Transformers

CNNs

What are CNNs really good at?

What assumptions do CNNs make about images?

CNNs

Can you think of a task where understanding the entire image at once is crucial?

How would CNNs handle that?

CNNs

If you wanted every part of the image to be aware of every other part, how would you design such a mechanism?

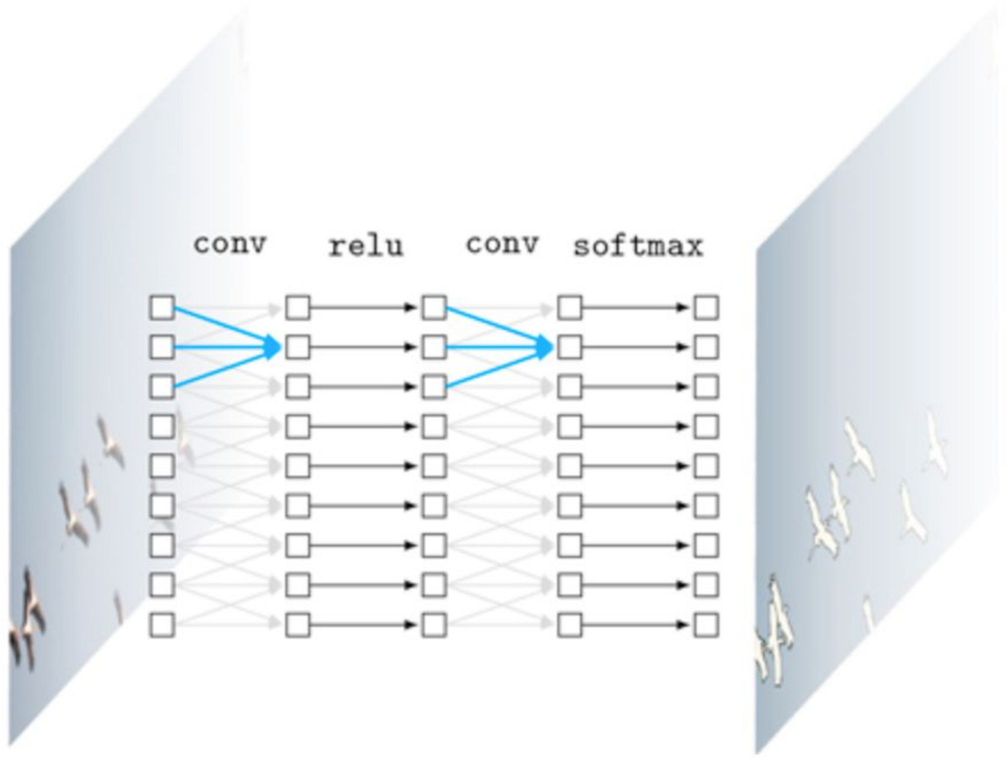


Figure 24.27: A 1D slice of a CNN that maps an image to an image. The input is the photo of size $3 \times N \times M$ and the output is a class probability map of size $K \times N \times M$; we visualize the corresponding label map on the right (per-pixel argmax over output probabilities). The blue arrows are the learnable parameters. The gray arrows share the weights of the blue arrows.

From RNNs to Transformers

Limitations of RNNs

1. Sequential Processing:
   a. RNNs process data sequentially, meaning they handle one element of a sequence at a time. This makes them inherently slow and inefficient, especially for long sequences.
   b. This sequential nature also hinders parallelization, limiting their ability to leverage the power of modern GPUs.
2. Vanishing/Exploding Gradients:
   a. RNNs struggle to capture long-range dependencies due to the vanishing or exploding gradient problem.
   b. As information propagates through many time steps, gradients can become too small or too large, making it difficult for the model to learn relationships between distant elements.
3. Limited Contextual Understanding:
   a. Even with improvements like LSTMs and GRUs, RNNs still have limitations in capturing long-range contextual information. They tend to focus more on recent inputs, making it challenging to understand complex dependencies in long sequences.

Transformers

1. Convolutional layers are not well-suited to globalizing information
2. Since the only way they can do so is by either **increasing the kernel size** of their filters or **stacking layers to increase the receptive field** of neurons on deeper layers.
3. Fully connected layers: They have tons of parameters, and it can take an exorbitant amount of time and data to fit all those parameters.

Look back selectively

1. When we look at a scene, our eyes flick around and we attend to certain elements that stand out, rather than taking in the whole scene at once.
2. A strategy for processing global information efficiently, focusing just on the parts of the signal that are most salient to the task at hand.

Look back selectively

1. In neural nets, attention follows the same intuitive idea. A set of neurons may attend to a set of neurons on the previous layer, in order to decide what their response should be.
2. If we “ask” that set of neurons to report the color of the car in the input image, then they should direct their attention to the neurons on the previous layer that represent the color of the car.

Pixels, Patches or Tokens?

1. Grouping of neurons: channels, tensors, batches …
2. Tokens a specific grouping of neurons
   a. Encapsulated information
   b. Special operators for accessing and modifying the internal state of tokens
3. One representation of tokens is a vector of neurons
4. Other representations might be
   a. Tensor-valued tokens
   b. Other richer token types e.g. in domains of 3D, physics, graphs
   c. E.g. to represent location, direction, surface norm, velocity, rotation

Pixels, Patches or Tokens?

1. To tokenize an image, we may simply represent each patch of pixels in the image with a token.
2. The token vector is the vectorized patch (stacking the three color channels one after the other),
3. or a lower-dimensional projection of the vectorized patch.

Pixels, Patches or Tokens?

1. Divide the image into non-overlapping square patches (e.g., 16×16).
2. Instead of non-overlapping patches, use sliding windows with overlaps.
3. Use a few convolutional layers (like in CNNs) to produce feature maps
4. Learns where to look or what tokens to extract.
5. Semantic regions as tokens
6. Tokenize at multiple scales (small patches → big patches)
7. Fourier / Frequency Domain Tokenization

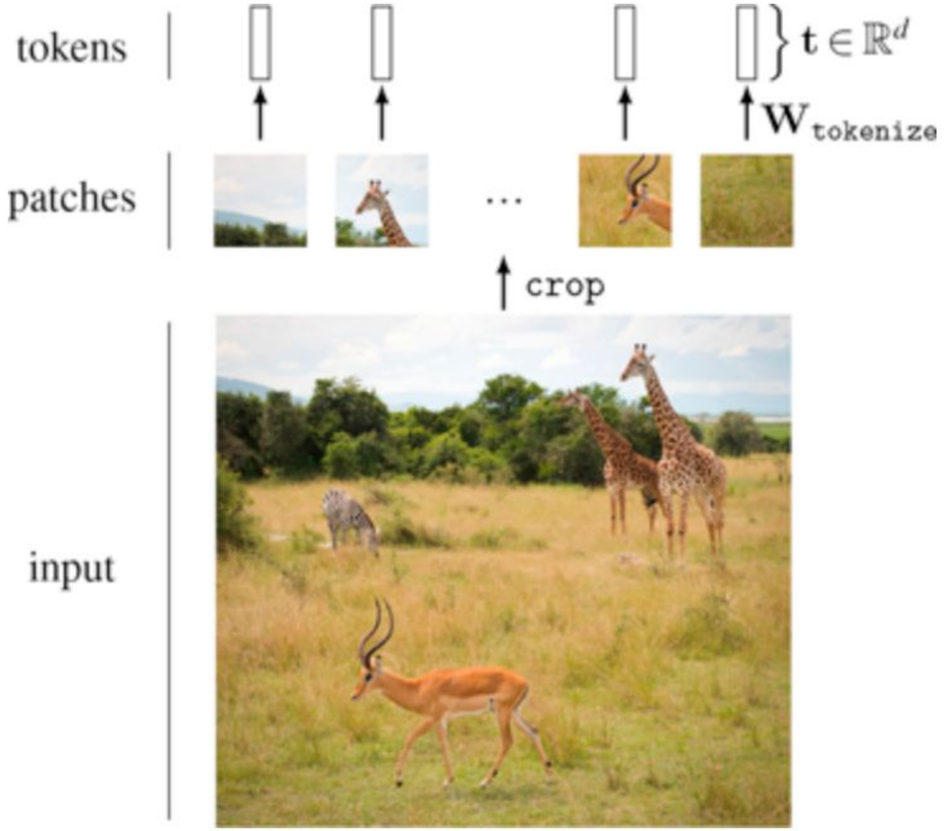

tokens
$\mathbf{t} \in \mathbb{R}^d$
$\mathbf{W}_{\texttt{tokenize}}$
patches
...
crop
input

Figure 26.2: Tokenization: converting an image to a set of vectors. $\mathbf{W}_{\text{tokenize}}$ is a learnable linear projection from the dimensionality of the vectorized crops to d dimensions. This is just one of many possible ways to tokenize an image.

Operations over tokens

1. Transformers consist of two main operations over tokens:
   a. (1) mixing tokens via a weighted sum, and
   b. (2) modifying each individual token via a nonlinear transformation.
   c. Analogous the linear layer and the pointwise nonlinearity in neural nets.
2. Tokens are to transformers as neurons are to neural nets.

Mixing Tokens

1. Linear combination of tokens is a weighted sum of vector-valued tokens.

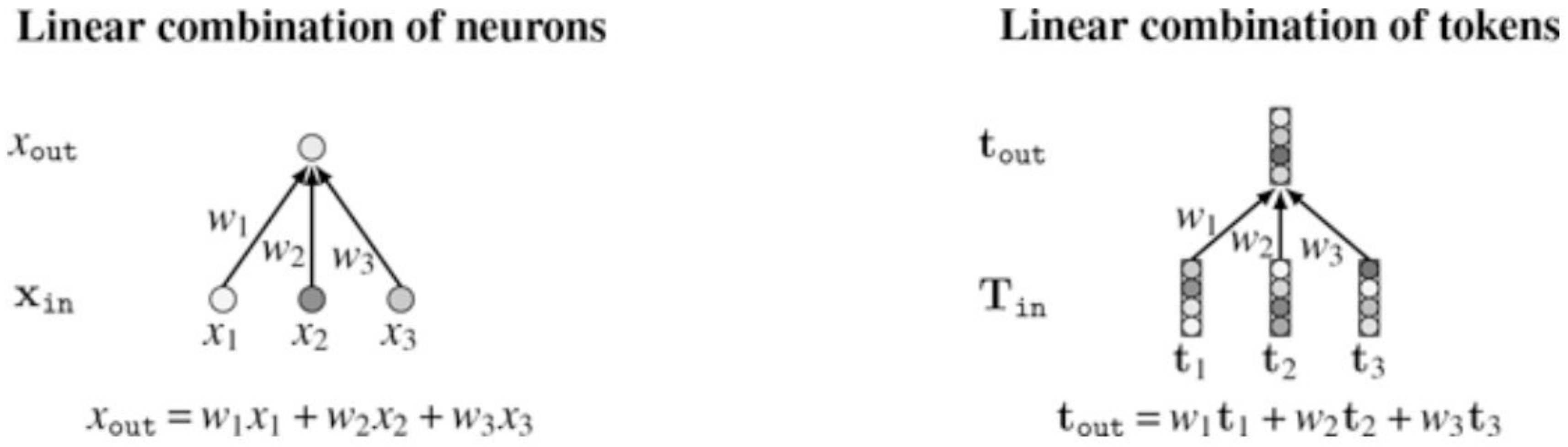

Linear combination of neurons
x_{out}
$\mathbf{x}_{\text{in}}$
w_1
w_2
w_3
x_1
x_2
x_3
$x_{\text{out}} = w_1 x_1 + w_2 x_2 + w_3 x_3$
Linear combination of tokens
$\mathbf{t}_{\text{out}}$
$\mathbf{T}_{\text{in}}$
w_1
w_2
w_3
$\mathbf{t}_1$
$\mathbf{t}_2$
$\mathbf{t}_3$
$\mathbf{t}_{\text{out}} = w_1 \mathbf{t}_1 + w_2 \mathbf{t}_2 + w_3 \mathbf{t}_3$

Figure 26.4: Linear combination of neurons versus tokens.

Modifying Tokens

1. Per-neuron vs per-token nonlinearity

$$\mathbf{x}_{\text{out}} = \begin{bmatrix} \texttt{relu}(x_{\texttt{in}}[0]) \\ \vdots \\ \texttt{relu}(x_{\texttt{in}}[N-1]) \end{bmatrix} \qquad \triangleleft \quad \text{per-neuron nonlinearity } (\texttt{relu})$$

$$\mathbf{T}_{\text{out}} = \begin{bmatrix} F_{\theta}(\mathbf{T}_{\texttt{in}}[0,:]) \\ \vdots \\ F_{\theta}(\mathbf{T}_{\texttt{in}}[N-1,:]) \end{bmatrix} \qquad \triangleleft \quad \text{per-token nonlinearity}$$

Modifying Tokens

$$\mathbf{T}_{\mathtt{out}} = \begin{bmatrix} F_\theta(\mathbf{T}_{\mathtt{in}}[0,:]) \\ \vdots \\ F_\theta(\mathbf{T}_{\mathtt{in}}[N-1,:]) \end{bmatrix} \quad \triangleleft \quad \text{per-token nonlinearity}$$

1. The Fθ may be any nonlinear function but some choices will work better than others.
2. One popular choice is for Fθ to be a multilayer perceptron (MLP). In this case,
   a. Fθ has learnable parameters θ, which are the weights and biases of the MLP.
   b. relus, and most other neuron-wise nonlinearities, have no learnable parameters, whereas Fθ typically does

Mixing and Modifying Tokens

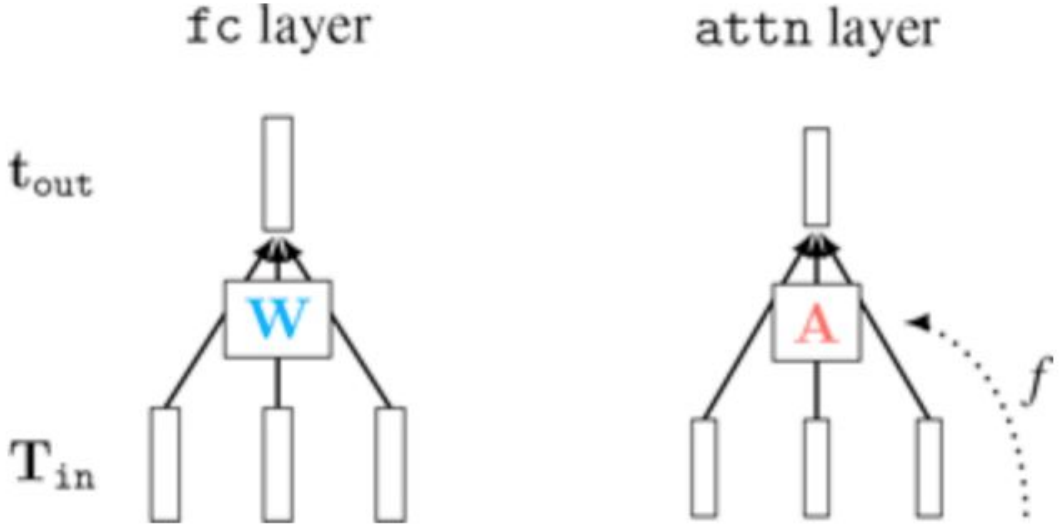
1. Mixing (linear): Who should I look at?
2. Modifying (non-linear): How should I transform what I have seen?

Analogy

3. Think of a Transformer block as a team meeting:
4. Mixing: People talk to each other and exchange ideas
5. MLP: Each person goes off to reflect and develop their own thoughts before the next meeting.
    a. Without that reflection (MLP), everyone just parrots what they heard.

The Attention Layer

1. Attention layers define a special kind of linear combination of tokens.
2. Rather than parameterizing the linear combination with a matrix of free parameters W, attention layers use a different matrix, which we call the attention matrix A.
3. The important difference between A and W is that A is data-dependent,
    a. that is, the values of A are a function the data input to the network.
    b. In addition, A typically only contains non-negative values, consistent with thinking of it as a matrix that allocates how much (non-negative) attention we pay to each input token.



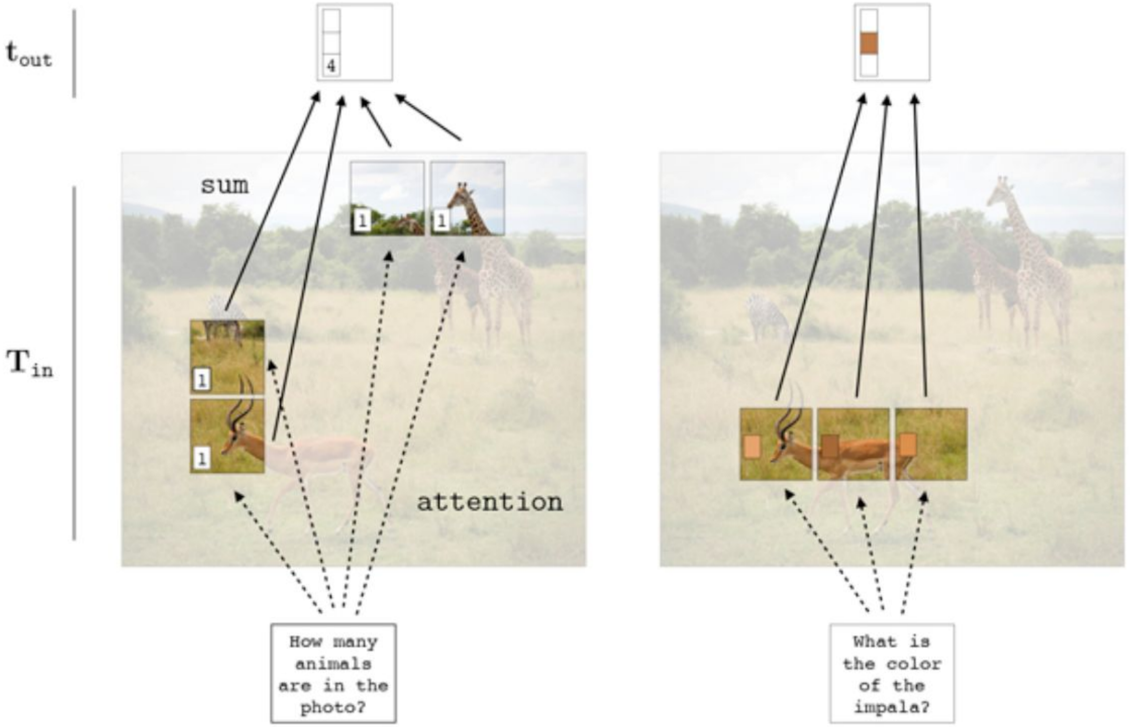
fc layer
attn layer
$\mathbf{t}_{\mathtt{out}}$
$\mathbf{T}_{\mathtt{in}}$
W
A
f

The Attention Layer

$$\mathbf{A} = f(\ldots) \qquad \triangleleft \text{ attention} \tag{26.9}$$

$$\mathbf{T}_{\texttt{out}} = \mathbf{A}\mathbf{T}_{\texttt{in}} \tag{26.10}$$

1. What exactly is f?
2. What inputs does f depend on and
3. what is f's mathematical form?
4. Let’s start with the intuition: f is a function that determines how much attention to apply to each token in T_{in}.
5. The f can depend on any number of input signals that tell the net what to pay attention to.



t_{out}
4
T_{in}
sum
1
1
1
1
attention
How many animals are in the photo?
What is the color of the impala?

Figure 26.7: How attention can be allocated across different regions (tokens) in an image. The token code vectors consist of multiple dimensions and each can encode a different attribute of the token. To the left we show a dimension that encodes number of animal heads. To the right we show a different dimension that encodes color (or this could be three dimensions, coding RGB). The output token is a weighted sum over all the tokens attended to.

Query-Key-Value Attention

1. Each token is associated with a **query vector**, a **key vector**, and a **value vector**.
2. We define these vectors as linear transformations of the token's code vector

$$\mathbf{q} = \mathbf{W}_q \mathbf{t} \qquad \triangleleft \text{ query}$$

$$\mathbf{k} = \mathbf{W}_k \mathbf{t} \qquad \triangleleft \text{ key}$$

$$\mathbf{v} = \mathbf{W}_v \mathbf{t} \qquad \triangleleft \text{ value}$$

Query-Key-Value Attention

1. In transformers, all inputs to the net are tokenized, so the textual question “How many animals are in the photo?” will also be represented as a token.
2. This token will submit its query vector, q_{question} to be matched against the keys of the tokens that represent different patches in the image
3. Querying each token in T_{in} in this way gives us a vector of similarities:

$$\mathbf{s} = [s_1, \ldots, s_N]^\mathsf{T} = [\mathbf{q}_{\texttt{question}}^\mathsf{T}\mathbf{k}_1, \ldots, \mathbf{q}_{\texttt{question}}^\mathsf{T}\mathbf{k}_N]^\mathsf{T}$$

Query-Key-Value Attention

1. Querying each token in T_{in} in this way gives us a vector of similarities:

$$\mathbf{s} = [s_1, \ldots, s_N]^\top = [\mathbf{q}_{\mathtt{question}}^\top \mathbf{k}_1, \ldots, \mathbf{q}_{\mathtt{question}}^\top \mathbf{k}_N]^\top$$

2. We then normalize the vector **s** using the softmax function to give us our attention weights **a** $\in \mathbb{R}^{N\times 1}$

$$\mathbf{a} = \mathtt{softmax}(\mathbf{s})$$

$$\mathbf{T}_{\mathtt{out}} = \begin{bmatrix} a_1 \mathbf{v}_1^\top \\ \vdots \\ a_N \mathbf{v}_N^\top \end{bmatrix}$$

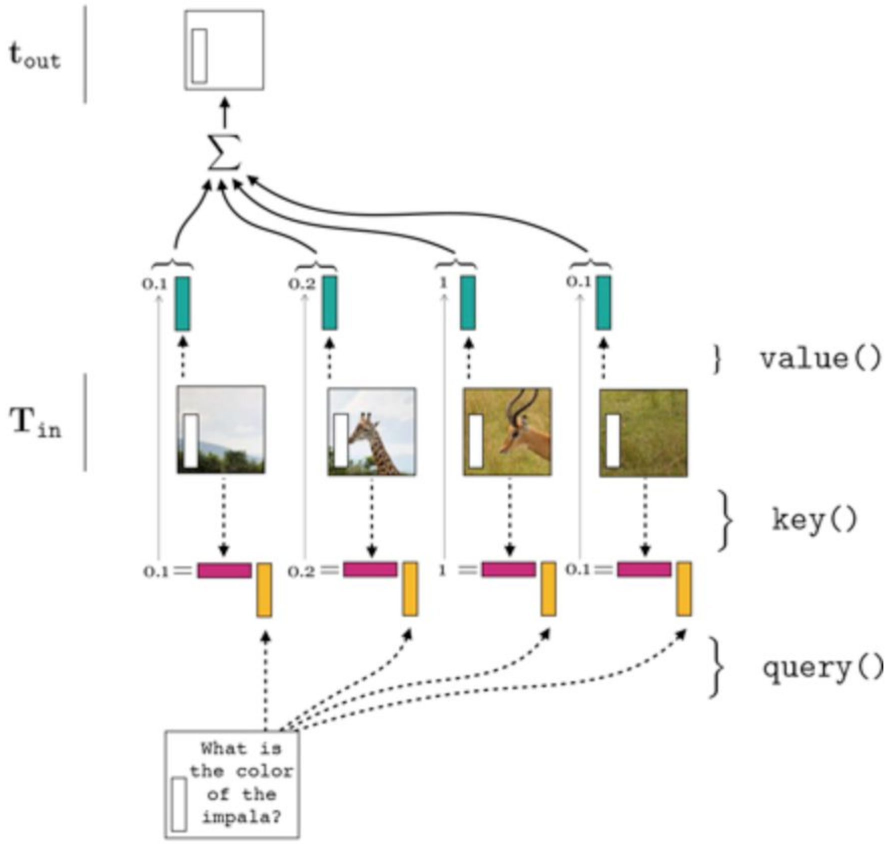


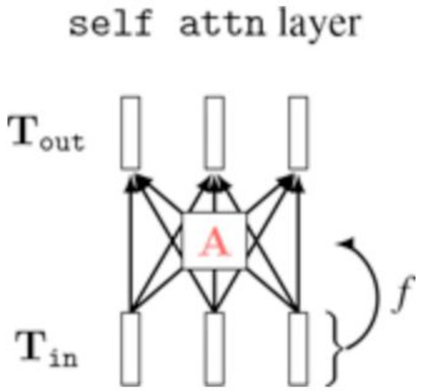
Figure 26.8: Mechanics of an attention layer. Queries from the question match keys from the tokens representing the impala; value vectors of the impala tokens then contribute the most to the sum that yields $\mathbf{t}_{\text{out}}$'s code vector. (Softmax omitted in this example.)

Self Attention

1. Attention is a general-purpose way of **dynamically pooling information** in one set of tokens based on queries from a different set of tokens
2. Previously we had hand designed queries. Can we make it general purpose where we don’t have to manually design queries?
3. Self-attention is such an architecture!

Self Attention

1. On a self-attention layer,
   a. all tokens submit queries,
   b. for each of these queries, we take a weighted sum over all tokens in that layer.
   c. If T_{in} is a set of N input tokens, then we have N queries, N weighted sums, and N output tokens to form T_{out}.



Self Attention

1. To compute the query, key, and value for a set of input tokens, T_{in}, we apply the same linear transformations to each token in the set, resulting in matrices Q_{in}, $K_{in} \in \mathbb{R}^{N\times m}$ and $V_{in} \in \mathbb{R}^{N\times d}$, where each row is the query/key/value for each token

$$\mathbf{Q}_{\mathtt{in}} = \begin{bmatrix}\mathbf{q}_1^\top \\ \vdots \\ \mathbf{q}_N^\top\end{bmatrix} = \begin{bmatrix}(\mathbf{W}_q\mathbf{t}_1)^\top \\ \vdots \\ (\mathbf{W}_q\mathbf{t}_N)^\top\end{bmatrix} = \mathbf{T}_{\mathtt{in}}\mathbf{W}_q^\top \qquad \triangleleft \quad \text{query matrix} \tag{26.17}$$

$$\mathbf{K}_{\mathtt{in}} = \begin{bmatrix}\mathbf{k}_1^\top \\ \vdots \\ \mathbf{k}_N^\top\end{bmatrix} = \begin{bmatrix}(\mathbf{W}_k\mathbf{t}_1)^\top \\ \vdots \\ (\mathbf{W}_k\mathbf{t}_N)^\top\end{bmatrix} = \mathbf{T}_{\mathtt{in}}\mathbf{W}_k^\top \qquad \triangleleft \quad \text{key matrix} \tag{26.18}$$

$$\mathbf{V}_{\mathtt{in}} = \begin{bmatrix}\mathbf{v}_1^\top \\ \vdots \\ \mathbf{v}_N^\top\end{bmatrix} = \begin{bmatrix}(\mathbf{W}_v\mathbf{t}_1)^\top \\ \vdots \\ (\mathbf{W}_v\mathbf{t}_N)^\top\end{bmatrix} = \mathbf{T}_{\mathtt{in}}\mathbf{W}_v^\top \qquad \triangleleft \quad \text{value matrix} \tag{26.19}$$

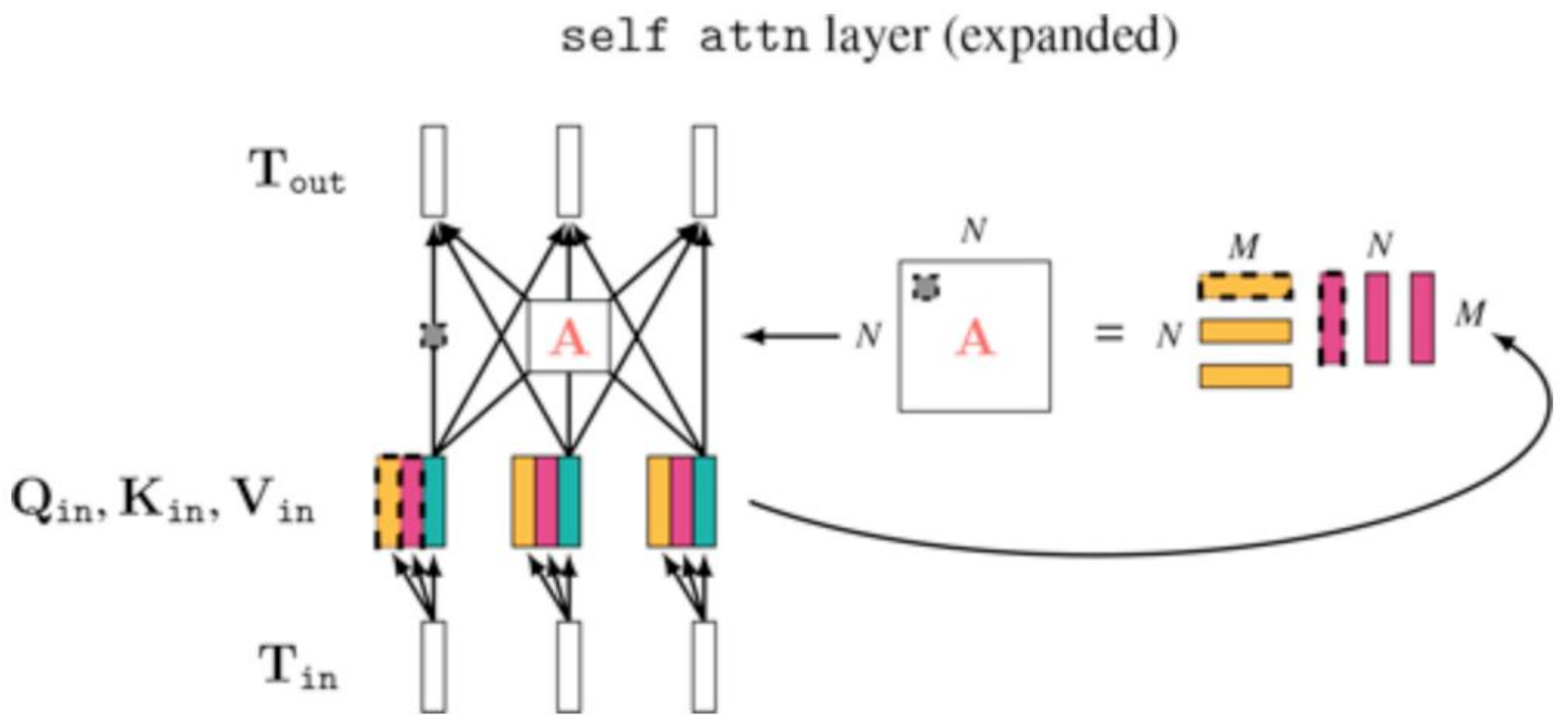

self attn layer (expanded)
$\mathbf{T}_{\texttt{out}}$
A
N
N
A
=
M
N
N
M
$\mathbf{Q}_{\texttt{in}}, \mathbf{K}_{\texttt{in}}, \mathbf{V}_{\texttt{in}}$
$\mathbf{T}_{\texttt{in}}$

Figure 26.10: Self-attention layer expanded. The nodes with the dashed outline correspond to each other; they represent one query being matched against one key to result in a scalar similarity value, in the gray box, which acts as a weight in the weighted sum computed by **A**.

Self Attention - Intuition

1. Consider our task is semantic segmentation
2. We have a token, that represents a patch of impala.
3. We wish to update this token via one layer of self-attention.
4. One way to do this would be to attend to the tokens representing other patches of the impala, and use them to refine it into a more abstracted token vector, capturing the label impala
5. The sum over the token code vectors has the effect of reducing noise that is not shared between the three attended impala patches, which amplifies the commonality between them – the label impala.

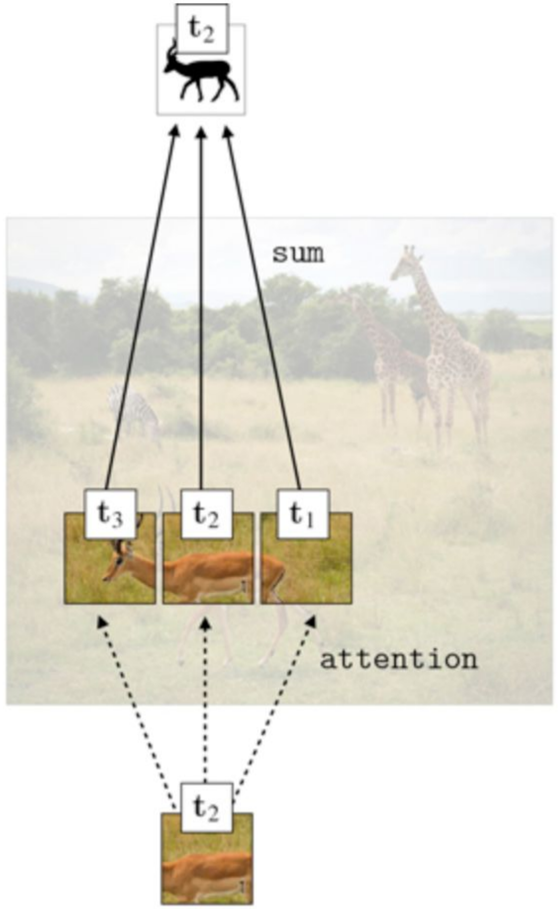


Figure 26.11: One way self-attention could be used to aggregate information across all patches containing the same object, and thereby arrive at a better representation of the object in $\mathbf{t}_2$, the query patch.

Self Attention - Intuition

1. More sophisticated refinements could be achieved via multiple layers of self-attention.
2. Further, the impala patch query could also retrieve information from the giraffe and zebra patches, as those patches provide additional context that could be informative
3. This is just one way self-attention could be used by the network. How it is actually used will be determined by the training data and task
4. Tokens on hidden layers do not necessarily represent spatially localized patches of pixels.

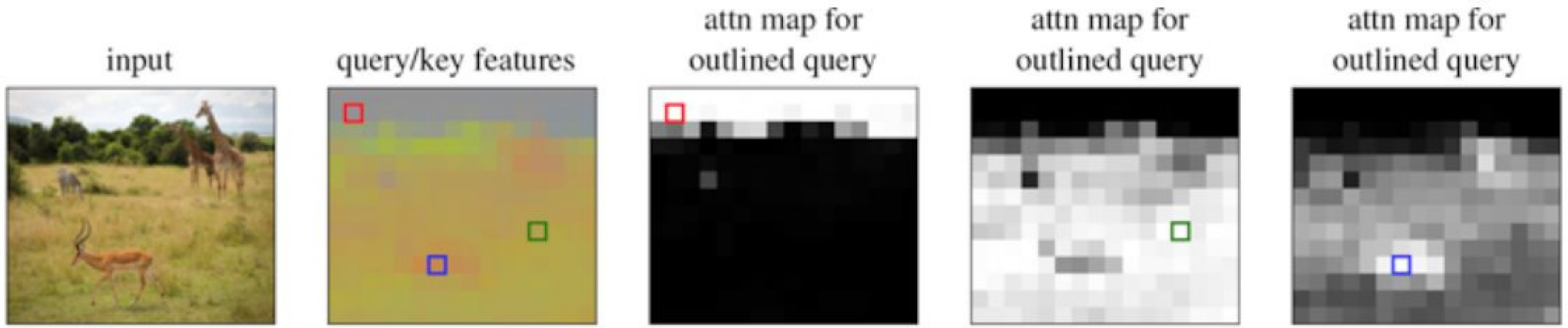

input
query/key features
attn map for outlined query
attn map for outlined query
attn map for outlined query

Figure 26.12: Example of self-attention maps where each token is an image patch and the query and key vectors are both set to the mean color of the patch, normalized to be a unit vector.

Multihead Self Attention

1. Despite their power, self-attention layers are still limited in that they **only have one set** of query/key/value projection matrices
2. What if we want to measure similarity in more than one way? For example, maybe we want our net to perform some set of computations based on **color similarity**, another based on **texture similarity**, and yet another based on **shape similarity**?

Multihead Self Attention

1. Running k attention layers in parallel.
2. All these layers are applied to the same input T_{in}.
3. This results in as set of k T_{out}
4. To merge these outputs, we concatenate all of them and project back to the original dimensionality of T_{in}.

Multihead Self Attention

1. The matrix W_{MSA} merges all the heads; its values are learnable parameters.

$$\mathbf{T}^{i}_{\texttt{out}} = \texttt{attn}^{i}(\mathbf{T}_{\texttt{in}}) \quad \text{for } i \in \{1, \ldots, k\} \tag{26.26}$$

$$\bar{\mathbf{T}}_{\texttt{out}} = \begin{bmatrix} \mathbf{T}^{1}_{\texttt{out}}[0,:] & \ldots & \mathbf{T}^{k}_{\texttt{out}}[0,:] \\ \vdots & \vdots & \vdots \\ \mathbf{T}^{1}_{\texttt{out}}[N-1,:] & \ldots & \mathbf{T}^{k}_{\texttt{out}}[N-1,:] \end{bmatrix} \quad \triangleleft \quad \bar{\mathbf{T}}_{\texttt{out}} \in \mathbb{R}^{N \times kv} \tag{26.27}$$

$$\mathbf{T}_{\texttt{out}} = \bar{\mathbf{T}}_{\texttt{out}} \mathbf{W}_{\texttt{MSA}} \quad \triangleleft \quad \mathbf{W}_{\texttt{MSA}} \in \mathbb{R}^{kv \times d} \tag{26.28}$$

The Full Transformer Architecture

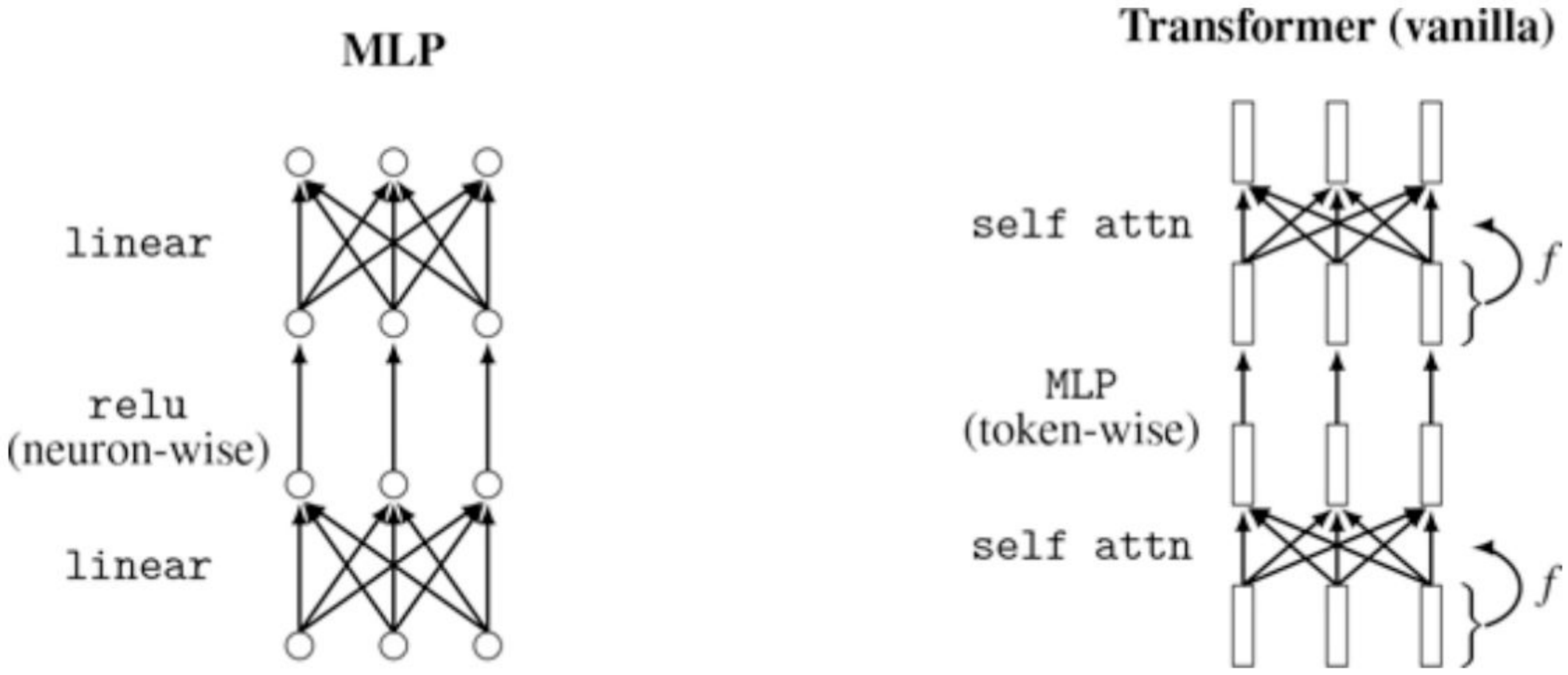
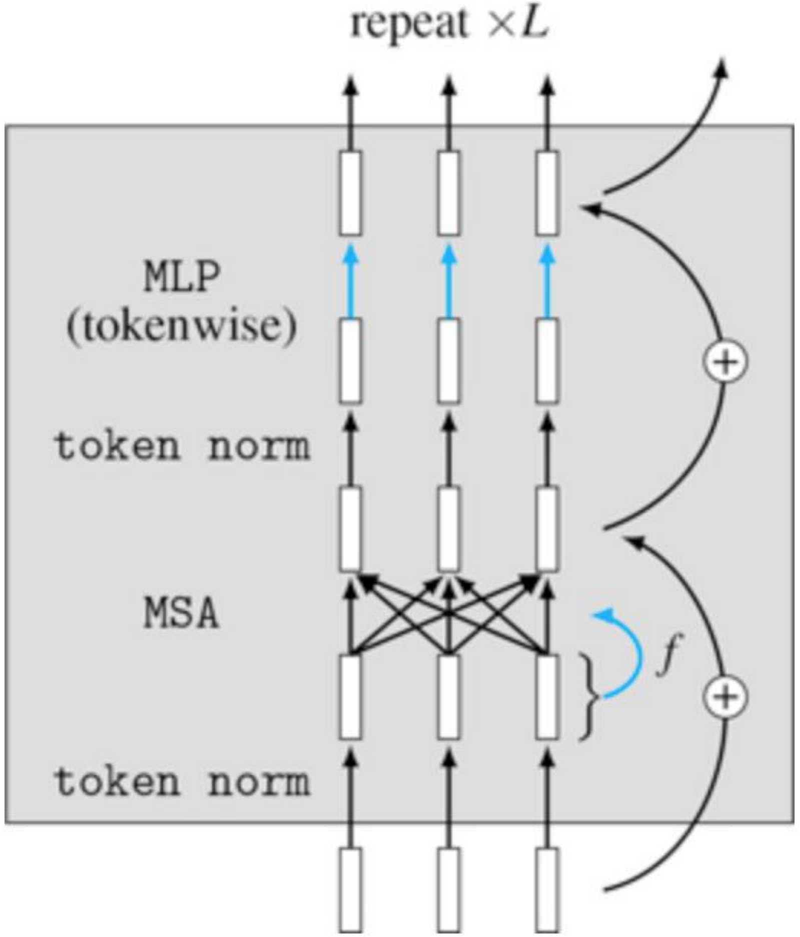


Figure 26.13: The basic transformer architecture versus an MLP.

Transformer (ViT) [109]



Why residual connections?

Residual Block: $x_{out} = F(x_{in}) + x_{in}$

Training deep neural networks is hard because of problems like:

1. Vanishing/exploding gradients

- As gradients flow backward through many layers, they can get very small (vanish) or very large (explode), making learning unstable or slow.

2. Information loss

- Each layer transforms the data, possibly distorting or losing important features.
- It's helpful if each layer can learn a *change* rather than a full replacement of the input.

3. Easier optimization

- Instead of learning the full function x_{out}, it's easier to learn a **residual** function
- $F(x_{in}) = x_{out} - x_{in}$ (i.e., just the *difference* from the input.)

Why residual connections?

1. It is easy for a residual block to simply perform an identity mapping
   a. it just has to learn to set F to zero.
2. Because of this, if we stack many residual blocks in a row, it can end up that the net learns to use only a subset of them.
3. If we set the number of stacked residual blocks to be very large then the net can essentially learn how deep to make itself, using as many blocks as necessary to solve the task.

Positional Encodings

1. Sometimes, we may wish to retain positional knowledge.
2. For example, knowing that a token is a representation of the top region of an image can help us identify that the token is likely to represent sky.
3. Positional encoding concatenates a code representing position within the signal onto each token.

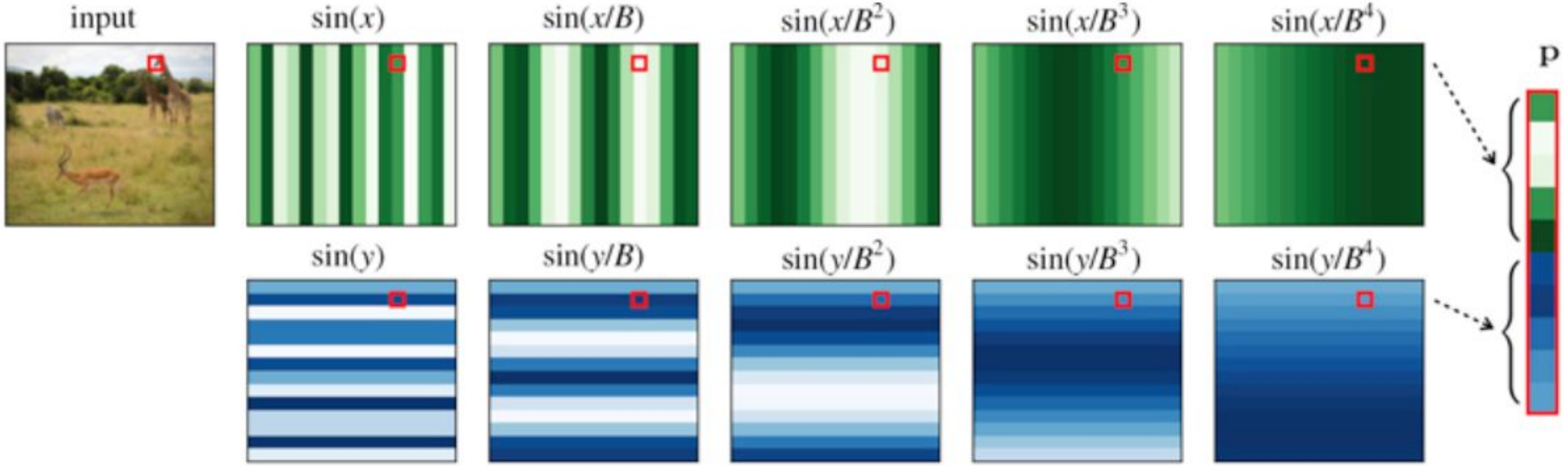
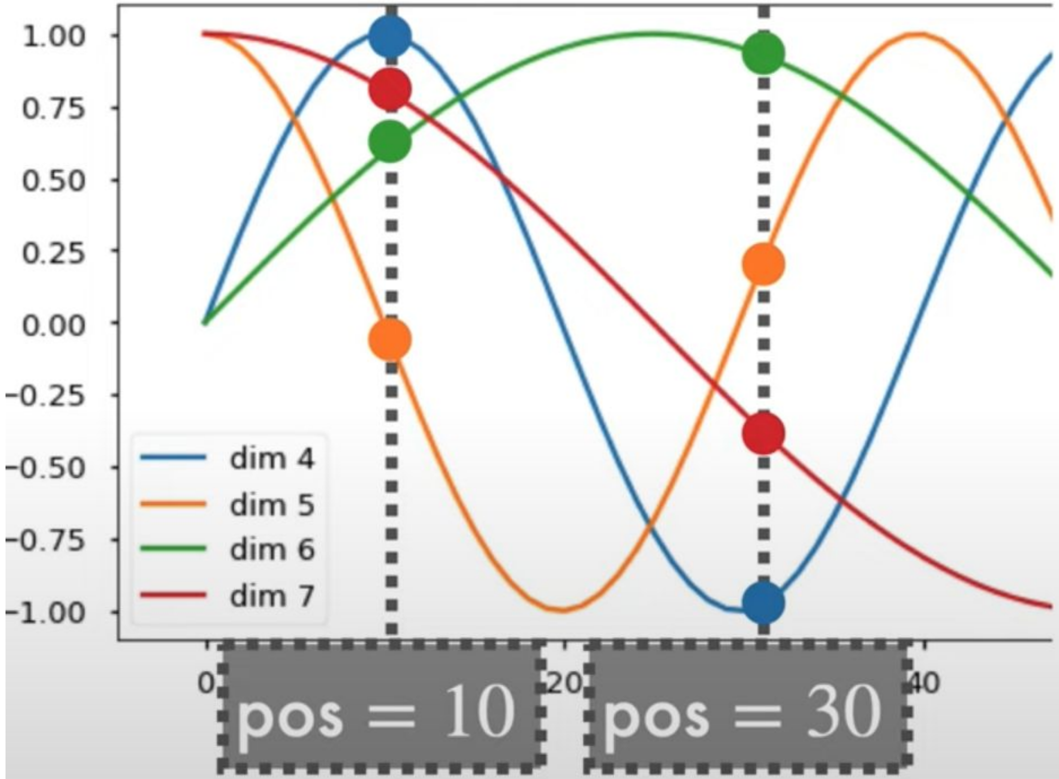


Figure 26.20: Positional codes.



Alternative (used in ViT): learn the embeddings from scratch

Transformers Tradeoffs

Effective Global Context with Fewer Layers:

- **CNNs:** To achieve a truly global receptive field in a CNN, you typically need a very deep network or very large kernel sizes.
- **Transformers:** The self-attention mechanism allows every token to directly interact with every other token in a single layer. This means that even with a relatively shallow Transformer network, information can propagate across the entire input sequence effectively, establishing global context much faster than in a CNN.

Selective Attention:

- While the initial formulation of self-attention has a quadratic complexity (O(n2d), the model learns to assign higher weights to the most relevant tokens and lower weights to less relevant ones for a given query token.
- This means that although *computationally* every token attends to every other token, *informationally*, the model can focus on a sparse subset of the input when making decisions. This selective attention is a powerful mechanism for capturing relevant global dependencies without being overwhelmed by irrelevant information.

Transformers Tradeoffs

Scalability Optimizations in Transformers:

- A significant amount of research has focused on developing more efficient attention mechanisms with sub-quadratic complexity. Some popular techniques include:
    - **Sparse Attention:** Restricting each token to attend to only a subset of other tokens (e.g., local windows, strided patterns, learned sparse patterns).
    - **Low-Rank Attention:** Approximating the attention matrix with lower-rank matrices to reduce the number of computations.
    - **Linformer, Performer, Reformer:** Architectures that introduce linear complexity (O(nd)) attention mechanisms.
    - **Hierarchical Attention:** Applying attention at different levels of granularity.
    - **Token pruning / distillation:** Drop unimportant tokens progressively

Parallel Processing:

- The self-attention mechanism is highly parallelizable. Unlike recurrent neural networks (RNNs), all token-to-token interactions in a Transformer layer can be computed in parallel, which can lead to faster training and inference on parallel hardware (like GPUs and TPUs).

Conclusion

1. Transformers are the dominant architecture in computer vision and in fact in most fields of artificial intelligence.
2. They combine many of the best ideas from earlier architectures—
    a. convolutional patchwise processing,
    b. residual connections,
    c. relu nonlinearities, and
    d. normalization layers—
3. with several newer innovations, in particular,
    a. vector-valued tokens,
    b. attention layers, and
    c. positional codes.
4. Transformers can also be considered as a special case of graph neural networks (GNNs). GNNs are a very general class of architecture for processing sets by forming a graph of operations over the set.
5. A transformer is doing exactly that: it takes an input set of tokens and, layer by layer, applies a network of transformations over that set until after enough layers a final representation or prediction is read out.

References

1. Foundations of Computer Vision - Chapter 26
2. https://www.youtube.com/watch?v=vsqKGZT8Qn8