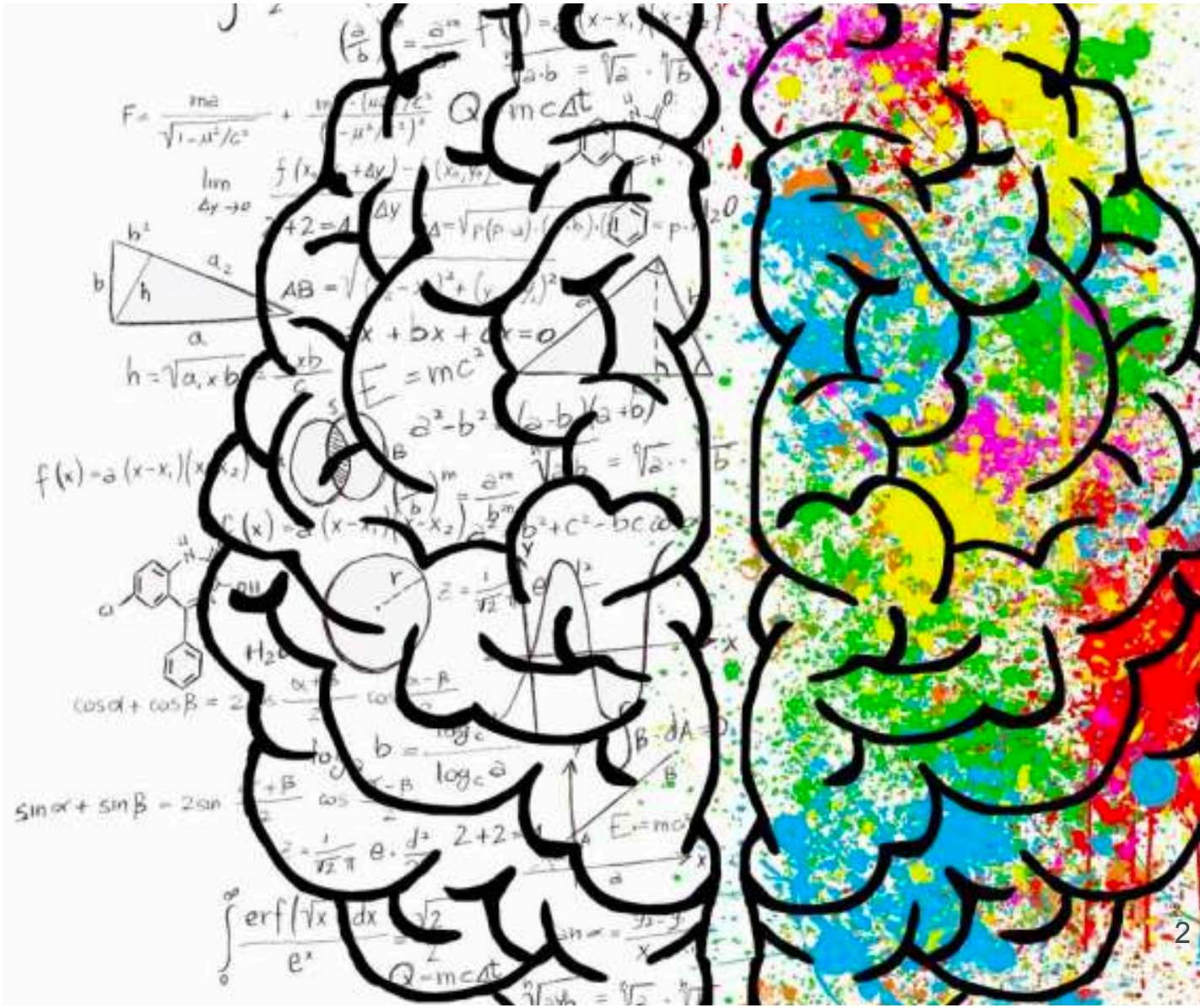
Introduction to Deep Learning
Attention and Transformer

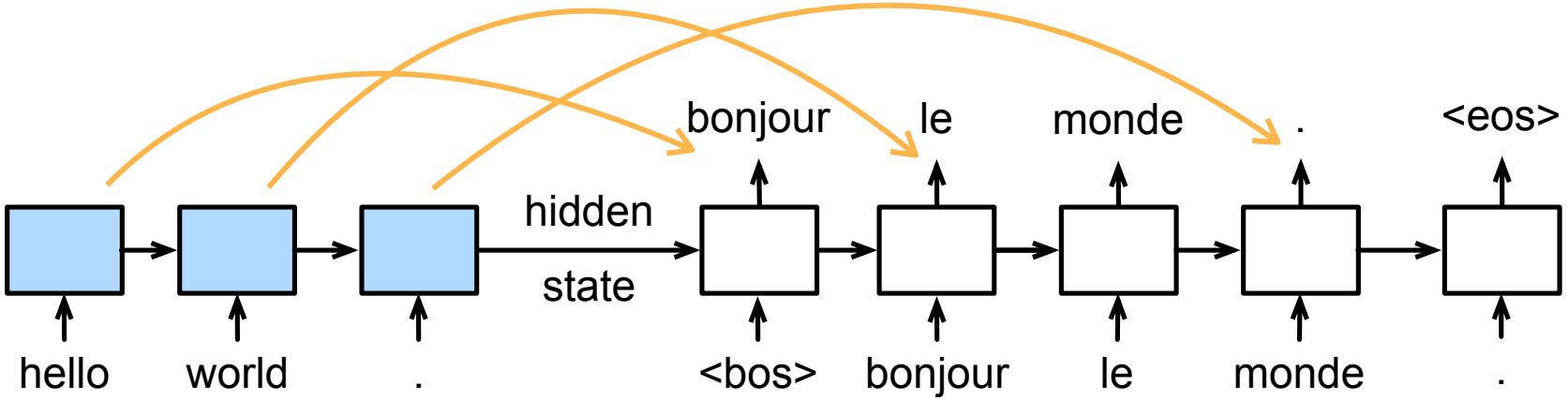
Slides Courtesy of Alex Smola and Mu Li

Attention



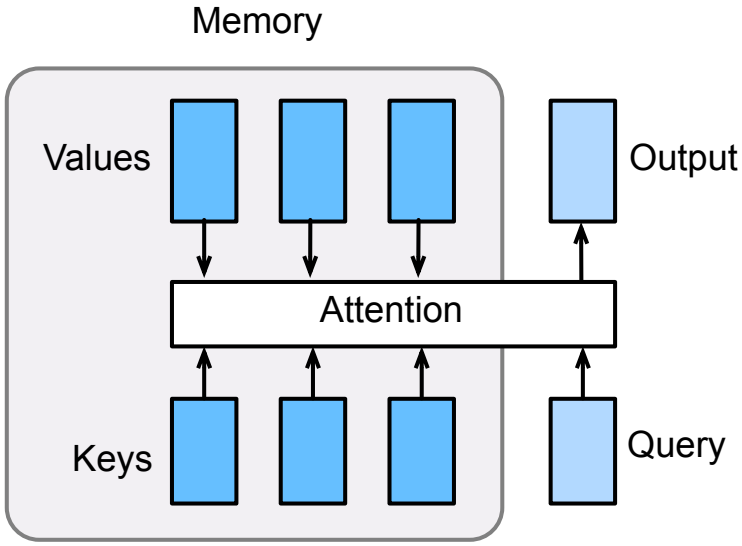
Motivation

- Each generated token might be related to different source tokens



Attention Layer

- An attention layer explicitly select related information
  - Its memory consists of key-value pairs
  - The output will be close to the values whose keys are similar to the query



  - Direct DB operation is not differentiable

Attention Layer

- Assume query $\mathbf{q} \in \mathbb{R}^{d_q}$ and the memory $(\mathbf{k}_1, \mathbf{v}_1), \ldots, (\mathbf{k}_n, \mathbf{v}_n)$ with $\mathbf{k}_i \in \mathbb{R}^{d_k}$ $\mathbf{v}_i \in \mathbb{R}^{d_v}$
- Compute n scores $a_1, \ldots, a_n$ with $a_i = \alpha(\mathbf{q}, \mathbf{k}_i)$
- Use softmax to obtain the attention weights

$$b_1, \ldots, b_n = \text{softmax}(a_1, \ldots, a_n)$$

- The output is a weighted sum of values

$$\mathbf{o} = \sum_{i=1}^{n} b_i \mathbf{v}_i$$

Vary α to get different attention layers, multi-head

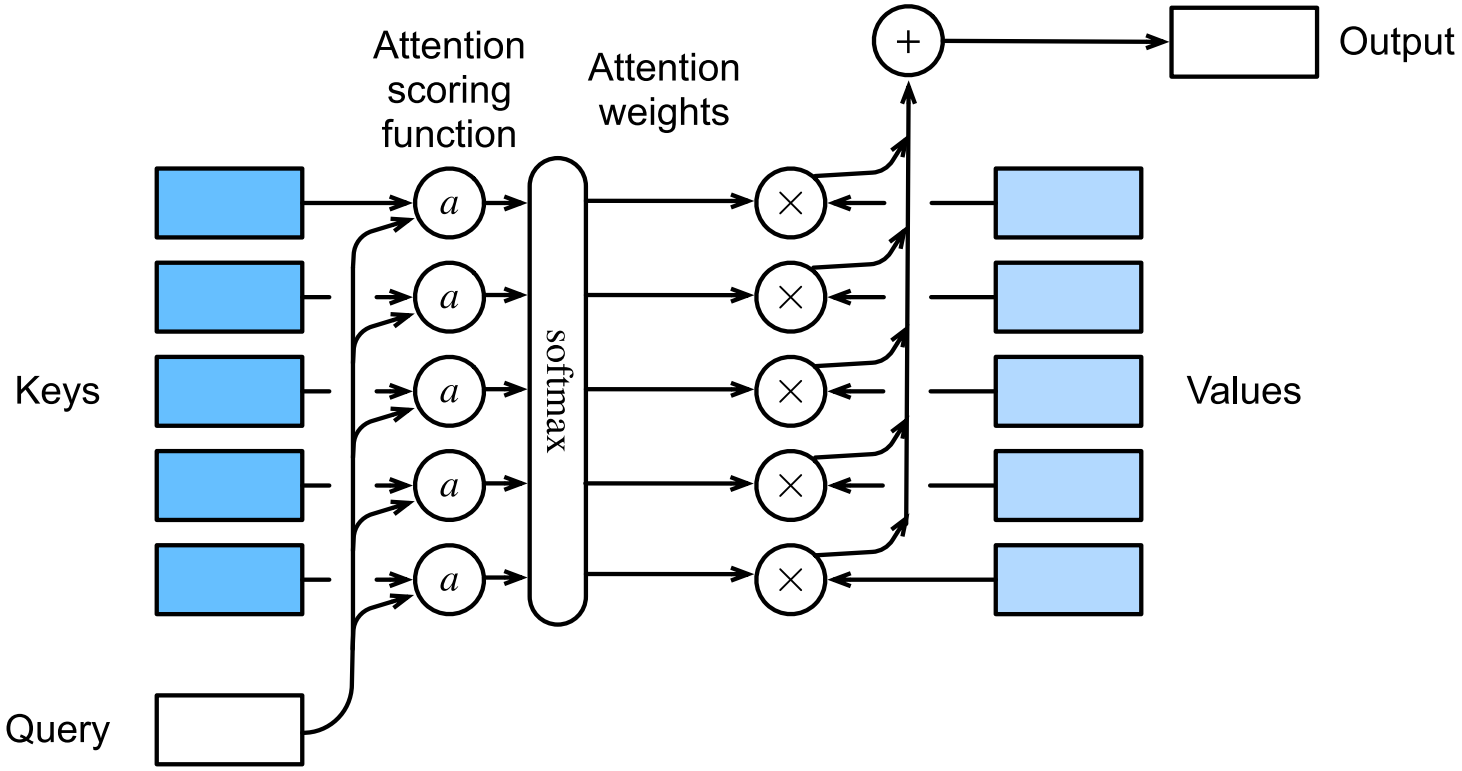
Dot Product Attention

- Assume the query has the same length as values $\mathbf{q}, \mathbf{k}_i \in \mathbb{R}^d$

$$\alpha(\mathbf{q}, \mathbf{k}) = \langle \mathbf{q}, \mathbf{k} \rangle / \sqrt{d}$$

- Vectorized version
  - *m* queries $\mathbf{Q} \in \mathbb{R}^{m \times d}$ and *n* keys $\mathbf{K} \in \mathbb{R}^{n \times d}$

$$\alpha(\mathbf{Q}, \mathbf{K}) = \mathbf{Q}\mathbf{K}^T / \sqrt{d}$$


Attention scoring function
Attention weights
Output
Keys
Values
Query
a
softmax
+

Multilayer Perception Attention

- Learnable parameters $\mathbf{W}_k \in \mathbb{R}^{h \times d_k}, \mathbf{W}_q \in \mathbb{R}^{h \times d_q},$ and $\mathbf{v} \in \mathbb{R}^h$

$$\alpha(\mathbf{k}, \mathbf{q}) = \mathbf{v}^T \tanh(\mathbf{W}_k \mathbf{k} + \mathbf{W}_q \mathbf{q})$$

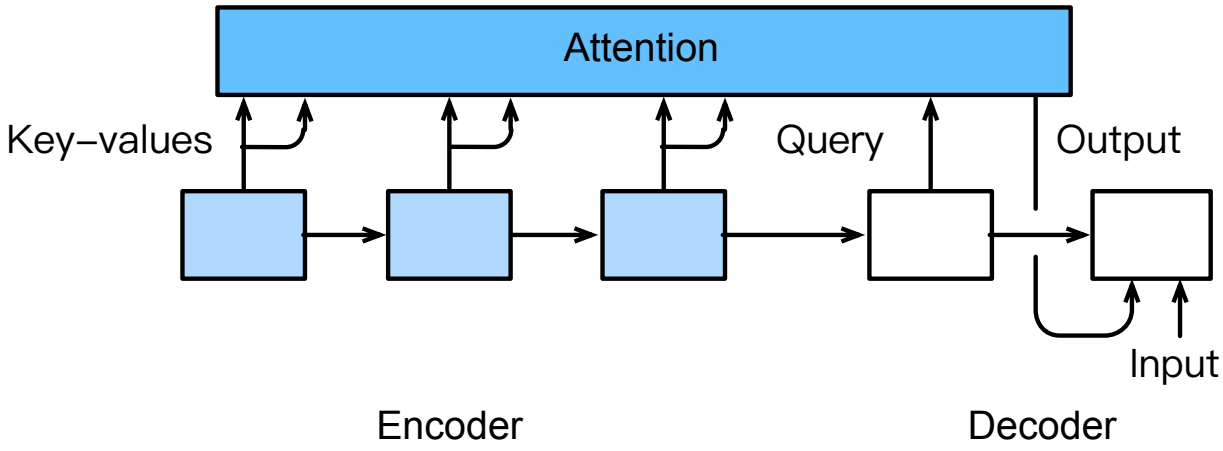
- Equals to concatenate the key and query, and then feed into a single hidden-layer perception with hidden size h and output size 1

Code…

Seq2seq with Attention

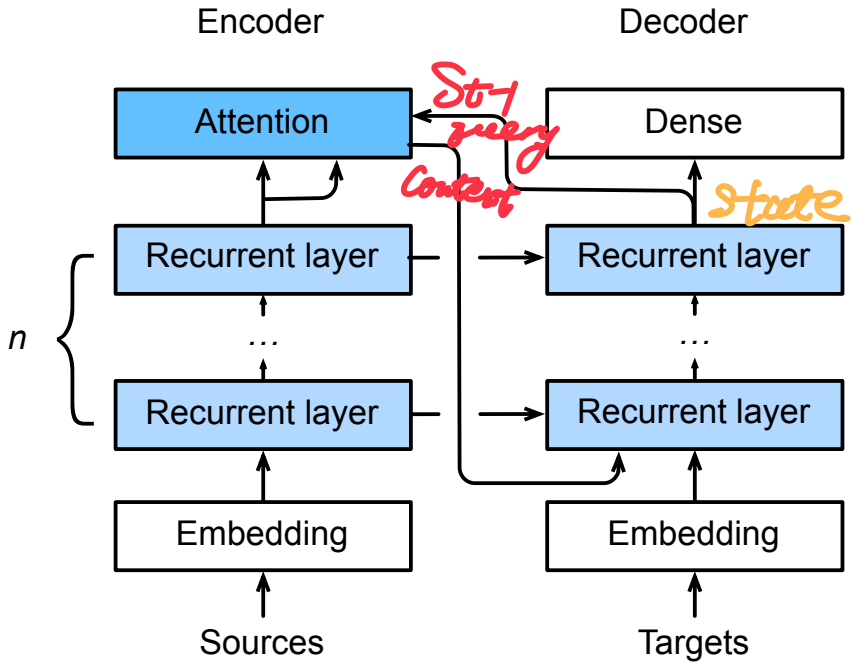
Model Architecture

- Each decode step might pay attention to particular encoder steps.
- Add an additional attention layer to use encoder's outputs as memory
- The attention output is used as the decoder's input



Encoder/Decoder Details

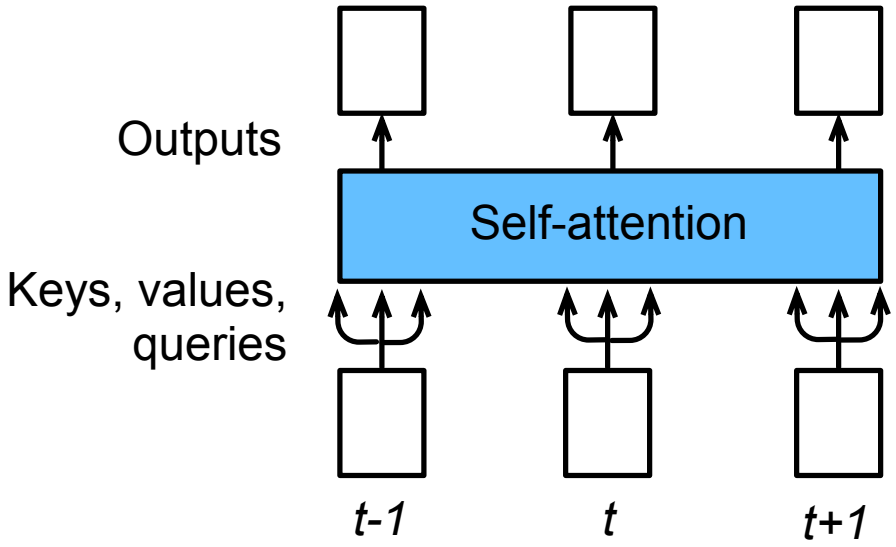
- We run through encoder, and save all states into Attention heads,
- Decoder h_{t-1} fetches attention
- The output of the last recurrent layer in the encoder is used
- The attention output is then concatenated with the embedding output to feed into the first recurrent layer in the decoder



Transformer

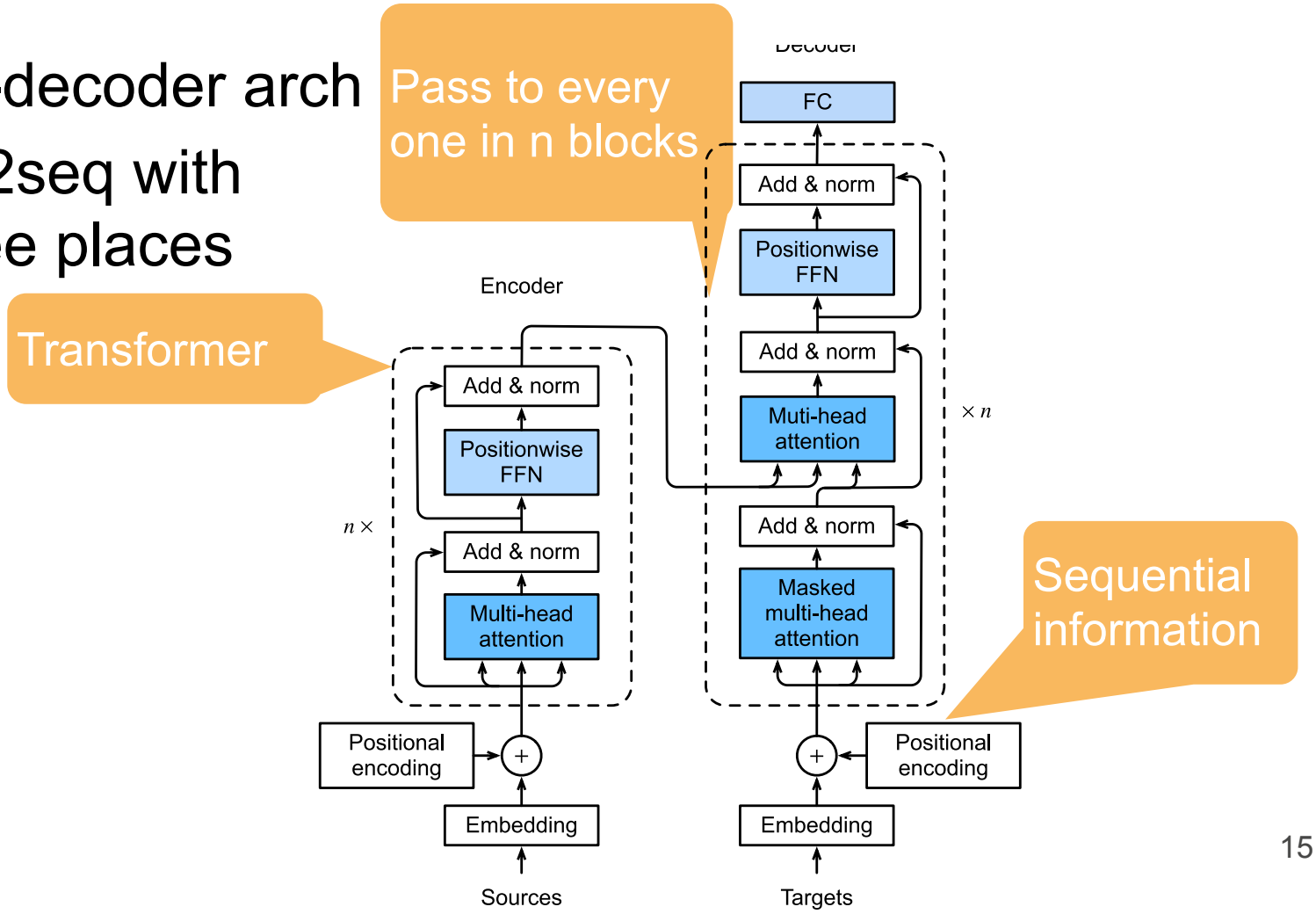
Self-attention

- To generate *n* outputs with *n* inputs, we can copy each input into a key, a value and a query
- No sequential information is preserved
- All outputs run in parallel (advantage)
- If you permute the input sequence, the output will be permuted, but values do not change.
- Must recover the sequential information.

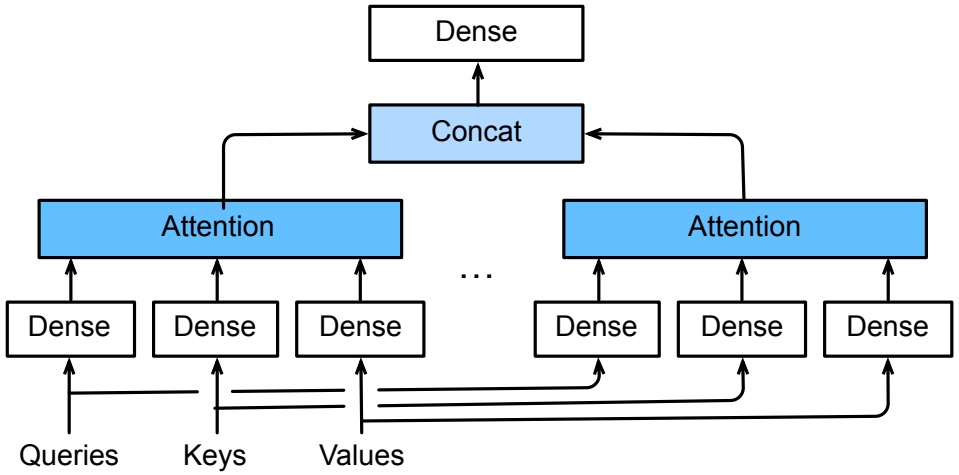


Transformer Architecture

- It's an encoder-decoder arch
- Differ from seq2seq with attention in three places



Multi-head Attention

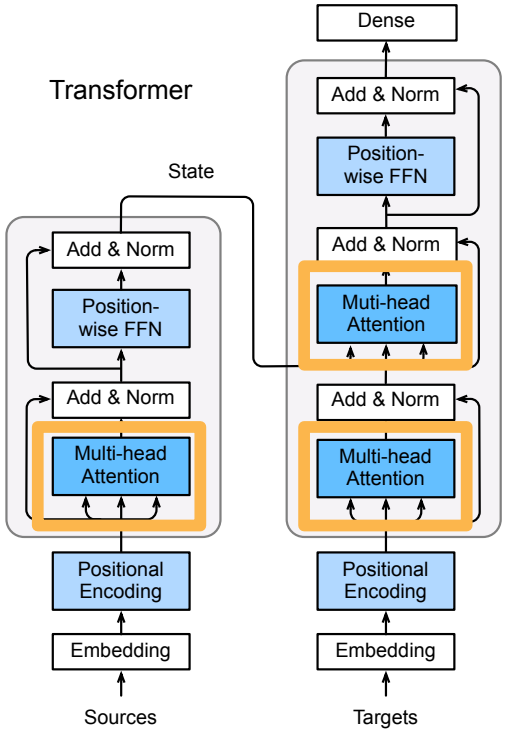


$$\mathbf{W}_q^{(i)} \in \mathbb{R}^{p_q \times d_q}, \mathbf{W}_k^{(i)} \in \mathbb{R}^{p_k \times d_k}, \text{ and } \mathbf{W}_v^{(i)} \in \mathbb{R}^{p_v \times d_v}$$

$$\mathbf{o}^{(i)} = \text{attention}(\mathbf{W}_q^{(i)}\mathbf{q}, \mathbf{W}_k^{(i)}\mathbf{k}, \mathbf{W}_v^{(i)}\mathbf{v})$$

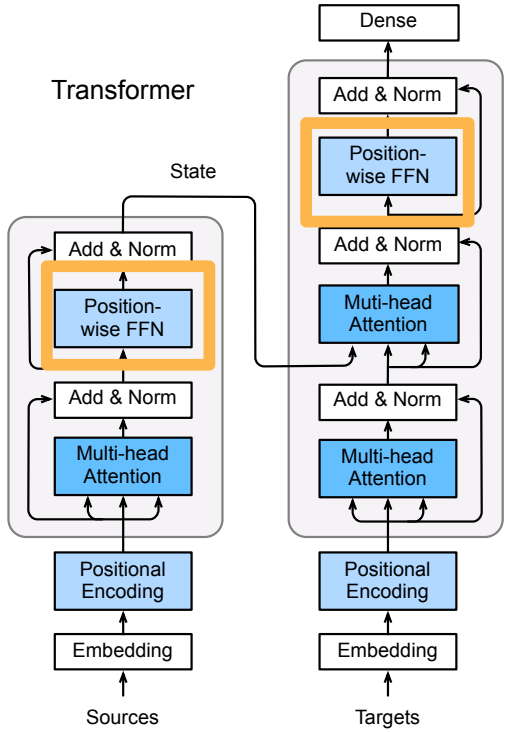
for $i = 1, \ldots, h$

$$\mathbf{o} = \mathbf{W}_o \begin{bmatrix} \mathbf{o}^{(1)} \\ \vdots \\ \mathbf{o}^{(h)} \end{bmatrix}$$



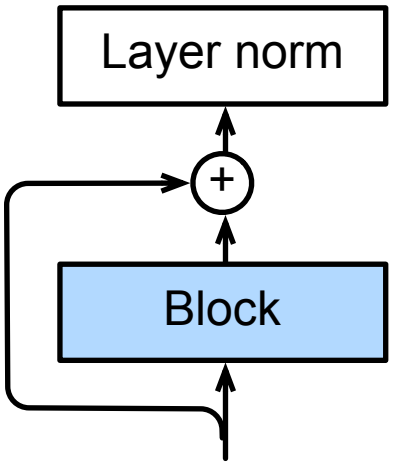
Position-wise Feed-Forward Networks

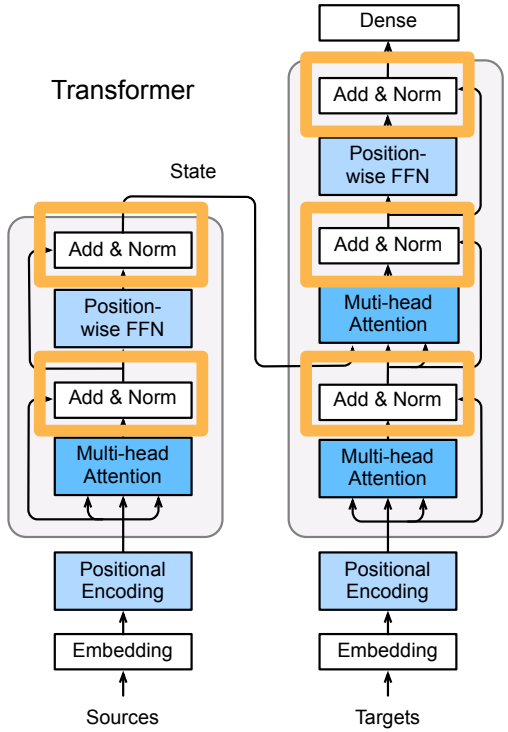
- Reshape input (batch, seq len, fea size) into (batch * seq len, fea size)
- Apply a two layer MLP
- Reshape back into 3-D
- Equals to apply two (1,1) conv layers



Add and Norm

- Layer norm is similar to batch norm
- But the mean and variances are calculated along the last dimension
- `X.mean(axis=-1)` instead of the first batch dimension in batch norm `X.mean(axis=0)`



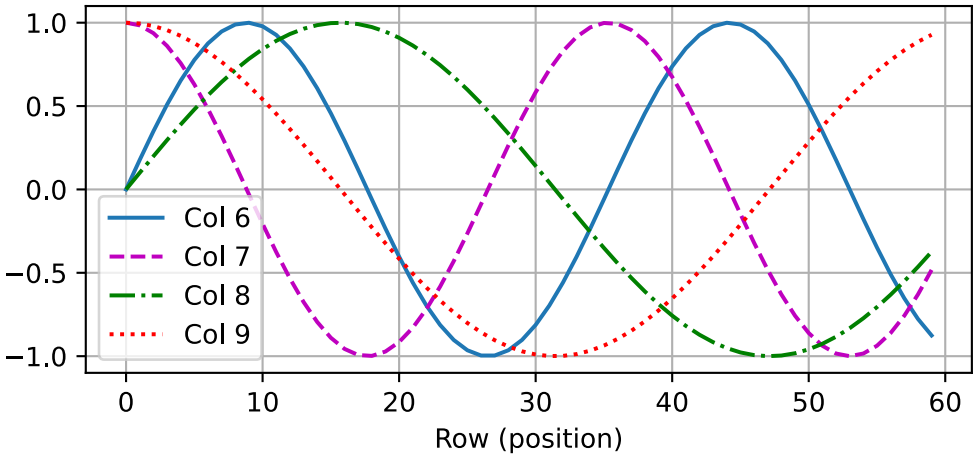


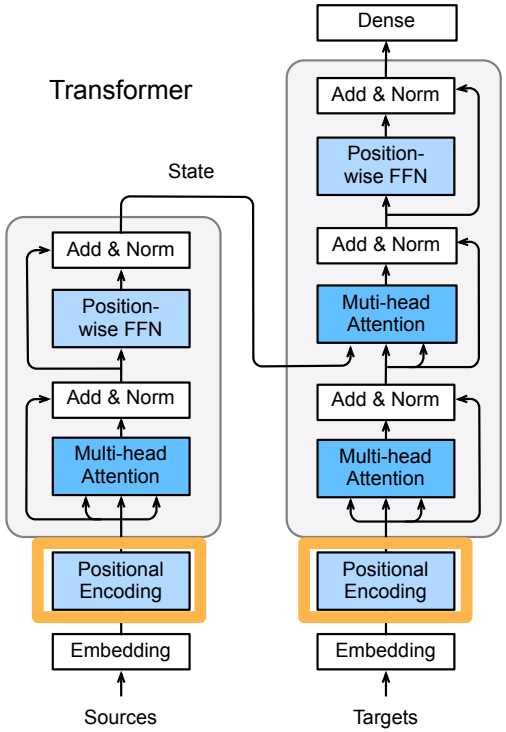
Positional Encoding

- Assume embedding output $X \in \mathbb{R}^{l \times d}$ with shape (seq len, embed dim)
- Create $P \in \mathbb{R}^{l \times d}$ with

$$P_{i,2j} = \sin\left(\frac{i}{10000^{\frac{2j}{d}}}\right) \qquad P_{i,2j+1} = \cos\left(\frac{i}{10000^{\frac{2j}{d}}}\right)$$

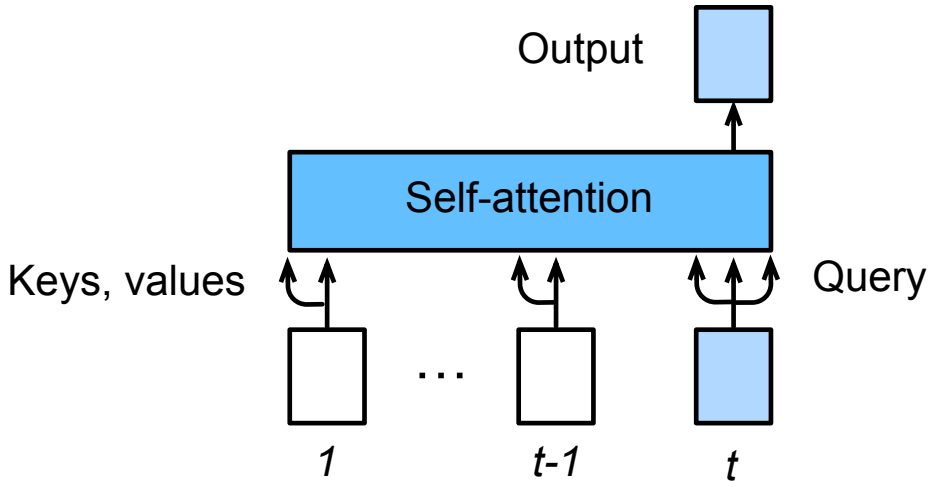
- Output $X + P$



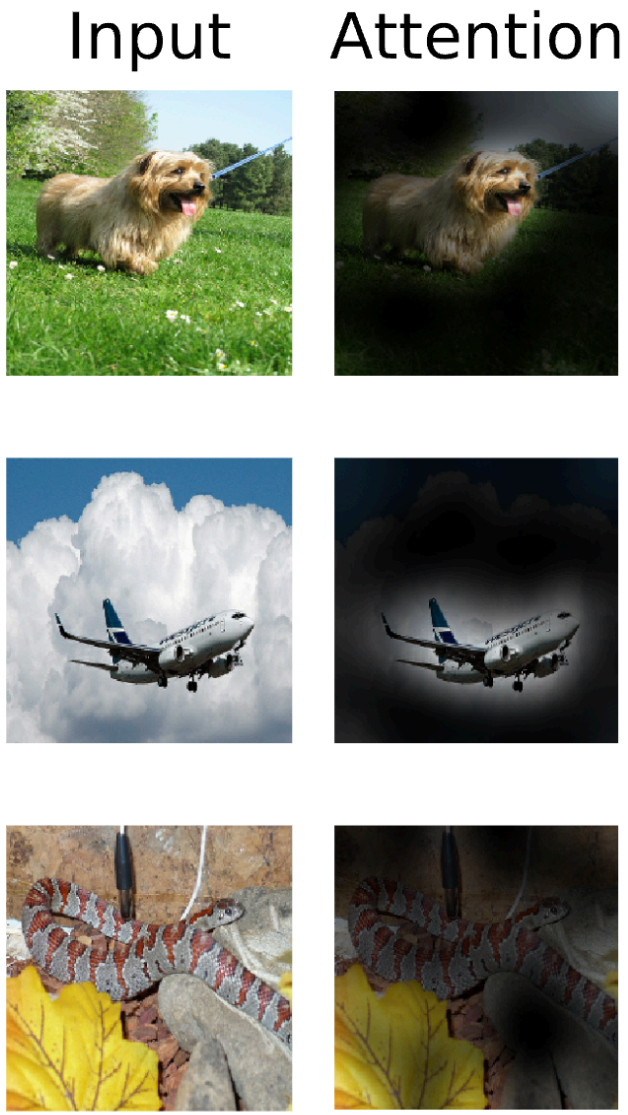


Predicting

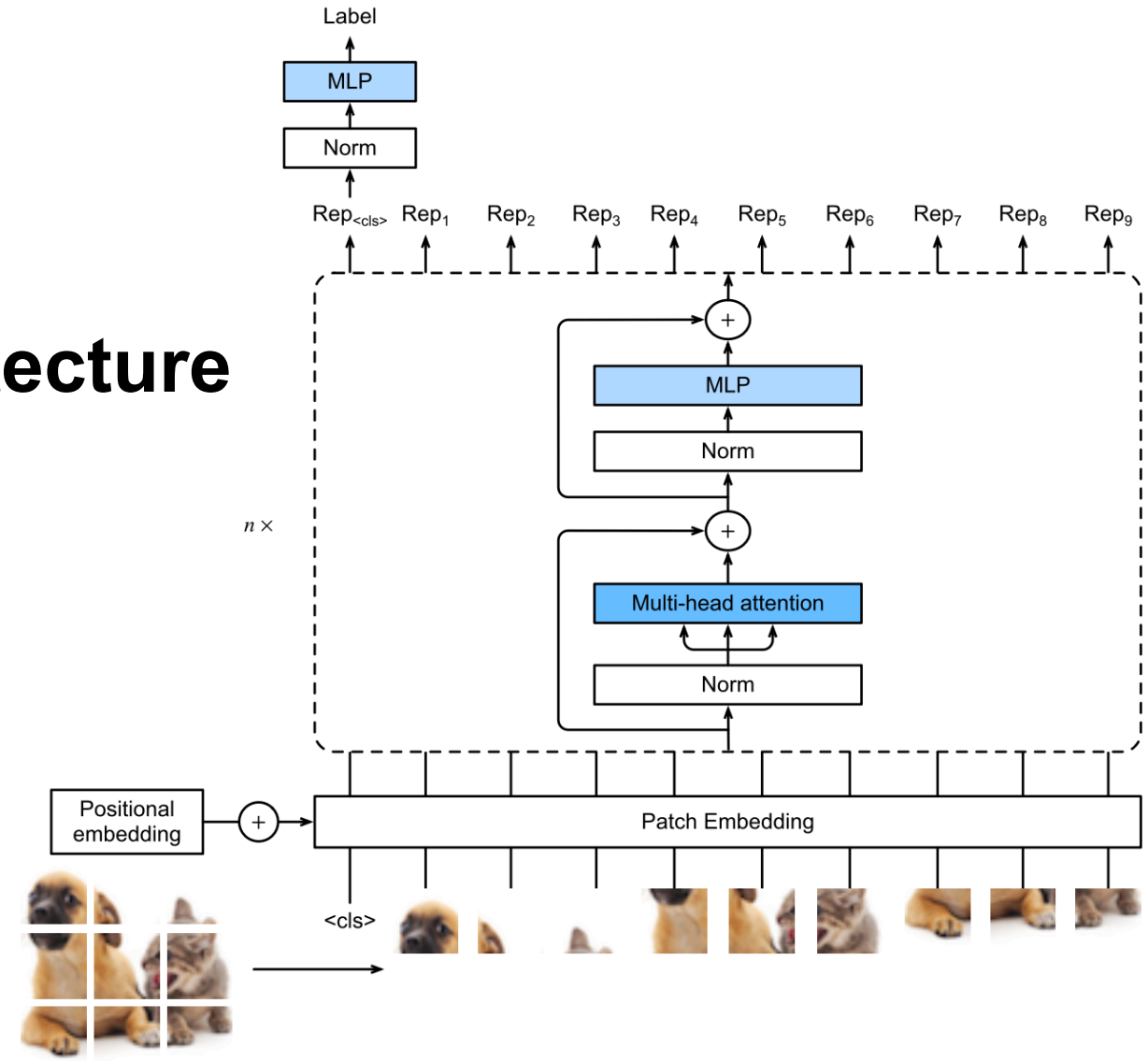
- Predict at time *t*:
  - Inputs of previous times as keys and values
  - Input at time *t* as query, as well as key and value, to predict output



Vision Transformer



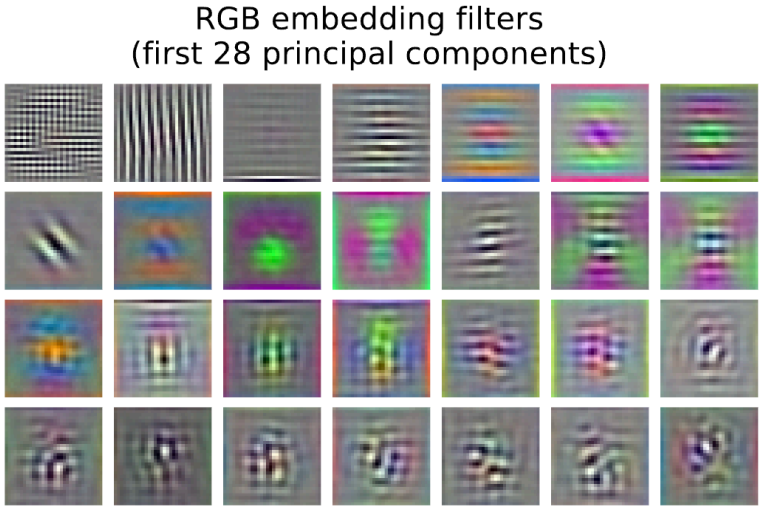
Model Architecture

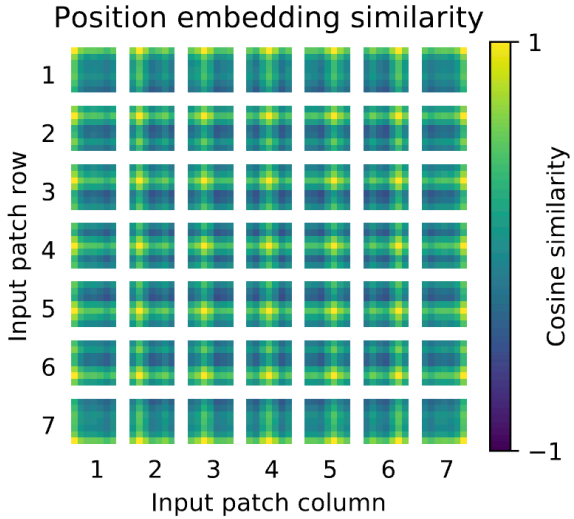


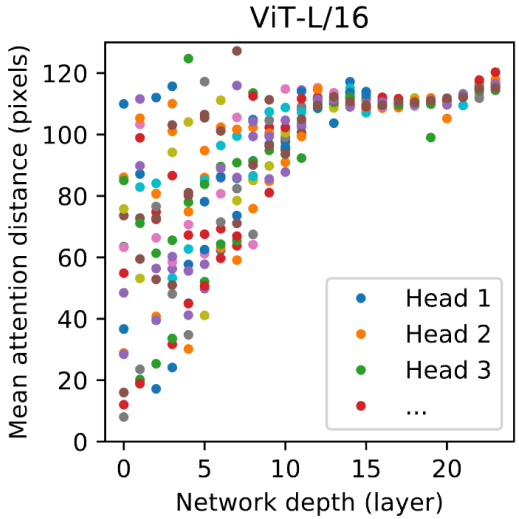
Patch Embedding

```
class PatchEmbedding(nn.Module):
    def __init__(self, img_size=96, patch_size=16, num_hiddens=512):
        super().__init__()
        def _make_tuple(x):
            if not isinstance(x, (list, tuple)):
                return (x, x)
            return x
        img_size, patch_size = _make_tuple(img_size), _make_tuple(patch_size)
        self.num_patches = (img_size[0] // patch_size[0]) * (
            img_size[1] // patch_size[1])
        self.conv = nn.LazyConv2d(num_hiddens, kernel_size=patch_size,
                                  stride=patch_size)

    def forward(self, X):
        # Output shape: (batch size, no. of patches, no. of channels)
        return self.conv(X).flatten(2).transpose(1, 2)
```


RGB embedding filters
(first 28 principal components)


Position embedding similarity
Input patch row
1
2
3
4
5
6
7
Input patch column
1
2
3
4
5
6
7
Cosine similarity
1
−1


ViT-L/16
Mean attention distance (pixels)
0
20
40
60
80
100
120
Network depth (layer)
0
5
10
15
20
Head 1
Head 2
Head 3
...

Experiment Results

		ViT-B/16	ViT-B/32	ViT-L/16	ViT-L/32	ViT-H/14
ImageNet	CIFAR-10	98.13	97.77	97.86	97.94	-
	CIFAR-100	87.13	86.31	86.35	87.07	-
	ImageNet	77.91	73.38	76.53	71.16	-
	ImageNet ReaL	83.57	79.56	82.19	77.83	-
	Oxford Flowers-102	89.49	85.43	89.66	86.36	-
	Oxford-IIIT-Pets	93.81	92.04	93.64	91.35	-
ImageNet-21k	CIFAR-10	98.95	98.79	99.16	99.13	99.27
	CIFAR-100	91.67	91.97	93.44	93.04	93.82
	ImageNet	83.97	81.28	85.15	80.99	85.13
	ImageNet ReaL	88.35	86.63	88.40	85.65	88.70
	Oxford Flowers-102	99.38	99.11	99.61	99.19	99.51
	Oxford-IIIT-Pets	94.43	93.02	94.73	93.09	94.82
JFT-300M	CIFAR-10	99.00	98.61	99.38	99.19	99.50
	CIFAR-100	91.87	90.49	94.04	92.52	94.55
	ImageNet	84.15	80.73	87.12	84.37	88.04
	ImageNet ReaL	88.85	86.27	89.99	88.28	90.33
	Oxford Flowers-102	99.56	99.27	99.56	99.45	99.68
	Oxford-IIIT-Pets	95.80	93.40	97.11	95.83	97.56

Table 5: Top1 accuracy (in %) of Vision Transformer on various datasets when pre-trained on ImageNet, ImageNet-21k or JFT300M. These values correspond to Figure 3 in the main text. Models are fine-tuned at 384 resolution. Note that the ImageNet results are computed without additional techniques (Polyak averaging and 512 resolution images) used to achieve results in Table 2.

ViT-B (base model) -L (large model), 16 v.s. 32. Patch size.