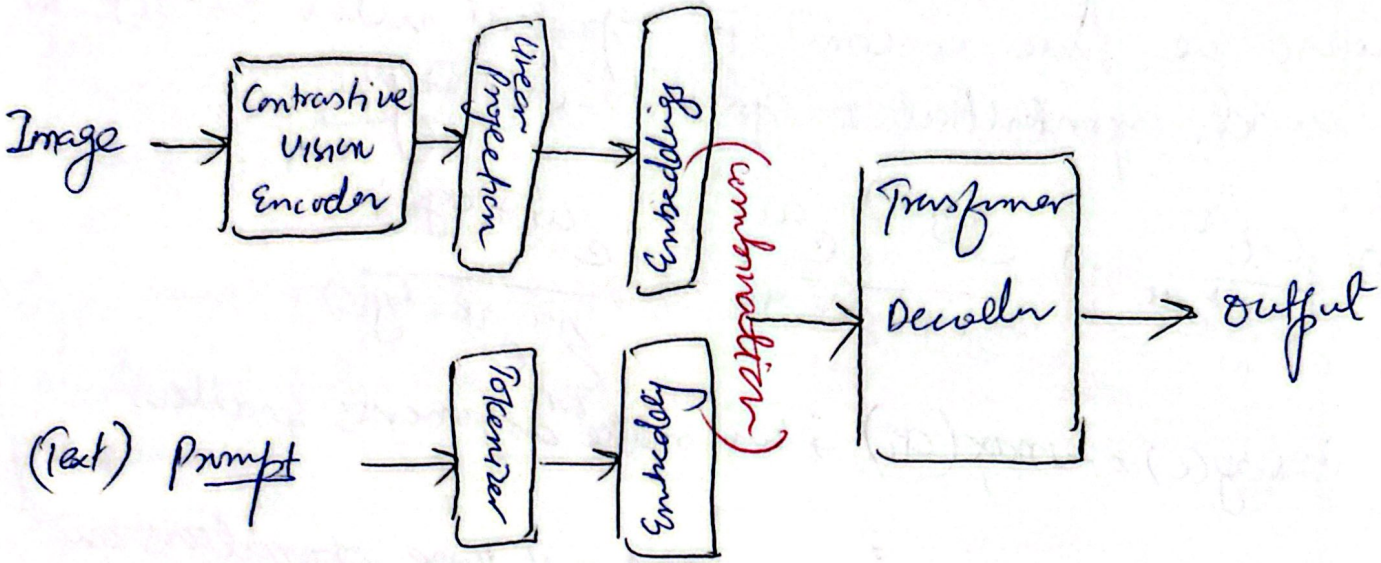
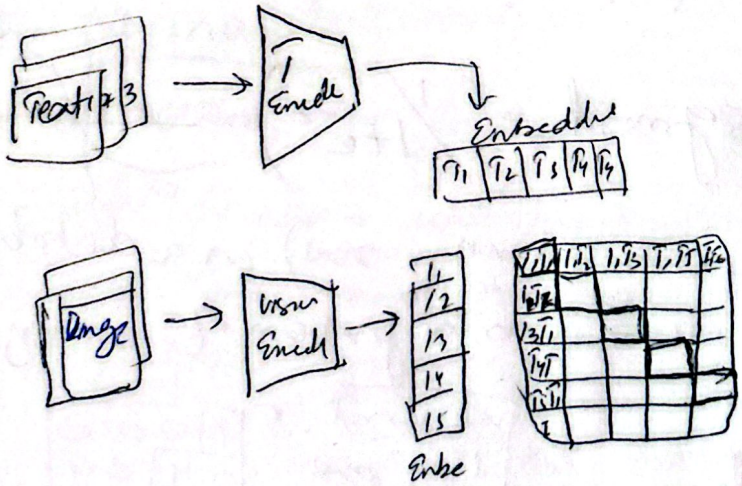
Multimodal. Vision Language Model (Paligemma)

→ Vision VLM → can extract info from image



→ Contrastive learning (CLIP → Sig2ip)

1) Contrastive Re-training.



we multiply each T embedding value with each I embedding to get a Matrix. and we want to train these encoder to produce a high dot product value for image and their corresponding text & low vice versa.

we find a loss function to make corresponding pair high by CrossEntropy loss. (we compute loss for both and then average it

Problem with CLIP:

softmax = $S_i = \frac{e^{a_i}}{\sum_{i=1}^{n} e^{a_k}}$

Numerically Stable Softmax-

The exponentials in softmax cause issue because they grow very faster and if big number comes its a issue because we have a limit to represent until 32-bit. So we avoid exponential to grow to infinity.

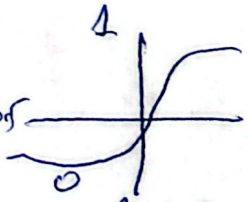
$$\Rightarrow \frac{C \cdot e^{a_i}}{C \cdot \sum_{k=1}^{N} e^{a_k}} \Rightarrow \frac{e^{\log(C)} \cdot e^{a_i}}{e^{\log C} \sum_{k=1}^{N} e^{a_k}} \Rightarrow \frac{e^{a_i + \log(C)}}{\sum_{k=1}^{N} e^{a_k + \log(C)}}$$

$\therefore \log(C) = -\max_i(a_i)$ → this make arguments smaller.

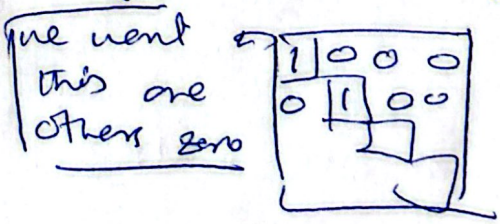
→ So, in dl we have to calculate all these operations on each row and column, which make it computationally heavy due to alot of operations.

So, In SIGLIP & → it help parallelize & high batch size.

Replace softmax with sigmoid $= \left(\frac{1}{1+e^{-z}}\right)$ or



So, instead of treating (each column or row) as a distribution we independently treat them as a problem of binary classification (1 vs all)

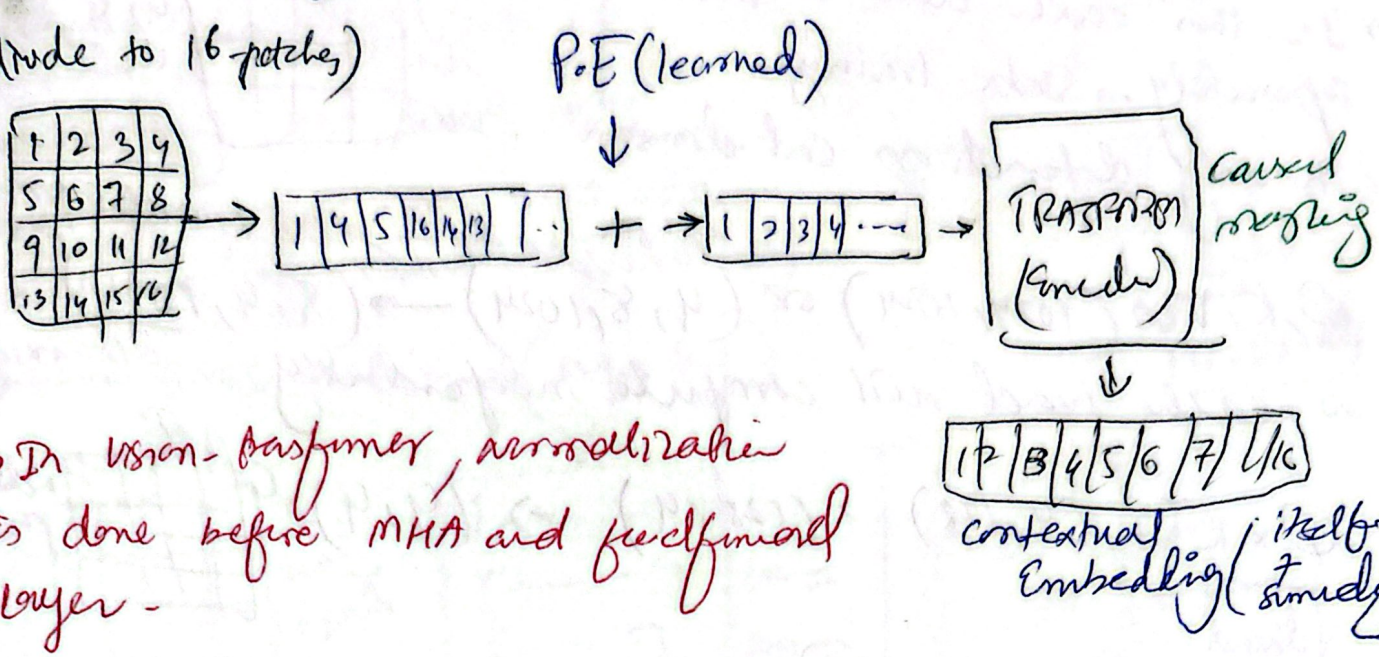


why contrastive encoder.

we want the embeddings that representation of image and have properties that can be better used with text embedding

Hence, contrastive + captions information of image.

Vision-Transformer (seq-seq Model)



* In Vision-Transformer, normalization is done before MHA and feedforward layer.

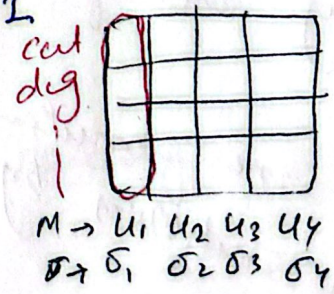
Normalization.

→ rapid changes between layer cause "covariance-shift" which cause the model to learn slowly.

Batch-Normalization.

we normalize the feature in input and output layers each feature independently by making it have the mean zero and variance of 1.

$$\hat{x}^{(k)} = \frac{x^k - E[x^k]}{\sqrt{Var(x^k)}}$$



→ The issue is that each statistics depends on what other items are in batch. For good results we need big-batch size.

(β, γ) are learnable parameters

layer - Normalisation. (Item-dimension)

→ In this each item is treated seperately → Stable training

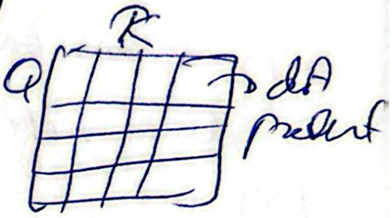
→ Its only depends on cat-dimension

Q, K, V ⇒ (1024, 1024) or (4, 8, 1024) $\xrightarrow{T}$ (8, 4, 1024) → for parallel processing

(seq length: 4, heads: 8, dimensions: 1024)

so, each head will compute independently.

$\frac{Q \times K^T}{\sqrt{d_{head}}} = (4,128) \times (128 \times 4) \rightarrow (4,4)$



$\times V = (4 \times 128) \rightarrow (4 \times 128)$ → weighted sum.

→ concatenate (heads) W^O. ⇒ Concat (128, 128 ...) 4 = 1024

→ W_O = (1024 × 1024), we do to mix result of different heads.

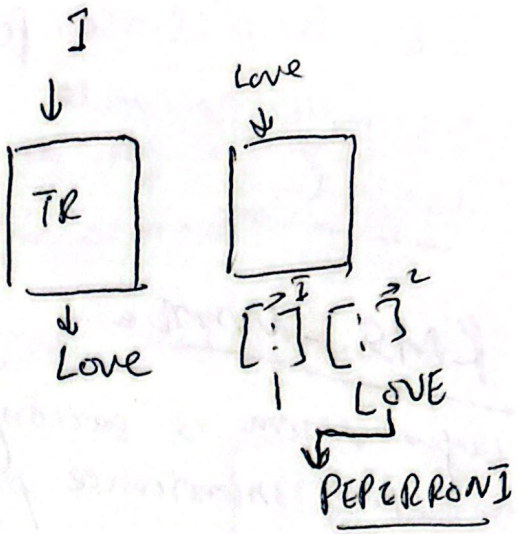
~~Linear~~

In layer norm, they treat each independent feature from batch, making it mean = 0, σ = 1, so it is coming out from distribution (gaussian)

[Same formula but we calculate statistics by row instead columns]

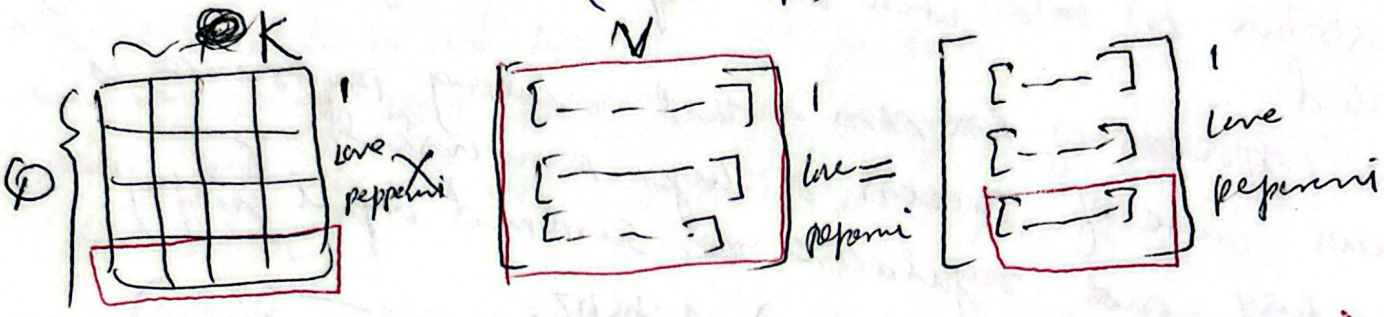
* Gamma and Beta are learnable parameters

KV-cache.



problem - generating alot of embedding, but using only one to project the next token (that have info about all previous ones)

attention scores matrix = (seq x seq)



→ peperoni is result of last row x with all V (columns)

So, we do.

instead of passing all previous tokens, we just passed last single token (Query) and we append it to previously created KV-cache buffer which we used for calculation of self-attention, and we'll get output embedding (1-token) contextualized

Prefill:

→ adding many embedding of token in KV-cache in single forward pass

Causal mask =

→ we can't mask image token, and are also not masking prompt mais if short & also because it define the task to do.

but, its a choise Paligemma, as we are not generating we only generate Target/suffix which we made causal.

RMS – Norm:

layer Norm is successful, because its recentering and rescaling invariance property. due to which model is re-sensitive to noise when both input and weights are randomly scaled.

The hypothesis of RMS_Norm is that rescaling invariance is main cause of success of layer Norm instead of recentering, and regularize the summed input simply according to (Root-mean-square) statistic.

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

g_i → learnable parameter

– when means of RMS sum = 0 {RMS = layer Norm}, although RMS doesn't re-center.

→ The goal to values around their mean, which can be any value {(not 0) as in layer Norm}.

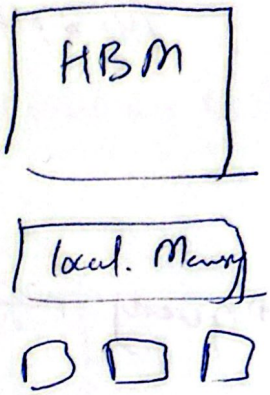
Scanned with CamScanner

Multi-Query Attention.

The difference is we have less heads in key and value which results in smaller projection for embedding of each token.

[1 ---- 1024] / 8 Heads → [1 -- 128] [129 -- 256] ---- [---1024]

Multi query paper.



→ The bottleneck is not how many dot products are we doing but the memory transfer from high-bandwidth to local memory (time taken)

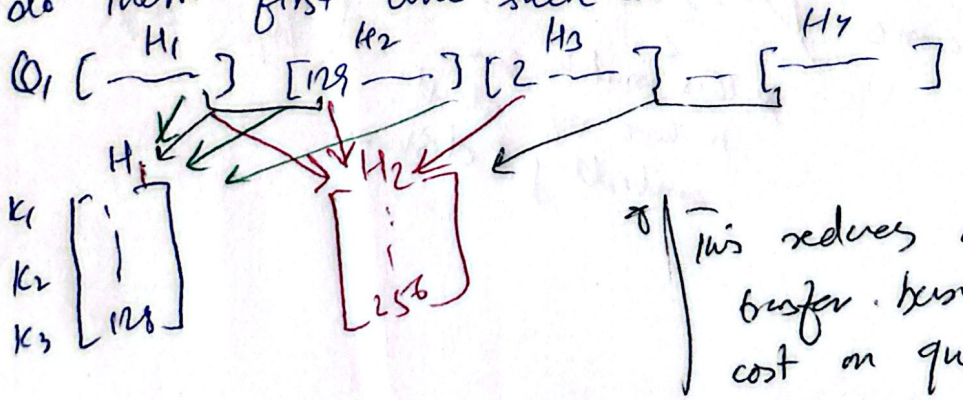
→ So, compute is less important than data transfer

Flash attention exploit this difference b/w data transfer and compute. It was faster to redue computation than copying different stuff from GPU. This is also why we do gradient checkpointing. During backward pass we do redo computations instead of copying them from HBM → LM.

To Reduce data transfer while we do multi head attention

→ one way is to use less heads in keys.

→ So, instead of copying each head of key for queries, we do them first time such as

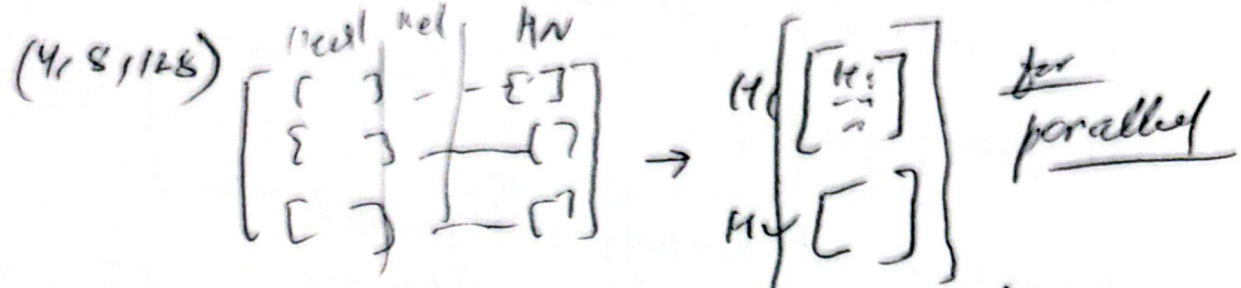


↑ This reduces data transfer. based on some cost on quality of model

- Multi-query attention = KV (heads = 1)

- Grouped Multi-query attention = KV (head = 2 or more)

→ also reduces the size of KV-cache



→ query needs to attend all past key and values

→ (expand = repeat)

Rotary Positional Encodings

→ They are not directly added to input, but they modifies attention mechanism in a way that it take in consideration the position of token.

$$f_q(x_m, m) f_k(x_n, n) = g(x_m, x_n, m-n)$$

→ The dot product depends on embedding + relative distance b/w them

2D-Case

$$\text{Rotation} = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix}, \quad f_{(q,k)}(x_m, m) = R \begin{pmatrix} W_{qk}^{1,1} & W_{qk}^{1,2} \\ W_{qk}^{2,1} & W_{qk}^{2,2} \end{pmatrix} \begin{pmatrix} x_m^1 \\ x_m^2 \end{pmatrix}$$

→ This make sure to have info about embedding & distance

General - form

$$R^d_{\Theta,m} = \begin{pmatrix} \cos m\theta & -\sin m\theta & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta & \cos m\theta & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta & -\sin m\theta & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta & \cos m\theta & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

→ It sparse, mean most of elements is 0, and we will be doing alot of computation.

So, better way: as we already know where are zeros (0)

Computationally efficient realization of Rotary Matrix Multiplication

using element-wise multiplication.

$$R^d_{\Theta,m}\, q = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \vdots \\ \cos m\theta_{d/2} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_d \\ x_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \vdots \\ \sin m\theta_{d/2} \end{pmatrix}$$

positions to encode

Embedding dimension

Rotated dimension Embedding

$\Theta_i = 10000^{-2i/d}$

$i = [1, 2, \ldots, d/2]$

decaying effect.

Similar = high

less = decay