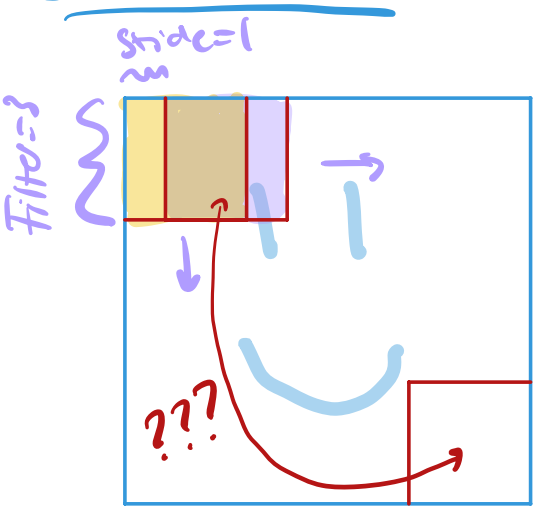
Convolutions



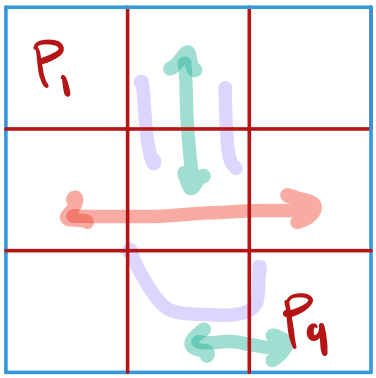
Sliding filter approach

→ Local computation focused on a small piece of image.

→ No learned relation btwn different located filters

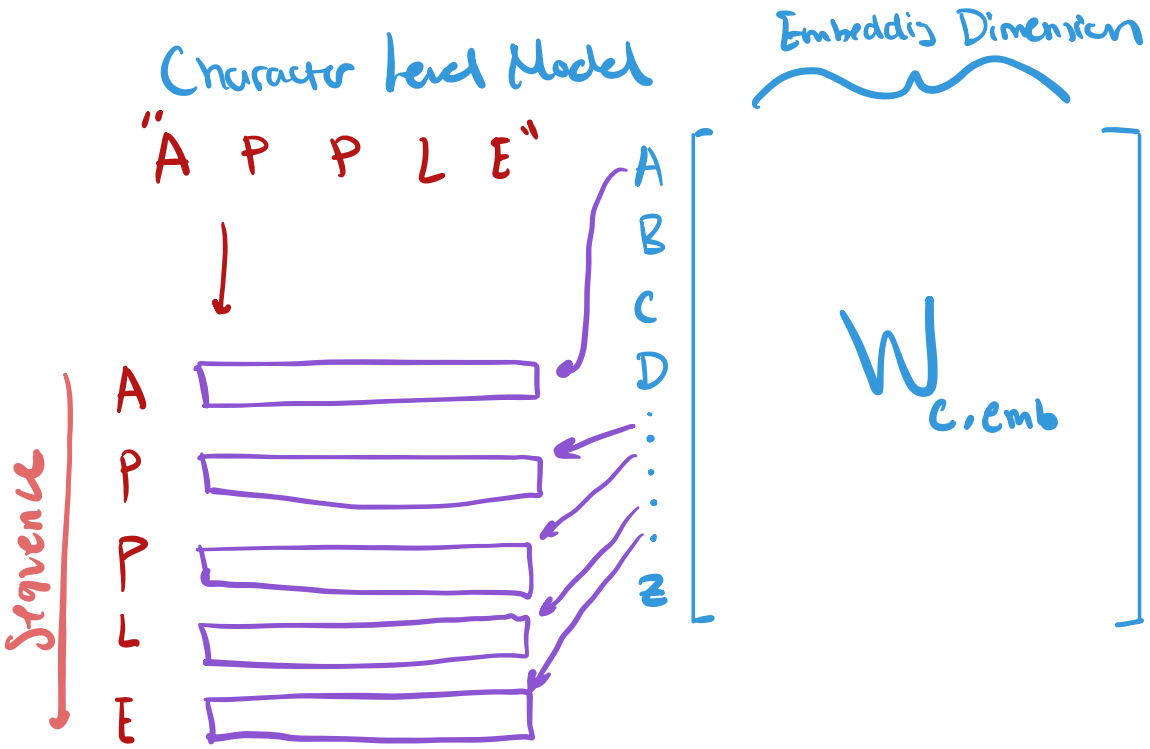
ViT

→ capture global image relations

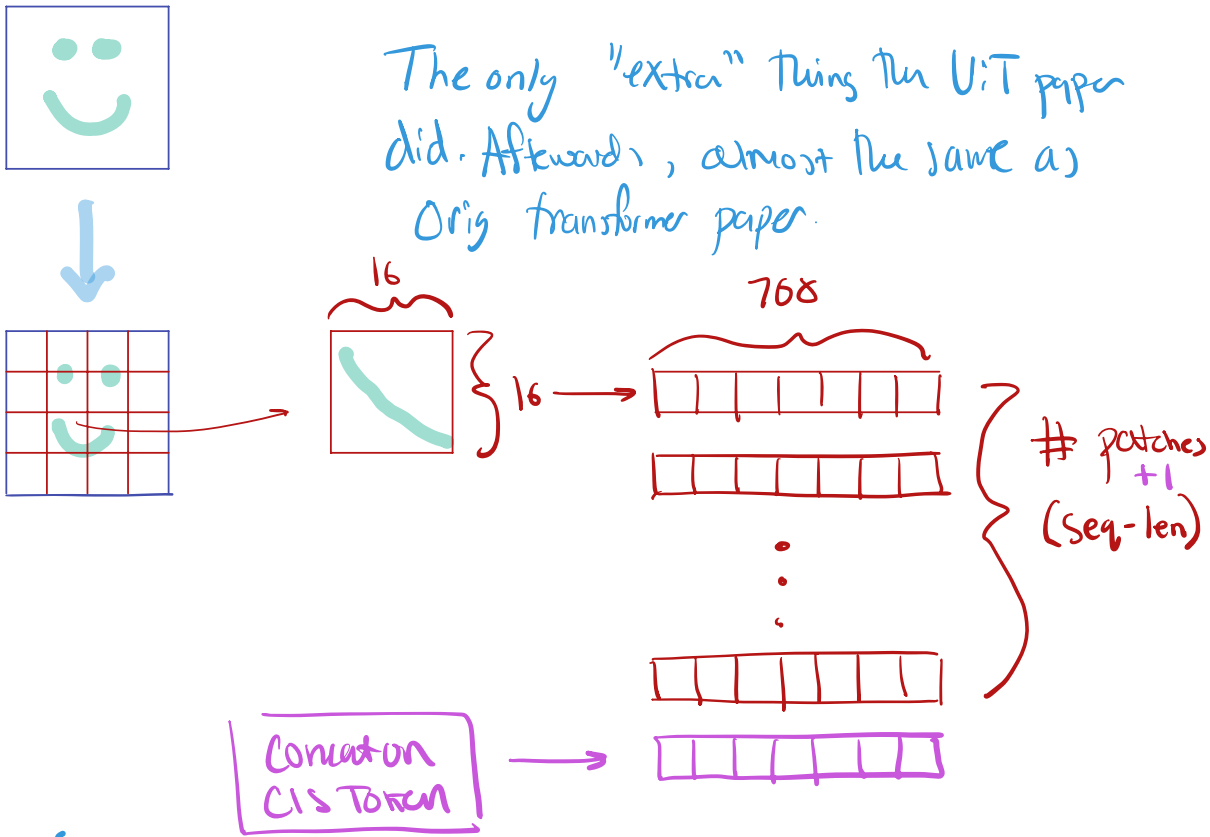


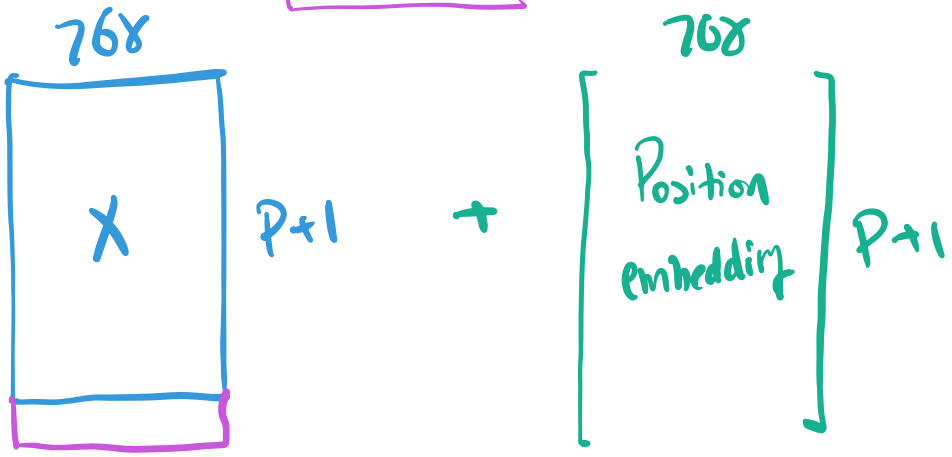
Q. How is patch 1 related to patch 9?

The typical NLP Task (Tokenizing/ Embeddi



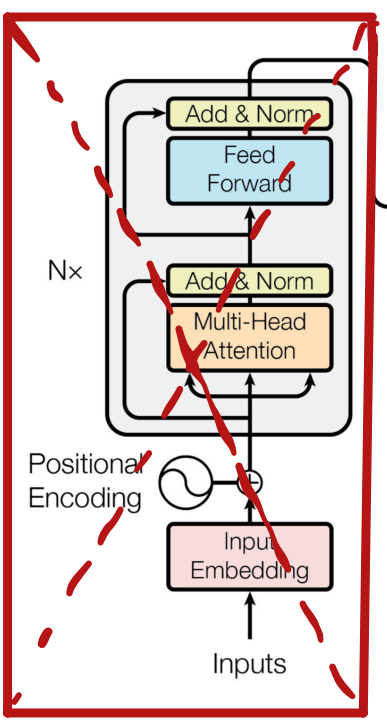
Vision Transformer → How do we convert a an image to an embedded sequence.





↖ inject positional info, otherwise permutation invariant

What exactly is a transformer?



Language translation

We only need this half!

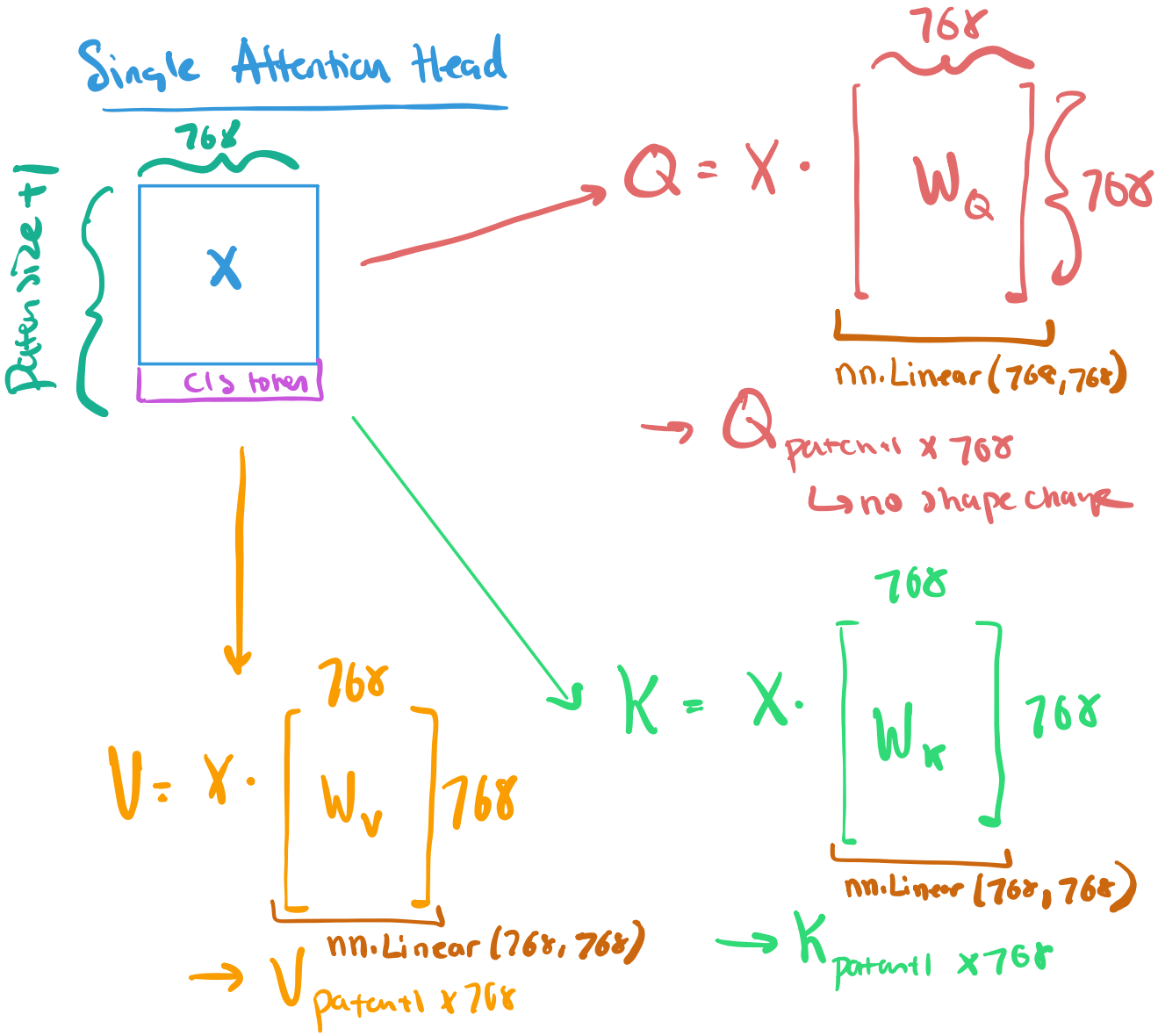
No masking this is not translation!

Super important!

encoder For input

Encoder For output

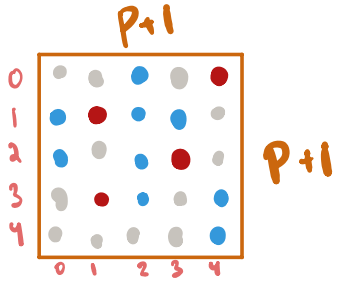
Single Attention Head



Attention Matrix (Fancy weighted average)

$Q_{p+1,768} \cdot K^{T}_{768,p+1} \rightarrow$ Attention mat.

$A_{p+1,\,p+1}$



$\frac{A_{p+1}}{\sqrt{768}}$ ← divide by sqrt head size
→ scaled attention.

Variance of A will scale w/ head size so we want to normalize

if $Var(A) = 768 \cdot Var(A^*)$

→ $Var\left(\frac{A}{\sqrt{768}}\right) = \left(\frac{1}{\sqrt{768}}\right)^2 \cdot 768 \cdot Var(A^*)$

$= Var(A^*)$

$\approx Var(Q), Var(K)$

$Softmax(A^*_{p+1,p+1})$ → scale to probability vector btwn $[0,1]$

$A^{*s}_{p+1,p+1} \cdot V_{p+1 \times 768}$ → $Out_{p+1,768}$

same shape!

$$A(Q,K,V) = Softmax\left(\frac{Q \cdot K^T}{\sqrt{d_K}}\right)V$$

$$\underset{p+1}{\left[\; A^{*s}_{p+1,p+1} \;\right]} \cdot \underset{768}{\left[\; V_{p+1,768} \;\right]}_{p+1}$$

Values are a projection of the original data X that encode each patch w/ a embedding vector. We want to do a weighted average of those embeddings.

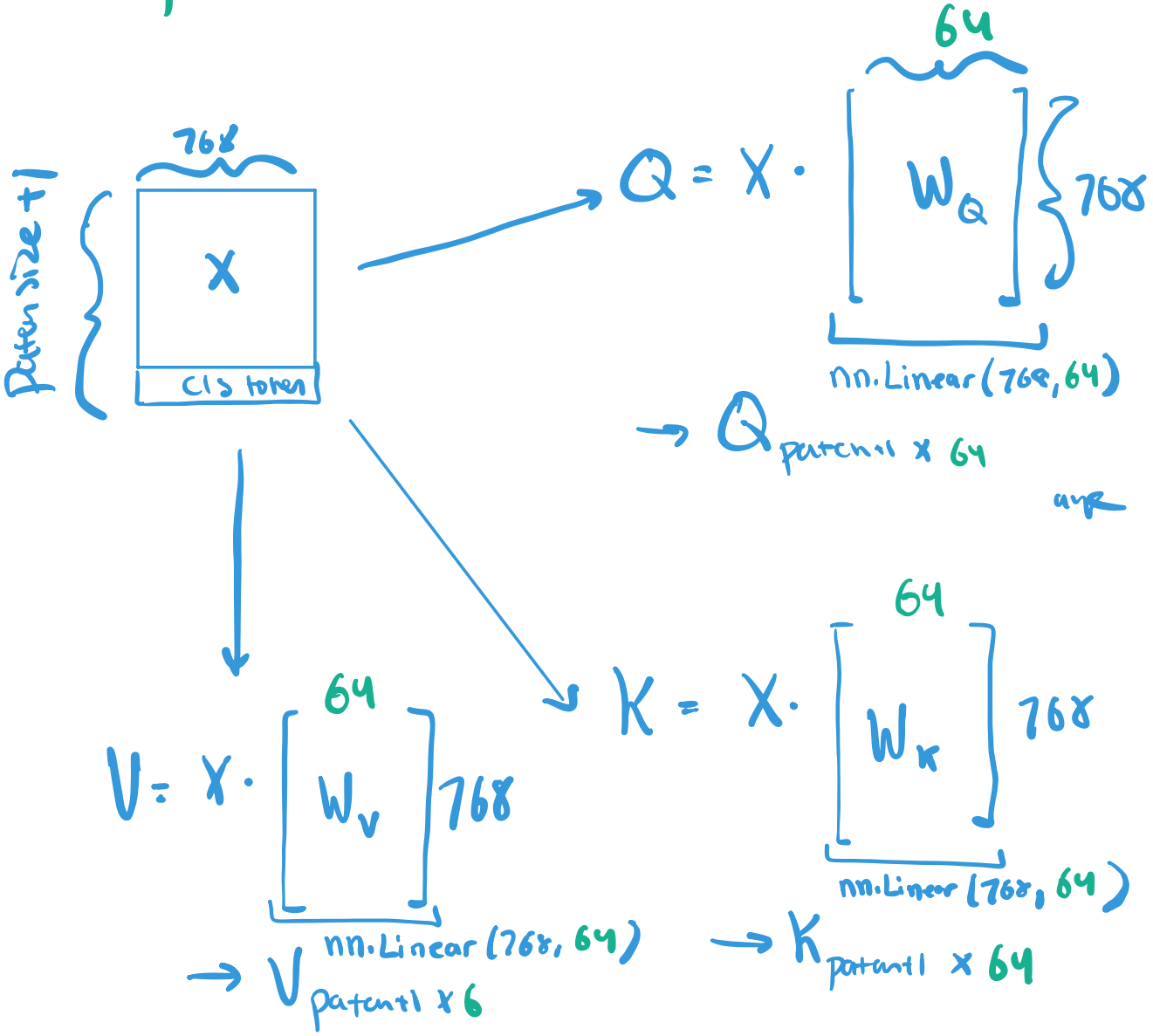
generated by W_V proj

$$\underset{p+1}{\left[\begin{matrix} 0.2 & 0.3 & 0.5 \\ \vdots & \vdots & \vdots \end{matrix}\right]}_{p+1\,(3)} \left[\begin{matrix} -\,e_1\,- \\ -\,e_2\,- \\ -\,e_3\,- \end{matrix}\right]_{p+1\,(3)}^{768}$$

$$\rightarrow \left[\begin{matrix} 0.2\,e_1 + 0.3\,e_2 + 0.5\,e_3 \\ \vdots \end{matrix}\right] \leftarrow \text{weighted average.}$$

Multiheaded Attention

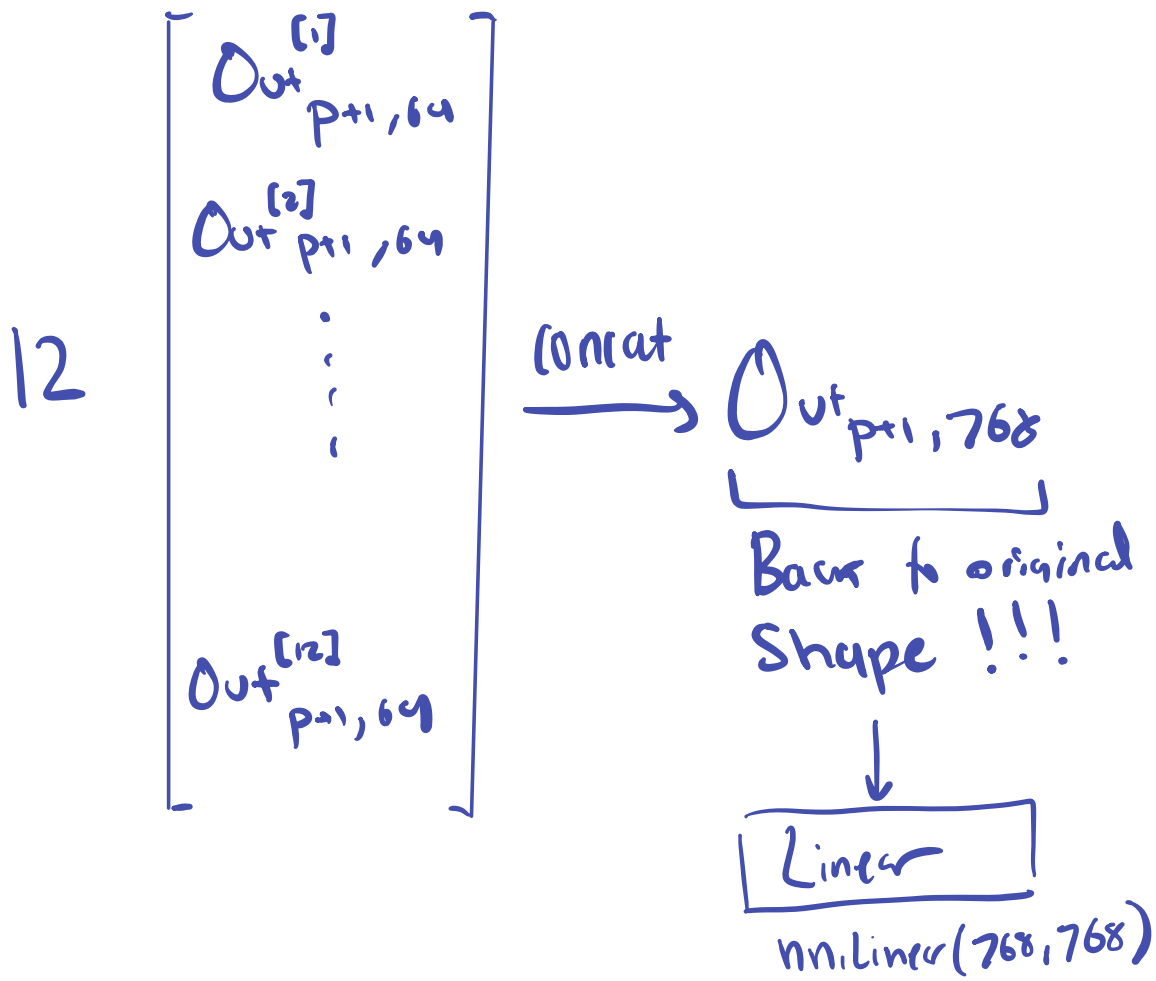
if we start with embedding = 768 and we want 12 heads → 768/12 = 64 dim per head.



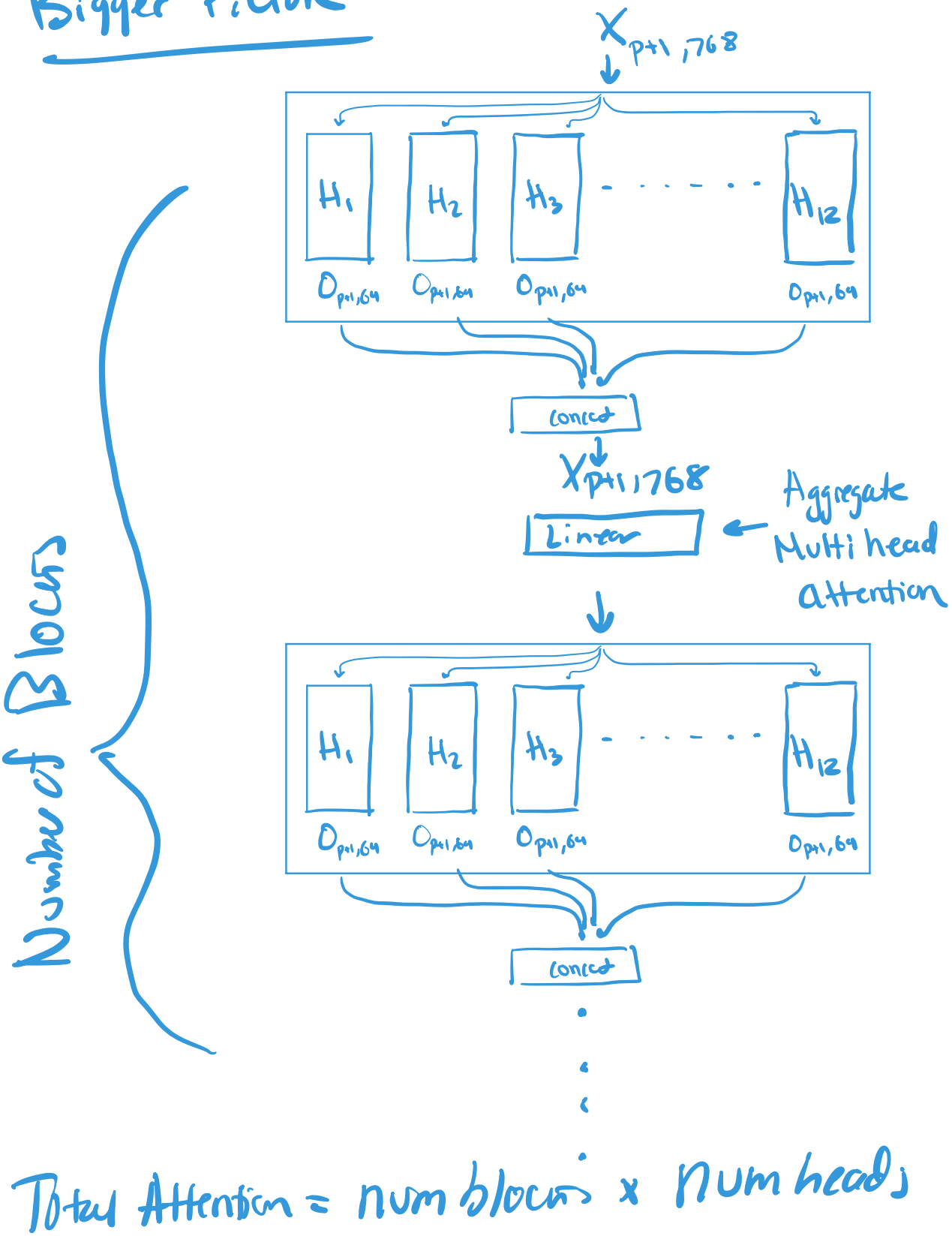
All previous calc. For single head identical so a single head returns:

$$Out_{patch+1,\,64}$$

But we have 12 heads so output will really be:

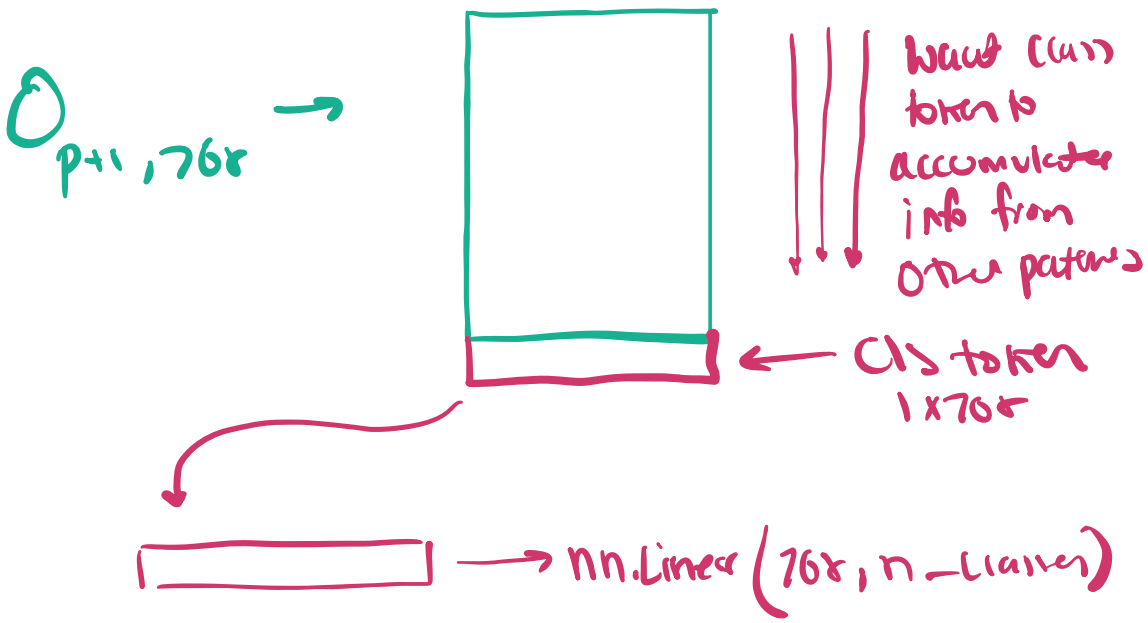


Bigger Picture

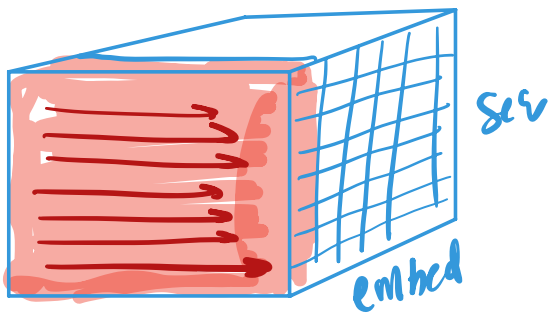


Total Attention = num blocks x num heads

Classifier

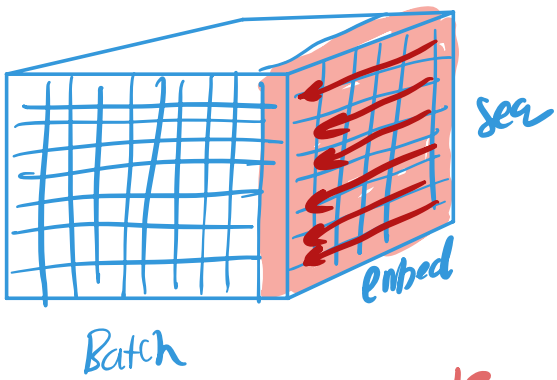

$O_{p+1,768}$
Want CLS token to accumulate info from other patches
CLS token
1x768
nn.Linear(768, n_classes)

Batch Norm



Calc μ and σ^2
For every pixel
across batch

Layernorm

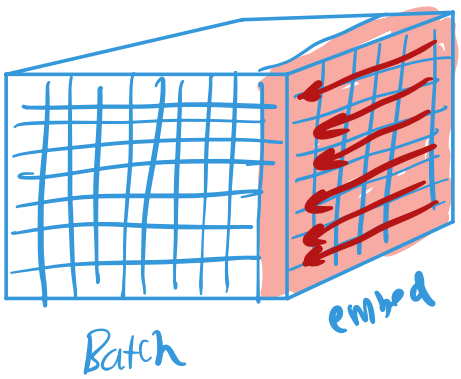


Normalize each sample individually

→ Don't need to store all of σ anymore!

Problems w/ batchnorm

- hard to use w/ diff'ed sequence lengths
- fails w/ small batch sizes
- hard to parallelize



Input to LN is
(batch, seq, embed)

Goal: Calculate μ, σ for each embedding vector for each sample in the batch

nn.LayerNorm(embed_dim)

each embedding vector is of length embed dim, so we have to let layer norm what to expect

From docs: "If single integer is used, this module will normalize over the last dimension expected to be of that specific size"