Vision transformer

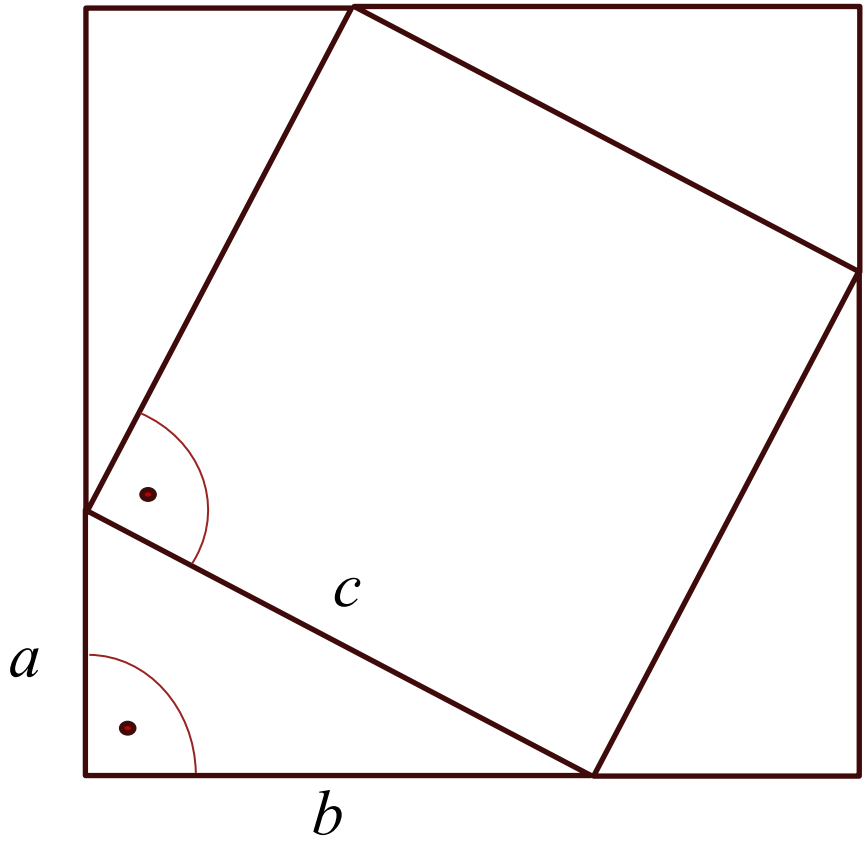
Andrej Lúčny

Cosine similarity. Embedding and wipe-out. Attention.
Vision transformer (ViT).

`https://github.com/andylucny/OAI`

Pythagoras' theorem

In a right triangle:



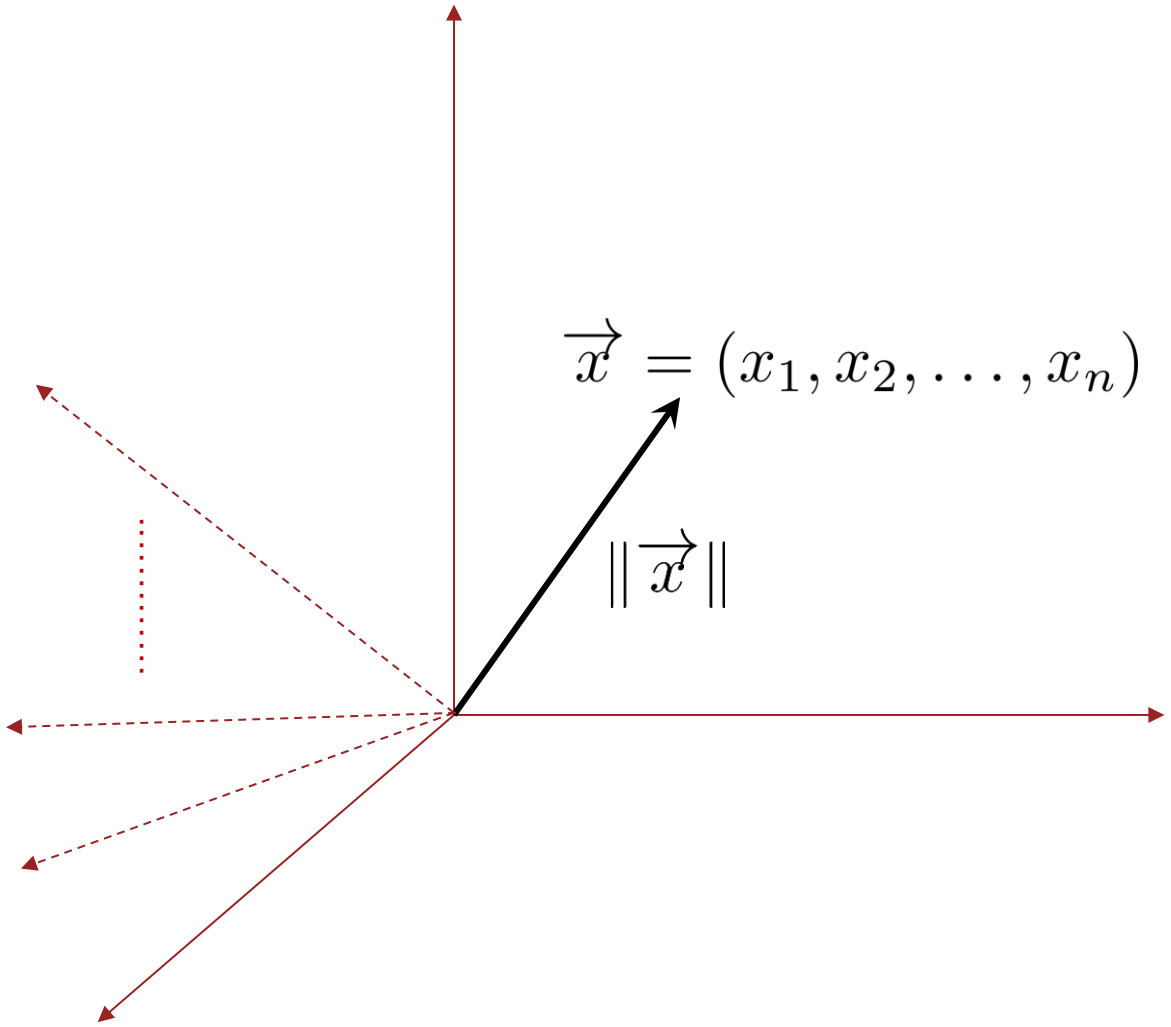
$$(a+b)^2 = c^2 + 4\frac{ab}{2}$$

$$\Longrightarrow$$

$$a^2 + 2ab + b^2 = c^2 + 2ab$$

$$\Longrightarrow$$

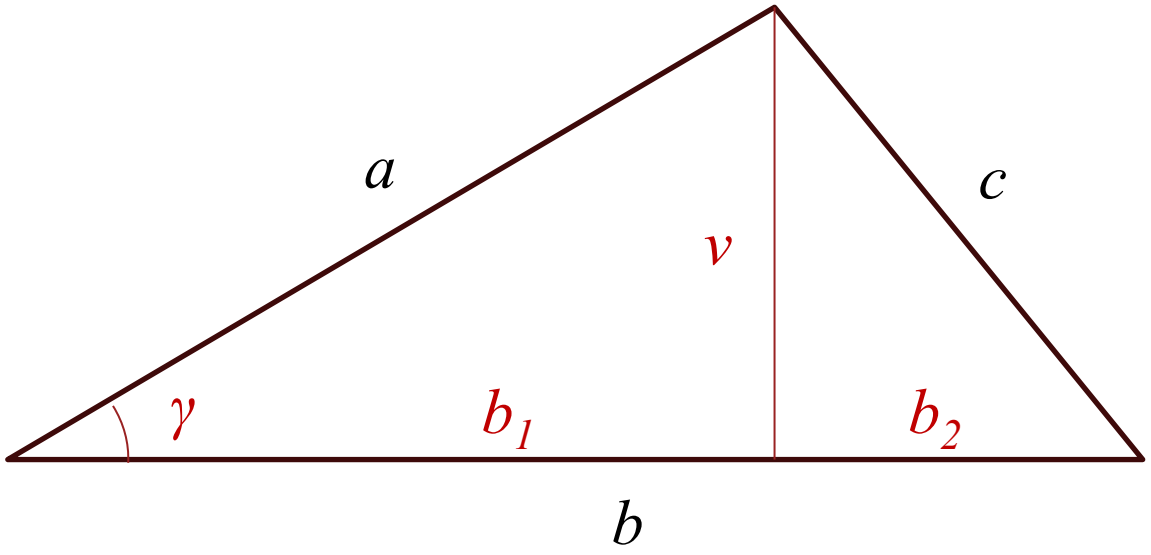
$$\boxed{a^2 + b^2 = c^2}$$



length of a vector in n-dimensional space :

$$\|\vec{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

Law of Cosines



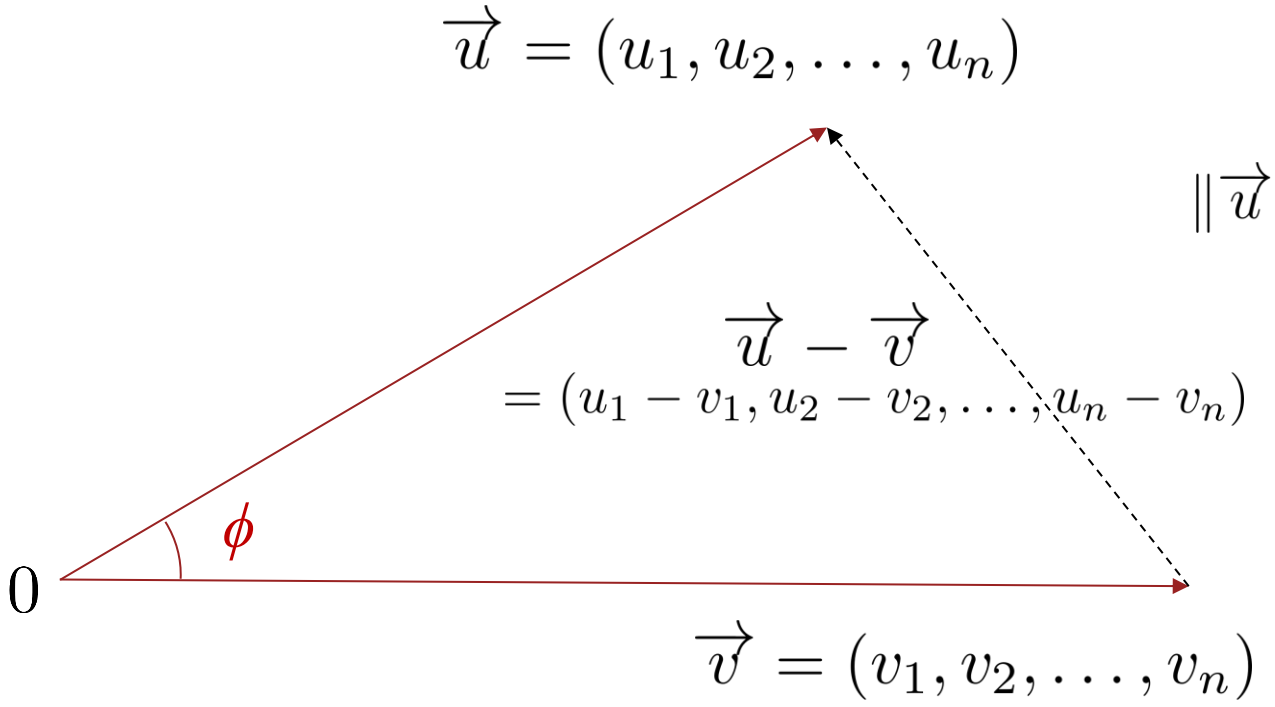
$$v^2 = a^2 - b_1^2$$
$$v^2 = c^2 - b_2^2 = c^2 - (b - b_1)^2 =$$
$$= c^2 - b^2 + 2bb_1 - b_1^2$$

$$a^2 - b_1^2 = c^2 - b^2 + 2bb_1 - b_1^2$$
$$a^2 = c^2 - b^2 + 2bb_1$$
$$c^2 = a^2 + b^2 - 2bb_1$$
$$c^2 = a^2 + b^2 - 2ab\frac{b_1}{a}$$

$\cos\gamma$

$$\boxed{c^2 = a^2 + b^2 - 2ab\cos\gamma}$$

Law of Cosines for Vectors



$$\|\vec{u} - \vec{v}\|^2 = \|\vec{u}\|^2 + \|\vec{v}\|^2 - 2\|\vec{u}\|\|\vec{v}\| cos\phi$$

$$cos\phi = \frac{\|\vec{u}\|^2 + \|\vec{v}\|^2 - \|\vec{u} - \vec{v}\|^2}{2\|\vec{u}\|\|\vec{v}\|}$$

$$cos\phi =$$

$$= \frac{\| \vec{u} \|^2 + \| \vec{v} \|^2 - \| \vec{u} - \vec{v} \|^2}{2 \| \vec{u} \| \| \vec{v} \|} =$$

$$= \frac{u_1^2 + u_2^2 + \cdots + u_n^2 + v_1^2 + v_2^2 + \cdots + v_n^2 - (u_1 - v_1)^2 - (u_2 - v_2)^2 - \cdots - (u_n - v_n)^2}{2 \| \vec{u} \| \| \vec{v} \|} =$$

$$= \frac{u_1^2 + u_2^2 + \cdots + u_n^2 + v_1^2 + v_2^2 + \cdots + v_n^2 - u_1^2 + 2u_1 v_1 - v_1^2 - u_2^2 + 2u_2 v_2 - v_2^2 - \cdots - u_n^2 + 2u_n v_n - v_n^2}{2 \| \vec{u} \| \| \vec{v} \|} =$$

$$= \frac{2u_1 v_1 + 2u_2 v_2 + \cdots + 2u_n v_n}{2 \| \vec{u} \| \| \vec{v} \|} =$$

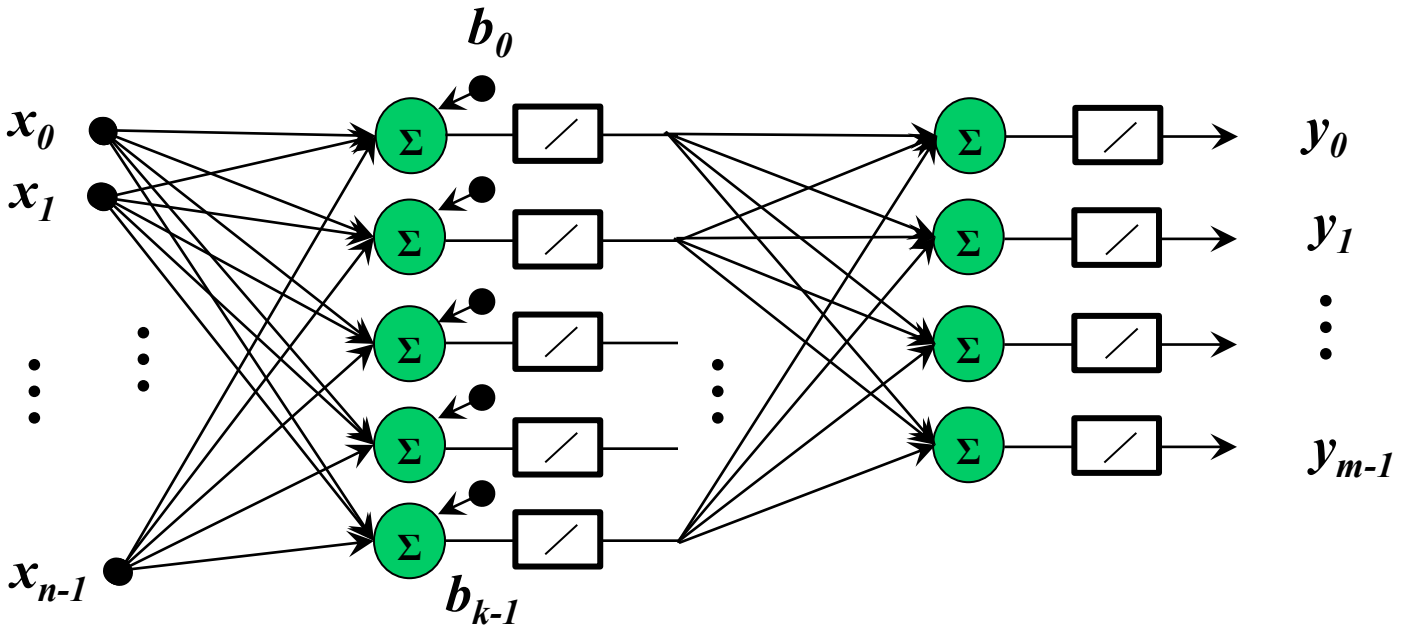
$$= \frac{u_1 v_1 + u_2 v_2 + \cdots + u_n v_n}{\| \vec{u} \| \| \vec{v} \|}$$

$\vec{u} \cdot \vec{v}$

Dot product

$$cos\phi = \frac{\vec{u} \cdot \vec{v}}{\| \vec{u} \| \| \vec{v} \|}$$

Perceptron

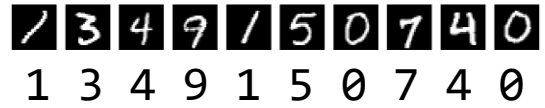


Let's take the simplest perceptron:

- Linear activation only
- No bias in the output layer

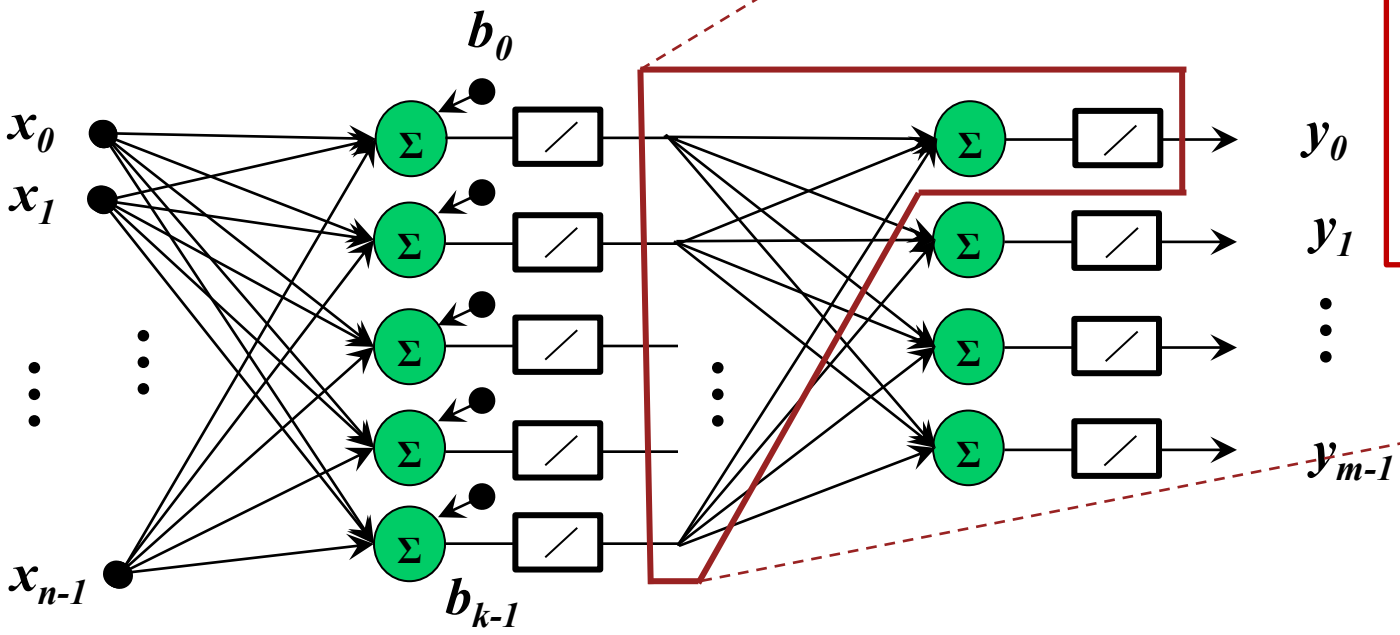
- such a model is not a universal approximator, but we still can train it

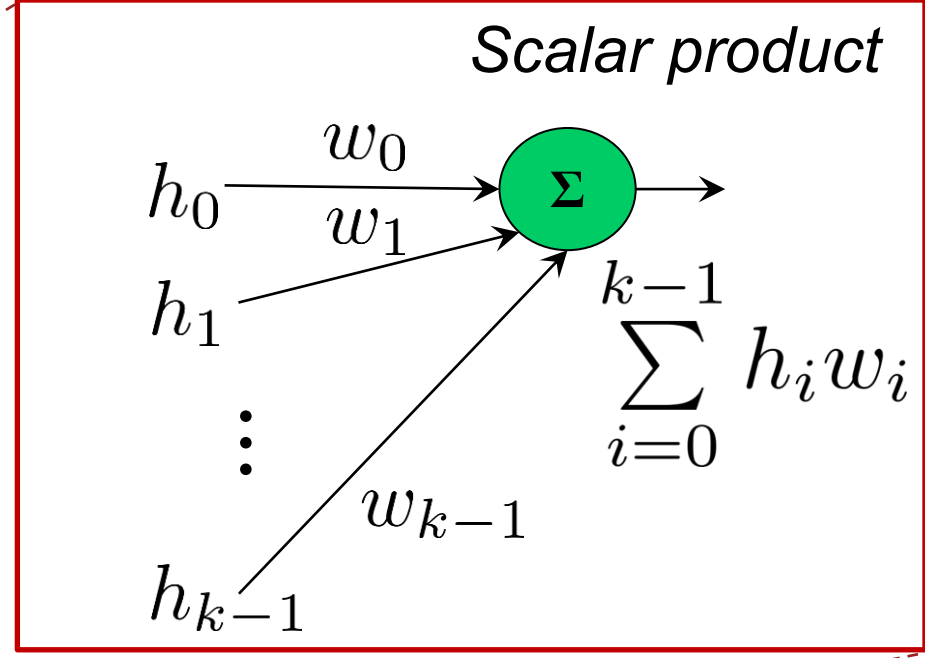
- A task example: MNIST digits classification



- input: n=784 intensities of 28 × 28 pixels
- output: m=10 logits
- k=1568 hidden neurons
- Loss function: BCE
- maximal logit corresponds to the most probable category

Nature of logits

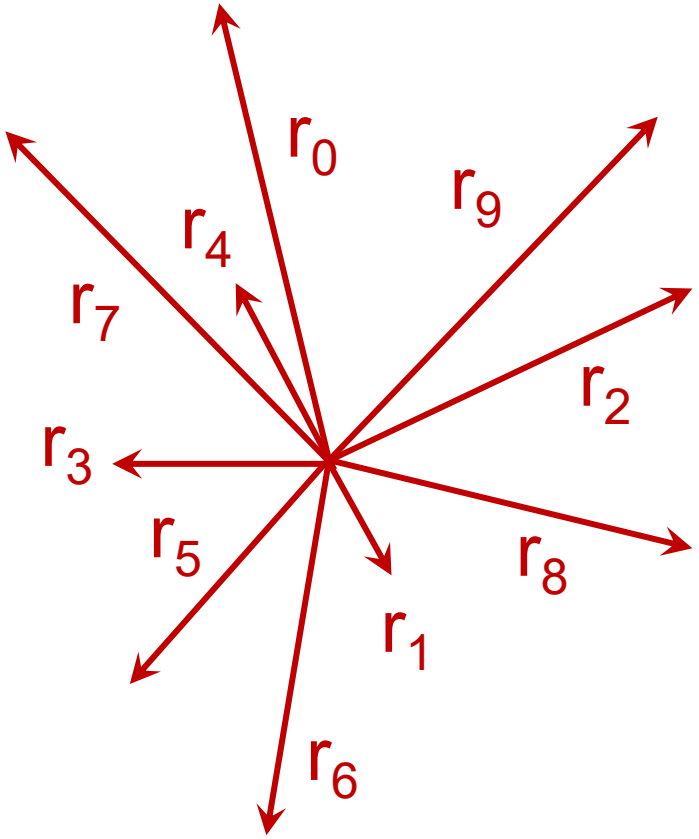


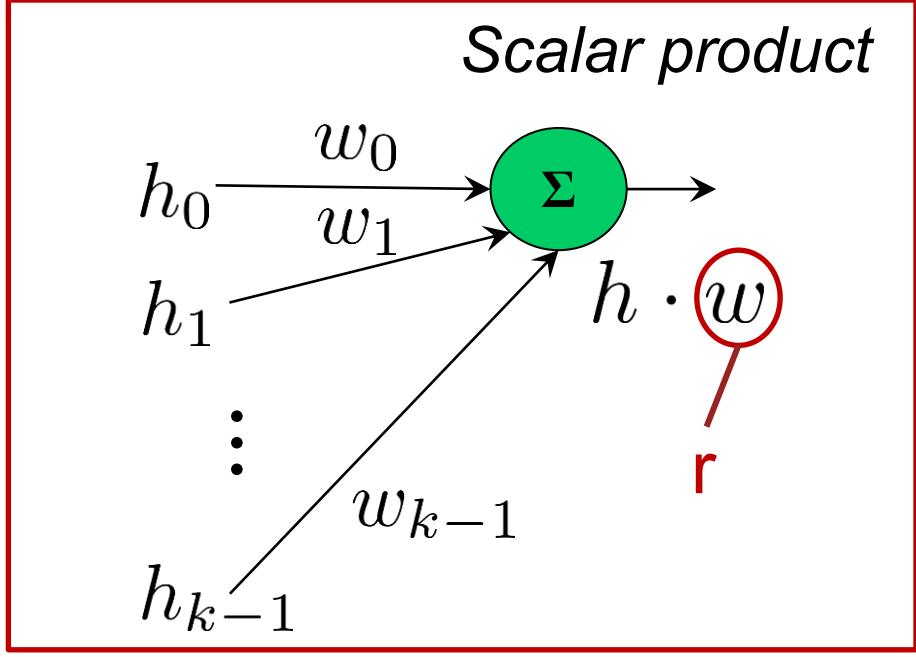


Let us look carefully how logits are calculated

Since we have no biases in the output layer, the output logits are the dot product of the hidden neurons' outputs and the output-layer weights.

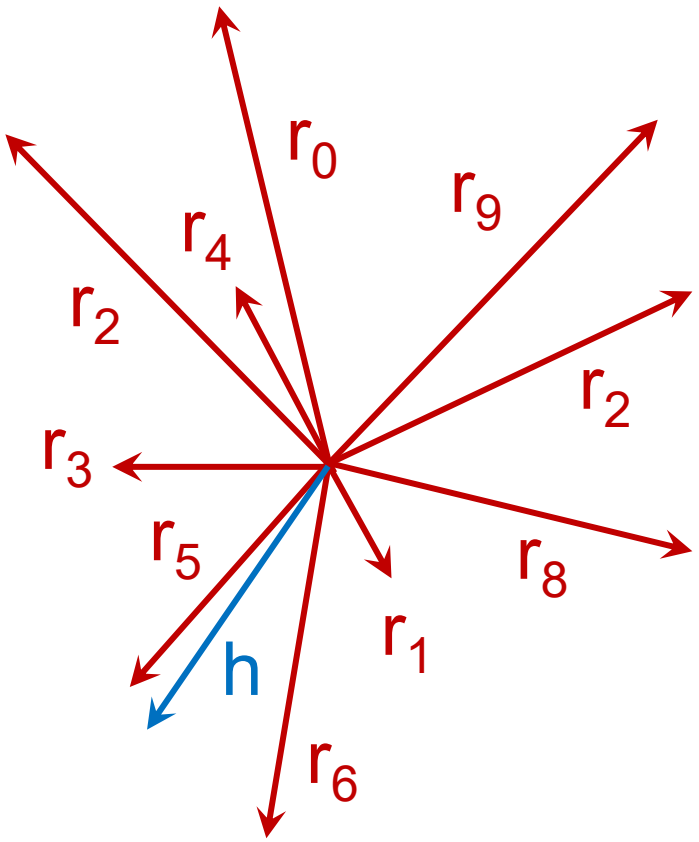
Representatives





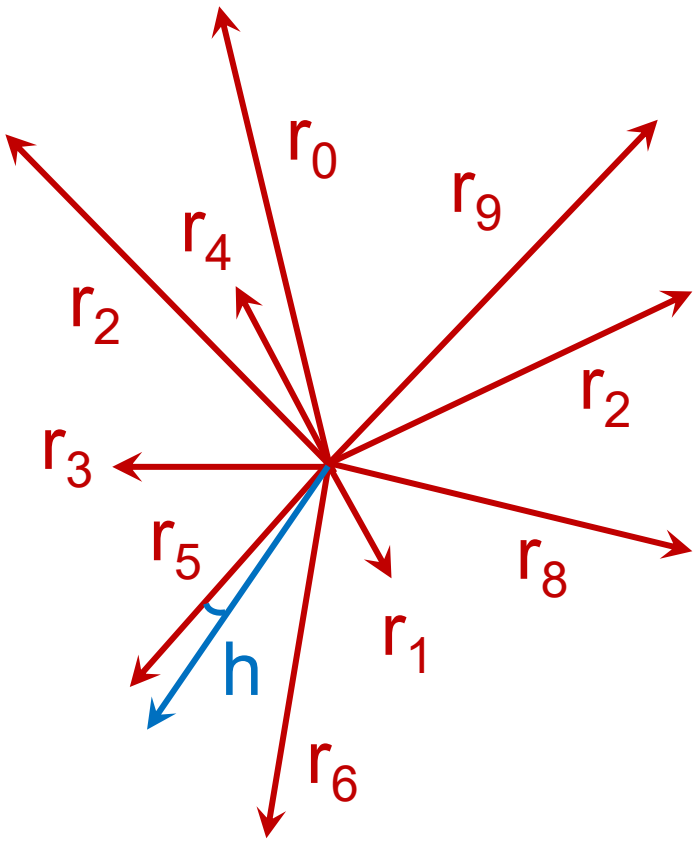
The weights of the output neurons can therefore be considered as m vectors representing individual categories in an n-dimensional space.

Wipeout



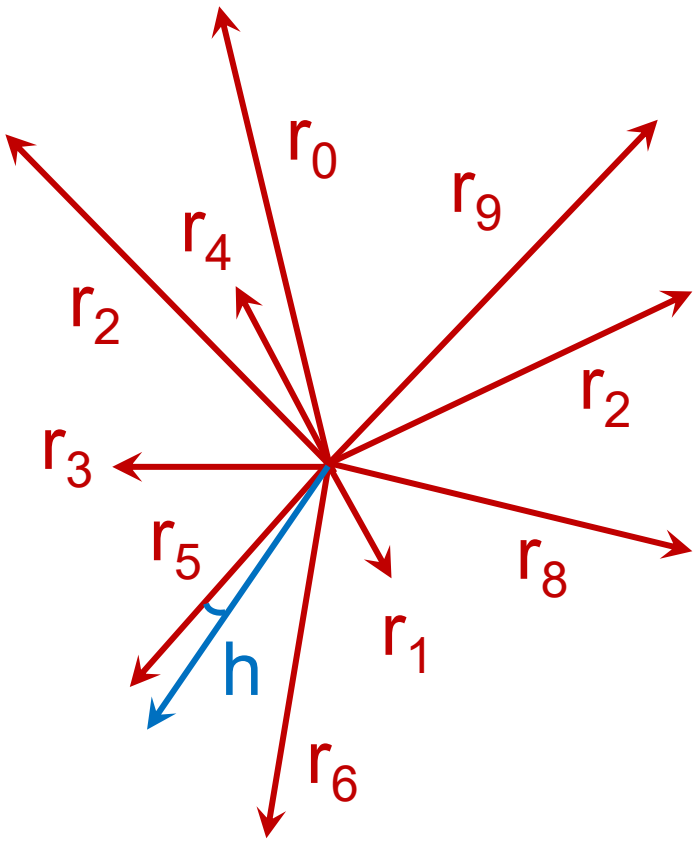
- When the output layer receives a hidden state h,
- it computes dot products with the representatives
- and assigns the index of the representative with the largest dot product as the specified category.

Wipeout



- According to the law of cosines for vectors, the scalar product is equal to the cosine of the angle formed by the vectors multiplied by the lengths of the vectors.
- So, if the representatives and all hidden states are approximately the same size,
- the network will declare the category to be the one whose representative forms the smallest angle with the hidden state (cosine similarity).

Wipeout



- We can call hidden state h a query q
- We can call representatives r_i keys k_i
- Then the most probable category is:

$$\arg\max_i qk_i$$

- and probabilities of belonging to a category are:

$$\mathrm{softmax}\left(qK^T\right)$$

Matrix notation

$$q = (q_0, q_1, \ldots, q_{d-1})$$

$$k_i = (k_{i,0}, k_{i,1}, \ldots, k_{i,d-1})$$

$$K = \begin{pmatrix} k_{0,0} & k_{0,1} & \cdots & k_{0,d-1} \\ k_{1,0} & k_{1,1} & \cdots & k_{1,d-1} \\ \vdots & \vdots & \ddots & \vdots \\ k_{l-1,0} & k_{l-1,1} & \cdots & k_{l-1,d-1} \end{pmatrix} \quad k_0$$

$$K^T = \begin{pmatrix} k_{0,0} & k_{1,0} & \cdots & k_{l-1,0} \\ k_{0,1} & k_{1,1} & \cdots & k_{l-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ k_{0,d-1} & k_{1,d-1} & \cdots & k_{l-1,d-1} \end{pmatrix} \quad k_0$$

$$qK^T = (q_0, q_1, \ldots, q_{d-1}) \begin{pmatrix} k_{0,0} & k_{1,0} & \cdots & k_{d-1,0} \\ k_{0,1} & k_{1,1} & \cdots & k_{d-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ k_{0,l-1} & k_{1,l-1} & \cdots & k_{d-1,l-1} \end{pmatrix} = (qk_0, qk_1, \ldots, qk_{l-1})$$

$$qk_i = \sum_{j=0}^{d-1} q_j k_{i,j}$$

Embedding

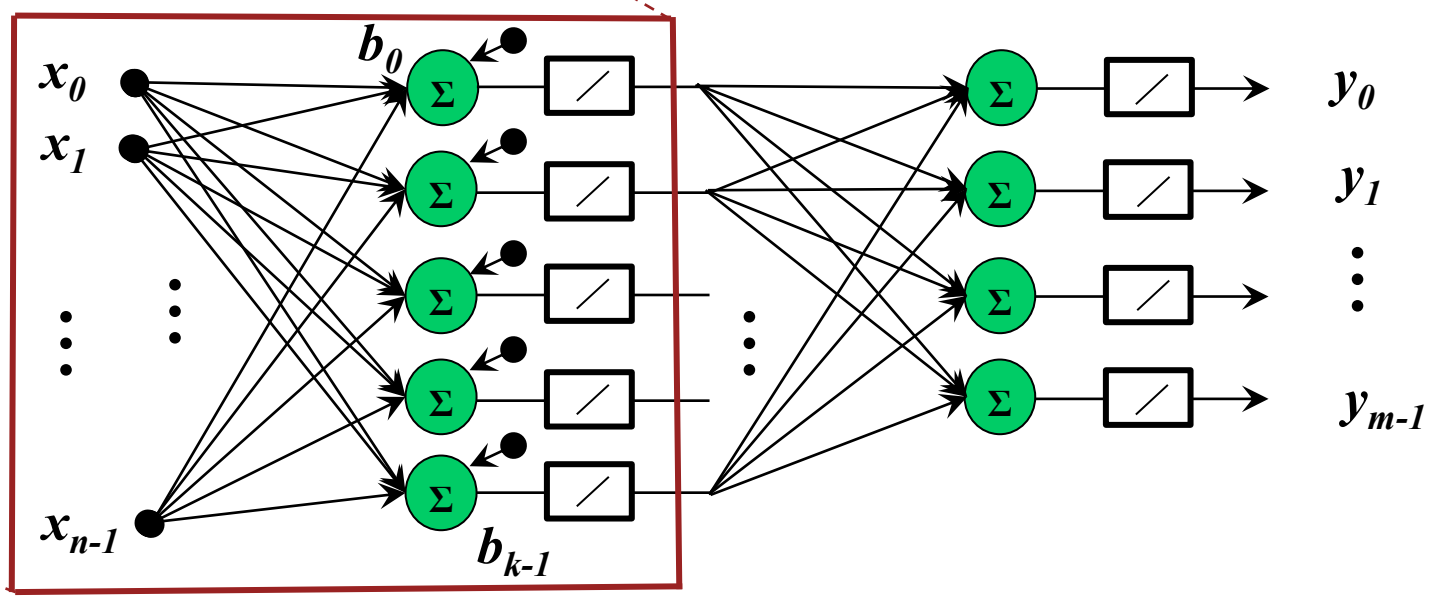
(for visual data)

afinne transformation

$$(x_0, x_1, \ldots, x_{n-1})$$

$$(h_0, h_1, \ldots, h_{k-1})$$

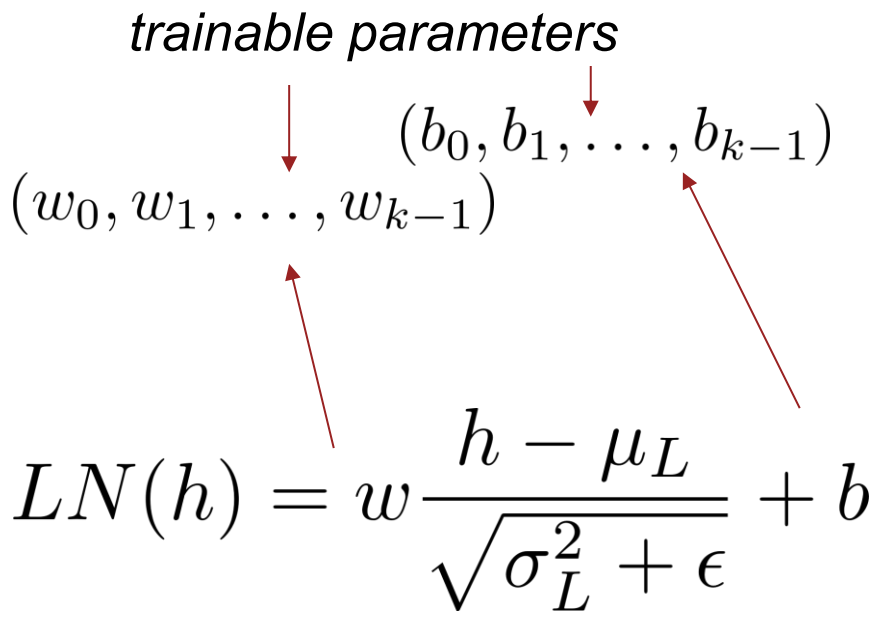
$$h_i = \sum_{j=0}^{n-1} x_j w_{i,j} + b_i$$



- We can think of the hidden state as an internal representation of the input
- The dimension of the hidden state corresponds to the number of features by which two inputs can differ from each other

Layer Normalization

- When we insert a building block between the embedding and wipeout that ensures the ranges and distributions of the individual features are roughly the same,
- this also ensures that the hidden states have roughly the same size, and
- the similarity between the hidden states is cosine similarity.

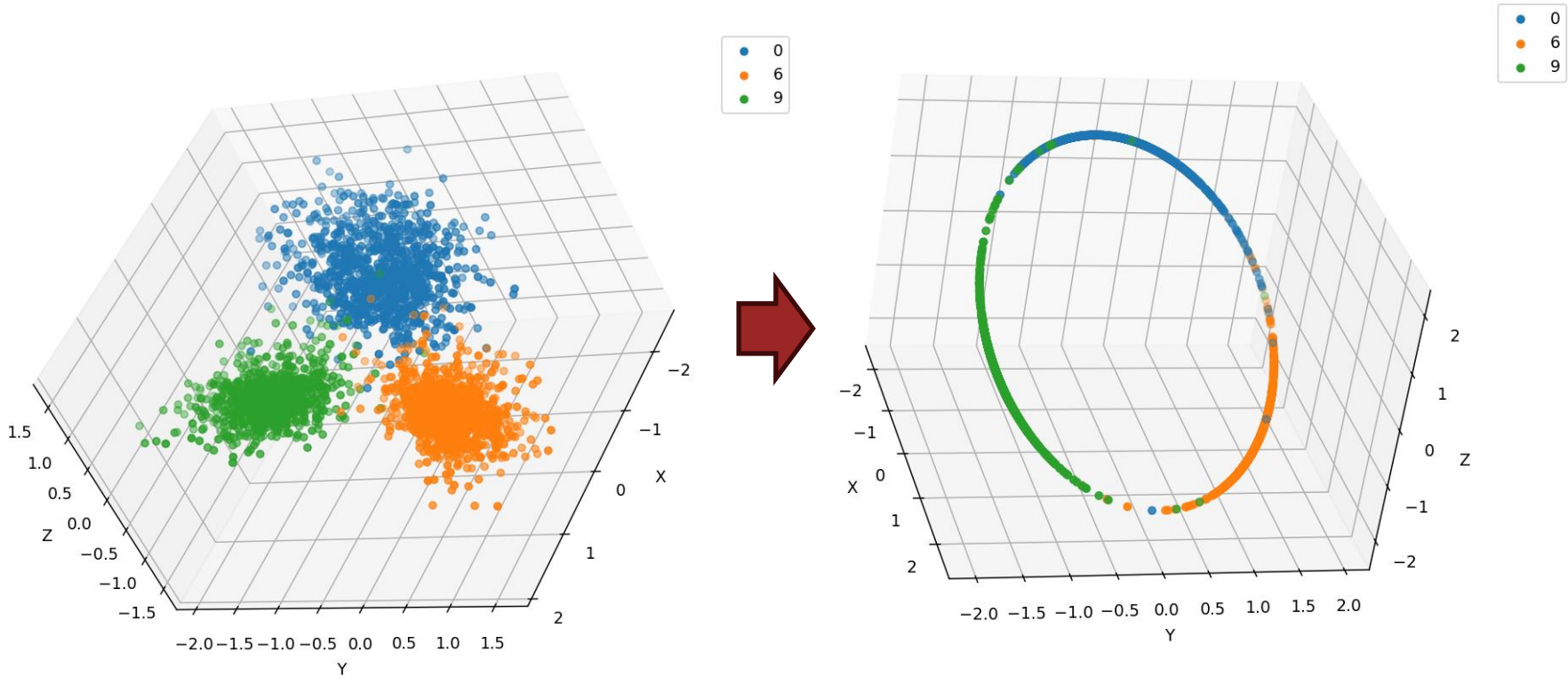


$$\mu_L = \frac{1}{k}\sum_{i=0}^{k-1} h_i$$

$$\sigma_L^2 = \frac{1}{k}\sum_{i=0}^{k-1}(h - \mu_L)^2$$

Layer Normalization

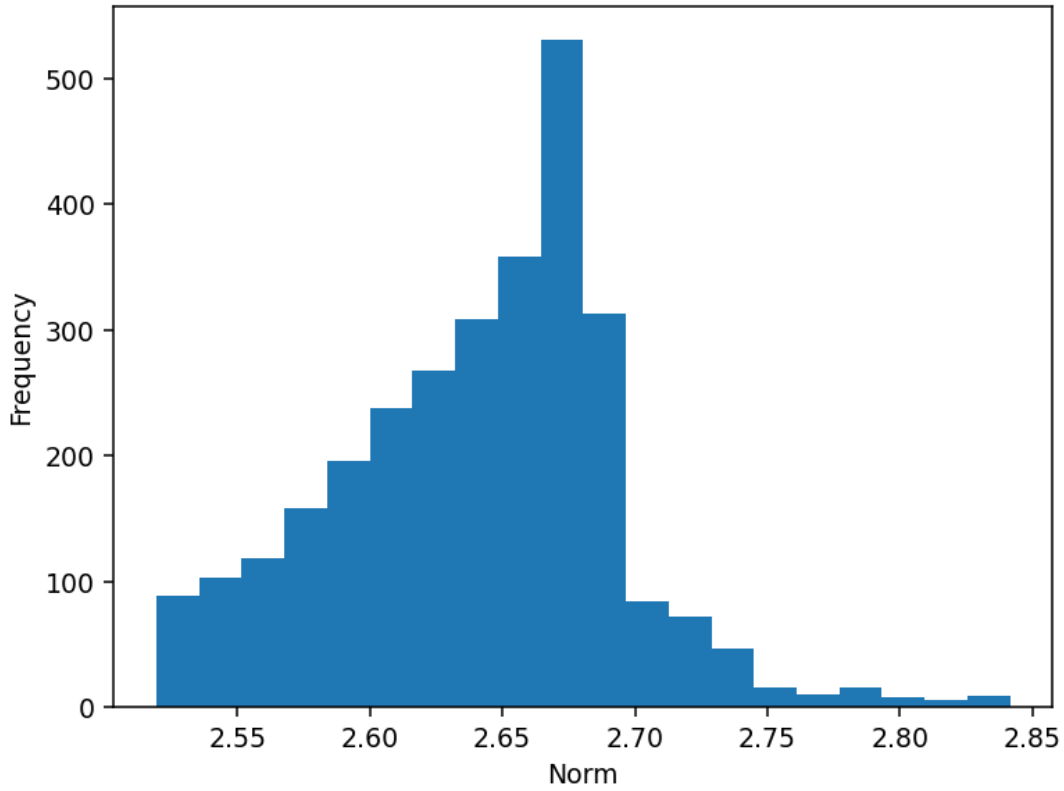
- projects samples to a hypersphere
- for d=3 on a circle,
- for d=4 on a sphere, ...



d = 3

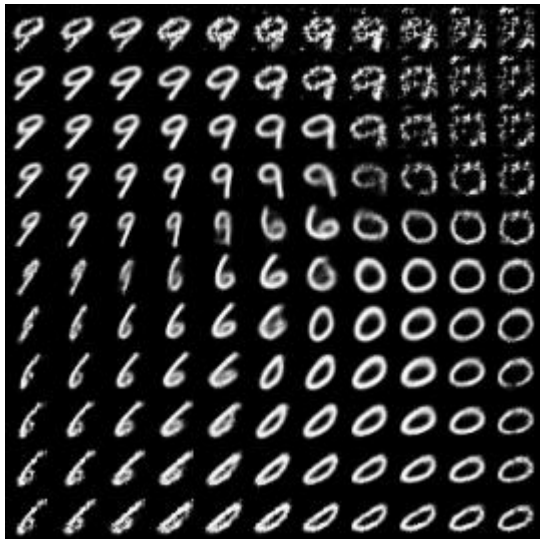
Layer Normalization

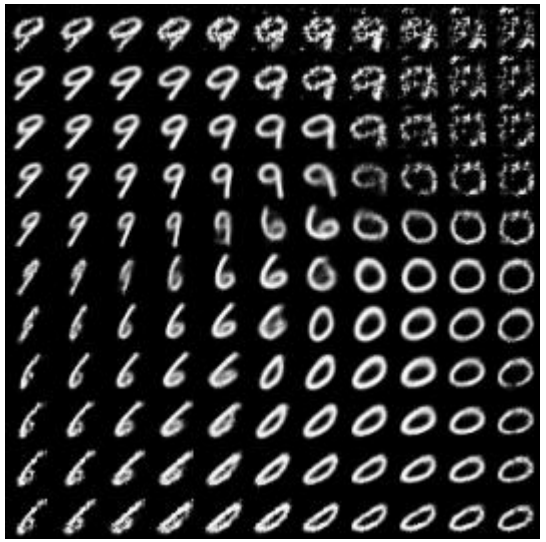
- As a result, feature vectors are similar in size, though not equal



Mapping Latent Spaces

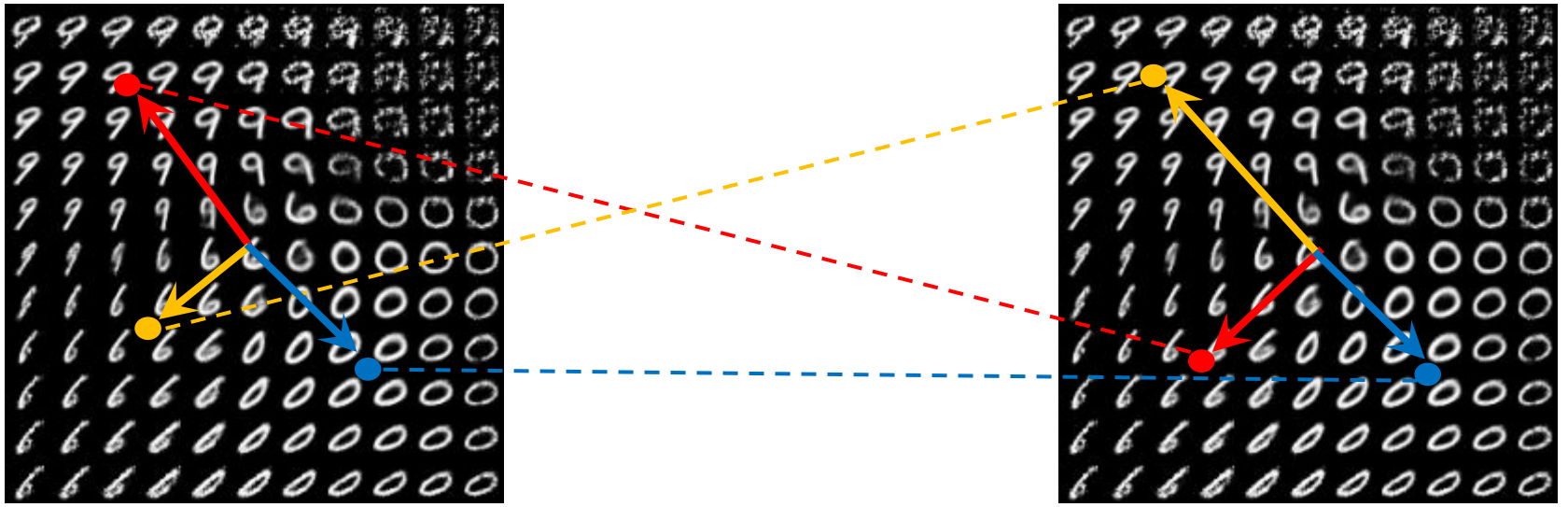
- Having an autoencoder of the digits 0, 6, 9
- How to refurbish it to a model that rotates the digits 0, 6, 9 by 180°?





Mapping Latent Spaces

- We need a mechanism that, for each instance, computes the corresponding instance based on a few associations we define.

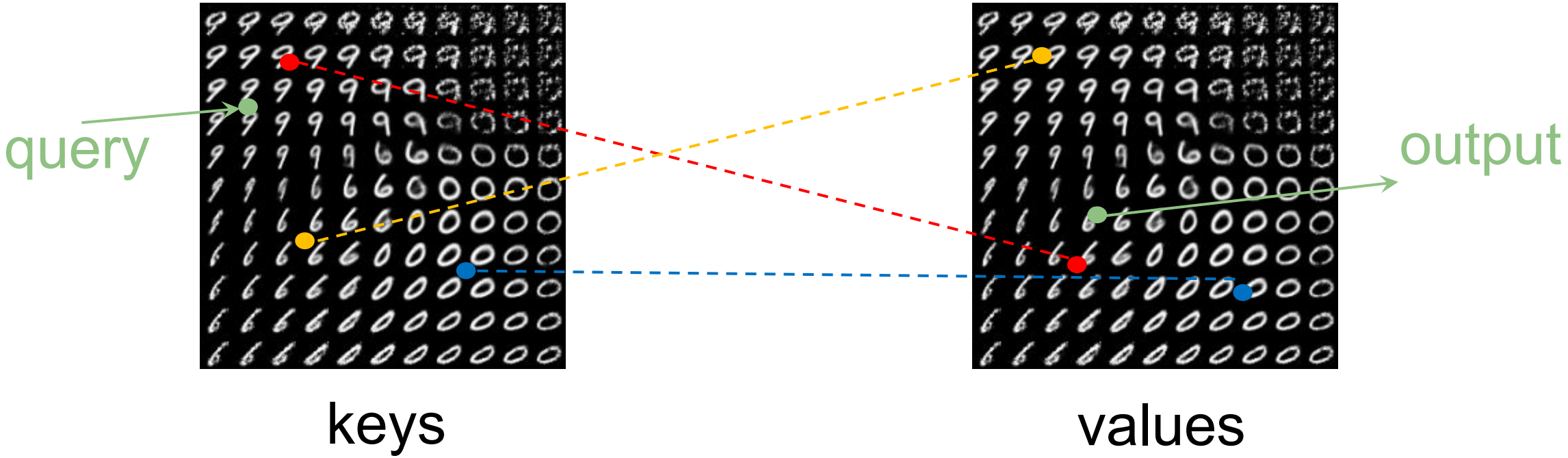


keys

values

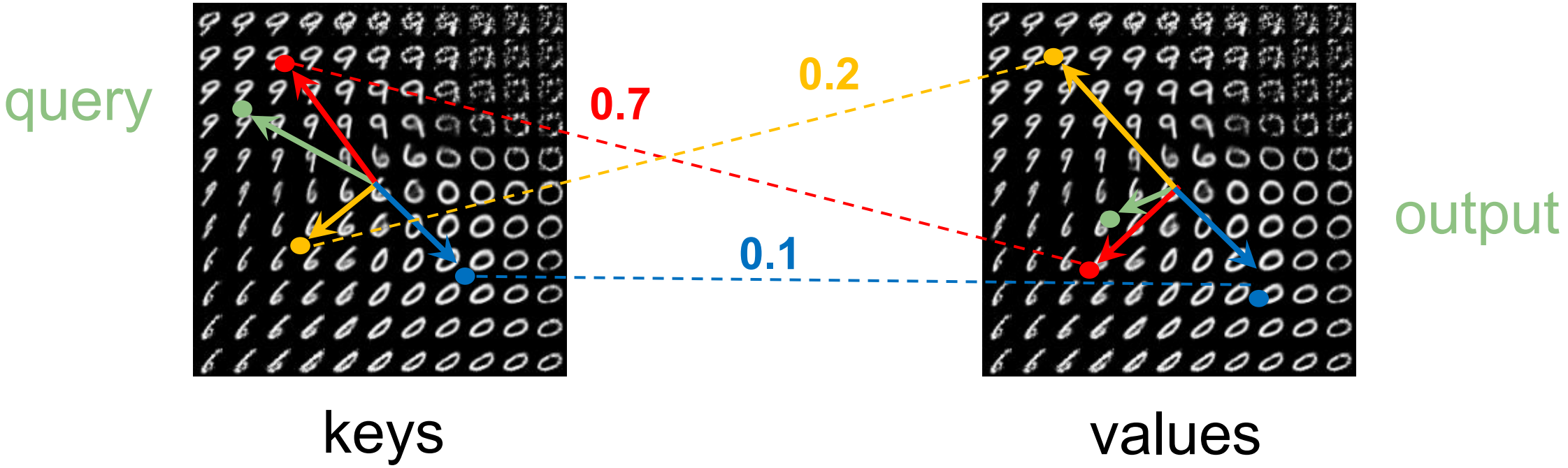
Mapping Latent Spaces

- How to calculate the correct output for any query using key-value pairs?



Mapping Latent Spaces

- We "mix" the query from the keys and produce the output as an analogous "mix" of values. The "mixing" coefficients, called attention weights, range from 0 to 1 and sum to 1.



Mapping Latent Spaces

- These weights are easy to calculate as long as the cosine similarity holds for the latent space of dimension d

$$K = \begin{pmatrix} k_0 \\ k_1 \\ \dots \\ k_{l-1} \end{pmatrix} \qquad V = \begin{pmatrix} v_0 \\ v_1 \\ \dots \\ v_{l-1} \end{pmatrix}$$

keys values

Mapping Latent Spaces

- How much the query q resembles keys $k_0, k_1, \ldots, k_{l-1}$ can be expressed in terms of their dot products:

$$(q \cdot k_0, q \cdot k_1, \ldots, q \cdot k_{l-1}) = qK^T$$

- The larger scalar product, the more the query resembles a key

Mapping Latent Spaces

- Can we compute the attention weights from

$$(q \cdot k_0, q \cdot k_1, \ldots, q \cdot k_{l-1}) = qK^T \quad ?$$

- Yes, of course:

$$c = softmax\left(qK^T\right)$$

Mapping Latent Spaces

$$c = softmax\left(\frac{qK^T}{s}\right)$$

- By adding a scale s, we can control how much of the more similar and the more dissimilar keys we mix.
- At sufficiently small scale, we get one-hot encoding.
- The natural scale is the square root of the dimension.

$$s = \sqrt{d}$$

Mapping Latent Spaces

$$o = cV = c_0 v_0 + c_1 v_1 + \cdots + c_{l-1} v_{l-1} =$$

$$= \left(\sum_{i=0}^{l-1} c_i v_{i,0}, \sum_{i=0}^{l-1} c_i v_{i,1}, \ldots, \sum_{i=0}^{l-1} c_i v_{i,d-1} \right)$$

- We then mix the output analogously from the values

Mapping Latent Spaces

- All together, we get formula:

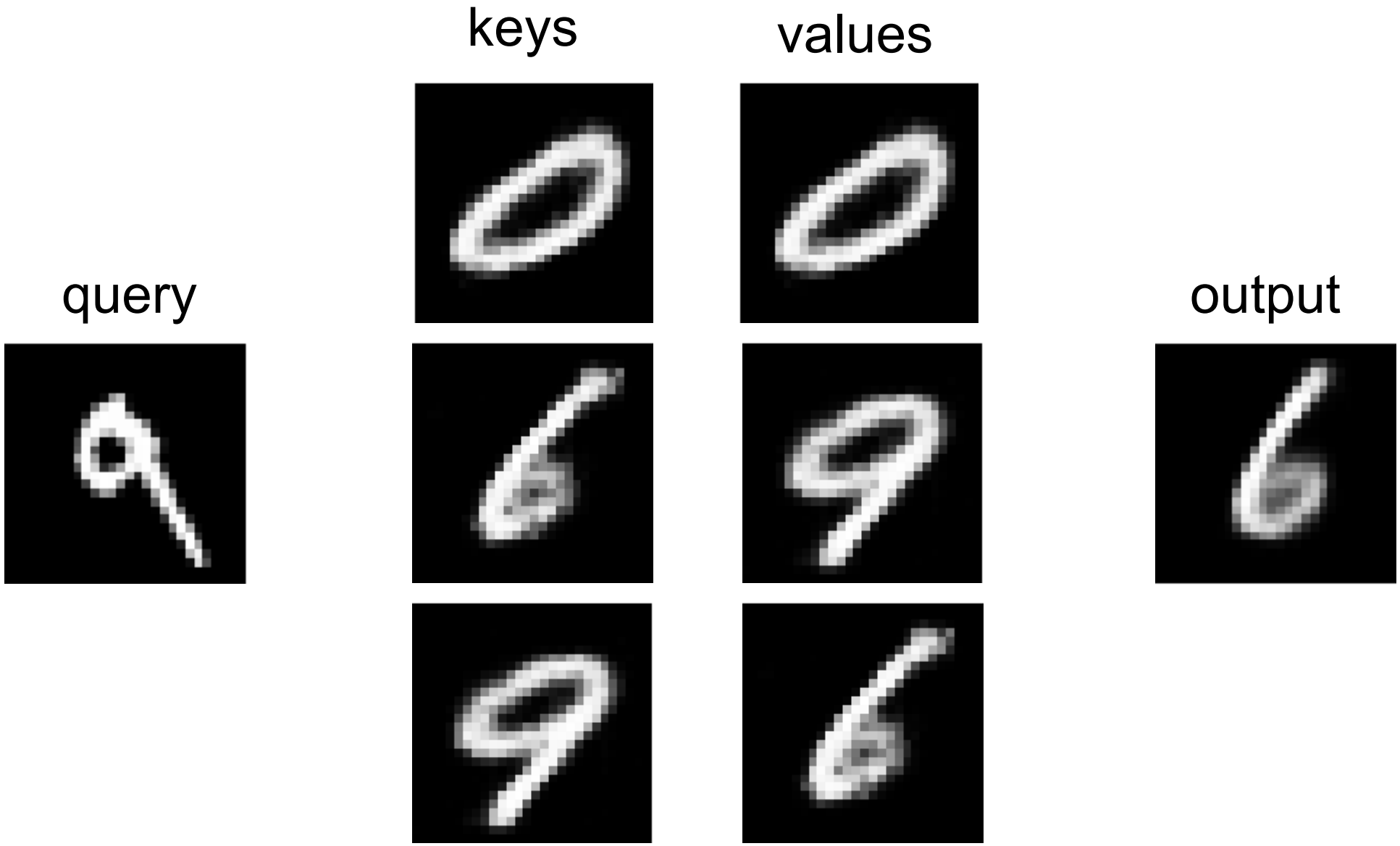
$$o = softmax\left(\frac{qK^T}{\sqrt{d}}\right)V$$

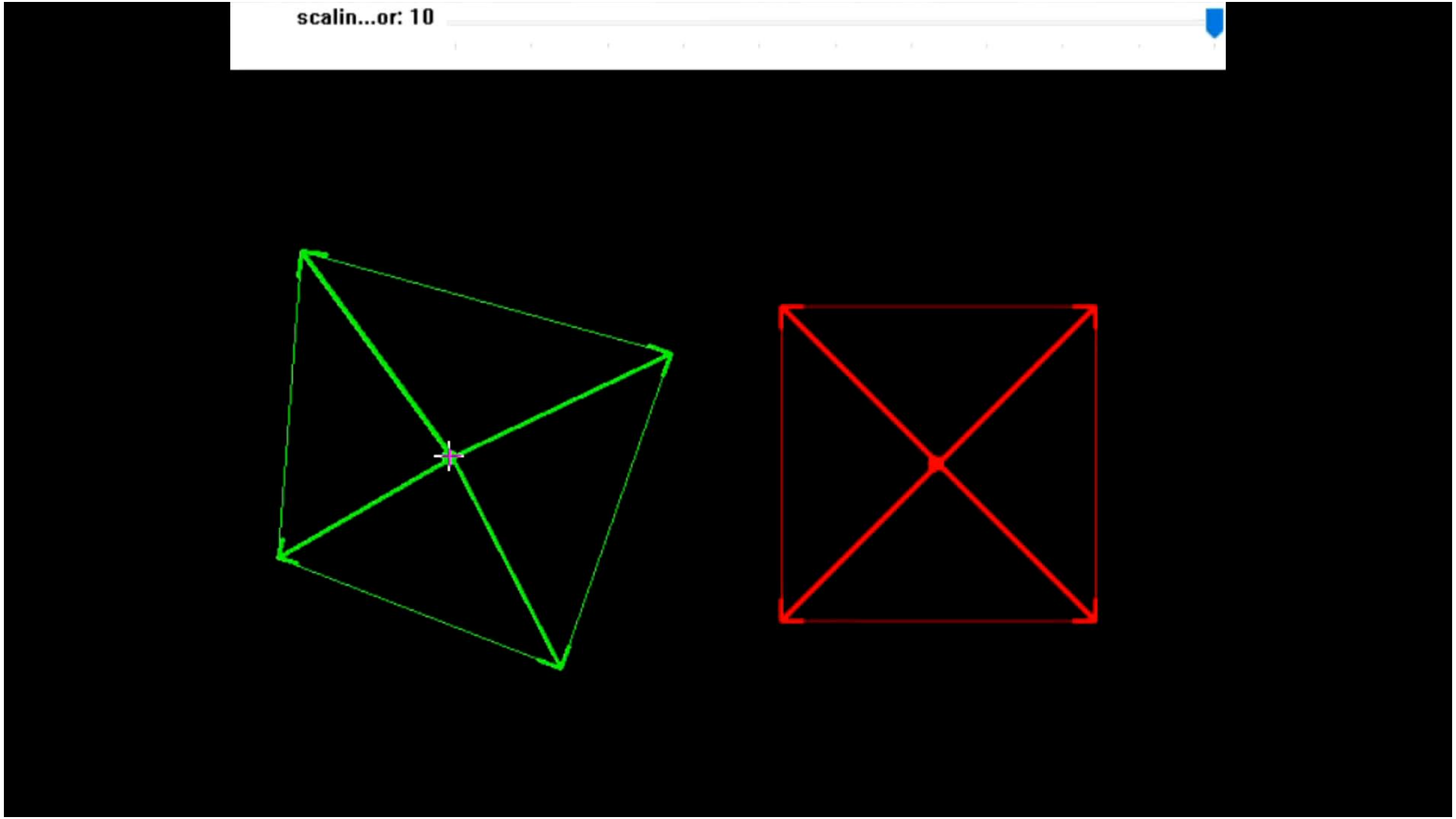
Mapping Latent Spaces

- For more queries $Q = \begin{pmatrix} q_0 \\ q_1 \\ \dots \\ q_{k-1} \end{pmatrix}$
  - we then get:

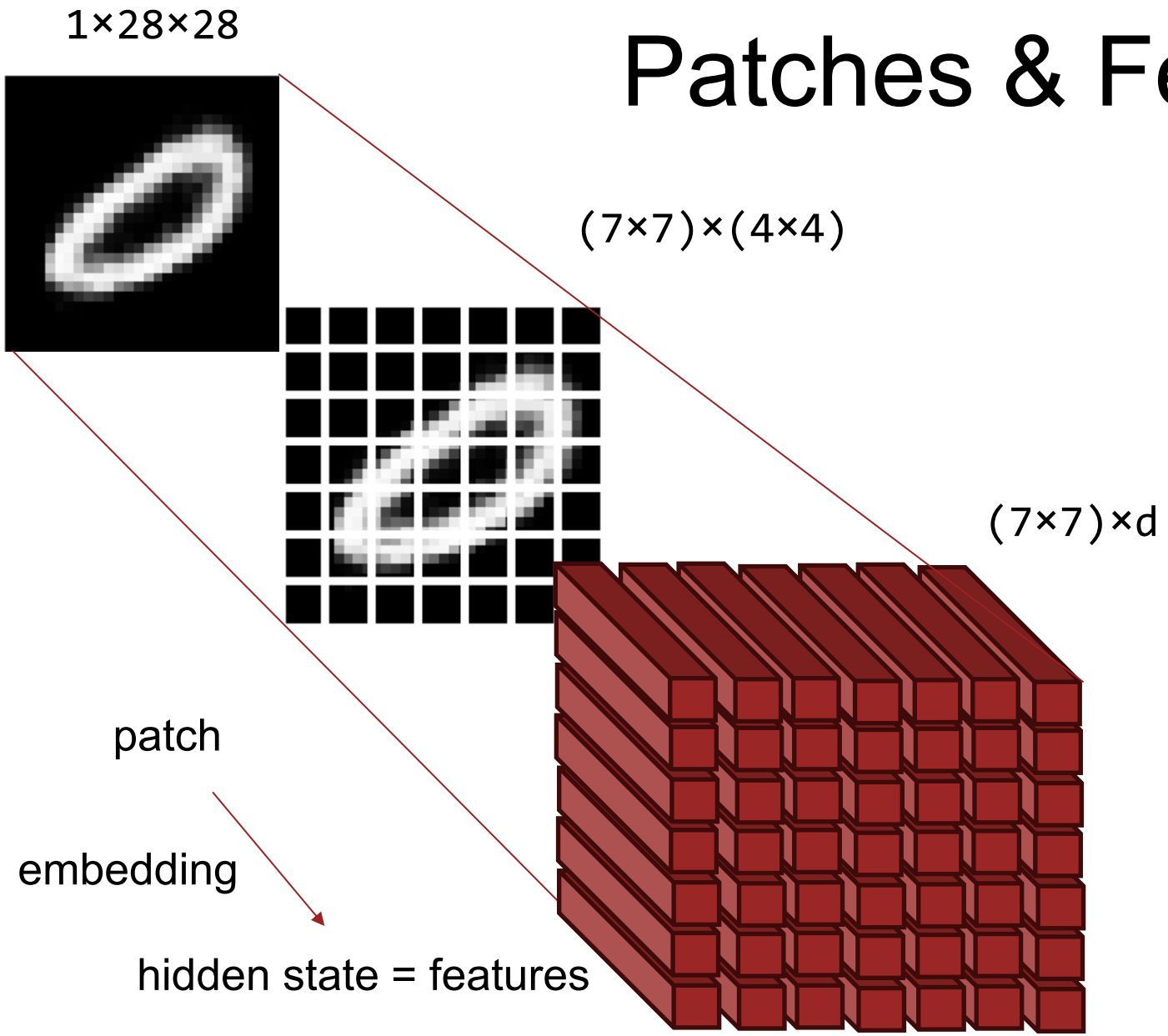
$$O = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

Attention


keys
values
query
output

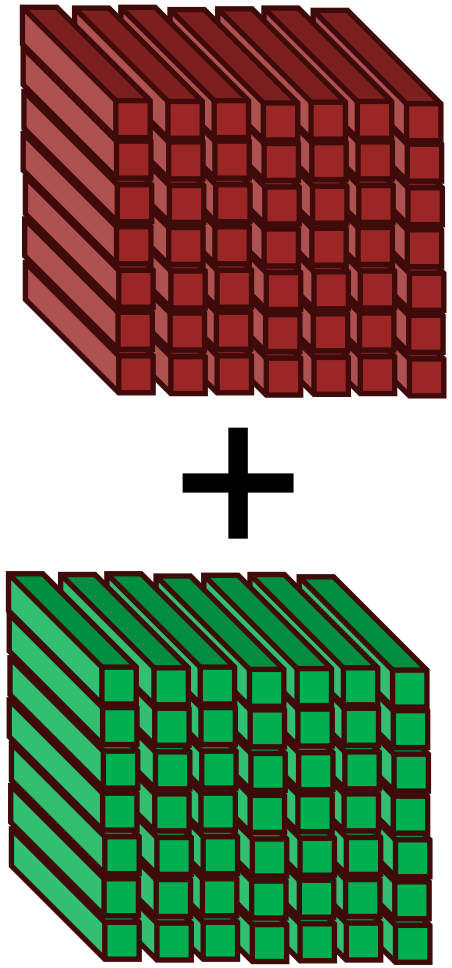

scalin...or: 10

Patches & Feature Maps



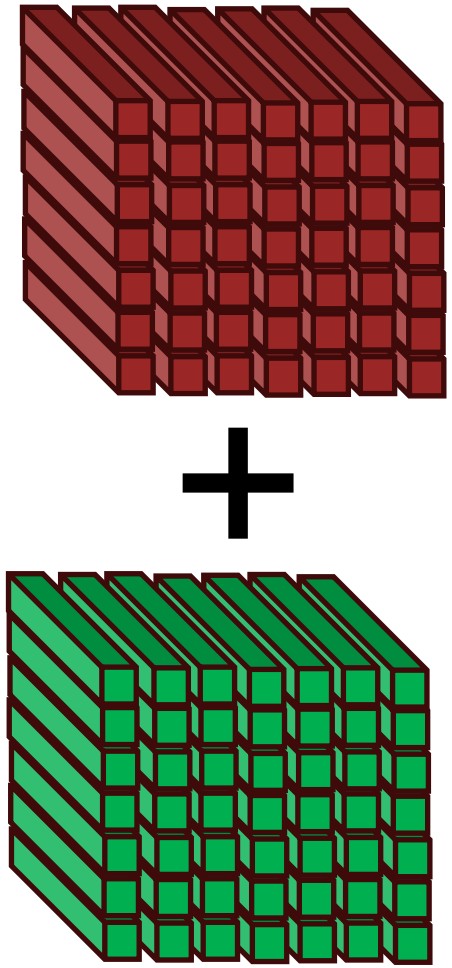
- The success of the embedding-wipeout architecture applied to the image as a whole is limited,
- but we can successfully apply it to its patches.

Trainable Position Encoding



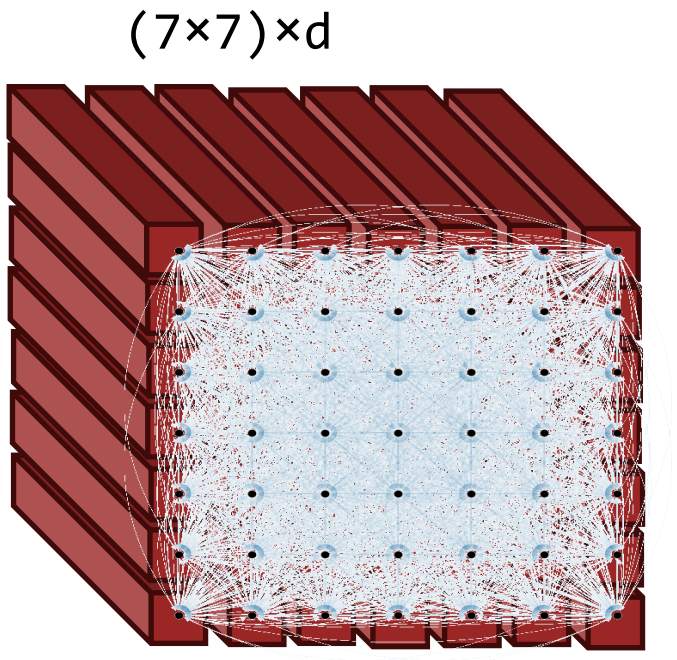
- The embedding of a patch does not depend on its position in the image.
- However, the position has an effect on its meaning.
- Therefore, we extend the system with a parameter (of embedding dimension), specific to each position, which is added to the embedding.
- Its value is trained from the dataset, along with the model weights and biases.

apples + pears ?



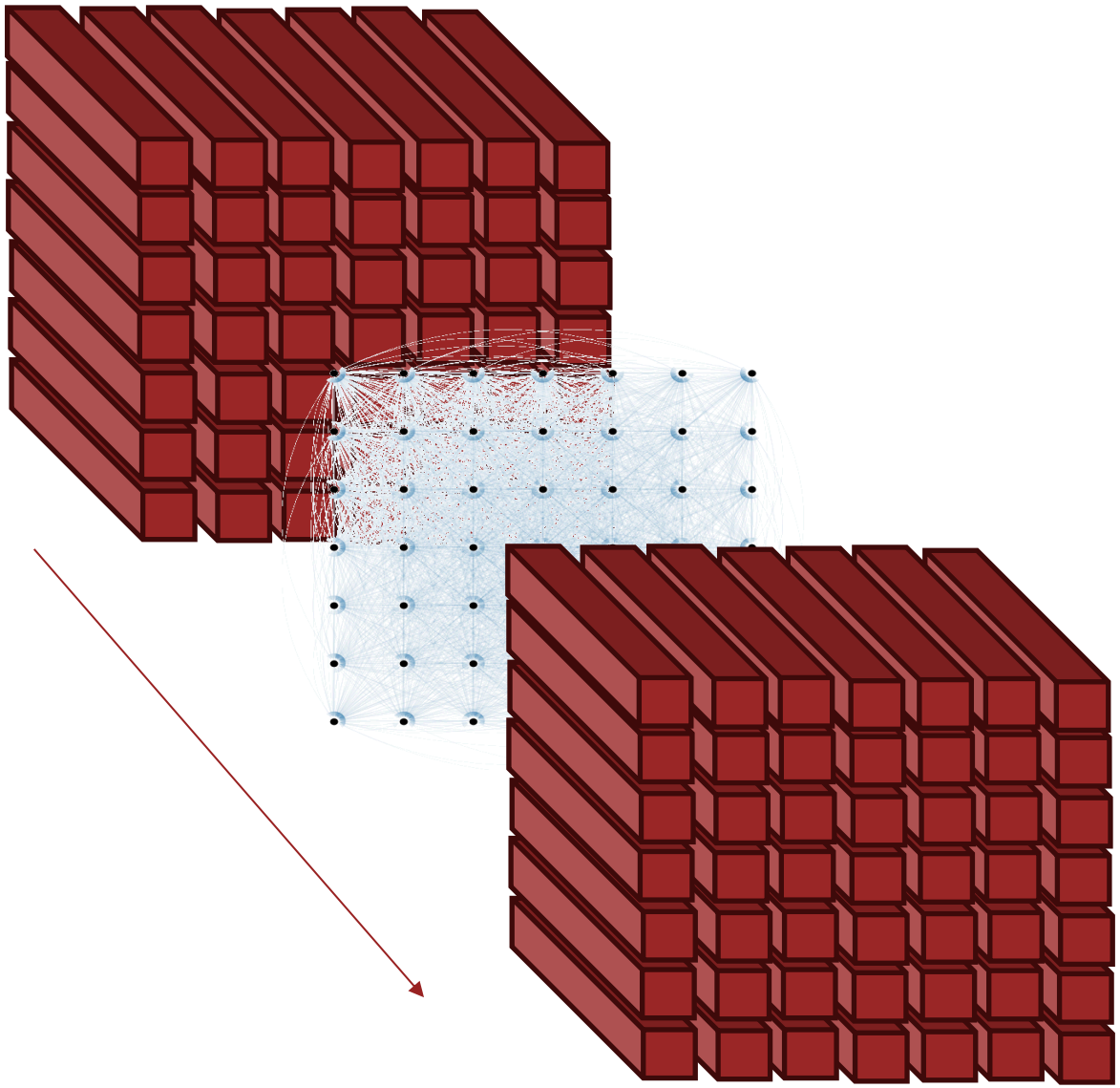
- How could this work?
- Though it is not granted to us that we succeed in training a model with sufficient accuracy,
- If we succeed and the model is designed to work well exactly when we sum apples + apples, it really will be apples and apples at the end of the training (despite starting with apples and pears)

Refining the meaning of hidden states according to context



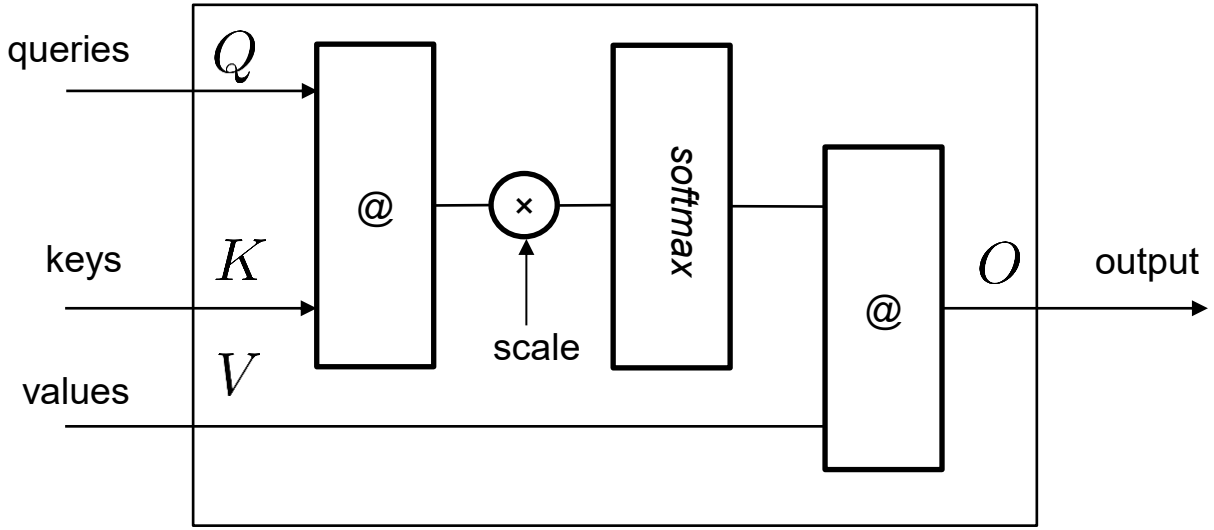
- Each patch is described by *d* features
- Can we refine a patch feature vector with context given by the presence of the other feature vectors?
- If we manage that patches appearing in the same images through our dataset have similar embeddings, we already know one such mechanism

Self-Attention



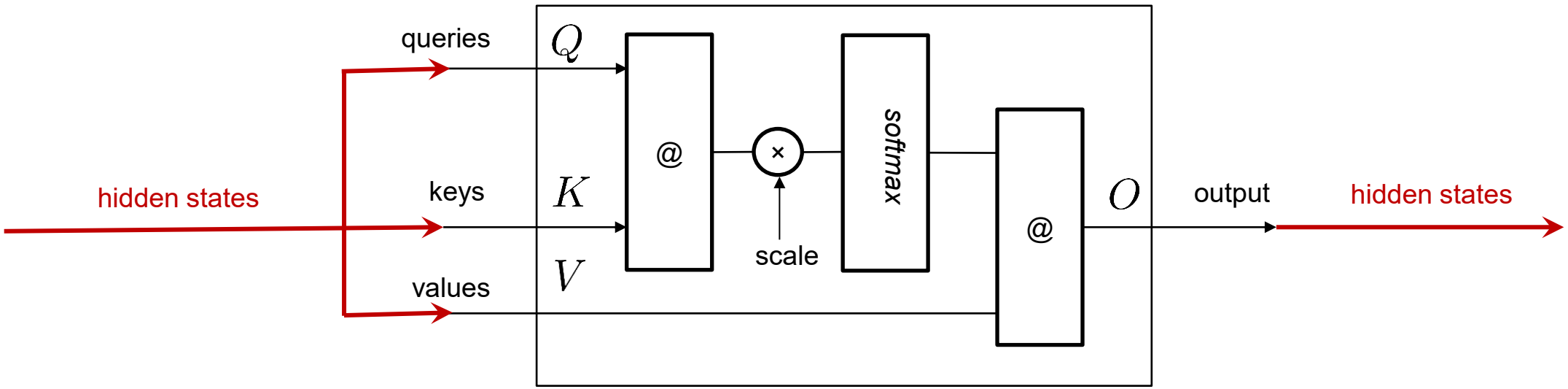
- The mechanism is Attention
- We mix every hidden state a bit with those others that appear with it in the same context (so, they are similar to it, in the sense of cosine similarity)

Attention



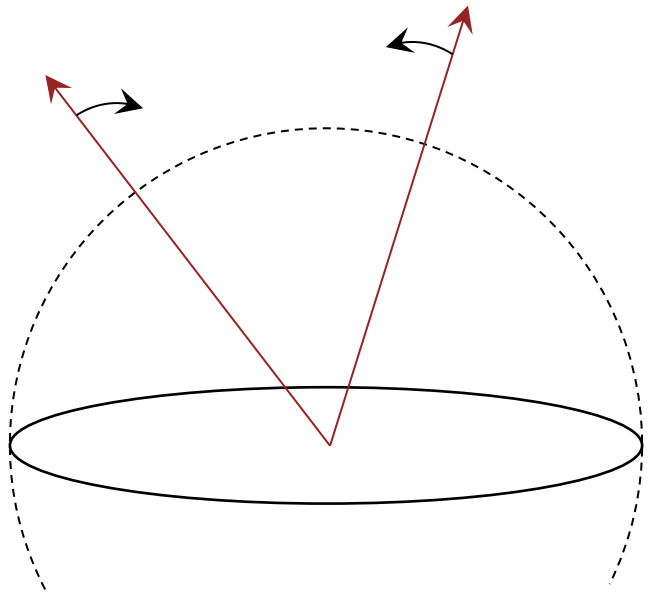
$$O = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V$$

Self-Attention



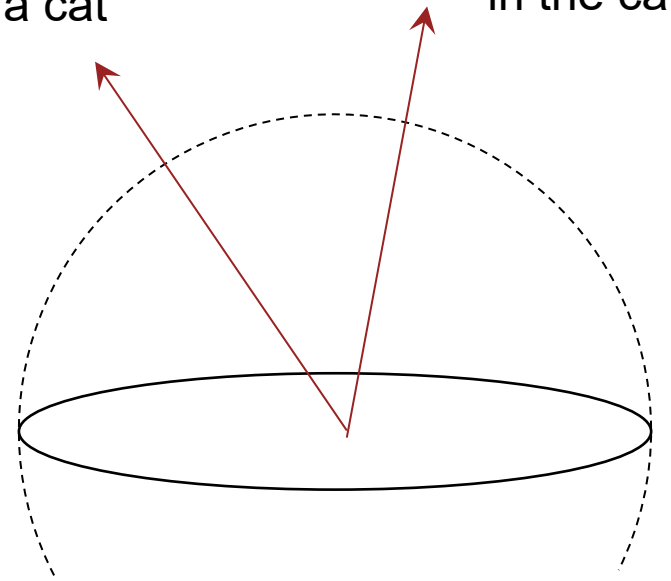
In square 10,10
we see a pointed
ear

In squares 12,13
we see a mouse
in the claws



In square 10,10
we see a pointed
ear of a cat

In squares 12,13
we see a mouse
in the cat's claws



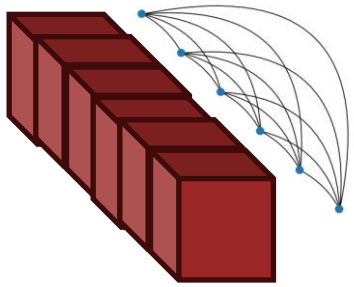
- *d-dimensional space represents the patches' meanings*
- *is sorted in such a way that the smaller the angle of representation they make, the more often they occur in the same context through the dataset, and the more they semantically influence each other*

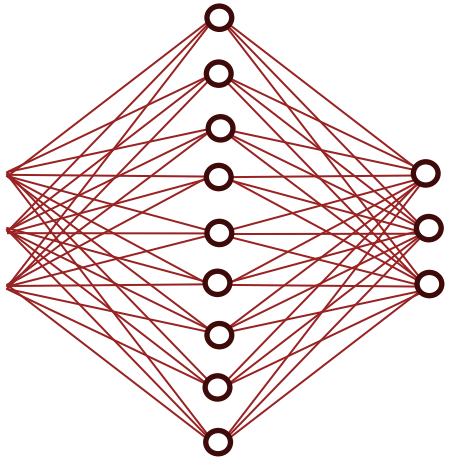
- *We calculate how each hidden state in the space of meanings should shift in response to the presence of other hidden states.*
- *Thus, the hidden states acquire a more global meaning.*

like Baron Munchausen pulling himself out of a swamp by his own hair

- What will guarantee that the hidden-state meaning space is properly organized?
- We obtain this by designing the transformer to work exactly when it is.
- If we succeed in training, the space will be organized.
- This trick is widely used in neural network design and WORKS 😯

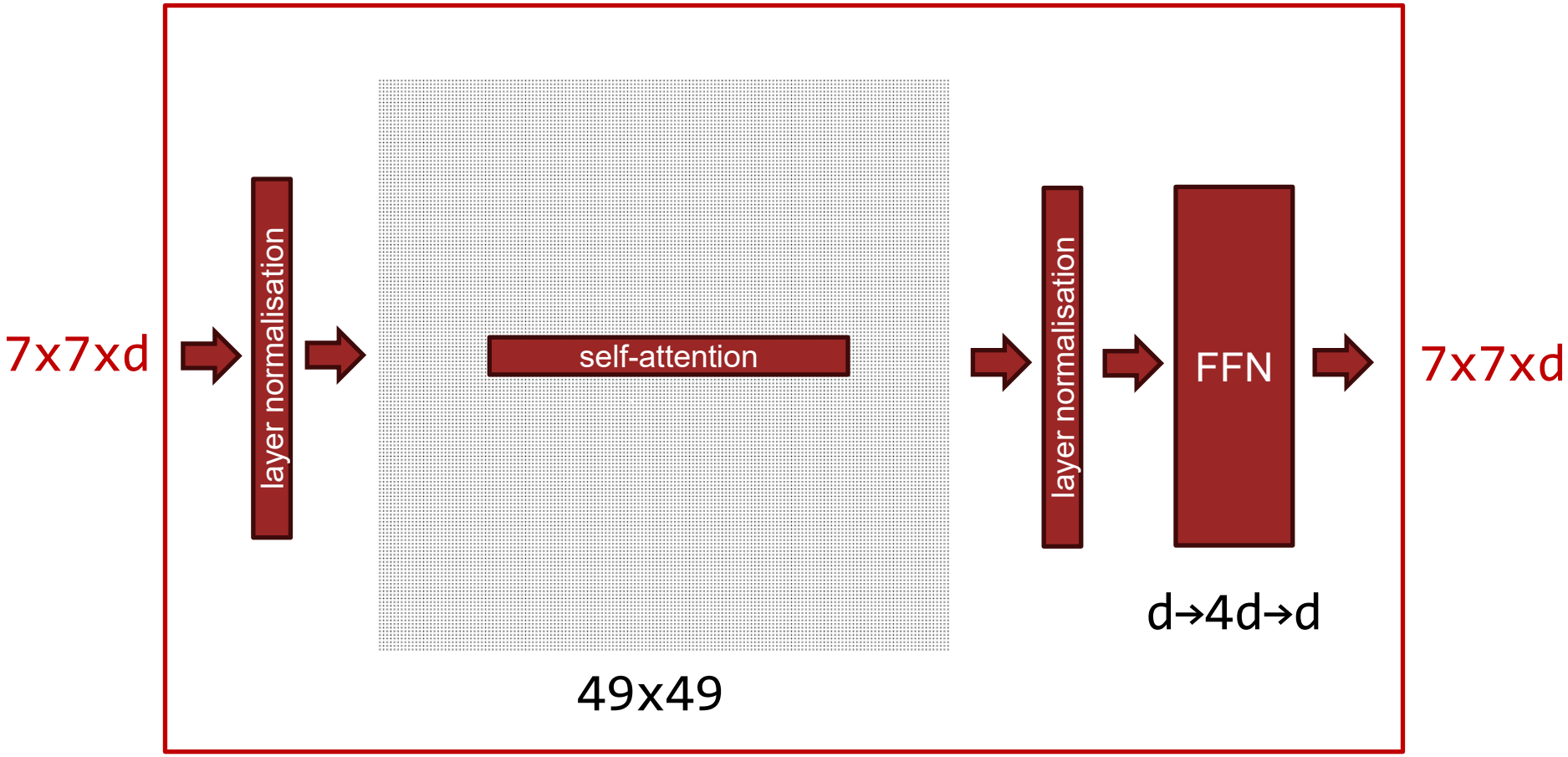
Feed Forward network (FFN)





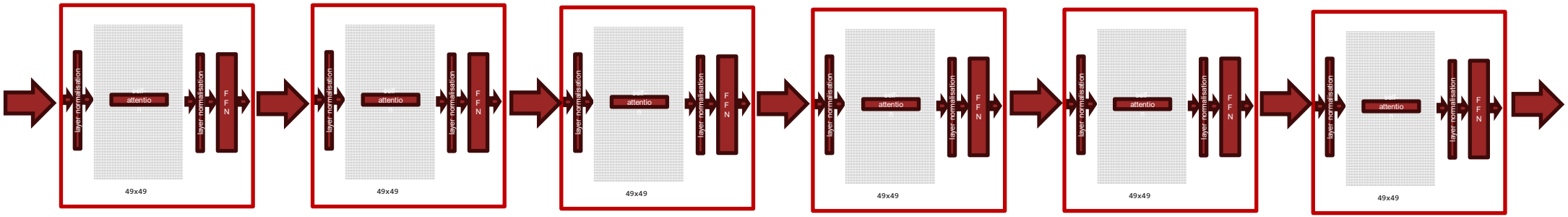
- The idea is also to let individual features influence each other
- For this, we will use a perceptron with as many inputs and outputs as features
- Each hidden state is processed independently by this perceptron
- This introduces nonlinearity into the processing; popular activations in the hidden layer of this perceptron are GELU and SiLU

Transformation block (layer)



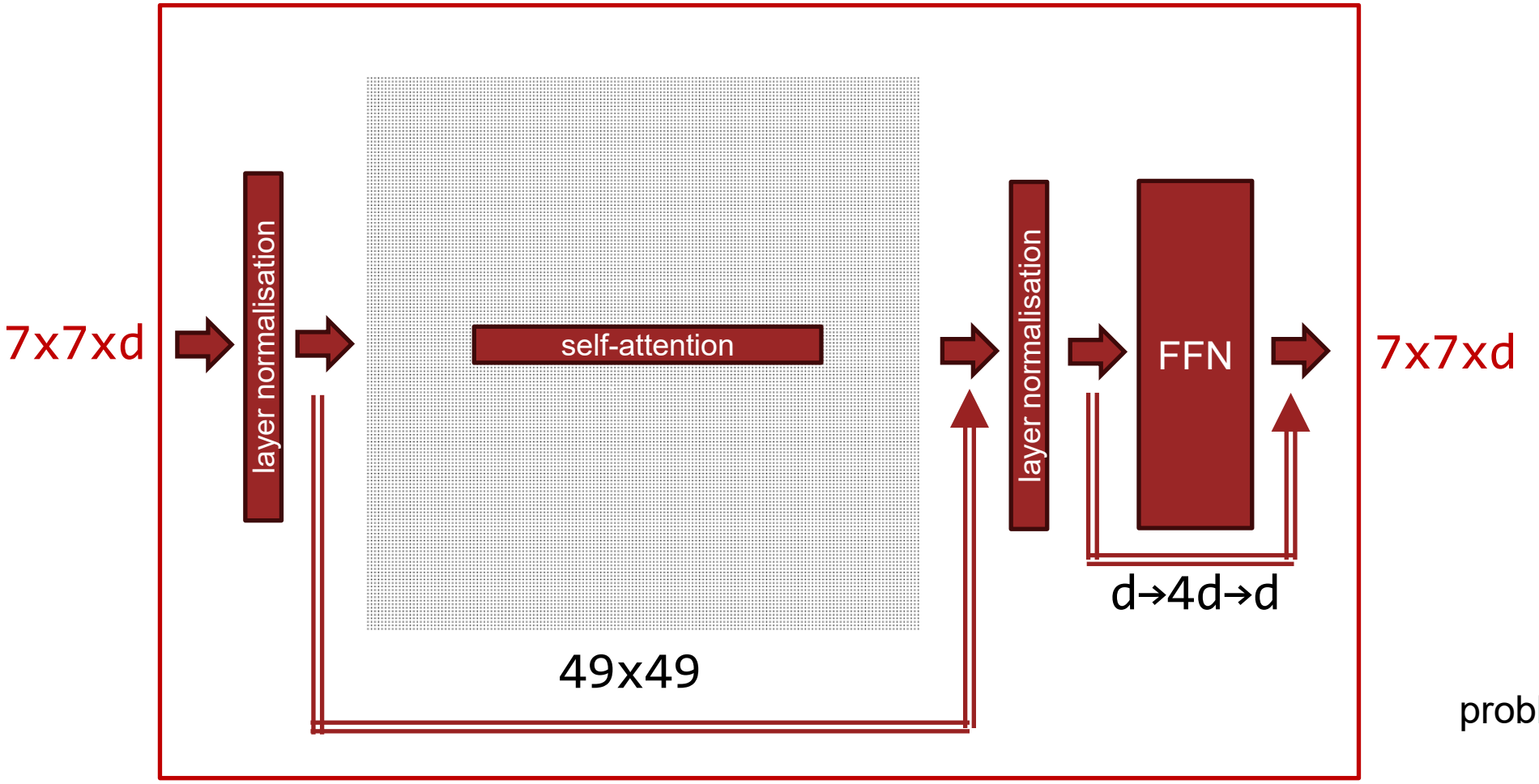
The need for residuals

- It is possible to connect several transformation blocks one after the other to obtain a more precise patches' meaning



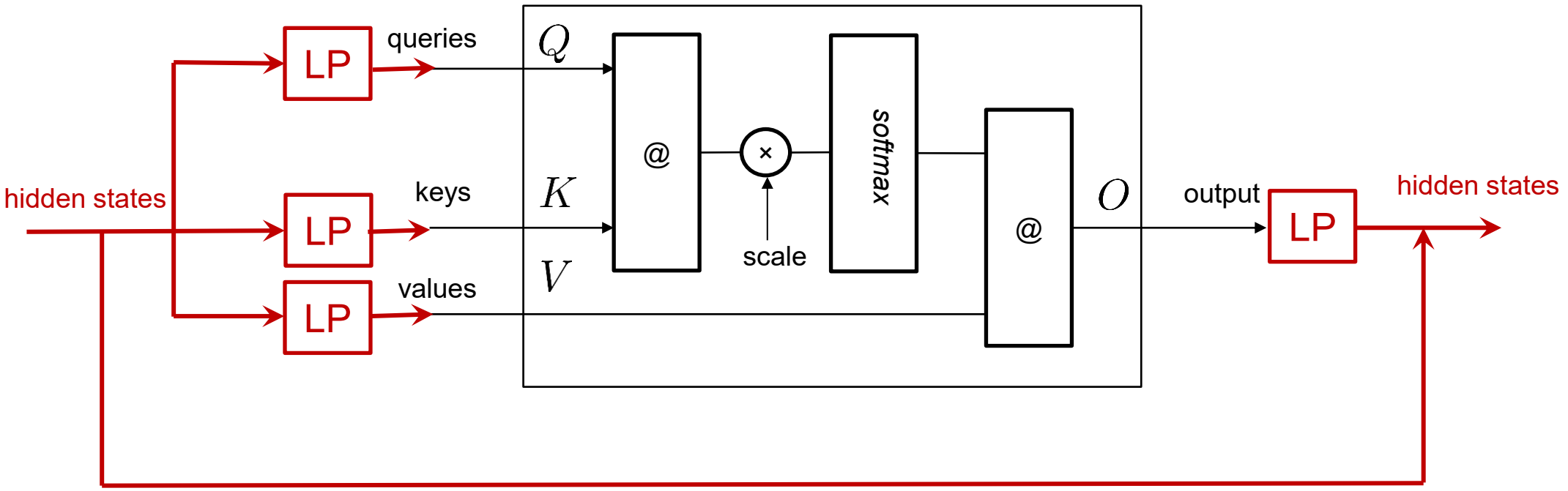
- But alas! We won't be able to train such a deep network
- But we already know what to do with it: we need to insert residual connections into the network

Transformation block with residuals



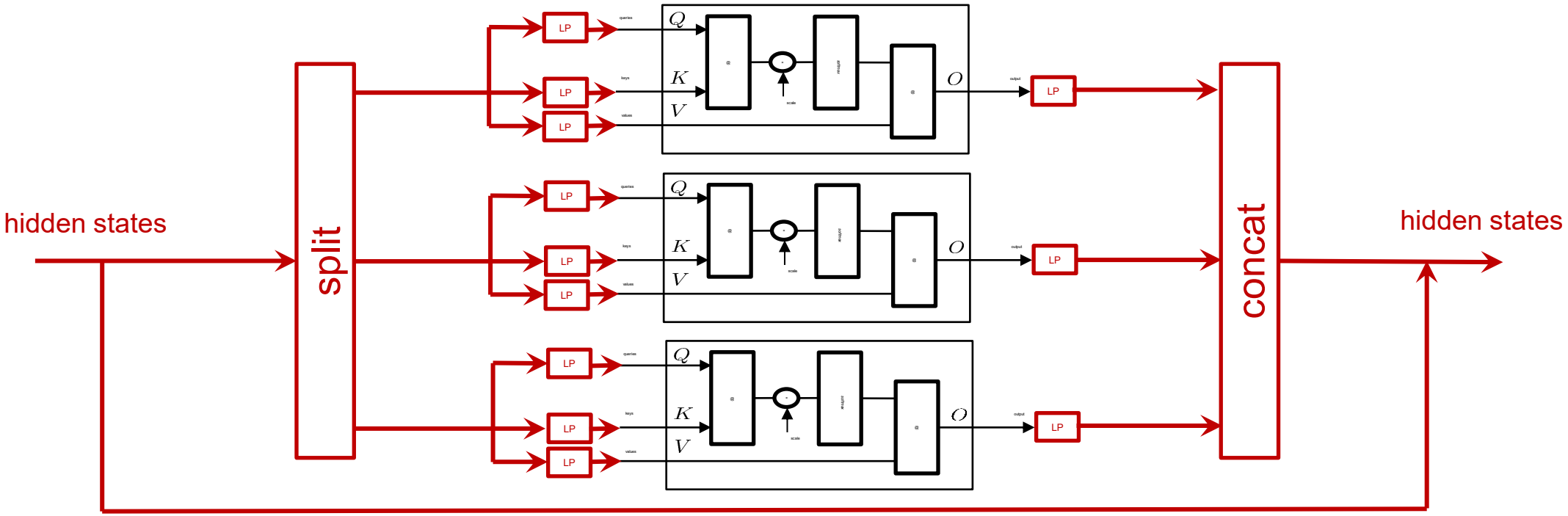
problem: Attention

Residual Self-Attention



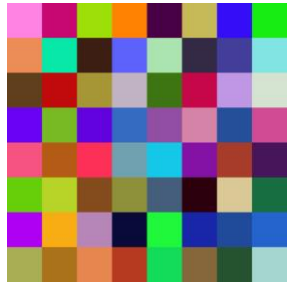
- It does not compute the changed hidden state, but only its change

Residual Multi-head Self-Attention

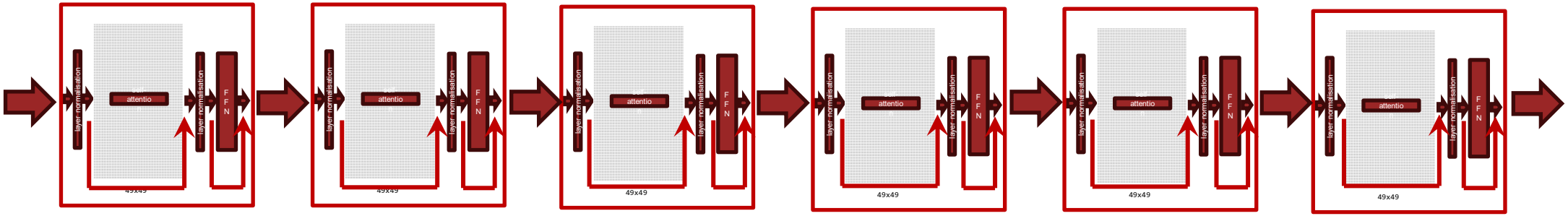


It divides features into groups that do not affect each other

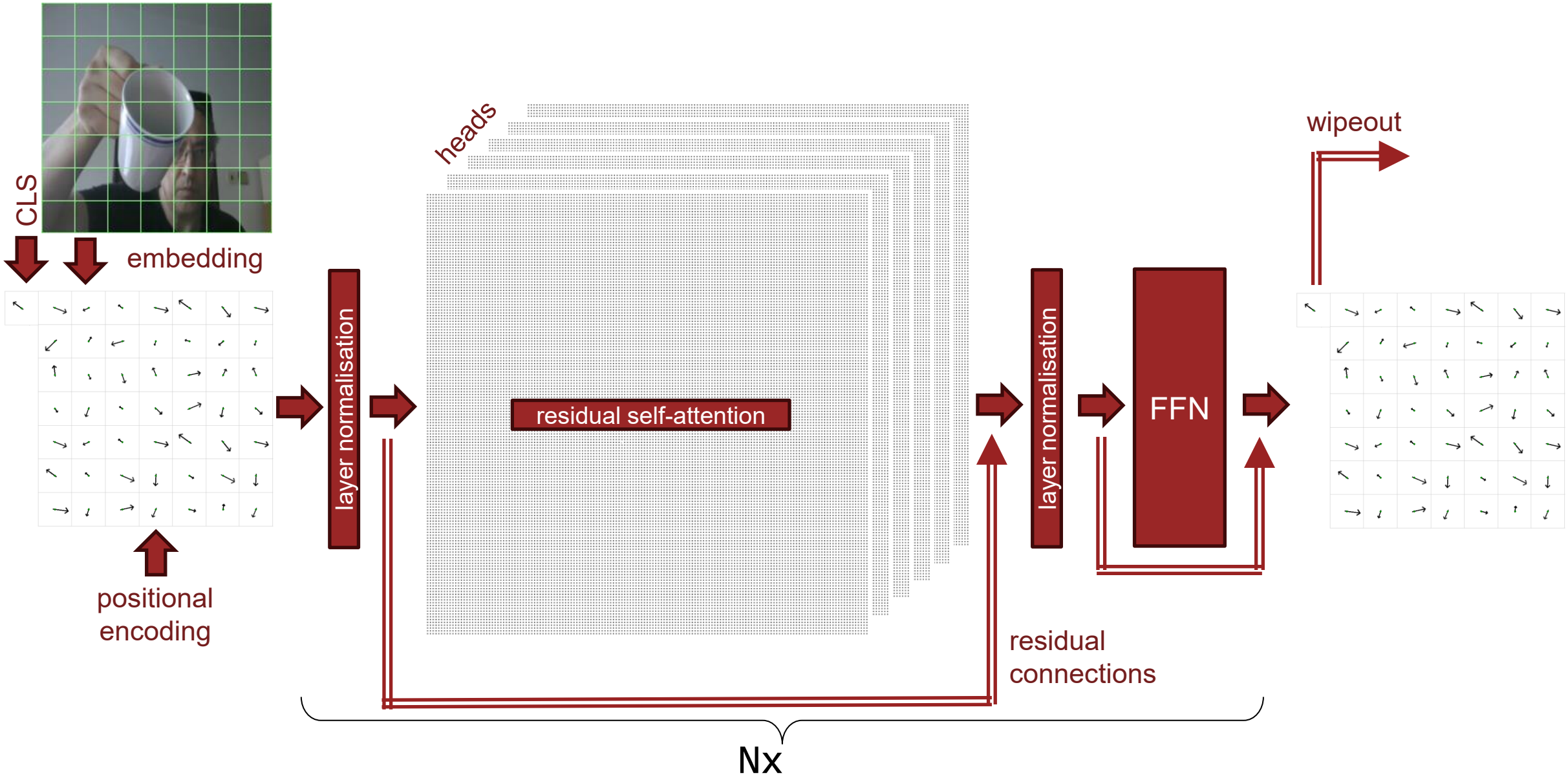
The need for CLS (Classification State)



- So far, the hidden states form a map of the size 7×7×d
- But from which state do we wipe out?



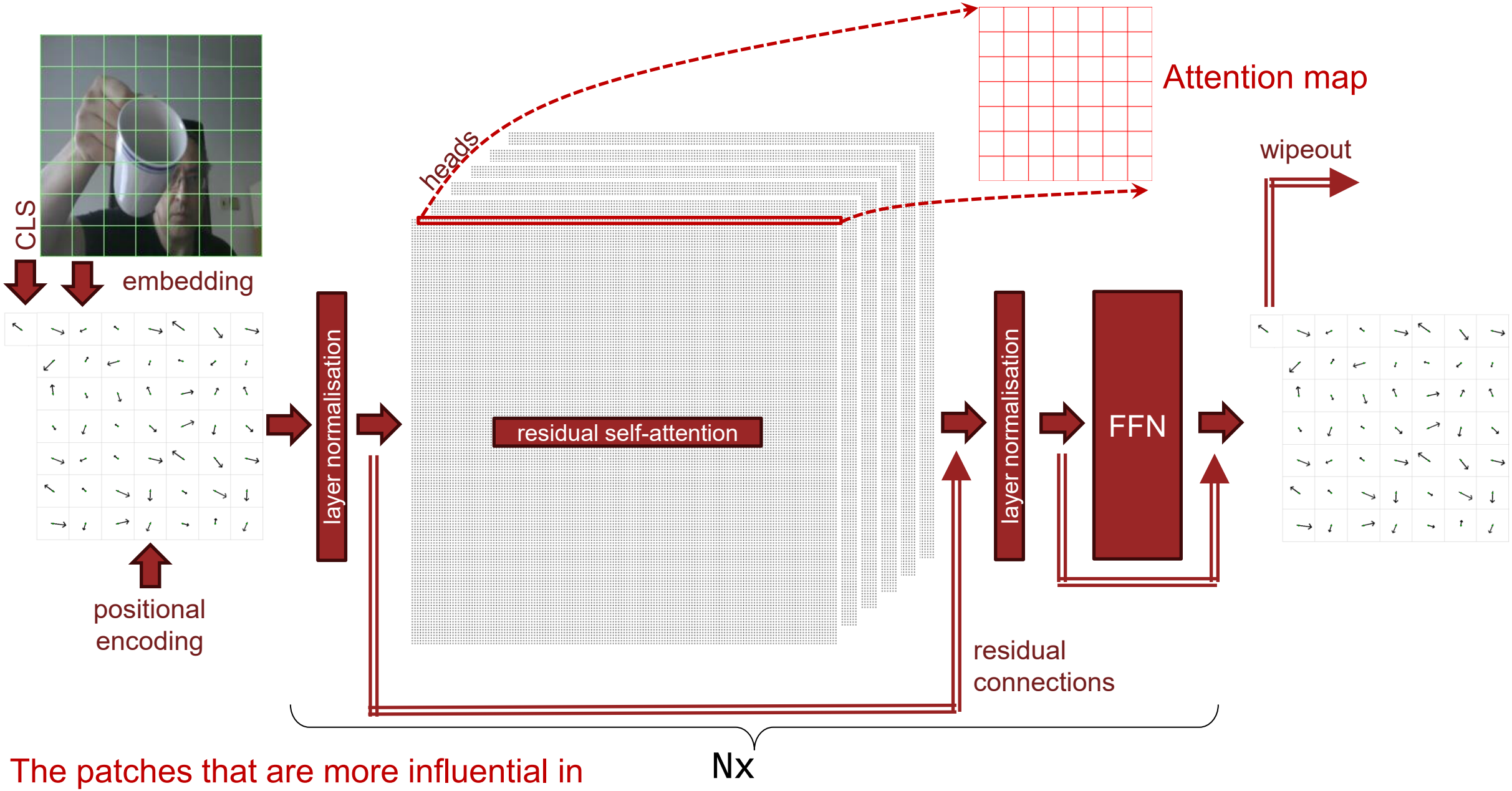
- From one of them? From all of them?
- The cleanest way is to add one more hidden state to capture the global meaning.
- We let the system learn its initial value, and wipe out the final one.



Vision Transformer

What makes ViT special?

- ViT is a classifier, but not only that
- ViT is also a detector that provides a mask of the objects seen
- All previous object detectors relied on brute-force methods, training a classifier and running it over many different regions of the image.
- **ViT does it differently. When it knows what it sees, it also knows where it sees it.**



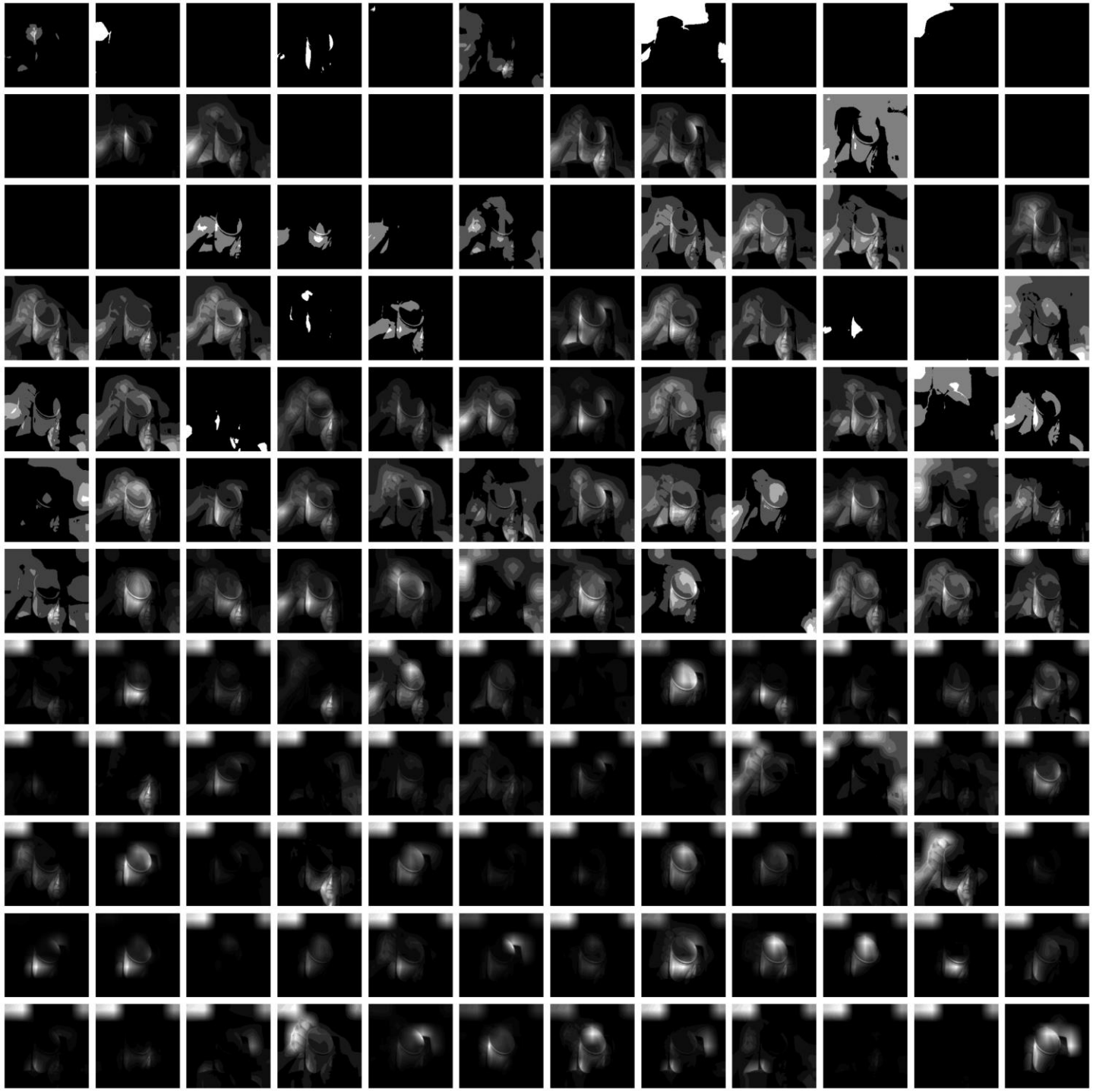
The patches that are more influential in classification are those on which an object is located.

Attention maps

12 heads

12 layer

Attention maps



Why are the best maps in the middle?