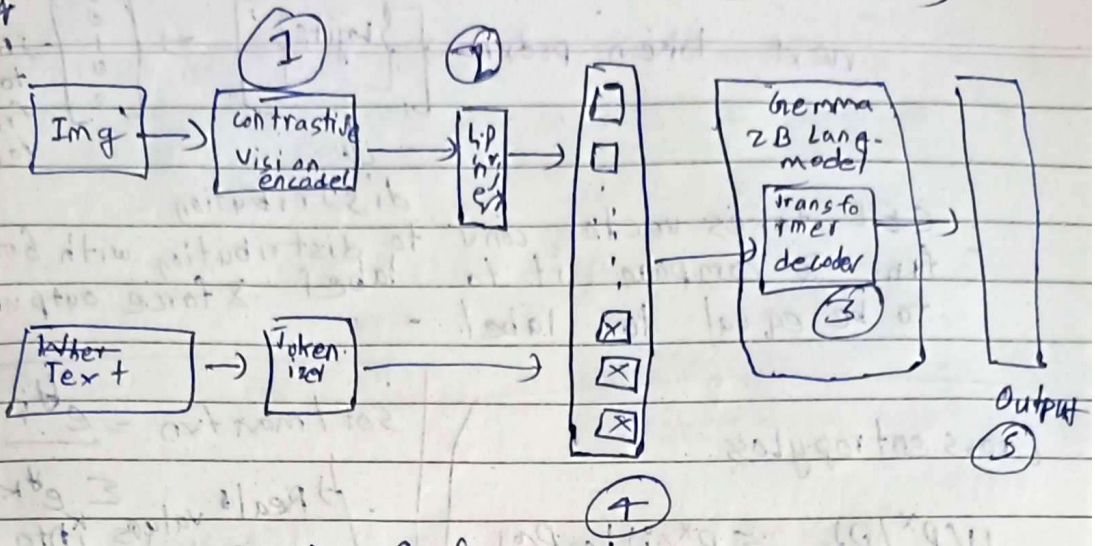
VLMs from scratch in Pytorch (PaliGemma)

0:34



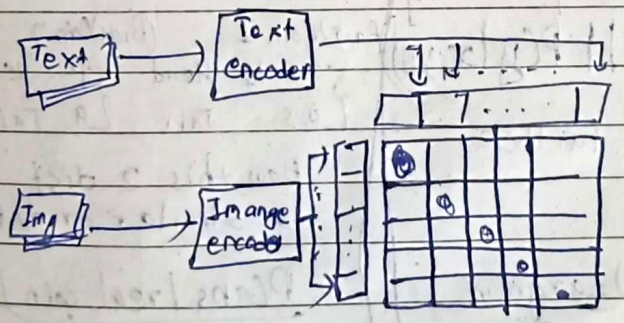
VLM can extract info from img.

Topics
- Vision Transformer
- Constrastive learning (Clip, SigLip)
- Language Model (Gemma)
  - KV-Cache
  - Rotary Positional encoding
  - Normalization (Batch, Layer, & RMS)

Part I

Q) What is contrastive learning?

1) Contrastive pre-training



{ we want when similar image & text things are dot product it shall give high value. if img & text are diff the value is low

We will first find ~~things~~ everything in ~~high & t~~ each box & then force to we will implement a loss fxn that maximizes the TL-BD diagonal to be high & others to be low. [Cross entropy loss]

Why cross entropy loss (CEL)?

next token prdxn → [{logits}] → [0 0 1 0 0] → next token from vocab

↓ distribution

CEL takes vector, conv to distribution with Softmax fxn & compare it to label & force output to be equal to label

Cross entropy loss:

$$H(P^*|P) = -\sum_i P^*(i) \log P(i)$$

Probablistic theory argues "most natural output will be entirely prob dist."

Param (θ)

x_i	Predicted: $P(y|x_i;\theta)$	True $P^*(y|x_i)$

y = class

An intuitive loss fxn app should be:

we should focus on minimizing KL divergence btwn model dist & true dist

$$D_{KL}(\overbrace{P^*(y|x)}^{\text{True}} \,||\, \overbrace{P(y|x_i;\theta)}^{\text{Predicted}})$$

which is further processed

$$\underset{\theta}{\operatorname{argmin}}\, D_{KL}(P^*||P) = \underset{\theta}{\operatorname{argmin}} - \sum_y P^*(y|x_i) \log P(y|x_i;\theta)$$

$$\underset{\theta}{\operatorname{argmin}}\, D_{KL}(P^*||P) = \underset{\theta}{\operatorname{argmin}}\, H(P^*, P)$$

out

for this to work our dist should be valid dist e such as sum is 1 & 0-1 only & this is maintained by softmax fxn IWD I DIDN'T GET 2nd part

softmax fxn $= \frac{e^{y_i}}{\sum_k e^{y_k}}$

- Real values into prabab
- Only used in output
- Higher prabab as actual output

KL divergence (Kullback Library)

dist measure btwn distributions

discrete: $D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$

continuous: $D_{KL}(P||Q) = \int P(x) \log \frac{P(x)}{Q(x)}$

[Trying to measure dist btwn prob dist]

(Fair) Coin 1: Head, Tail — (Bias) Coin 2: Head, Tail

How this 2 dist is similar or diff

$$\frac{P(\text{obs} \mid \text{real coin})}{P(\text{obs} \mid \text{coin 2})}$$

A natural measurement of dist btwn prob dist. motivated by looking at how likely 2nd dist will be able to generate samples from 1st distribution

Problem with clip: [computationally expensive] softmax $\frac{e^{a_i}}{\sum_{k=1}^{K} e^{a_k}}$

we are using clip

softmax has a problem: it is numerically unstable as exp fxn can grow fast & may not fit in 32 bit FP number

soln: do not make exp grow to ∞

$$s_i = \frac{c \cdot e^{a_i}}{c \cdot \sum_{k=1}^{N} e^{a_k}} = \frac{e^{\log(c)} e^{a_i}}{e^{\log(c)} \cdot \sum_{k=1}^{N} e^{a_k}} = \frac{e^{a_i + \log c}}{\sum_{k=1}^{N} e^{a_k + \log c}}$$

Normally choose $\log(c) = -\max(a_i)$

This will make the arg. of exp towards negative & the exp itself towards zero.

Too many component & not parallelizing efficiently too

assymetryicity in that table we need to calc softmax for every row & every column

---

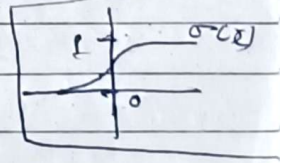
SigLip ~~Soft~~ Improvement on Clip

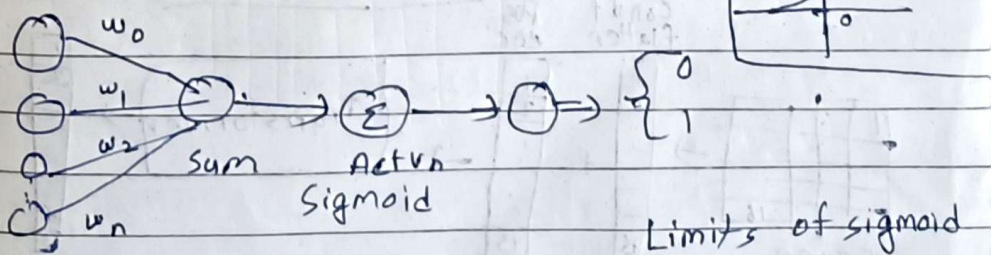
To use sigmoid loss instead of cross entropy loss.

Sigmoid fxn

Plays imp role in logistic regression

Logistic reg: technique to predict outcome of binary classif. problem. $\sigma(x) = \frac{1}{1+e^{-x}}$





Making predictions

Limits of sigmoid $\therefore [0, 1]$

every prdxn $\geq 0.5 \approx 1$ prd_m

$\leq 0.5 \approx 0$

Each box in table is treated independently as binary classif. problem. but in clip we opted towards depending on boxes itself.

And this is done via sigmoid fn.

So, by this way the principal diagonal becomes 1 and other becomes 0 coz of sigmoid & helps in parallelizing & no dependency.

Part 2

Why contrastive vision encoder?

why not normal encoder to capture meaning of any img?

Because we need to integrate it with text & this app. we discussed just does that.

Cheaper to train coz each img has it's description in web

- Noisy dataset

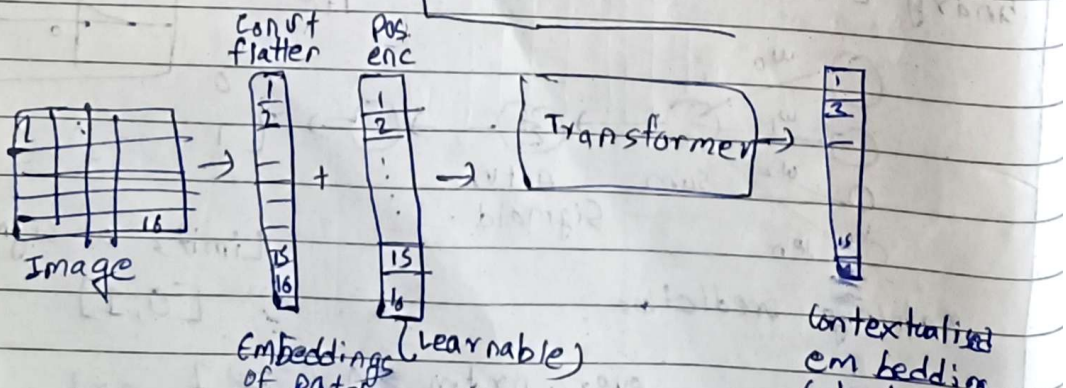
but it is trained on trillions so, eventually it becomes good

---

Part 3 Vision Transformer

Explained in paper

"An Image is worth 16x16 words:..."



With LM we do casual mask but with ~~VLM~~ ViT we don't care about auto regressive nature like I output is depending on it's previous part only

Coding Siglip & input is image [patch embedding] flattened to things but outputs are also 16 things not 1 single thing

- make config ~~& input is~~
  hidden_size = embedding size

- structure [siglipVisionModel]
  __init__ {config, call transformers}
  forward (self, pixel_values) -> tuple:
  [Batch_size, Channels, Height, Width] ->
  [Batch_size, num_patches, embed_Dim]

- SigLIP Transformer
  ~~config~~
  __init__
  config, embed_dim, embeddings,
  encoder, post_layernorm,

  ~~forward (self, pixel_values: tensor) -> torch.Te~~
  hidden_states = self.embedding
  last_hidden_state, (last hidden_state)
  encoder which is basically transformer's encoder
  return last_hidden_state
  in ViT norm is done before FFN & MHA

- SiglipVisionEmbedding
  ~~init~~ __init__
  ~~config~~ siglip
  config, embeddings, encoder,
  postlayernorm conv2D
  num_patch
  ~~num~~ positions
  pos_embeddings
  ~~forward~~ forward (pixel_values)
  patch embed
  embedding = patch_embed .flatten
  embeddings.transpose(1, 2)
  emb = embedding + post_embedding

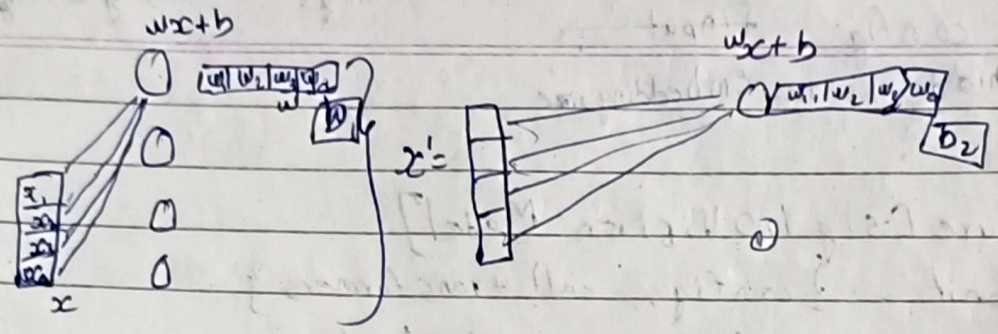
PostLN

Conv2D
input_dim = (H, W, C)
K = Filter
F = kernel size
S = stride (jumping thing)
P = padding

Filter is for each channel diff so, if K = 2, then it has basically 6 diff filter.

element wise mult.
conv is implemented.
[Unfold + Matrix mult + Resize]

Normalization



Problem : Covariate shift

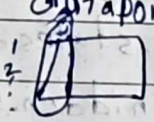
when input vector changes from 1 batch to anthr that changes in magn the output also changes in magn, & it simultaneously affect loss, gradient & stuff, back propagation & affects model

Network learns slowly

Soln:

(1) Batch norm:

dist. datapoints in mean = 0 & $\sigma \approx 1$



we do it along ↓ in

problem with BN is that each statistic depends on what other items are in batch. to get good result we must use big batch size

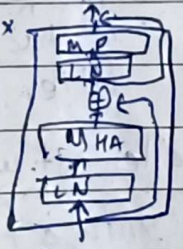
(2) Layer Norm

we do it along row

same mean = μ, $\sigma = 1$

& independent of each other data points

Contextualised embedding



Patch + Postnal embedding.

siglip Encoder
decoding Siglip Encoder

SiglipEncoder Layer
__init__()
super
embed_dim
self_attn = SiglipAttention( )
layer norm1
mlp
layer_norm2

forward
residual = x
hidden_s = layer_norm1(x)
h_s, _ = self_attn(x)
h_s = residual, h_s
residual = h_s
h_s = layer_norm2(h_s)
h_s = mlp(h_s) → why mlp?
h_s = residual + h_s

non-linearity & understand complexity

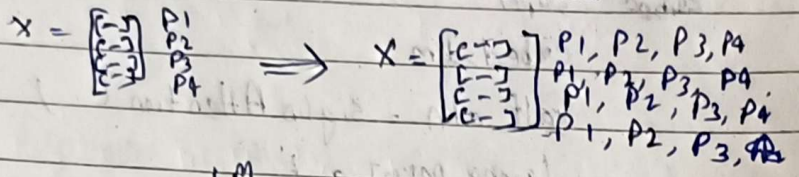
Siglip MLP
__init__
super
config, fc1, fc2
forward
h_s = fc1
h_s = nn.gelu("tanh")
h_s = fc2

Understanding MHA

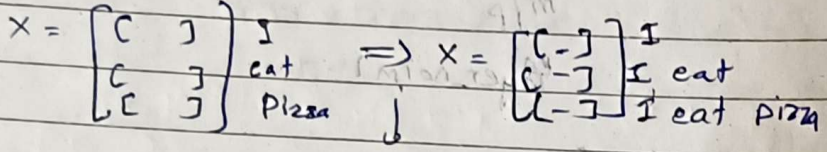
Way of Contextualizing stuff

goal : transform intial seq. of uncontextualised embeddings to a seq. of contextualised embedding

Vision Transformer



LM



It is done in 1/C ~~pos~~ it is powerful coz of this)

Siglip Attention

__init__

__super__

config, embed_dim

num_heads, head_dim

forward

from x to Q, K, V

$Q = X \times W_q = (4, 1024) \times (1024, 128) = (4, 8, 128)$

$K = X \times W_k$

$V = X \times W_v$

Coding

(4, 1024) (4, 8, 128)

seq embed_dim seq n_head head_dim

S1: from x to Q, K, V.

I have anthr note for oth thing I will just binge watch this section

Each head will just watch a section of embeding like a head watches 1st 128 dim & so on...

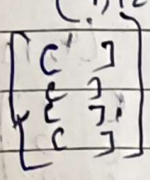
because it is more eff

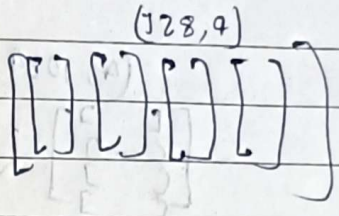
(4, 8, 128) to (8, 4, 128)

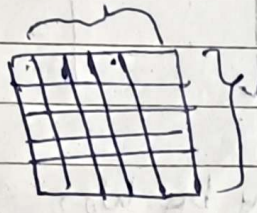
In this way, (4,8,128) to (8,4,128)
we can understand that each section can be computed independently as those 8 sections carries very imp meaning to see context in diff. way

why?
- we want to parallelise comp
- each head should learn to relate tokens (or patches) diff

S-3: Calc. attn for each head in //

Q_{Head_1} = (4,128) 

$K^T_{Head_1}$ = (128,4) 

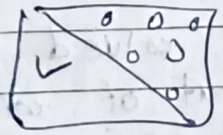
$\frac{Q \times K^T}{\sqrt{d_{head}}}$ = 

This rep relationship btwn tokens bigger value is goo relationship

Attn masking

because we tell we don't want current token to know about future token

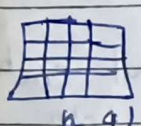
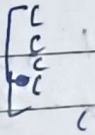
$softmax\left(\frac{Q \times K^T}{\sqrt{d_{head}}}\right)$

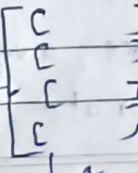
$softmax\left(\frac{QK^T}{\sqrt{d_{head}}} + mask\right)$ = 

Attention weight

Step4. Multiply by V seq

each row rep weighted sum of

 (4,4) ×  (9,128)

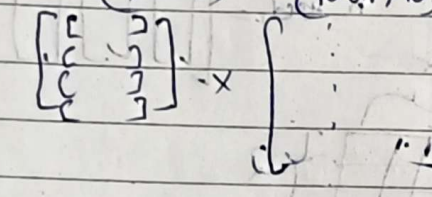
=  (4,128)

Step 5: Transpose back
(8, 4, 128) to (4, 8, 128) ~~to (4, 1024)~~

Step 6: Concatenate all head
Given each head ~~in computing~~ contextualized
(4, 8, 128) to (4, 1024)

Step 7: Multiply by W_o
parameter

$(4, 1024) \times (1024, 1024) = (4, 1024)$



Why multiply by W_o?
- To combine info from attn head
- To restore correct dim.

We understand attention and attention rep. things in multiple dimension (seq. of embeddings) not only 1 for particular input token but in img we must have (n x 1) dim for any particular image which is the main input for contrastive learning.

① This is solved via by only taking 1st output part of contextialize embedding from transformer as a representative for whole img, force model to put all info in 1st contextualised embedding

② Just take avg of all output embedding to generate 1 single embedding

We are done with understanding how to build contrastive vision encoder

part- 4

Processing Paligemma

we need to make tokeniser in this ~~pape~~ Paligemma we are making the use of gemma tokeniser but our approach to understand image also, so, some modifications are maintained to achieve what we want to achieve

Paligemma further can be used for img segmentation or obj detection as well

But we are doing poligemma for conditional modeling and other aspect is out of our scope.

Image normalisation mean & standard deviation same meaning as norm. that we do in neural network, so that we want always have same dist.

Imagenet_standard_mean = [0.5, 0.5, 0.5]
Imagenet_standard_std = [0.5, 0.5, 0.5]

preprocess img:

Step1: take img.
S2: resize
S3: change to np array
S4: rescale
S5 = ~~normalize~~ with std_mean & std dev
S6 = transpose to Cchannel, Height, width

Passed this S6 to make ~~tensor~~ ~~of~~ single tensor further to torch tensor

And add ~~img~~ text aspect as well

Coding Gemma

PaliGemma for Conditional Generation

__init__

config

vision_tower

multi_modal_projector

vocab_size

language_model

pad_token_id

tie_weight — weight tying technique of using param of 1 layer for anthr

forward

input_ids

pixel_values

attn_mask

kv_cache

input_imbeds

selected_image_feature

image_feature → Linear Projector

merge_input_id_with_input_features

outputs = langmodel(

=

=

=

)

GemmaConfig

GemmaConfig

group query attn

when q is & k, v has diff no. of heads

so, we just make place to fit every thing with final_embedding

& then we make,

text_mask
image_mask
pad_mask

& we make it applicable to make it compatible with given dim

text_mask_expanded
pad_mask_expanded
Image_mask_expanded

Now add text embed

final_embedding [add text mask exp] if true with some where fxn.

~~same for~~

~~final~~

Now add image things with ~~.scatter~~ masked_scatter [can't use where]

~~final_embedding~~

Now for padding just insert ~~zeros~~.

K-V cache

Problem k-V cache is solving:

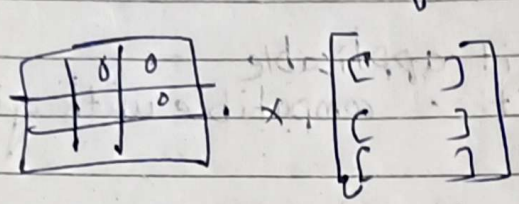
Casual mask : helps in understanding casual mask of present token & all the previous tokens.
if we don't use casual mask it understands everything in a way [vision encoders kind of follows it]

Some explanation of transformer on how it works during inferencing

During inference we are generating embeddings that we aren't using.

If it has 1000s token attn mech. will gen 1000 x 1000 matrix from that only a small portion will be used

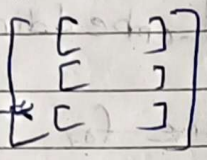
And KV cache exactly helps with that helping us to exactly do that generate embeddings that matters



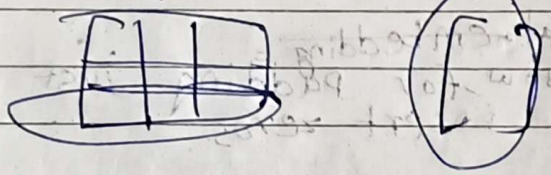
(3,3) (3, 128)

Attn ~~matrix~~ weight

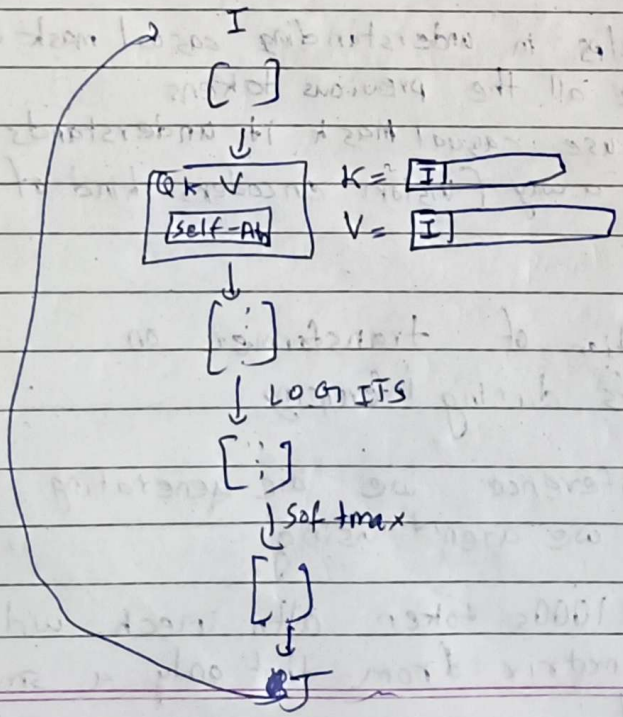
previously we used to multiply all Attn wt with V to create final



only this is imp, but this can be gen by just using last row of Attn weight & value



[. . .]



KV cache basically allow us to during inferencing to during token genin to avoid generating all embeddings of all input seq but only last contextualised embedding

Prefilling : meaning like we can do it for multiple as user can give multiple tokes & gene all that can be // maintained

Only during prefilling phase we allow gen, of multiple output embeddings & discard that we don't need.

because it will make it fast & gpu can be effectively utilised

2 step
i) Prefilling
ii) Token generation

In paligemma we don't mask out anything like later like in vanilla transformer for img token we need to understand each thing with another

why prompt isn't casual masking as well to tokens of the prompt.

coz textual prompt itself is very short

& this prompt actually represents what we want our model to perform we want all the tokens to understand each other

Image, prefix, suffix / Target

We only mask out what we expect model to output.

Itis what paligemma choosed.

The output gen aren't that long but they can be fine-tuned to be so as well.

During traing we will have casual mask for suffix but during inference we don't have any casual mask working with KV cache.

Rotary positional encoding (RoPE)

Absolute posn embedding

- Learned from data
  - Posn vect for 1-512
  - Max length is bounded
- Sinusoidal fxn

Empirically: Both approach have similar performance

Another problem is every single posn embedding is indep of each other
i.e. Pos. 1 & 2 is not diff from 1 & 500 which is intuitively wrong

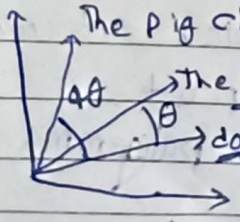
Relative posn embedding

It kinda solves the distance problem in words
like 1 & 3 will have same effectiveness as 501 & 503

This approach is slow
- Extra step in self-attn layer
- Difficult for KV cache

Rope:

Instead of adding posn in vector to encode posn of word in a sentence, they propose to apply rotation to vector.



It takes best out of both worlds

Gemmafor CausalLM

Transformer model + Lang. Modeling Head

~~GemmaModel~~

init

config

tie_weights

forward

(attn_mask

pos_ids

inputs_embeds

kv_cache)

output = attn_mask, pos_ids, inputs_embeds, kv_cache

hiden = outputs

logits = ~~set~~ lm_heads (hiden)

logits = [logits.float)

GemmModel

init

config

~~padding_idx~~

vocab_size

embed_tokens

layers = Module list [Gemma Decodet layer]

norm = Gemma RMSNorm

Rotary posn is find out during calc of attn not at earlier like in other ways

get_input_embeds

forward (attn mask, pos_id, input_embeds, kv_cache)

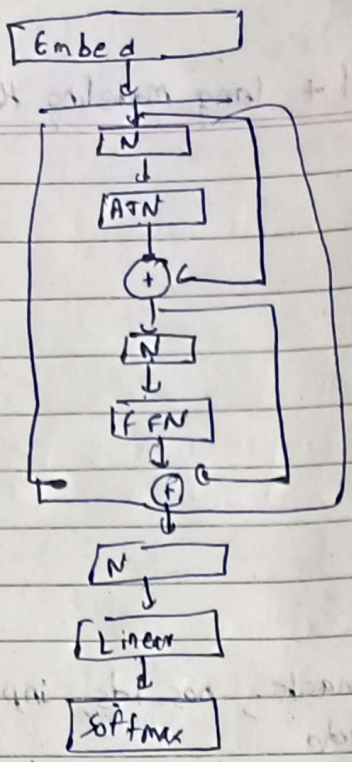
hid_states = inputembeds

normaliser

hiddenstates = hid_states * normaliser

for loop

all transformer block



N = RMSnorm

RMS norm

Claim: success of layernorm is not bcz of re-centring invariance but coz of re-scaling invariance

what they mean is the mean need not be around 1. It could be around any no. given that it is normally dist. around that number

we only need to compute RMs instead of σ, & μ like in LN

$$\bar{a}_i = \frac{a_i}{RMS(a)} g_i, \quad RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{} a_i^2}$$

faster than LN

class GemmaRMSNorm

init

eps

weight

norm

forward

eps is req if no is close to 0 division by 0 eps is added

class GemmaDecoderLayer

init ~~self~~

GemmaAttn

GemmaMLP

~~LN~~ RMS1

RMS2

forward

like in fig

class GemmaMLP ~~(gelu)~~

init

config

hidden_size

intermediate_size

gate_proj

up_proj

down_proj

forward

gelu actvn fxn

Gelu

motvn

- combines both relu & dropout

- deterministic non-linearity

weighted sum - x

$\phi(x) = GELU(x) = x * m$

m = cumulative Dist. fxn of Normal dist

-

we have gate_proj now here which is basically upping & downing no. of neurons

class GemmaAttention(config, layer_idx)

Gemma is a decoder only layer.

init

self

key value heads

key value groups

rope_theta

is_causal

q_proj (hiddensize, num_heads * head_dim, attention_bias)

k_proj —

v_proj —

o_proj —

It is diff from normal attn.

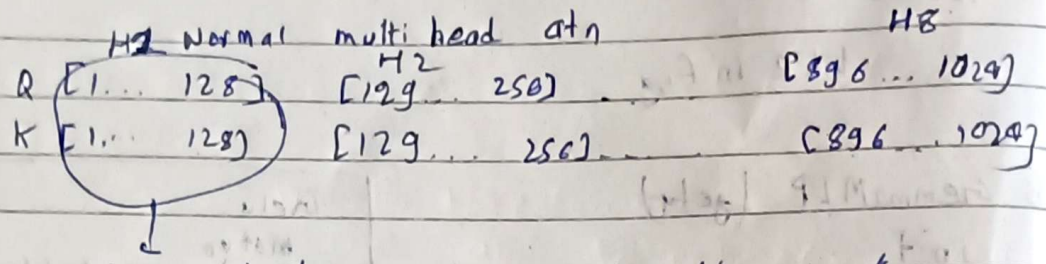
W_q size [1024, 1024] # W_o size = [1024, 1024]

W_k size = [1024, 128]

W_v size [1024, 128]

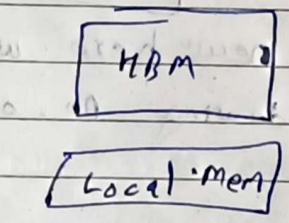
We have less head in key & values then query which is Grouped query attn concept

Grouped query attn



Problem with MHA:

claims: problem isn't with no. of computations that we are doing But the no. of data transfer happening in GPU [Not how many dot products but how many components How GPU transfers form HBM to local Mem works]



matrix opn can be done in //

mem copying is very slow.

Similar Concept in Flash attn we redo computation instead of saving coz saving means copying which costs time

So, Our approach is to reduce data transfer
1) Use less head for keys

- A group of ~~keys~~ query will use 1 key

In a way it reduces some quality but not much which we can negotiate based on our usecase

Also reduces size of kv cache.
kv cache is also have been bottleneck and it also helps in that

```
rotary_embeds = Embeddings(
        head_dim,
        max_pose_embs,
        base = rope_theta
        )
forward
        ~~xen~~, as usual + rotary embeds
        +
        kv-cache thing

class kvcache():
    init
        key_cach
        value_cache
        num_items
        if  == 0
            return 0
        else
            return key_cache[0].shape[-2]
    update
        if never done
        else(done)
            update
```

repeat_kv
(we are basically repeating things with same kernel instead of using
)

repeat_kv [basically repeats for the keys & values ~~of the query~~ that are missing for heads of the query]
we are doing this coz we don't have
custom cuda kernel

attn weight
softmax
dropout
attn output
return

Now positional encoding used in decoder layer,

in vanilla transformer
we use absolute ~~embe~~ posn embedding
i.e. sinusoidal thing

But now adays we use Rope [Relative posn embedding]
We don't ~~di~~ add directly into ~~token~~ embedding of each token
but they modify attn mech itself
so that attn mechanism takes into consideration the posn of token to relate them differently based on the posn

paper explores ~~qsn~~
can we find encoding of embedding vectors such that when we do dot product that encodes posn/ embedding
or, output of q & k's dot product only depends on embedding of 1st token, 2nd token & relative dist btwn them

Eg
Huggingface implementation of Rope is explained which is a bit diff. from the approach we have in paper but the result is same