, before Activation, MLP and unembed

each input layr - mean 0, variance 1
- then learnbie scaling and bastation

$y_0 \in \mathbb{R}^n$

$y_1 = y_0 - \sum \frac{y_0}{n}$

$\Rightarrow y_1 = y_0 - \langle\langle y_0, (1,\dots 1)\rangle \frac{1}{\sqrt{n}}, (1,\dots 1)\rangle \frac{1}{\sqrt{n}}$ ↙ equivalet to setting first coord to 0 (good aim)

$\Rightarrow y_2 = \frac{y_1}{\left(\frac{|y_1|}{\sqrt{n}}\right)}$ and this makes $|y_2| = \sqrt{n}$ or var = 1

↳ std of y_1

$\Rightarrow y_3 = y_2 \odot w$ ← representation of residue stream

↑ elent wise multiplication

where $w \in \mathbb{R}^n$

$= y_2 D_w$ ← diagonal with w as diagonal

$\Rightarrow y_4 = y_3 + b$

where $b \in \mathbb{R}^n$

Note: As layr norm is just scale and traslate, we can apply layer norm before auter linear map (linear x linear = linear) so we can just fold this into one single effetive linear layer

positional information: Attention operates over all pair of position, so the attention weight from token 5 to token 1 and 5 to 2 will be calulated same but

Nearby tokens are more relevant

So a new lookup table mapping index of the position of each token to a residual stream vetor, add it to embed.

we add rather than concatate

Matu logic:

LayerNorm:

$X \in \mathbb{R}^{B \times P \times D}$

B = Batch size

P = NO of positions (tokens)

D = hidden dimension

$$\mu_{b,p,1} = \frac{1}{D}\sum_{i=1}^{D} x_{b,p,i}$$

$$x'_{b,p,i} = x_{b,p,i} - \mu_{b,p,1}$$

$$\sigma^2_{b,p} = \frac{1}{D}\sum_{i=1}^{D}(x_{b,p,i} - \mu_{b,p,1})^2$$

$$scale_{b,p,1} = \sqrt{\sigma^2_{b,p} + \varepsilon}$$

$$\hat{x}_{b,p,i} = \frac{x_{b,p,i} - \mu_{b,p,1}}{\sqrt{\sigma^2_{b,p,1} + \varepsilon}}$$ ← to prevent 0 division

if $[2,4,768] \longrightarrow [2,4,768]$
$x_{b,p,d}$ $\hat{x}_{b,p,d}$ (every column is $\hat{x}_{b,p,i}$)

Embedding:

for each $X \in \mathbb{R}^{B \times P}$
↳ input tokens
$x_{b,p} \in [0, V-1]$ ↳ no of vocabs
$X = [x_{b,p}]$
↑ matrix ↳ now, $Embed_{b,p,*} = W_E[t_{b,p}]$
vector (column)
and so $Embed_{b,p,d}$ = cols of $Embed_{b,p,*}$
$\Rightarrow Embed \in \mathbb{R}^{b \times p \times d}$
↳ matrix

so $X \in \mathbb{R}^{b \times p}$
$\Rightarrow W\_E(X) = Embed_{b \times p \times d}$
↑ d is model dimension

position Embed:

$pos\_embed_{p,D} = W\_pos[X_{b,*}]$
small matrix for given batch
↑ find batch ↖ all positions that batch

pos_emb = all values are like $W\_pos[x_{b,*}]$
full matrix $\in \mathbb{R}^{B \times P \times D}$ ↑ find column

Attention:

$\hat{x} \in \mathbb{R}^{B \times P \times D}$ (normalised input)

$$Q[b,i,h,:] = \sum_{j=1}^{D} x[b,i,j] \cdot W_Q[h,j,:] + b_Q[h,:]$$

$$K[b,i,h,:] = \sum_{j=1}^{D} x[b,i,j] \cdot W_K[h,j,:] + b_K[h,:]$$

$$V[b,i,h,:] = \sum_{j=1}^{D} x[b,i,j] \cdot W_V[h,j,:] + b_V[h,:]$$

$Q, K, V \in \mathbb{R}^{B \times P \times H \times d}$
↑ ↑ dim of head
n head

$$attention\_score = \frac{Q[b,i,h,:] \cdot K[b,j,h,:]}{\sqrt{d}}$$

$$\text{att\_score} \in \mathbb{R}^{B \times H \times P \times P}$$

for each pos i, head h

$$\text{attn\_score}[i,j] = -\infty \text{ if } j > i$$

$$\text{pattern}[b,h,i,:] = \text{softmax}(\text{att\_score}[b,h,i,:])$$

$$\text{pattern} \in \mathbb{R}^{B \times H \times P \times P}$$

$$Z[b,i,h,:] = \sum_{j=1}^{P} \text{pattern}[b,h,i,j] \cdot V[b,j,h,:]$$

$$Z \in \mathbb{R}^{B \times P \times H \times d}$$

$$\text{attn\_out}[b,i,:] = \sum_{h=1}^{H} \sum_{k=1}^{d} Z[b,i,h,k] \cdot W_o[h,k,:] + b_o$$

$$\text{attn\_out} \in \mathbb{R}^{B \times P \times D}$$

MLP:

$\hat{X} \xrightarrow{L}$ Activation $\xrightarrow{L} X' \longrightarrow \hat{X}'$

Unrolled: Same as previous