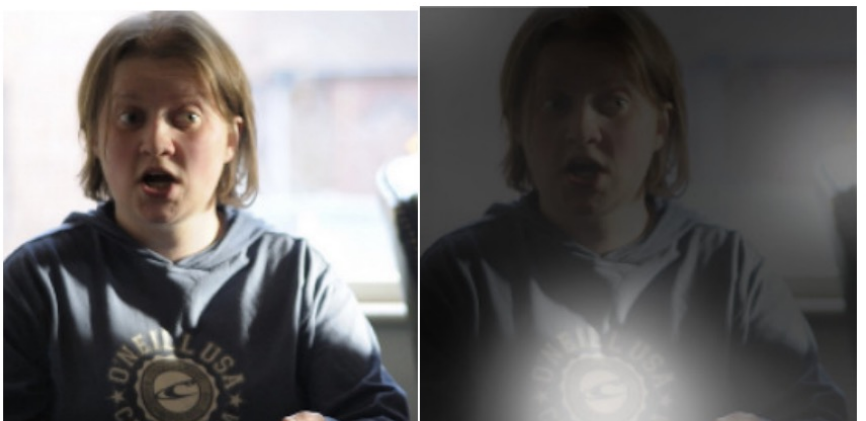
Vision Transformers

attention, transformers, vision transformers, CLIP



A woman holding a clock in her hand.

CSC420
David Lindell
University of Toronto
cs.toronto.edu/~lindell/teaching/420
Slide credit: FCV, Bill Freeman, Phillip Isola, Tianhong Li | CS231n, Fei Fei Li, Justin Johnson

TCIG
Toronto Computational Imaging Group


Course Overview
Machine Learning
Digital Photography
Dynamic Vision
3D Vision
1. fully-connected networks
2. CNNs
3. vision transfomers & generative models I
4. generative models II & embedded ethics
5. digital photoraphy
6. radiometry & photometry
7. 3D vision & stereo I
8. epipolar geometry & stereo II
9. structure from motion & radiance fields
10. optical flow & tracking
11. modeling humans

Outline

- motivation / why use different architectures?
- limitations of CNNs
- attention
- transformers
- generative models

Outline

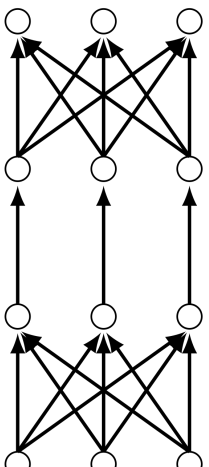
- motivation / why use different architectures?
- limitations of CNNs
- attention
- transformers
- generative models

Multilayer Perceptron

linear comb. of neurons ▷

neuron-wise nonlinearity ▷

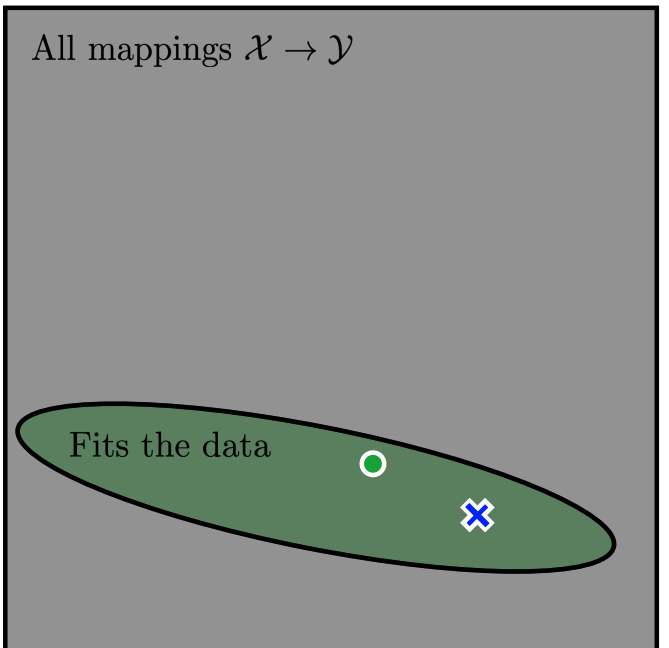
linear comb. of neurons ▷



+ Universal
+ Simple (elegant theory)
+ Embarrassingly parallel

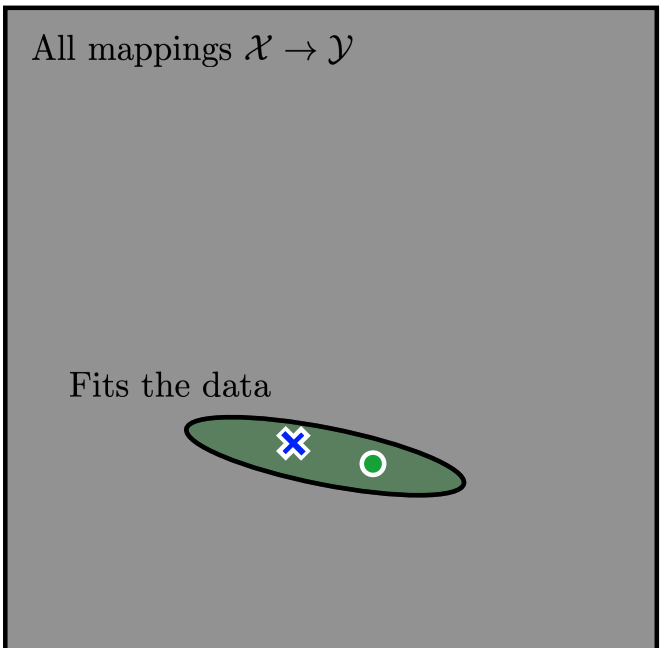
- Weak inductive biases
- Sample inefficient / data hungry
- Dense (fully-connected) linear layers take a lot of compute

Why use other architectures?



- True solution
- Learned solution

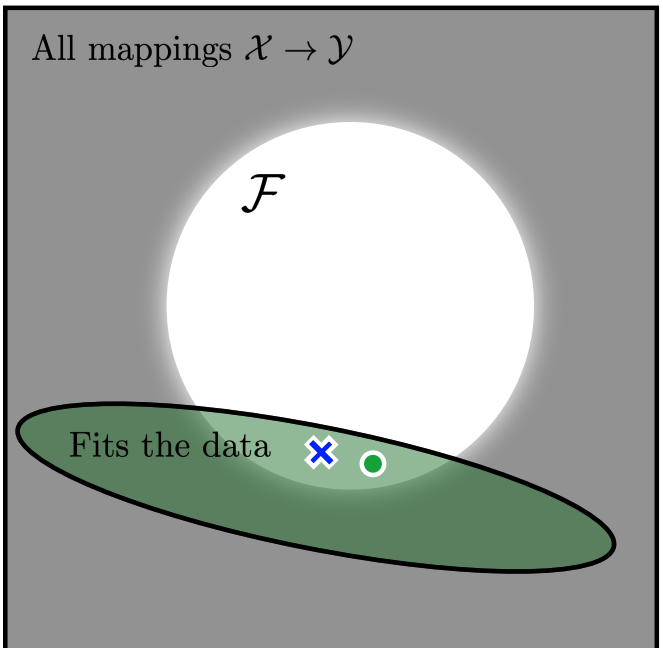
Why use other architectures?



- True solution
- Learned solution

Effect of adding more data.

Why use other architectures?



$\mathcal{F}$ – Hypothesis space

● True solution

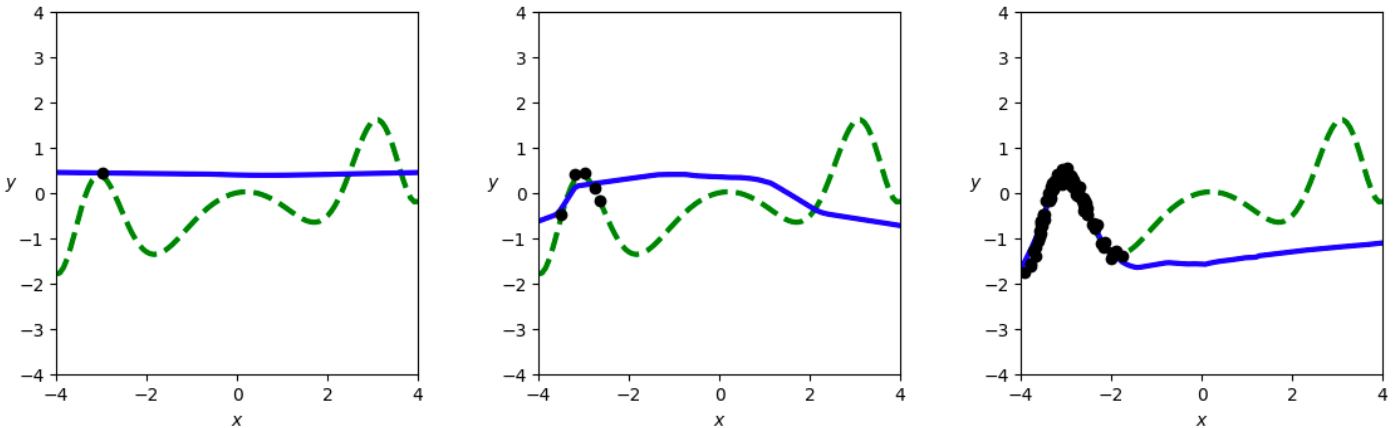
✖ Learned solution

Less data, better architecture.

We can pin down truth either by adding more data, or by using a more constrained architecture.

$y = f(x)$

f: 5 layer ReLU-net

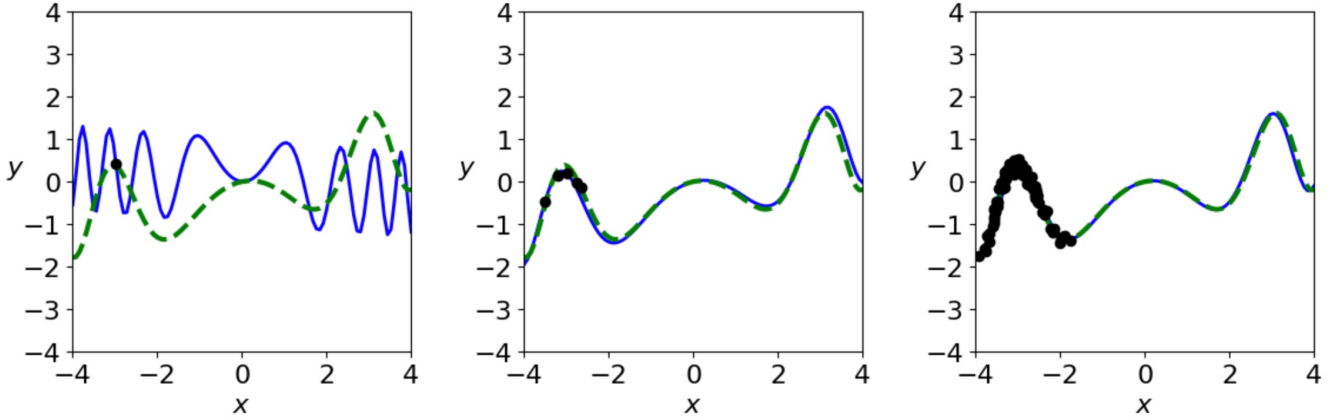


True solution

Learned solution

Training data

$$y = ax + \sin(bx^2)$$



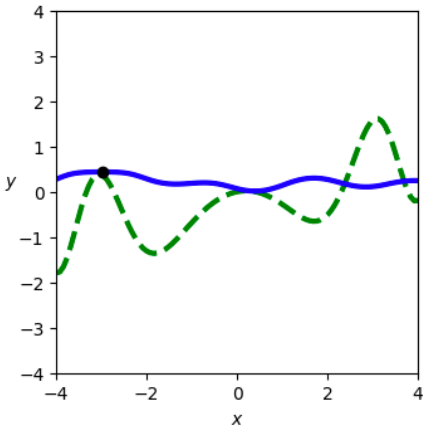
- True solution
- Learned solution
- Training data

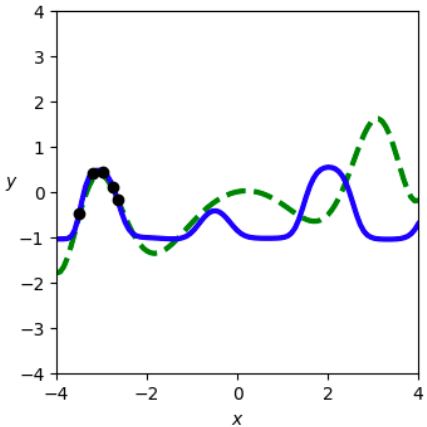
Architectures enable us to generalize outside the training distribution.

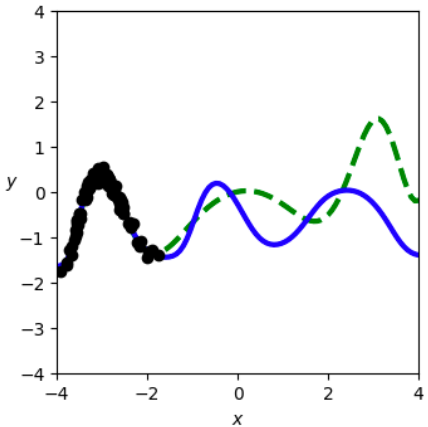
A good architecture is one that can represent the true function and is otherwise minimal (and is also easy to search over via gradient-based learning, easy to parallelize, fast on GPU, etc).

$y = f(x)$

f: 5 layer sin-net (SIREN)







- True solution
- Learned solution
- Training data

[Sitzmann*, Martel*, Bergman, Lindell, Wetzstein, NeurIPS 2020]

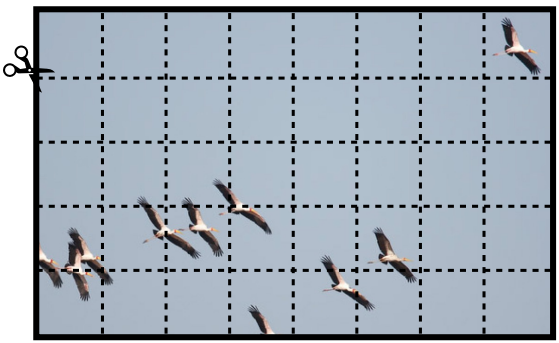
Outline

- motivation / why use different architectures?
- limitations of CNNs
- attention
- transformers
- generative models

A Limitation of CNNs

How many birds are in this image?

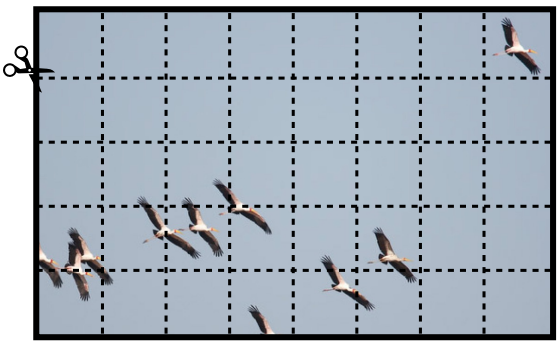
A Limitation of CNNs



How many birds are in this image?

Is the top right bird the same species as the bottom left bird?

A Limitation of CNNs

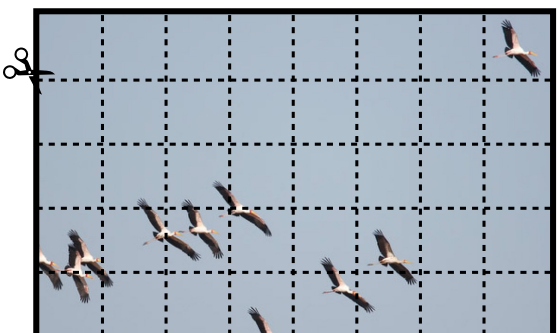


How many birds are in this image?

Is the top right bird the same species as the bottom left bird?

can a CNN answer these questions?

A Limitation of CNNs

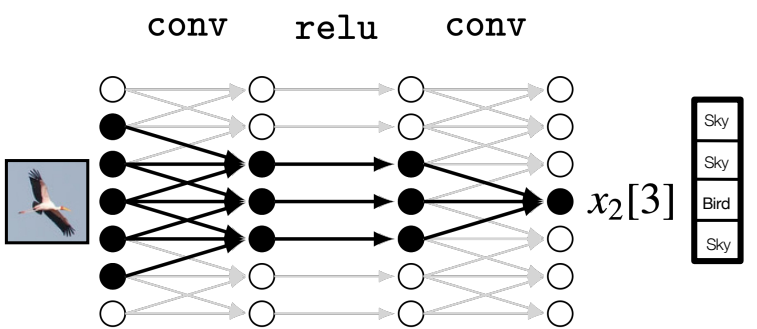


How many birds are in this image?

Is the top right bird the same species as the bottom left bird?

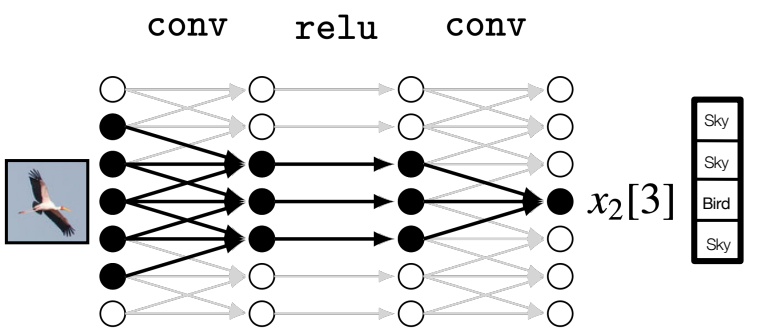
CNNs are built around the idea of locality, and are not well-suited to modeling long distance relationships

Receptive fields



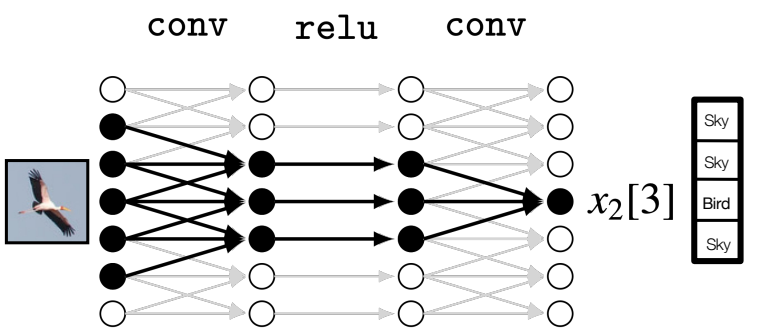
which input pixels does each output pixel depend on?

Receptive fields

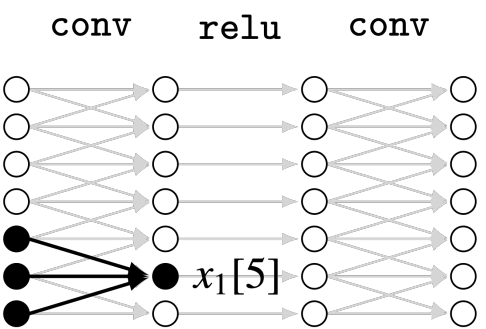


which input pixels does each output pixel depend on?
this is called the *receptive field*

Receptive fields

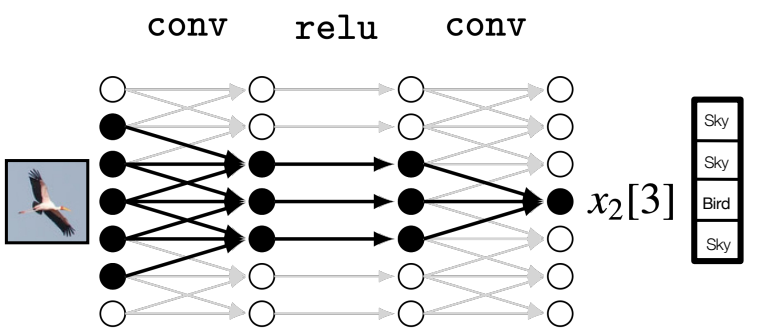


which input pixels does each output pixel depend on?
this is called the *receptive field*

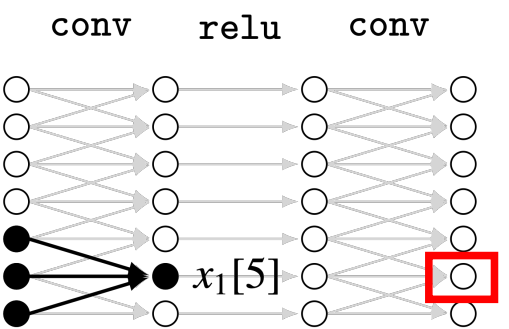


this is the receptive field for which output pixel?

Receptive fields

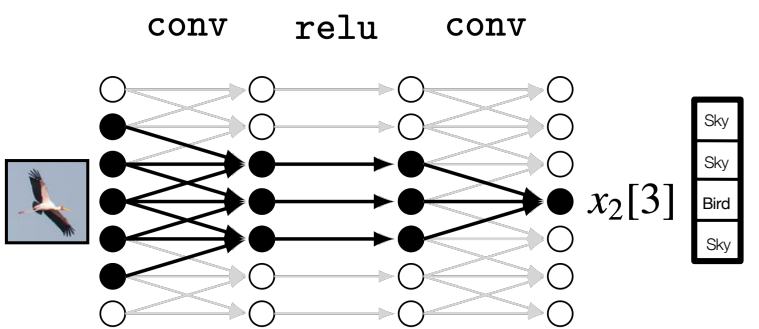


which input pixels does each output pixel depend on?
this is called the *receptive field*

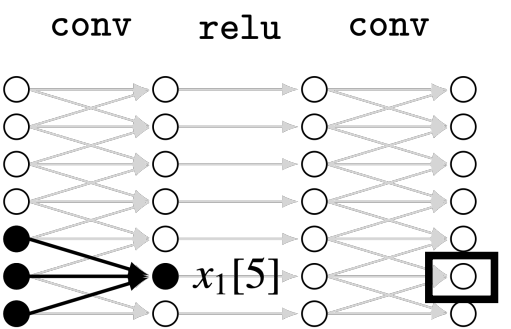


this is the receptive field for which output pixel?

Receptive fields



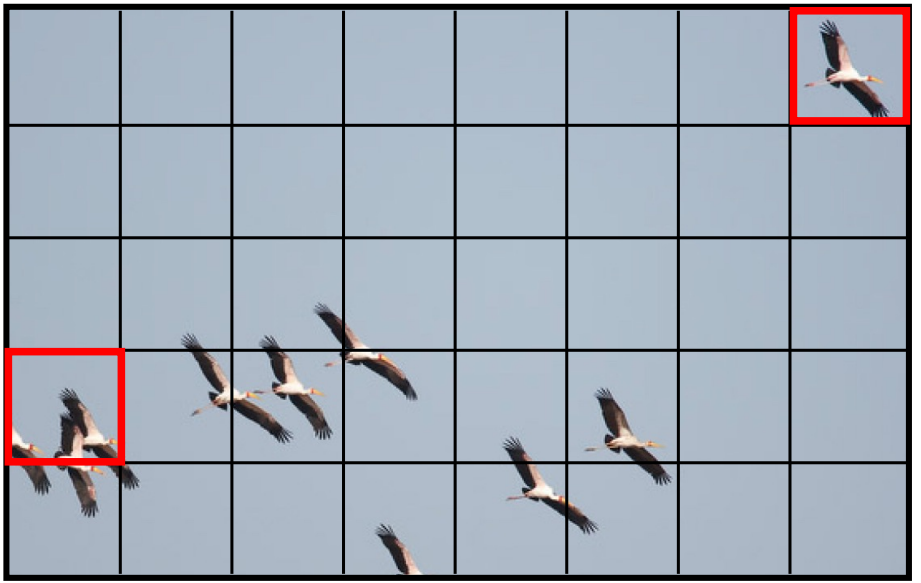
which input pixels does each output pixel depend on?
this is called the *receptive field*



this is the receptive field for which output pixel?

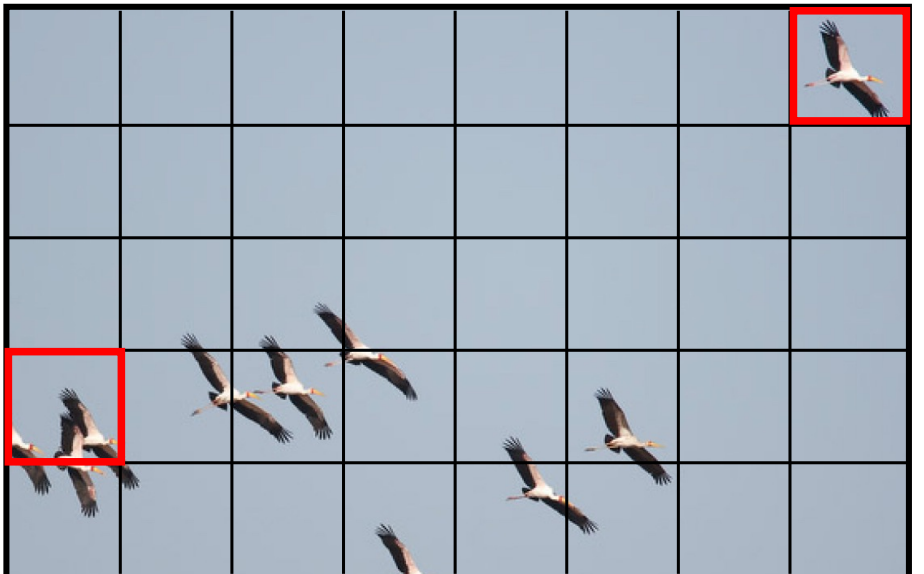
how could we increase/decrease the size of the receptive field?

Receptive fields



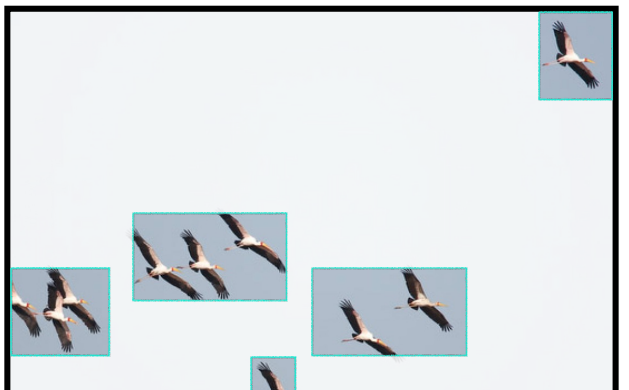
how much do distant patches affect the CNN output?

Receptive fields



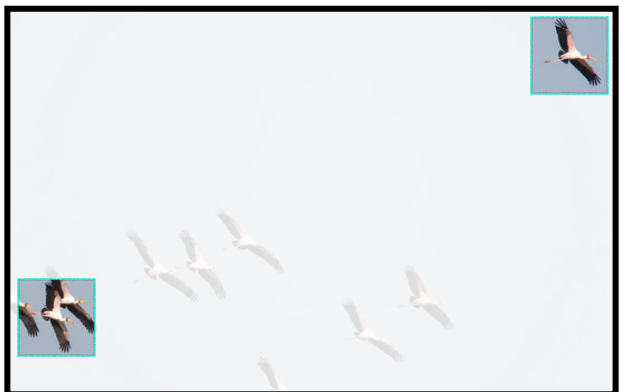
they may not interact at all!

The Idea of Attention



How many birds are in this image?

The Idea of Attention



Is the top right bird the same species as the bottom left bird?

The Idea of Attention

What's the color of the sky?

Outline

- motivation / why use different architectures?
- limitations of CNNs
- attention
- transformers
- generative models

Three Key Architectural Innovations

1. Tokens
2. Attention
3. Positional Codes

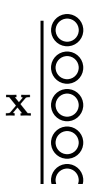
New idea #1: tokens

A New Data Type: Tokens

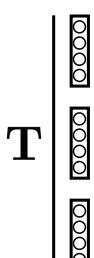
- A token is just a vector of neurons.

- But the connotation is that a token is an encapsulated bundle of information; with transformers we will operate over tokens rather than over neurons.

array of neurons



array of tokens



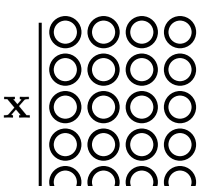
Note: sometimes the word "token" is instead used to refer to the atomic units of the data sequence we will model. In this usage tokens are the representation of the data only at the input and output layers. We use a more general definition where tokens are the representation of the data at any layer.

A new data structure: Tokens

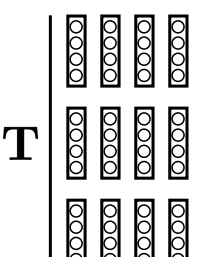
- A token is just a vector of neurons.

- But the connotation is that a token is an encapsulated bundle of information; with transformers we will operate over tokens rather than over neurons.

array of neurons



array of tokens

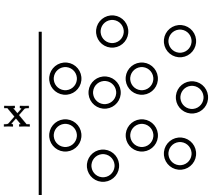


A new data structure: Tokens

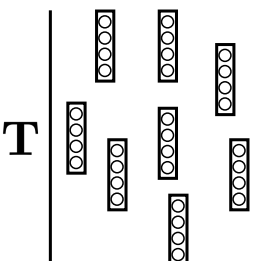
- A token is just a vector of neurons.

- But the connotation is that a token is an encapsulated bundle of information; with transformers we will operate over tokens rather than over neurons.

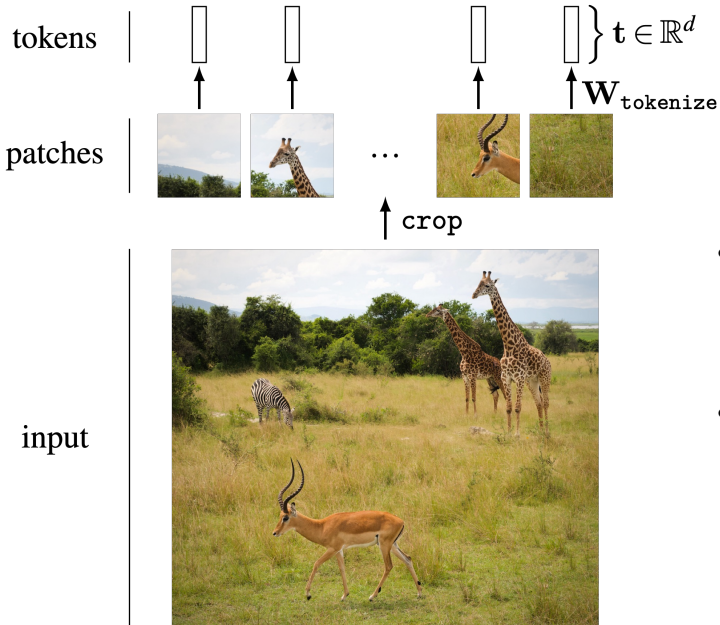
set of neurons



set of tokens



Tokenizing the input data

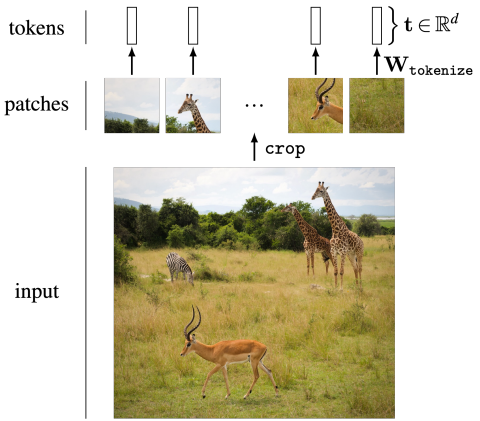


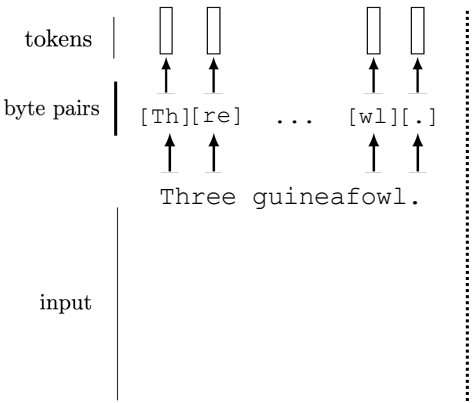
- When operating over neurons, we represent the input as an array of scalar-valued measurements (e.g., pixels)

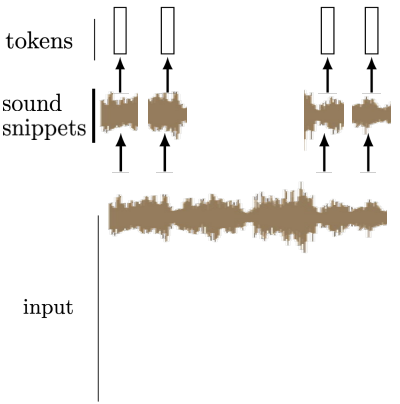
- When operating over tokens, we represent the input as an array of vector-valued measurements

Tokenizing the input data

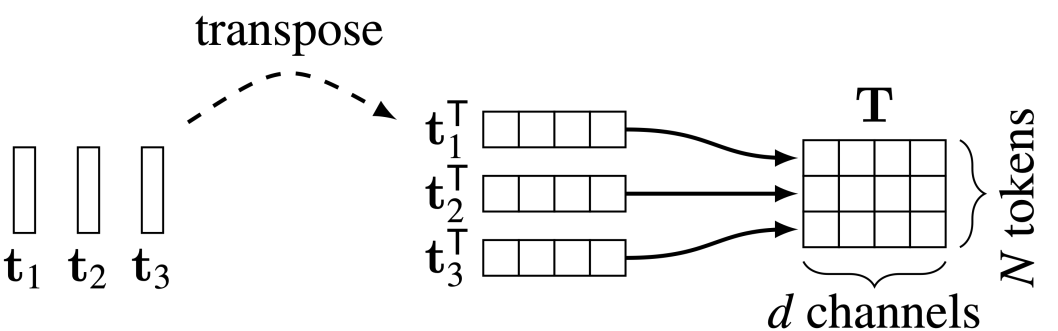
You can tokenize anything.
General strategy: chop the input up into chunks, project each chunk to a vector.





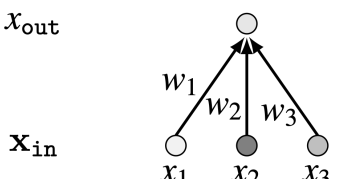


Notation


transpose
$\mathbf{t}_1$ $\mathbf{t}_2$ $\mathbf{t}_3$
$\mathbf{t}_1^\mathsf{T}$
$\mathbf{t}_2^\mathsf{T}$
$\mathbf{t}_3^\mathsf{T}$
$\mathbf{T}$
N tokens
d channels

Linear combination of tokens

Linear combination of neurons

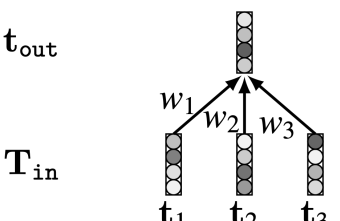


$$x_{\text{out}} = w_1 x_1 + w_2 x_2 + w_3 x_3$$

$$x_{\text{out}}[i] = \sum_{j=1}^{N} w_{ij} x_{\text{in}}[j]$$

$$\mathbf{x}_{\text{out}} = \mathbf{W}\mathbf{x}_{\text{in}}$$

Linear combination of tokens



$$\mathbf{t}_{\text{out}} = w_1 \mathbf{t}_1 + w_2 \mathbf{t}_2 + w_3 \mathbf{t}_3$$

$$\mathbf{T}_{\text{out}}[i,:] = \sum_{j=1}^{N} w_{ij} \mathbf{T}_{\text{in}}[j,:]$$

$$\mathbf{T}_{\text{out}} = \mathbf{W}\mathbf{T}_{\text{in}}$$

Token-wise nonlinearity

$$\mathbf{x}_{\texttt{out}} = \begin{bmatrix} \texttt{relu}(x_{\texttt{in}}[0]) \\ \vdots \\ \texttt{relu}(x_{\texttt{in}}[N-1]) \end{bmatrix}$$

$$\mathbf{T}_{\texttt{out}} = \begin{bmatrix} F_\theta(\mathbf{T}_{\texttt{in}}[0,:]) \\ \vdots \\ F_\theta(\mathbf{T}_{\texttt{in}}[N-1,:]) \end{bmatrix}$$

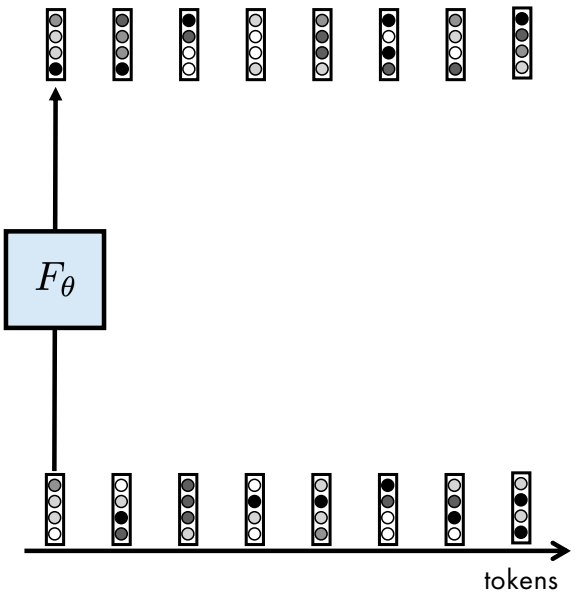
F is typically an MLP

Equivalent to a CNN with 1x1 kernels run over token sequence

Token-wise nonlinearity

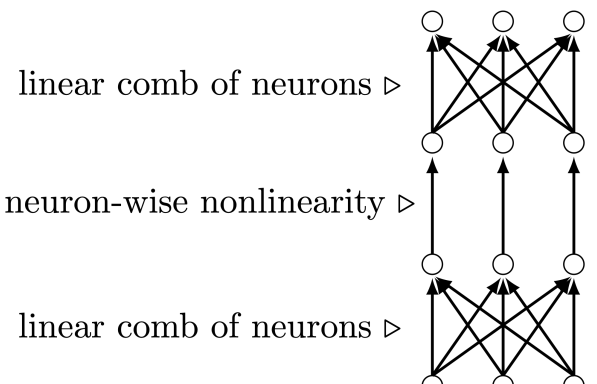
$$\mathbf{x}_{\text{out}} = \begin{bmatrix} \texttt{relu}(x_{\texttt{in}}[0]) \\ \vdots \\ \texttt{relu}(x_{\texttt{in}}[N-1]) \end{bmatrix}$$

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



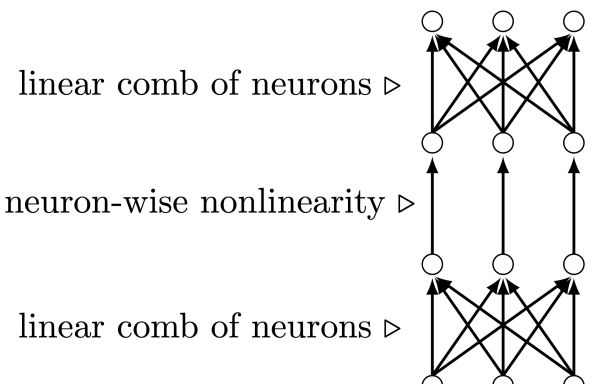
Token nets

Neural net

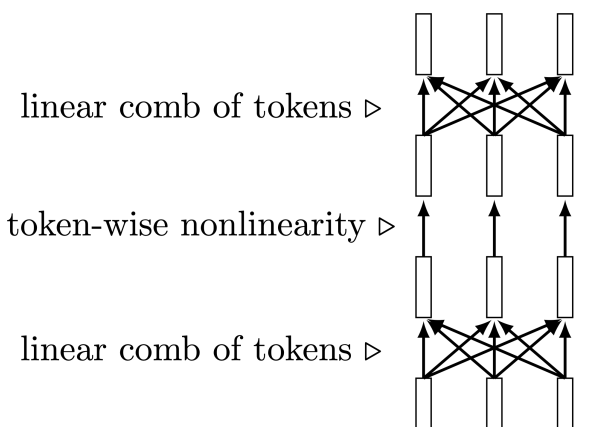


Token nets

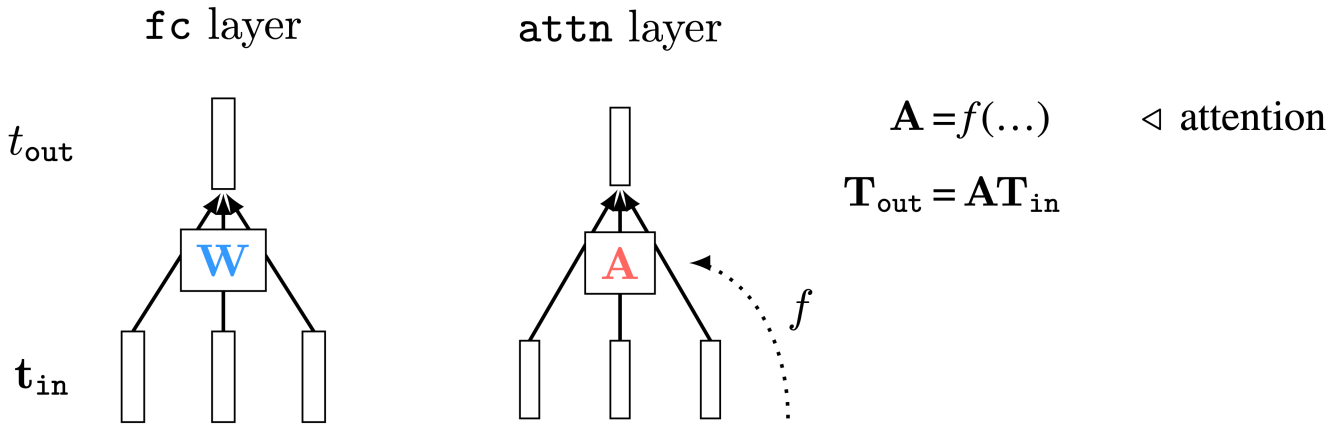
Neural net



Token net



New idea #2: attention



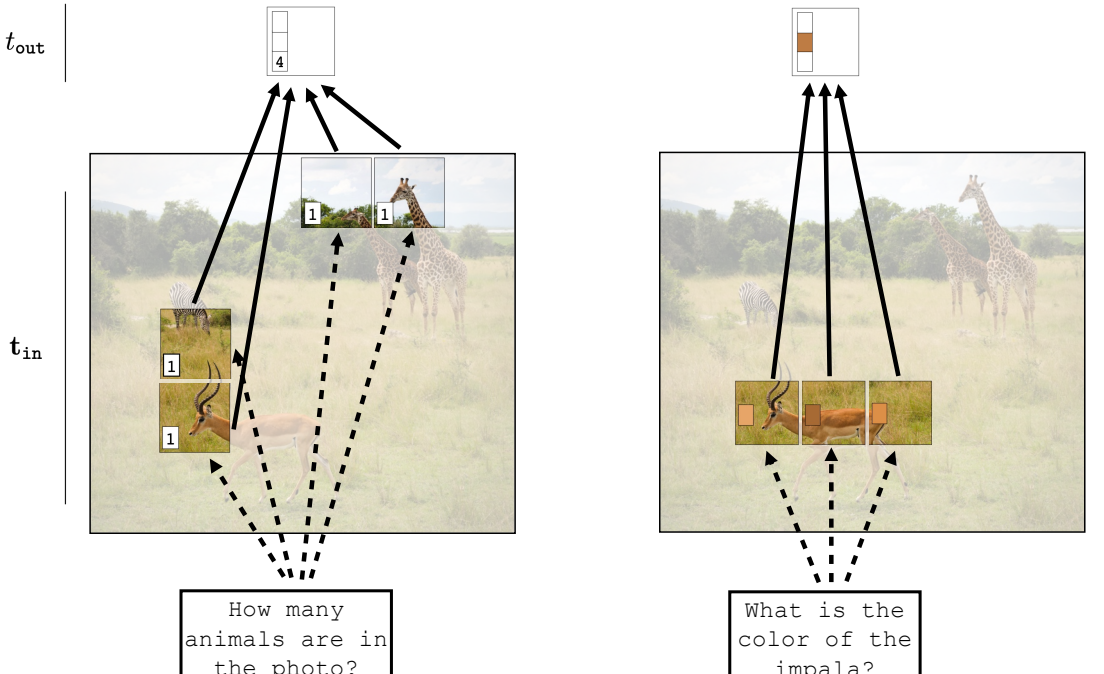
W is free parameters.

A is a function of some input data. The data tells us which tokens to attend to (assign high weight in weighted sum)

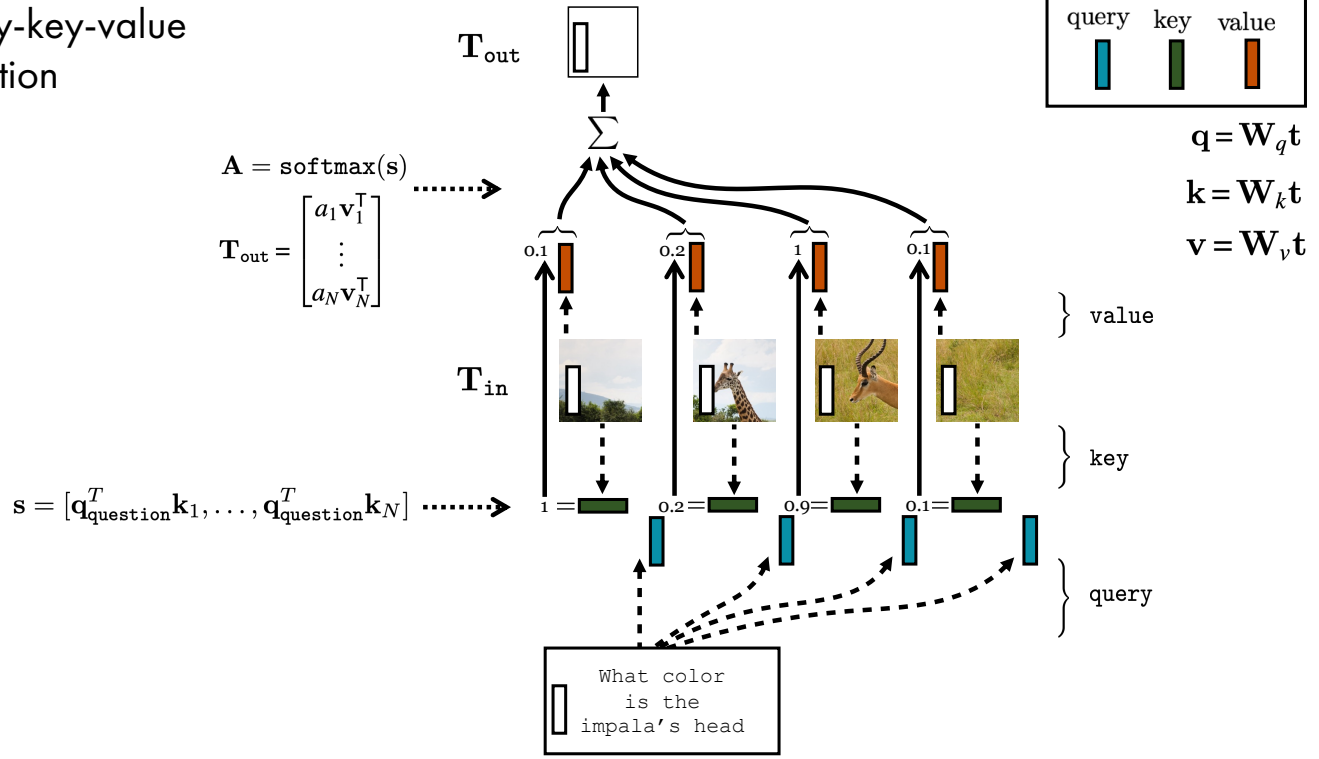
t_{out}

$\mathbf{t}_{\text{in}}$

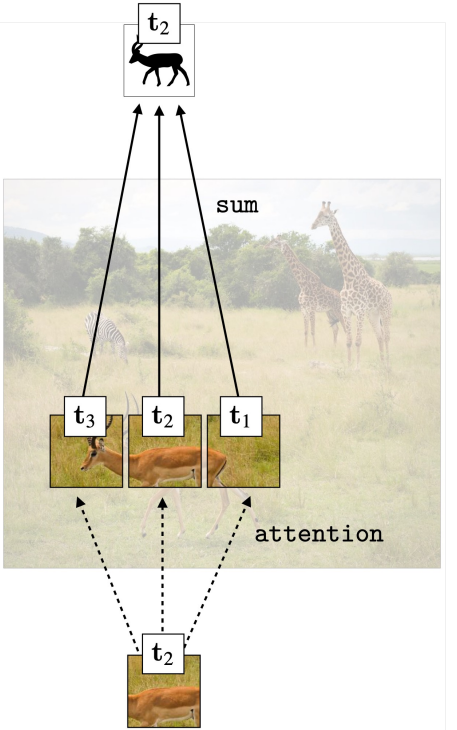
How many animals are in the photo?


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

query-key-value attention



Self-attention

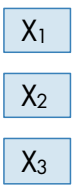


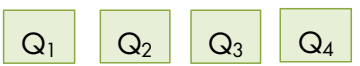
Attention Layer

Inputs:

Query vector: Q [N_Q x D_Q]

Data vectors: X [N_X x D_X]





slide credit: Justin Johnson, Fei Fei Li

Attention Layer

Inputs:

Query vector: Q [N_Q x D_Q]

Data vectors: X [N_X x D_X]

the input data
(e.g., image tokens)

X_1

X_2

X_3

the query that will attend to input data (e.g., text)

Q_1 Q_2 Q_3 Q_4

slide credit: Justin Johnson, Fei Fei Li

Attention Layer

Inputs:

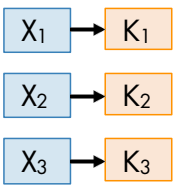
Query vector: Q $[N_Q \times D_Q]$

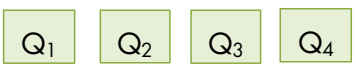
Data vectors: X $[N_X \times D_X]$

Key matrix: W_K $[D_X \times D_Q]$

Computation:

Keys: $K = XW_K$, $[N_X \times D_Q]$





slide credit: Justin Johnson, Fei Fei Li

Attention Layer

Inputs:

Query vector: Q $[N_Q \times D_Q]$

Data vectors: X $[N_X \times D_X]$

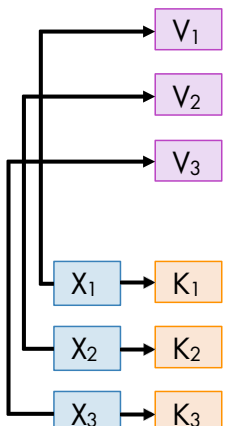
Key matrix: W_K $[D_X \times D_Q]$

Value matrix: W_V $[D_X \times D_V]$

Computation:

Keys: $K = XW_K$, $[N_X \times D_Q]$

Values: $V = XW_V$, $[N_X \times D_V]$



slide credit: Justin Johnson, Fei Fei Li

Attention Layer

Inputs:

Query vector: Q [N_Q x D_Q]

Data vectors: X [N_X x D_X]

Key matrix: W_K [D_X x D_Q]

Value matrix: W_V [D_X x D_V]

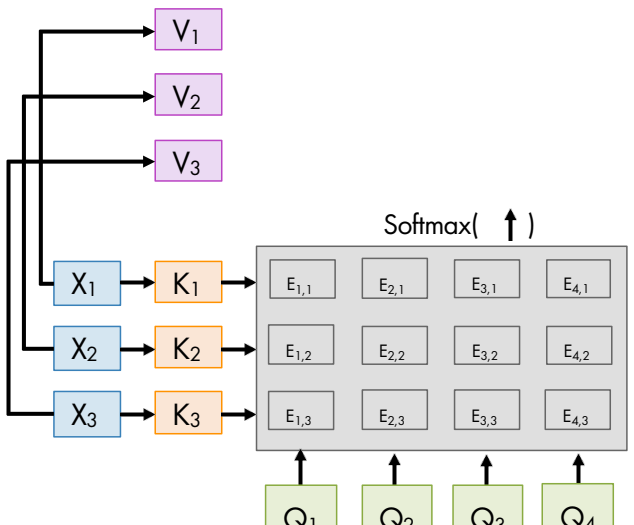
Computation:

Keys: $K = XW_K$, [N_X x D_Q]

Values: $V = XW_V$, [N_X x D_V]

Similarities: $E = QK^T / \sqrt{D_Q}$, [N_Q x N_X]

$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$



slide credit: Justin Johnson, Fei Fei Li

Attention Layer

Inputs:

Query vector: Q [N_Q x D_Q]

Data vectors: X [N_X x D_X]

Key matrix: W_K [D_X x D_Q]

Value matrix: W_V [D_X x D_V]

Computation:

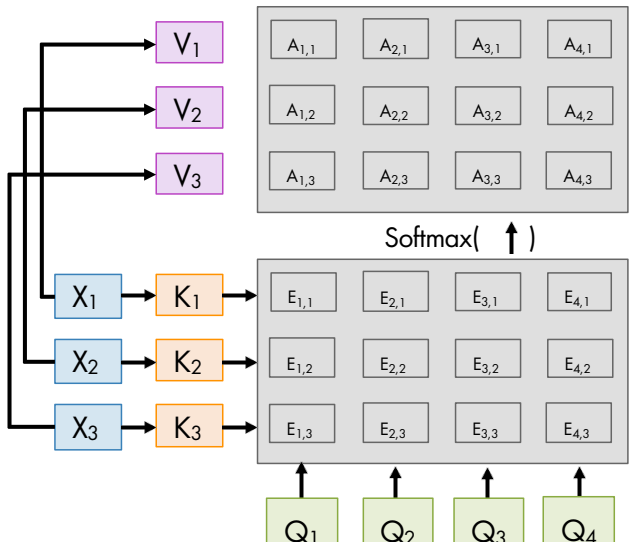
Keys: $K = XW_K$, [N_X x D_Q]

Values: $V = XW_V$, [N_X x D_V]

Similarities: $E = QK^T / \sqrt{D_Q}$, [N_Q x N_X]

$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Attention Weights: A = softmax(E, dim=1) [N_Q x N_X]



slide credit: Justin Johnson, Fei Fei Li

Attention Layer

Inputs:

Query vector: Q $[N_Q \times D_Q]$

Data vectors: X $[N_X \times D_X]$

Key matrix: W_K $[D_X \times D_Q]$

Value matrix: W_V $[D_X \times D_V]$

Computation:

Keys: $K = XW_K$, $[N_X \times D_Q]$

Values: $V = XW_V$, $[N_X \times D_V]$

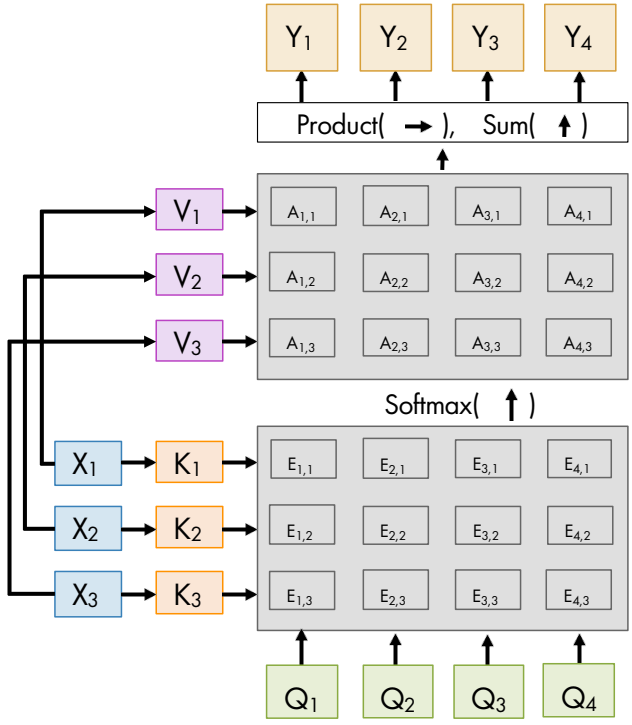
Similarities: $E = QK^T / \sqrt{D_Q}$, $[N_Q \times N_X]$

$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Attention Weights: $A = \text{softmax}(E, \text{dim}=1)$ $[N_Q \times N_X]$

Output vector: $Y = AV$ $[N_Q \times D_V]$

$Y_i = \sum_j A_{ij} V_j$



slide credit: Justin Johnson, Fei Fei Li

Attention Layer

Inputs:

Query vector: Q $[N_Q \times D_Q]$

Data vectors: X $[N_X \times D_X]$

Key matrix: W_K $[D_X \times D_Q]$

Value matrix: W_V $[D_X \times D_V]$

Com

Keys

Valu

Each output is a linear combination of all values, weighted by attention weights

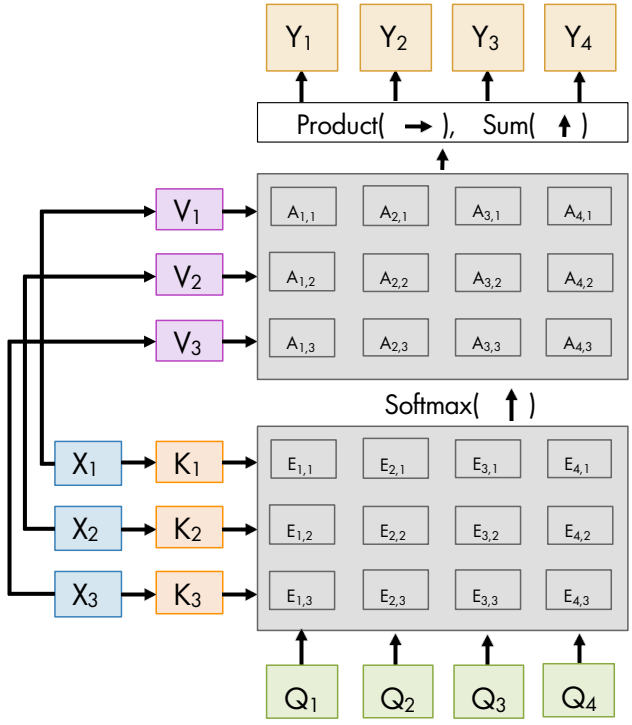
Similarities: $E = QK^T / \sqrt{D_Q}$, $[N_Q \times N_X]$

$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Attention Weights: $A = \text{softmax}(E, \text{dim}=1)$ $[N_Q \times N_X]$

Output vector: $Y = AV$ $[N_Q \times D_V]$

$Y_i = \sum_j A_{ij} V_j$



slide credit: Justin Johnson, Fei Fei Li

Cross-Attention Layer

Inputs:

Query vector: Q $[N_Q \times D_Q]$

Data vectors: X $[N_X \times D_X]$

Key matrix: W_K $[D_X \times D_Q]$

Value matrix: W_V $[D_X \times D_V]$

Computation:

Keys: $K = XW_K$, $[N_X \times D_Q]$

Values: $V = XW_V$, $[N_X \times D_V]$

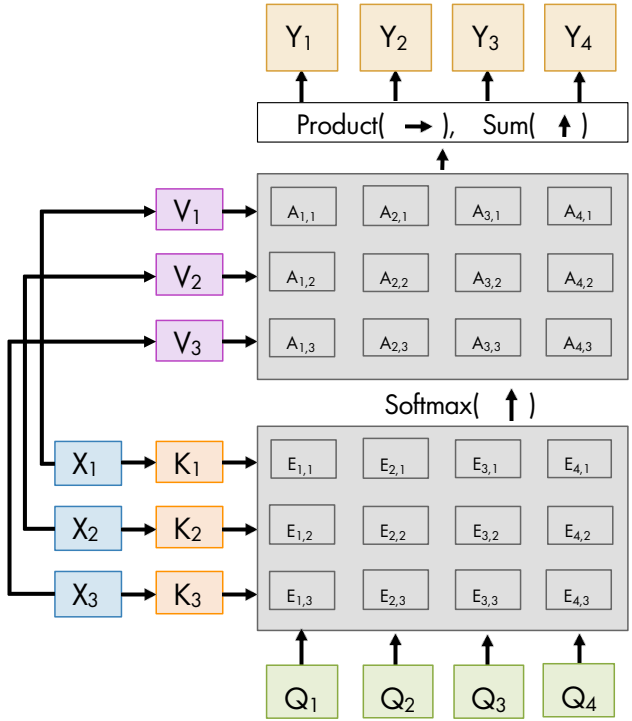
Similarities: $E = QK^T / \sqrt{D_Q}$, $[N_Q \times N_X]$

$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Attention Weights: $A = \text{softmax}(E, \text{dim}=1)$ $[N_Q \times N_X]$

Output vector: $Y = AV$ $[N_Q \times D_V]$

$Y_i = \sum_j A_{ij} V_j$



slide credit: Justin Johnson, Fei Fei Li

Cross-Attention Layer

Inputs:

Query vector: Q [N_Q x D_Q]

Data vectors: X [N_X x D_X]

Key matrix: W_K [D_X x D_Q]

Value matrix: W_V [D_X x D_V]

Com

Keys

Valu

Similarities: $E = QK^T / \sqrt{D_Q}$, [N_Q x N_X]

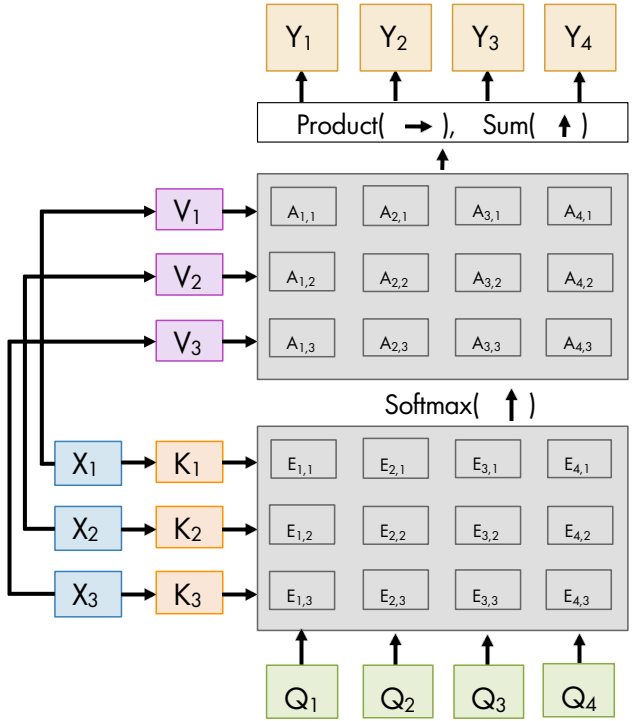
$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Attention Weights: A = softmax(E, dim=1) [N_Q x N_X]

Output vector: Y = AV [N_Q x D_V]

$Y_i = \sum_j A_{ij} V_j$

Each query produces one output, which is a mix of information in the data vectors



slide credit: Justin Johnson, Fei Fei Li

Self-Attention Layer

Self-Attention Layer

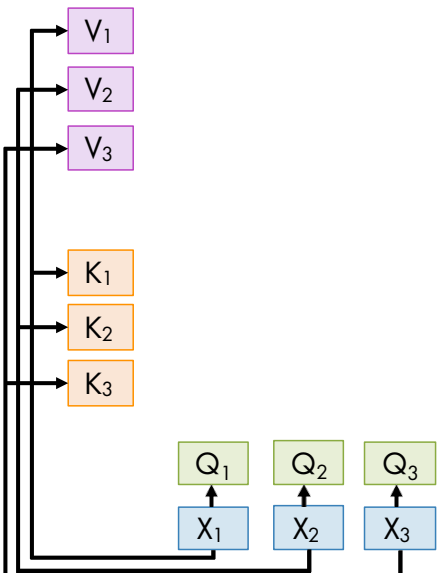
Inputs:

Query matrix: W_Q [D_{in} x D_{out}]

Data vector: X [N x D_{in}]

Key matrix: W_K [D_{in} x D_{out}]

Value matrix: W_V [D_{in} x D_{out}]



slide credit: Justin Johnson, Fei Fei Li

Self-Attention Layer

Inputs:

Query matrix: W_Q [D_{in} x D_{out}]

Data vector: X [N x D_{in}]

Key matrix: W_K [D_{in} x D_{out}]

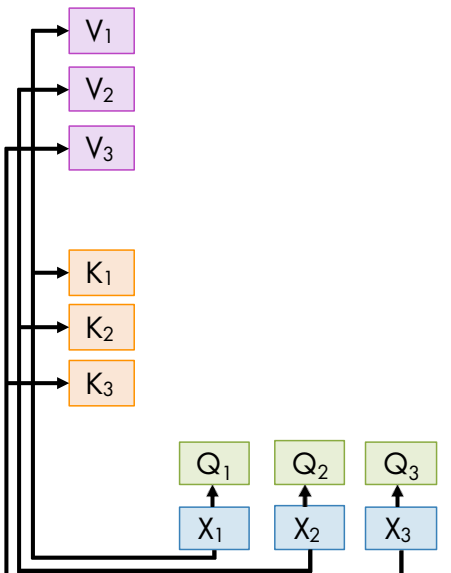
Value matrix: W_V [D_{in} x D_{out}]

From each input: compute a query, key, and value vector

Often fused to one matmul:

$[Q\ K\ V] = X\ [W_Q\ W_K\ W_V]$

[N x 3Dout] = [N x Din] [Din x 3Dout]



slide credit: Justin Johnson, Fei Fei Li

Self-Attention Layer

Inputs:

Query matrix: W_Q [D_{in} x D_{out}]

Data vector: X [N x D_{in}]

Key matrix: W_K [D_{in} x D_{out}]

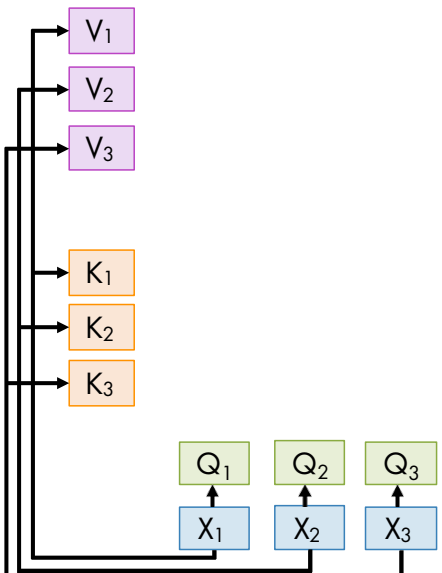
Value matrix: W_V [D_{in} x D_{out}]

Computation:

Queries: $Q = XW_Q$, [N x D_{out}]

Keys: $K = XW_K$, [N x D_{out}]

Values: $V = XW_V$, [N x D_{out}]



slide credit: Justin Johnson, Fei Fei Li

Self-Attention Layer

Inputs:

Query matrix: W_Q [D_{in} x D_{out}]

Data vector: X [N x D_{in}]

Key matrix: W_K [D_{in} x D_{out}]

Value matrix: W_V [D_{in} x D_{out}]

Computation:

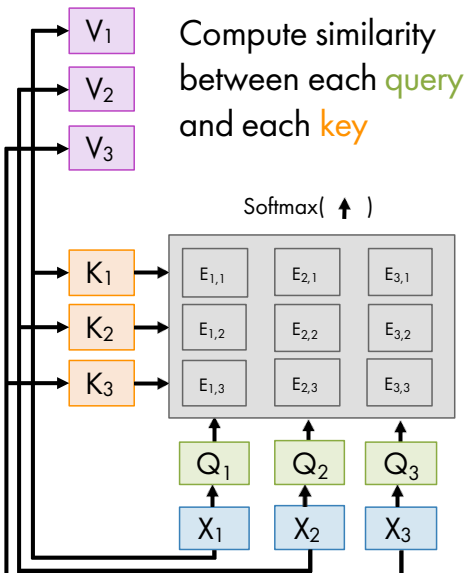
Queries: $Q = XW_Q$, [N x D_{out}]

Keys: $K = XW_K$, [N x D_{out}]

Values: $V = XW_V$, [N x D_{out}]

Similarities: $E = QK^T / \sqrt{D_Q}$, [N x N]

$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$



slide credit: Justin Johnson, Fei Fei Li

Self-Attention Layer

Inputs:

Query matrix: W_Q [D_{in} x D_{out}]

Data vector: X [N x D_{in}]

Key matrix: W_K [D_{in} x D_{out}]

Value matrix: W_V [D_{in} x D_{out}]

Computation:

Queries: $Q = XW_Q$, [N x D_{out}]

Keys: $K = XW_K$, [N x D_{out}]

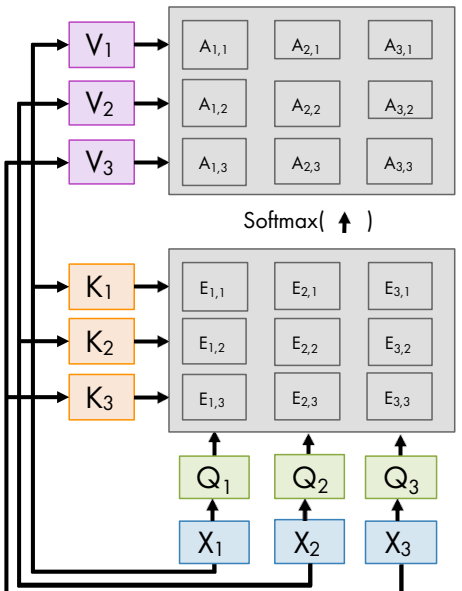
Values: $V = XW_V$, [N x D_{out}]

Similarities: $E = QK^T / \sqrt{D_Q}$, [N x N]

$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Attention Weights: A = softmax(E, dim=1) [N x N]

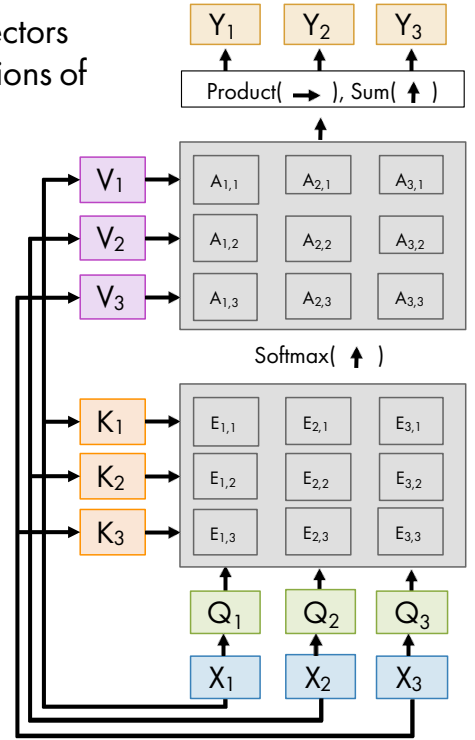
Normalize over each column: each query computes a distribution over keys



slide credit: Justin Johnson, Fei Fei Li

Self-Attention Layer

Compute output vectors as linear combinations of value vectors



Inputs:

Query matrix: W_Q [D_{in} x D_{out}]

Data vector: X [N x D_{in}]

Key matrix: W_K [D_{in} x D_{out}]

Value matrix: W_V [D_{in} x D_{out}]

Computation:

Queries: $Q = XW_Q$, [N x D_{out}]

Keys: $K = XW_K$, [N x D_{out}]

Values: $V = XW_V$, [N x D_{out}]

Similarities: $E = QK^T / \sqrt{D_Q}$, [N x N]

$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Attention Weights: A = softmax(E, dim=1) [N x N]

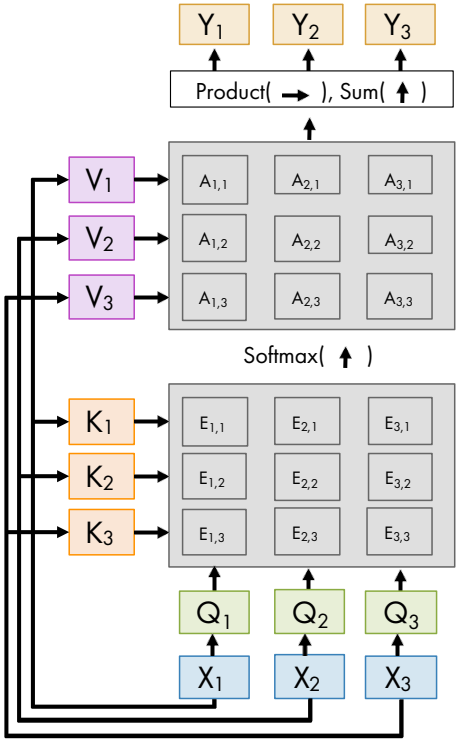
Output vector: $Y = AV$ [N x D_{out}] , $Y_i = \sum_i A_{ij} V_i$

slide credit: Justin Johnson, Fei Fei Li

Self-Attention Layer

Each input produces one output, which is a mix of information from all inputs

Shapes get a little simpler:
- N input vectors, each D_{in}
- Almost always D = D = D



slide credit: Justin Johnson, Fei Fei Li

Masked Self-Attention Layer

Inputs:

Query matrix: Q [D_{in} x D_{out}]

Data vector: X [N x D_{in}]

Key matrix: W_K [D_{in} x D_{out}]

Value matrix: W_V [D_{in} x D_{out}]

Computation:

Queries: Q = XW$_Q$, [N x D_{out}]

Keys: K = XW$_K$, [N x D_{out}]

Values: V = XW$_V$, [N x D_{out}]

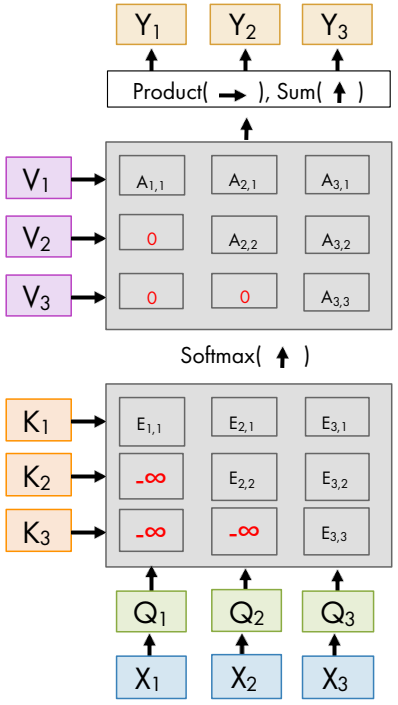
Similarities: $E = QK^T / \sqrt{D_Q}$, [N x N]

$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Override similarities with -inf; this controls which inputs each vector is allowed to look at.

Attention Weights: A = softmax(E, dim=1) [N x N]

Output vector: Y = AV [N x D_{out}] , $Y_i = \sum_j A_{ij} V_j$



slide credit: Justin Johnson, Fei Fei Li

Masked Self-Attention Layer

Inputs:

Query matrix: Q [D_{in} x D_{out}]

Data vector: X [N x D_{in}]

Key matrix: W_K [D_{in} x D_{out}]

Value matrix: W_V [D_{in} x D_{out}]

Computation:

Queries: $Q = XW_Q$, [N x D_{out}]

Keys: $K = XW_K$, [N x D_{out}]

Values: $V = XW_V$, [N x D_{out}]

Similarities: $E = QK^T / \sqrt{D_Q}$, [N x N]

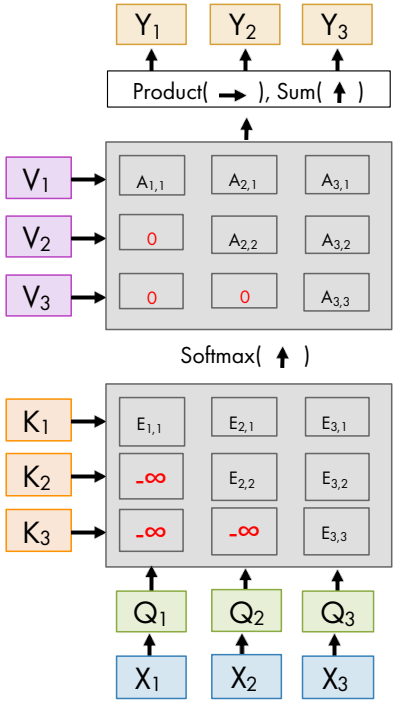
$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Attention Weights: $A = \text{softmax}(E, \text{dim}=1)$ [N x N]

Output vector: $Y = AV$ [N x D_{out}] , $Y_i = \sum_j A_{ij} V_j$

why is this useful?

Override similarities with -inf; this controls which inputs each vector is allowed to look at.



slide credit: Justin Johnson, Fei Fei Li

Masked Self-Attention Layer

Inputs:

Query matrix: Q [D_{in} x D_{out}]

Data vector: X [N x D_{in}]

Key matrix: W_K [D_{in} x D_{out}]

Value matrix: W_V [D_{in} x D_{out}]

Computation:

Queries: $Q = XW_Q$, [N x D_{out}]

Keys: $K = XW_K$, [N x D_{out}]

Values: $V = XW_V$, [N x D_{out}]

Similarities: $E = QK^T / \sqrt{D_Q}$, [N x N]

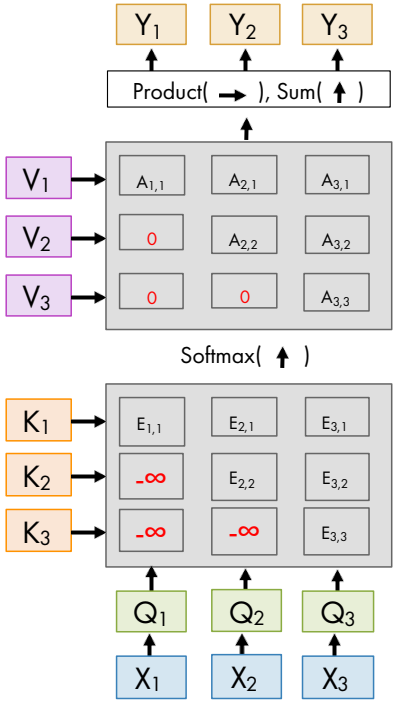
$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Attention Weights: $A = \text{softmax}(E, \text{dim}=1)$ [N x N]

Output vector: $Y = AV$ [N x D_{out}] , $Y_i = \sum_j A_{ij} V_j$

why is this useful?

Override similarities with -inf; this controls which inputs each vector is allowed to look at.



slide credit: Justin Johnson, Fei Fei Li

Multi-Headed Self-Attention Layer

Inputs:

Query matrix: Q [D_{in} x D_{out}]

Data vector: X [N x D_{in}]

Key matrix: W_K [D_{in} x D_{out}]

Value matrix: W_V [D_{in} x D_{out}]

Computation:

Queries: $Q = XW_Q$, [N x D_{out}]

Keys: $K = XW_K$, [N x D_{out}]

Values: $V = XW_V$, [N x D_{out}]

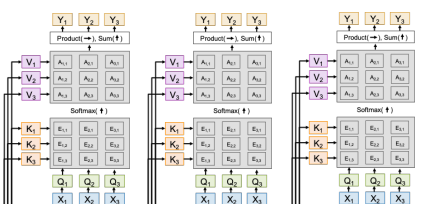
Similarities: $E = QK^T / \sqrt{D_Q}$, [N x N]

$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Attention Weights: A = softmax(E, dim=1) [N x N]

Output vector: $Y = AV$ [N x D_{out}] , $Y_i = \sum_j A_{ij} V_j$

H = 3 independent self-attention layers (called heads), each with their own weights



X_1 X_2 X_3

slide credit: Justin Johnson, Fei Fei Li

Multi-Headed Self-Attention Layer

Inputs:

Query matrix: Q [D_{in} x D_{out}]

Data vector: X [N x D_{in}]

Key matrix: W_K [D_{in} x D_{out}]

Value matrix: W_V [D_{in} x D_{out}]

Computation:

Queries: $Q = XW_Q$, [N x D_{out}]

Keys: $K = XW_K$, [N x D_{out}]

Values: $V = XW_V$, [N x D_{out}]

Similarities: $E = QK^T / \sqrt{D_Q}$, [N x N]

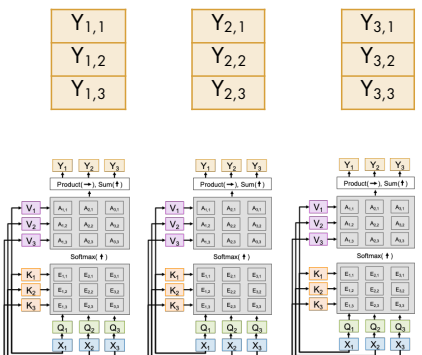
$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Attention Weights: A = softmax(E, dim=1) [N x N]

Output vector: $Y = AV$ [N x D_{out}] , $Y_i = \sum_j A_{ij} V_j$

Stack up the H independent outputs for each input X

H = 3 independent self-attention layers (called heads), each with their own weights



slide credit: Justin Johnson, Fei Fei Li

Multi-Headed Self-Attention Layer

Inputs:

Query matrix: Q [D_{in} x D_{out}]

Data vector: X [N x D_{in}]

Key matrix: W_K [D_{in} x D_{out}]

Value matrix: W_V [D_{in} x D_{out}]

Computation:

Queries: $Q = XW_Q$, [N x D_{out}]

Keys: $K = XW_K$, [N x D_{out}]

Values: $V = XW_V$, [N x D_{out}]

Similarities: $E = QK^T / \sqrt{D_Q}$, [N x N]

$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

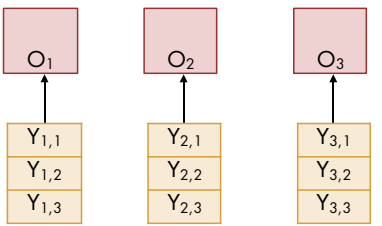
Attention Weights: A = softmax(E, dim=1) [N x N]

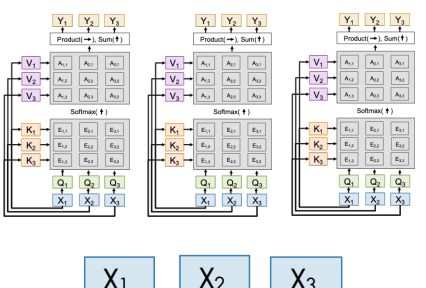
Output vector: $Y = AV$ [N x D_{out}] , $Y_i = \sum_j A_{ij} V_j$

Output projection fuses data from each head

Stack up the H independent outputs for each input X

H = 3 independent self-attention layers (called heads), each with their own weights



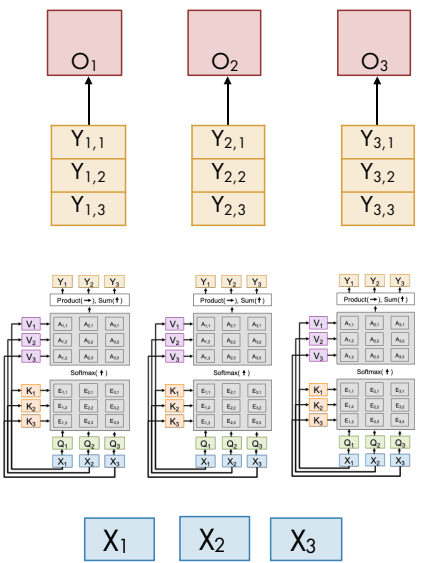


slide credit: Justin Johnson, Fei Fei Li

Multi-Headed Self-Attention Layer

implementation details:

- Each of the H parallel layers use a QKV dim of D_H = "head dim"

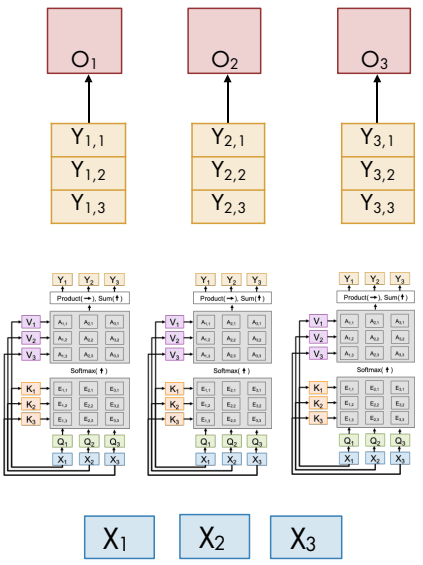


slide credit: Justin Johnson, Fei Fei Li

Multi-Headed Self-Attention Layer

implementation details:

- Each of the H parallel layers use a QKV dim of D_H = "head dim"
- Usually $D_H = D / H$, so the inputs and outputs have the same dimension

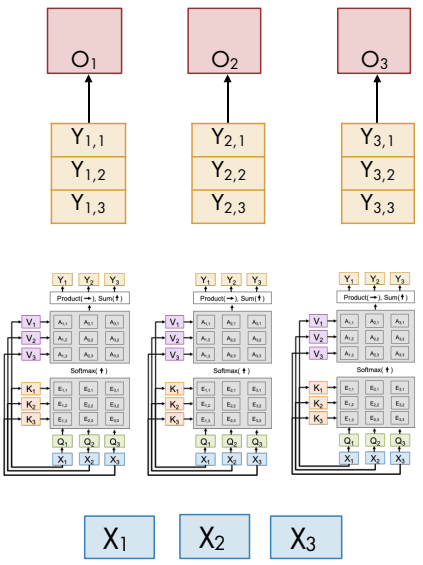


slide credit: Justin Johnson, Fei Fei Li

Multi-Headed Self-Attention Layer

implementation details:

- Each of the H parallel layers use a QKV dim of D_H = "head dim"
- Usually $D_H = D / H$, so the inputs and outputs have the same dimension
- All heads are processed in parallel using batched matrix multiplies

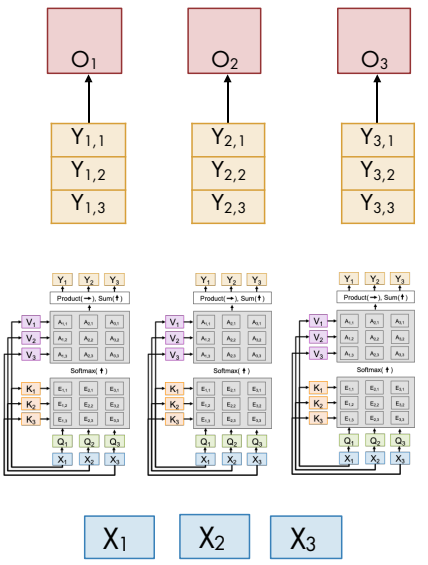


slide credit: Justin Johnson, Fei Fei Li

Multi-Headed Self-Attention Layer

implementation details:

- Each of the H parallel layers use a QKV dim of D_H = "head dim"
- Usually $D_H = D / H$, so the inputs and outputs have the same dimension
- All heads are processed in parallel using batched matrix multiplies
- most architectures are multi-headed in practice



slide credit: Justin Johnson, Fei Fei Li

Self-Attention is Four Matrix Multiplies

Inputs:

Query matrix: Q [D_{in} x D_H]

Data vector: X [N x D]

Key matrix: W_K [D_{in} x HD_H]

Value matrix: W_V [D_{in} x HD_H]

Computation:

Queries: $Q = XW_Q$, [H x N x D_{out}]

Keys: $K = XW_K$, [H x N x D_{out}]

Values: $V = XW_V$, [H x N x D_{out}]

Similarities: $E = QK^T / \sqrt{D_Q}$, [H x N x N]

$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Attention Weights: A = softmax(E, dim=1) [H x N x N]

Head outputs: $Y = AV$ [H x N x D_H]

Outputs: $O = YW_O$ [N x D]

slide credit: Justin Johnson, Fei Fei Li

Self-Attention is Four Matrix Multiplies

Inputs:

Query matrix: Q $[D_{in} \times D_H]$

Data vector: X $[N \times D]$

Key matrix: W_K $[D_{in} \times HD_H]$

Value matrix: W_V $[D_{in} \times HD_H]$

Computation:

Queries: $Q = XW_Q$, $[H \times N \times D_{out}]$

Keys: $K = XW_K$, $[H \times N \times D_{out}]$

Values: $V = XW_V$, $[H \times N \times D_{out}]$

Similarities: $E = QK^T / \sqrt{D_Q}$, $[H \times N \times N]$

$E_{ij} = Q_i K_i^T / \sqrt{D_Q}$

Attention Weights: $A = \text{softmax}(E, \text{dim}=1)$ $[H \times N \times N]$

Head outputs: $Y = AV$ $[H \times N \times D_H]$

Outputs: $O = YW_O$ $[N \times D]$

1. QKV Projection
   $[N \times D]$ $[D \times 3HD_H]$ => $[N \times 3HD_H]$
   Split and reshape to get Q, K, V each of shape $[H \times N \times D_H]$

slide credit: Justin Johnson, Fei Fei Li

Self-Attention is Four Matrix Multiplies

Inputs:

Query matrix: Q [D_{in} x D_H]

Data vector: X [N x D]

Key matrix: W_K [D_{in} x HD_H]

Value matrix: W_V [D_{in} x HD_H]

Computation:

Queries: $Q = XW_Q$, [H x N x D_{out}]

Keys: $K = XW_K$, [H x N x D_{out}]

Values: $V = XW_V$, [H x N x D_{out}]

Similarities: $E = QK^T / \sqrt{D_Q}$, [H x N x N]

$E_{ij} = Q_i K_i^T / \sqrt{D_Q}$

Attention Weights: A = softmax(E, dim=1) [H x N x N]

Head outputs: $Y = AV$ [H x N x D_H]

Outputs: $O = YW_O$ [N x D]

1. QKV Projection
   [N x D] [D x $3HD_H$] => [N x $3HD_H$]
   Split and reshape to get Q, K, V each of shape [H x N x D_H]
2. QK Similarity
   [H x N x D_H] [H x D_H x N] => [H x N x N]

slide credit: Justin Johnson, Fei Fei Li

Self-Attention is Four Matrix Multiplies

Inputs:

Query matrix: Q [D_{in} x D_H]

Data vector: X [N x D]

Key matrix: W_K [D_{in} x HD_H]

Value matrix: W_V [D_{in} x HD_H]

Computation:

Queries: $Q = XW_Q$, [H x N x D_{out}]

Keys: $K = XW_K$, [H x N x D_{out}]

Values: $V = XW_V$, [H x N x D_{out}]

Similarities: $E = QK^T / \sqrt{D_Q}$, [H x N x N]

$E_{ij} = Q_i K_i^T / \sqrt{D_Q}$

Attention Weights: A = softmax(E, dim=1) [H x N x N]

Head outputs: $Y = AV$ [H x N x D_H]

Outputs: $O = YW_O$ [N x D]

1. QKV Projection
   [N x D] [D x $3HD_H$] => [N x $3HD_H$]
   Split and reshape to get Q, K, V each of shape [H x N x D_H]
2. QK Similarity
   [H x N x D_H] [H x D_H x N] => [H x N x N]
3. V-Weighting
   [H x N x N] [H x N x D_H] => [H x N x D_H]
   Reshape to [N x HD_H]

slide credit: Justin Johnson, Fei Fei Li

Self-Attention is Four Matrix Multiplies

Inputs:

Query matrix: Q $[D_{in} \times D_H]$

Data vector: X $[N \times D]$

Key matrix: W_K $[D_{in} \times HD_H]$

Value matrix: W_V $[D_{in} \times HD_H]$

Computation:

Queries: $Q = XW_Q$, $[H \times N \times D_{out}]$

Keys: $K = XW_K$, $[H \times N \times D_{out}]$

Values: $V = XW_V$, $[H \times N \times D_{out}]$

Similarities: $E = QK^T / \sqrt{D_Q}$, $[H \times N \times N]$

$E_{ij} = Q_i K_i^T / \sqrt{D_Q}$

Attention Weights: $A = \text{softmax}(E, \text{dim}=1)$ $[H \times N \times N]$

Head outputs: $Y = AV$ $[H \times N \times D_H]$

Outputs: $O = YW_O$ $[N \times D]$

1. QKV Projection
   $[N \times D]$ $[D \times 3HD_H]$ => $[N \times 3HD_H]$
   Split and reshape to get Q, K, V each of shape $[H \times N \times D_H]$
2. QK Similarity
   $[H \times N \times D_H]$ $[H \times D_H \times N]$ => $[H \times N \times N]$
3. V-Weighting
   $[H \times N \times N]$ $[H \times N \times D_H]$ => $[H \times N \times D_H]$
   Reshape to $[N \times HD_H]$
4. Output Projection
   $[N \times HD_H]$ $[HD_H \times D]$ => $[N \times D]$

slide credit: Justin Johnson, Fei Fei Li

Self-Attention is Four Matrix Multiplies

Inputs:

Query matrix: Q $[D_{in} \times D_H]$

Data vector: X $[N \times D]$

Key matrix: W_K $[D_{in} \times HD_H]$

Value matrix: W_V $[D_{in} \times HD_H]$

Computation:

Queries: $Q = XW_Q$, $[H \times N \times D_{out}]$

Keys: $K = XW_K$, $[H \times N \times D_{out}]$

Values: $V = XW_V$, $[H \times N \times D_{out}]$

Similarities: $E = QK^T / \sqrt{D_Q}$, $[H \times N \times N]$

$$E_{ij} = Q_i K_i^T / \sqrt{D_Q}$$

Attention Weights: $A = \text{softmax}(E, \text{dim}=1)$ $[H \times N \times N]$

Head outputs: $Y = AV$ $[H \times N \times D_H]$

Outputs: $O = YW_O$ $[N \times D]$

1. QKV Projection
   $[N \times D]$ $[D \times 3HD_H]$ => $[N \times 3HD_H]$
   Split and reshape to get Q, K, V each of shape $[H \times N \times D_H]$
2. QK Similarity
   $[H \times N \times D_H]$ $[H \times D_H \times N]$ => $[H \times N \times N]$
3. V-Weighting
   $[H \times N \times N]$ $[H \times N \times D_H]$ => $[H \times N \times D_H]$
   Reshape to $[N \times HD_H]$
4. Output Projection
   $[N \times HD_H]$ $[HD_H \times D]$ => $[N \times D]$

How much compute does this take as the number of vectors N increases?

slide credit: Justin Johnson, Fei Fei Li

Self-Attention is Four Matrix Multiplies

Inputs:

Query matrix: Q $[D_{in} \times D_H]$

Data vector: X $[N \times D]$

Key matrix: W_K $[D_{in} \times HD_H]$

Value matrix: W_V $[D_{in} \times HD_H]$

Computation:

Queries: $Q = XW_Q$, $[H \times N \times D_{out}]$

Keys: $K = XW_K$, $[H \times N \times D_{out}]$

Values: $V = XW_V$, $[H \times N \times D_{out}]$

Similarities: $E = QK^T / \sqrt{D_Q}$, $[H \times N \times N]$

$$E_{ij} = Q_i K_i^T / \sqrt{D_Q}$$

Attention Weights: $A = \text{softmax}(E, \text{dim}=1)$ $[H \times N \times N]$

Head outputs: $Y = AV$ $[H \times N \times D_H]$

Outputs: $O = YW_O$ $[N \times D]$

1. QKV Projection
   $[N \times D]$ $[D \times 3HD_H]$ => $[N \times 3HD_H]$
   Split and reshape to get Q, K, V each of shape $[H \times N \times D_H]$
2. QK Similarity
   $[H \times N \times D_H]$ $[H \times D_H \times N]$ => $[H \times N \times N]$
3. V-Weighting
   $[H \times N \times N]$ $[H \times N \times D_H]$ => $[H \times N \times D_H]$
   Reshape to $[N \times HD_H]$
4. Output Projection
   $[N \times HD_H]$ $[HD_H \times D]$ => $[N \times D]$

How much compute does this take as the number of vectors N increases?

slide credit: Justin Johnson, Fei Fei Li

Self-Attention is Four Matrix Multiplies

Inputs:

Query matrix: Q [D_{in} x D_H]

Data vector: X [N x D]

Key matrix: W_K [D_{in} x HD_H]

Value matrix: W_V [D_{in} x HD_H]

Computation:

Queries: $Q = XW_Q$, [H x N x D_{out}]

Keys: $K = XW_K$, [H x N x D_{out}]

Values: $V = XW_V$, [H x N x D_{out}]

Similarities: $E = QK^T / \sqrt{D_Q}$, [H x N x N]

$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Attention Weights: A = softmax(E, dim=1) [H x N x N]

Head outputs: $Y = AV$ [H x N x D_H]

Outputs: $O = YW_O$ [N x D]

1. QKV Projection
   [N x D] [D x $3HD_H$] => [N x $3HD_H$]
   Split and reshape to get Q, K, V each of shape [H x N x D_H]
2. QK Similarity
   [H x N x D_H] [H x D_H x N] => [H x N x N]
3. V-Weighting
   [H x N x N] [H x N x D_H] => [H x N x D_H]
   Reshape to [N x HD_H]
4. Output Projection
   [N x HD_H] [HD_H x D] => [N x D]

How much compute does this take as the number of vectors N increases? $O(N^2)$

slide credit: Justin Johnson, Fei Fei Li

Self-Attention is Four Matrix Multiplies

Inputs:

Query matrix: Q $[D_{in} \times D_H]$

Data vector: X $[N \times D]$

Key matrix: W_K $[D_{in} \times HD_H]$

Value matrix: W_V $[D_{in} \times HD_H]$

Computation:

Queries: $Q = XW_Q$, $[H \times N \times D_{out}]$

Keys: $K = XW_K$, $[H \times N \times D_{out}]$

Values: $V = XW_V$, $[H \times N \times D_{out}]$

Similarities: $E = QK^T / \sqrt{D_Q}$, $[H \times N \times N]$

$E_{ij} = Q_i K_i^T / \sqrt{D_Q}$

Attention Weights: $A = \text{softmax}(E, \text{dim}=1)$ $[H \times N \times N]$

Head outputs: $Y = AV$ $[H \times N \times D_H]$

Outputs: $O = YW_O$ $[N \times D]$

1. QKV Projection
   $[N \times D]$ $[D \times 3HD_H]$ => $[N \times 3HD_H]$
   Split and reshape to get Q, K, V each of shape $[H \times N \times D_H]$
2. QK Similarity
   $[H \times N \times D_H]$ $[H \times D_H \times N]$ => $[H \times N \times N]$
3. V-Weighting
   $[H \times N \times N]$ $[H \times N \times D_H]$ => $[H \times N \times D_H]$
   Reshape to $[N \times HD_H]$
4. Output Projection
   $[N \times HD_H]$ $[HD_H \times D]$ => $[N \times D]$

How much memory does this take as the number of vectors N increases?

slide credit: Justin Johnson, Fei Fei Li

Self-Attention is Four Matrix Multiplies

Inputs:

Query matrix: Q $[D_{in} \times D_H]$

Data vector: X $[N \times D]$

Key matrix: W_K $[D_{in} \times HD_H]$

Value matrix: W_V $[D_{in} \times HD_H]$

Computation:

Queries: $Q = XW_Q$, $[H \times N \times D_{out}]$

Keys: $K = XW_K$, $[H \times N \times D_{out}]$

Values: $V = XW_V$, $[H \times N \times D_{out}]$

Similarities: $E = QK^T / \sqrt{D_Q}$, $[H \times N \times N]$

$E_{ij} = Q_i K_i^T / \sqrt{D_Q}$

Attention Weights: $A = \text{softmax}(E, \text{dim}=1)$ $[H \times N \times N]$

Head outputs: $Y = AV$ $[H \times N \times D_H]$

Outputs: $O = YW_O$ $[N \times D]$

1. QKV Projection
   $[N \times D]$ $[D \times 3HD_H]$ => $[N \times 3HD_H]$
   Split and reshape to get Q, K, V each of shape $[H \times N \times D_H]$
2. QK Similarity
   $[H \times N \times D_H]$ $[H \times D_H \times N]$ => $[H \times N \times N]$
3. V-Weighting
   $[H \times N \times N]$ $[H \times N \times D_H]$ => $[H \times N \times D_H]$
   Reshape to $[N \times HD_H]$
4. Output Projection
   $[N \times HD_H]$ $[HD_H \times D]$ => $[N \times D]$

How much memory does this take as the number of vectors N increases?

slide credit: Justin Johnson, Fei Fei Li

Self-Attention is Four Matrix Multiplies

Inputs:

Query matrix: Q [D_{in} x D_H]

Data vector: X [N x D]

Key matrix: W_K [D_{in} x HD_H]

Value matrix: W_V [D_{in} x HD_H]

Computation:

Queries: $Q = XW_Q$, [H x N x D_{out}]

Keys: $K = XW_K$, [H x N x D_{out}]

Values: $V = XW_V$, [H x N x D_{out}]

Similarities: $E = QK^T / \sqrt{D_Q}$, [H x N x N]

$E_{ij} = Q_i K_j^T / \sqrt{D_Q}$

Attention Weights: A = softmax(E, dim=1) [H x N x N]

Head outputs: $Y = AV$ [H x N x D_H]

Outputs: $O = YW_O$ [N x D]

1. QKV Projection
   [N x D] [D x $3HD_H$] => [N x $3HD_H$]
   Split and reshape to get Q, K, V each of shape [H x N x D_H]
2. QK Similarity
   [H x N x D_H] [H x D_H x N] => [H x N x N]
3. V-Weighting
   [H x N x N] [H x N x D_H] => [H x N x D_H]
   Reshape to [N x HD_H]
4. Output Projection
   [N x HD_H] [HD_H x D] => [N x D]

How much memory does this take as the number of vectors N increases? $O(N^2)$

slide credit: Justin Johnson, Fei Fei Li

Self-Attention is Four Matrix Multiplies

Inputs:

Query matrix: Q $[D_{in} \times D_H]$

Data vector: X $[N \times D]$

Key matrix: W_K $[D_{in} \times HD_H]$

Value matrix: W_V $[D_{in} \times HD_H]$

Computation:

Queries: $Q = XW_Q$, $[H \times N \times D_{out}]$

Keys: $K = XW_K$, $[H \times N \times D_{out}]$

Values: $V = XW_V$, $[H \times N \times D_{out}]$

Similarities: $E = QK^T / \sqrt{D_Q}$, $[H \times N \times N]$

$$E_{ij} = Q_i K_i^T / \sqrt{D_Q}$$

Attention Weights: $A = \text{softmax}(E, \text{dim}=1)$ $[H \times N \times N]$

Head outputs: $Y = AV$ $[H \times N \times D_H]$

Outputs: $O = YW_O$ $[N \times D]$

1. QKV Projection
   $[N \times D]$ $[D \times 3HD_H]$ => $[N \times 3HD_H]$
   Split and reshape to get Q, K, V each of shape $[H \times N \times D_H]$
2. QK Similarity
   $[H \times N \times D_H]$ $[H \times D_H \times N]$ => $[H \times N \times N]$
3. V-Weighting
   $[H \times N \times N]$ $[H \times N \times D_H]$ => $[H \times N \times D_H]$
   Reshape to $[N \times HD_H]$
4. Output Projection
   $[N \times HD_H]$ $[HD_H \times D]$ => $[N \times D]$

If N=100K, H=64 then HxNxN attention weights take 1.192 TB! GPUs don't have that much memory...

slide credit: Justin Johnson, Fei Fei Li

Self-Attention is Four Matrix Multiplies

Inputs:

Query matrix: Q $[D_{in} \times D_H]$

Data vector: X $[N \times D]$

Key matrix: W_K $[D_{in} \times HD_H]$

Value matrix: W_V $[D_{in} \times HD_H]$

Computation:

Queries: $Q = XW_Q$, $[H \times N \times D_{out}]$

Keys: $K = XW_K$, $[H \times N \times D_{out}]$

Values: $V = XW_V$, $[H \times N \times D_{out}]$

Similarities: $E = QK^T / \sqrt{D_Q}$, $[H \times N \times N]$

$E_{ij} = Q_i K_i^T / \sqrt{D_Q}$

Attention Weights: $A = \text{softmax}(E, \text{dim}=1)$ $[H \times N \times N]$

Head outputs: $Y = AV$ $[H \times N \times D_H]$

Outputs: $O = YW_O$ $[N \times D]$

1. QKV Projection
   $[N \times D]$ $[D \times 3HD_H]$ => $[N \times 3HD_H]$
   Split and reshape to get Q, K, V each of shape $[H \times N \times D_H]$
2. QK Similarity
   $[H \times N \times D_H]$ $[H \times D_H \times N]$ => $[H \times N \times N]$
3. V-Weighting
   $[H \times N \times N]$ $[H \times N \times D_H]$ => $[H \times N \times D_H]$
   Reshape to $[N \times HD_H]$
4. Output Projection
   $[N \times HD_H]$ $[HD_H \times D]$ => $[N \times D]$

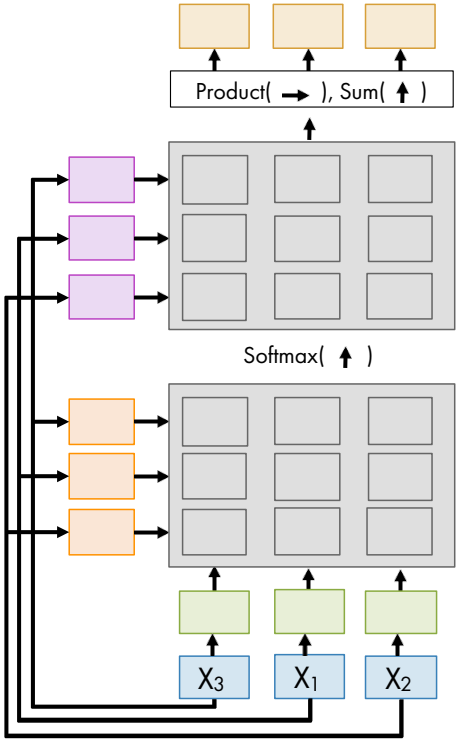
FlashAttention reduces memory to O(N) by processing in tiles w/out storing full NxN matrix.

"Flashattention: Fast and memory-efficient exact attention with IO-awareness."

slide credit: Justin Johnson, Fei Fei Li

Self-Attention Layer

Consider permuting inputs

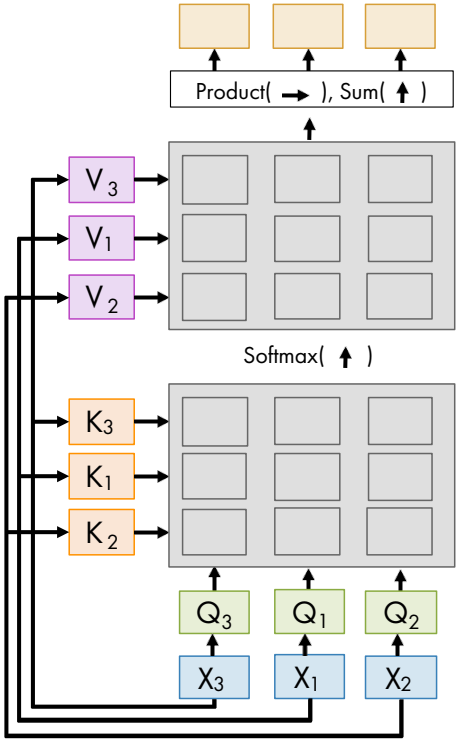


slide credit: Justin Johnson, Fei Fei Li

Self-Attention Layer

Consider permuting inputs

- Queries, keys, and values will be the same but permuted

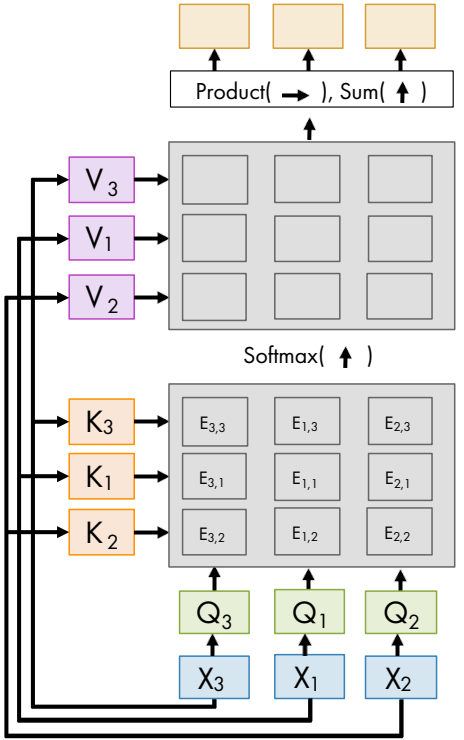


slide credit: Justin Johnson, Fei Fei Li

Self-Attention Layer

Consider permuting inputs

- Queries, keys, and values will be the same but permuted
- Similarities are the same but permuted

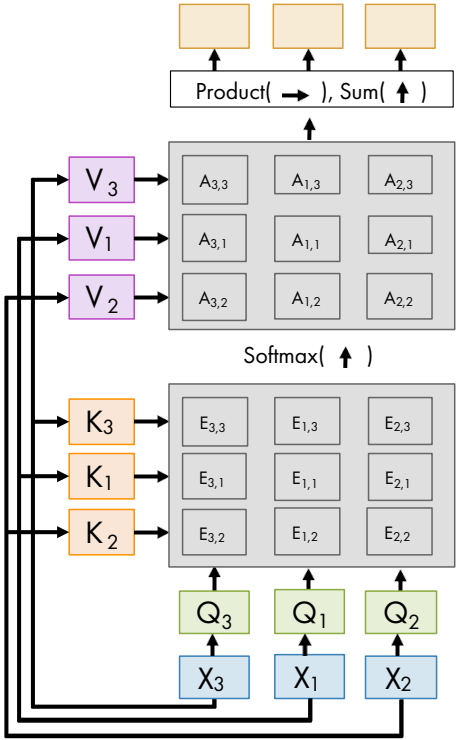


slide credit: Justin Johnson, Fei Fei Li

Self-Attention Layer

Consider permuting inputs

- Queries, keys, and values will be the same but permuted
- Similarities are the same but permuted
- Attention weights are the same but permuted

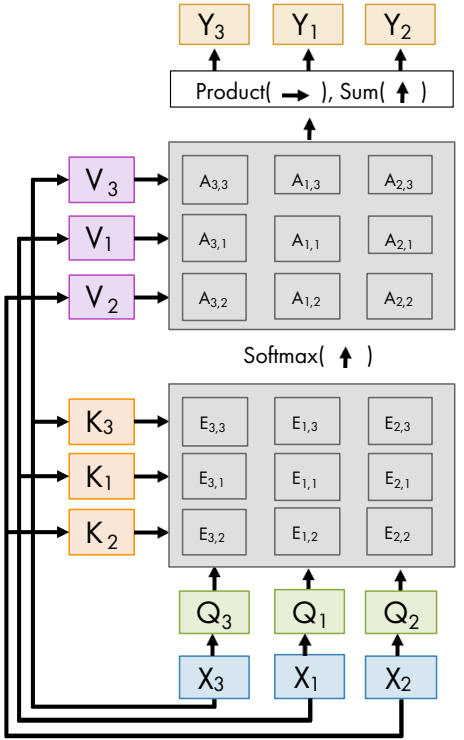


slide credit: Justin Johnson, Fei Fei Li

Self-Attention Layer

Consider permuting inputs

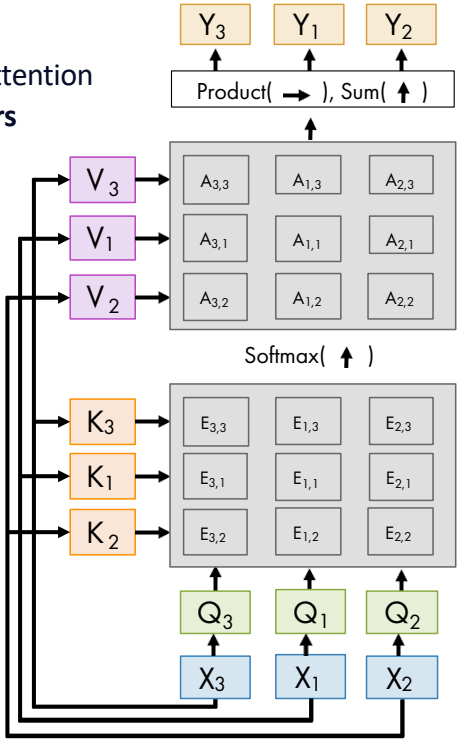
- Queries, keys, and values will be the same but permuted
- Similarities are the same but permuted
- Attention weights are the same but permuted
- Outputs are the same but permuted



slide credit: Justin Johnson, Fei Fei Li

Self-Attention Layer

This means that Self-Attention works on **sets of vectors**



Consider permuting inputs

- Queries, keys, and values will be the same but permuted
- Similarities are the same but permuted
- Attention weights are the same but permuted
- Outputs are the same but permuted
- Self-Attention is permutation equivariant: $F(\sigma(X)) = \sigma(F(X))$

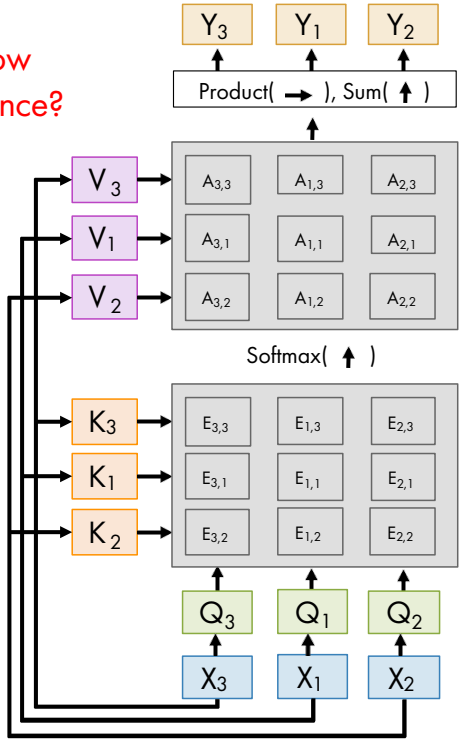
slide credit: Justin Johnson, Fei Fei Li

Self-Attention Layer

Does self-attention know the order of the sequence?

Consider permuting inputs

- Queries, keys, and values will be the same but permuted
- Similarities are the same but permuted
- Attention weights are the same but permuted
- Outputs are the same but permuted
- Self-Attention is permutation equivariant: $F(\sigma(X)) = \sigma(F(X))$



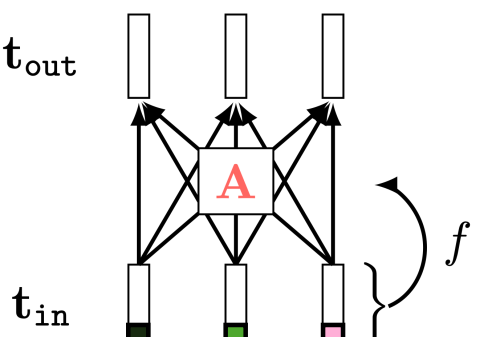
slide credit: Justin Johnson, Fei Fei Li

New idea #3: positional encoding

What if you don't want to be permutation invariant?

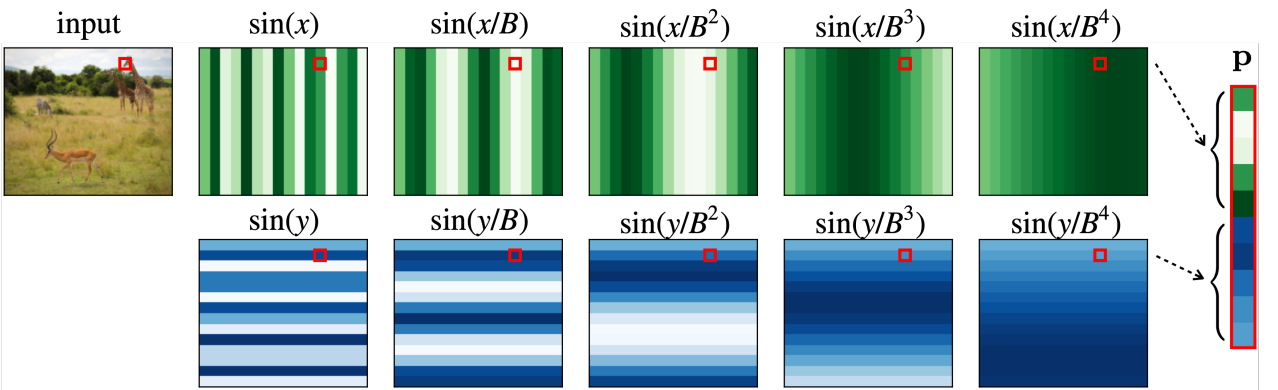
1. Use an architecture that is not permutation invariant (e.g., MLP)

2. Add location information to the token code vectors — this is called positional encoding



Fourier positional codes

Represent coordinates on Fourier basis



Outline

- motivation / why use different architectures?
- limitations of CNNs
- attention
- transformers
- generative models

Transformers

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Vaswani, Ashish, et al. "Attention is all you need." *Proc. NeurIPS* (2017).

The Transformer

Transformer Block

Input: Set of vectors x

Vaswani et al, "Attention is all you need," NeurIPS 2017

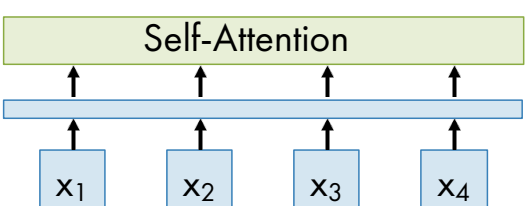
slide credit: Justin Johnson, Fei Fei Li

The Transformer

Transformer Block

Input: Set of vectors x

All vectors interact through
(multiheaded) Self-Attention



Vaswani et al, "Attention is all you need," NeurIPS 2017

slide credit: Justin Johnson, Fei Fei Li

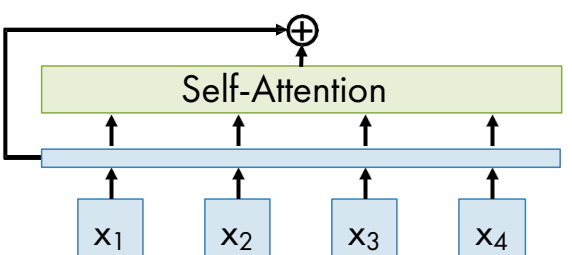
The Transformer

Transformer Block

Input: Set of vectors x

Residual connection

All vectors interact through
(multiheaded) Self-Attention



Vaswani et al, "Attention is all you need," NeurIPS 2017

slide credit: Justin Johnson, Fei Fei Li

The Transformer

Transformer Block

Input: Set of vectors x

Recall Layer Normalization:

Given $h_1, ..., h_N$ (Shape: D)

scale: γ (Shape: D)

shift: β (Shape: D)

$\mu_i = (\sum_i h_{i,i})/D$ (scalar)

$\sigma_i = (\sum_i (h_{i,i} - \mu_i)^2/D)^{1/2}$ (scalar)

$z_i = (h_i - \mu_i) / \sigma_i$

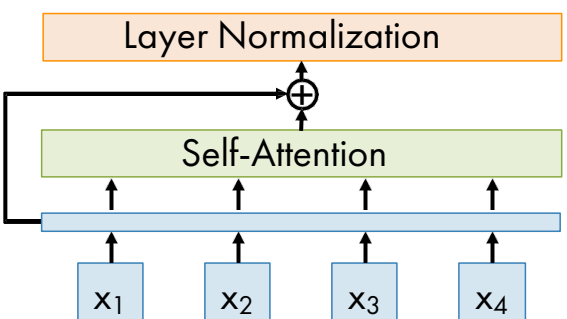
$y_i = \gamma * z_i + \beta$

Ba et al, 2016

Layer normalization normalizes all vectors

Residual connection

All vectors interact through (multiheaded) Self-Attention



Vaswani et al, "Attention is all you need," NeurIPS 2017

slide credit: Justin Johnson, Fei Fei Li

The Transformer

Transformer Block

Input: Set of vectors x

Usually a two-layer MLP;
classic setup is
D => 4D => D

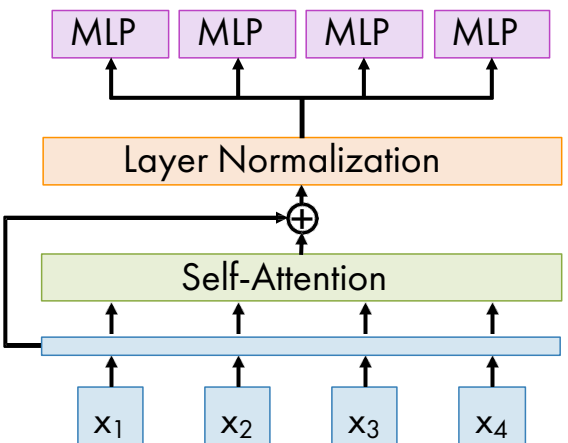
Also sometimes called FFN
(Feed-Forward Network)

MLP independently
on each vector

Layer normalization
normalizes all vectors

Residual connection

All vectors interact through
(multiheaded) Self-Attention



Vaswani et al, "Attention is all you need," NeurIPS 2017

slide credit: Justin Johnson, Fei Fei Li

The Transformer

Transformer Block

Input: Set of vectors x

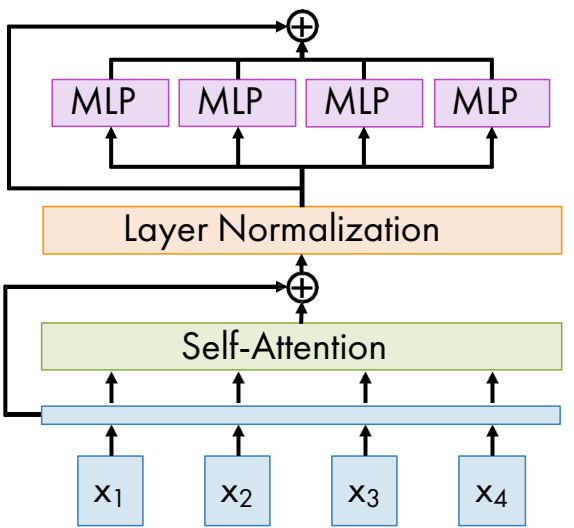
Residual connection

MLP independently on each vector

Layer normalization normalizes all vectors

Residual connection

All vectors interact through (multiheaded) Self-Attention



Vaswani et al, "Attention is all you need," NeurIPS 2017

slide credit: Justin Johnson, Fei Fei Li

The Transformer

Transformer Block

Input: Set of vectors x

Another Layer Norm

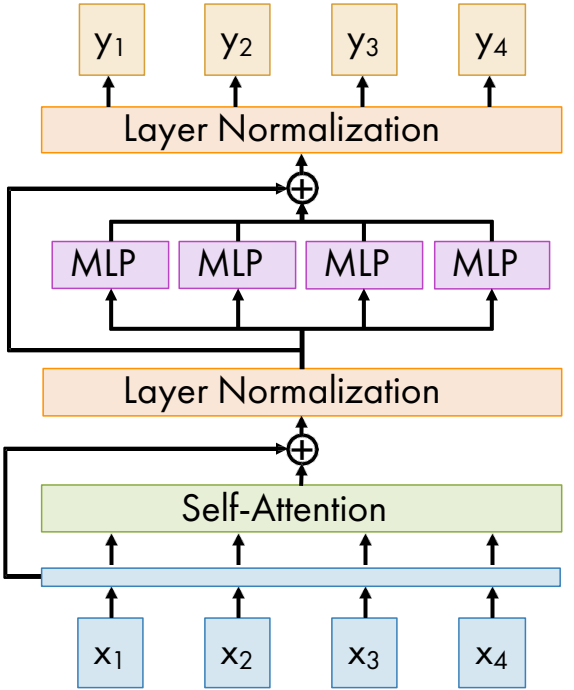
Residual connection

MLP independently on each vector

Layer normalization normalizes all vectors

Residual connection

All vectors interact through (multiheaded) Self-Attention



Vaswani et al, "Attention is all you need," NeurIPS 2017

slide credit: Justin Johnson, Fei Fei Li

The Transformer

Transformer Block

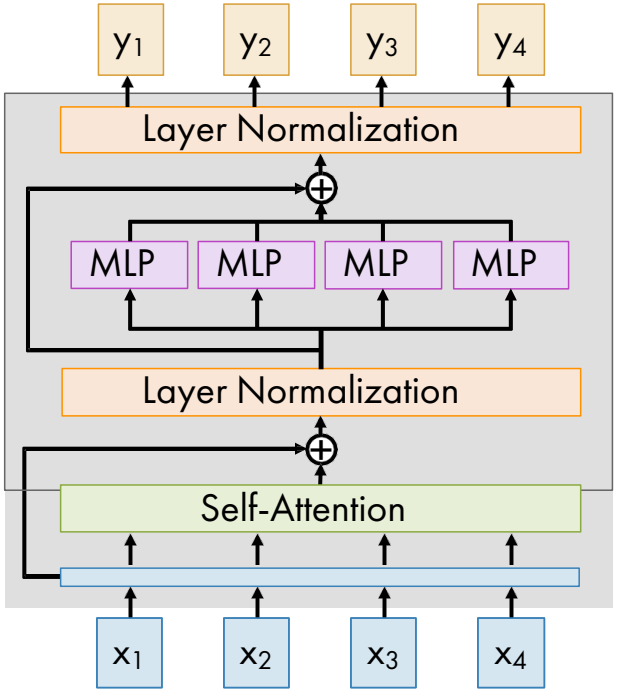
Input: Set of vectors x
Output: Set of vectors y

Self-Attention is the only interaction between vectors

LayerNorm and MLP work on each vector independently

Highly scalable and parallelizable, most of the compute is just 6 matmuls:

4 from Self-Attention
2 from MLP



Vaswani et al, "Attention is all you need," NeurIPS 2017

slide credit: Justin Johnson, Fei Fei Li

The Transformer

Transformer Block

Input: Set of vectors x
Output: Set of vectors y

Self-Attention is the only interaction between vectors

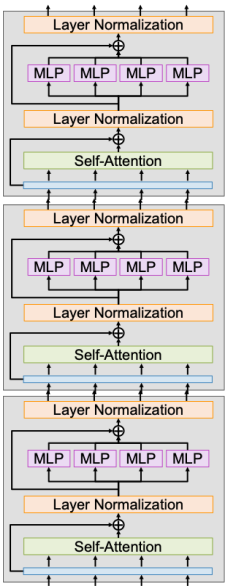
LayerNorm and MLP work on each vector independently

Highly scalable and parallelizable, most of the compute is just 6 matmuls:

4 from Self-Attention
2 from MLP

A Transformer is just a stack of identical Transformer blocks!

They have not changed much since 2017... but have gotten a lot bigger



Vaswani et al, "Attention is all you need," NeurIPS 2017

slide credit: Justin Johnson, Fei Fei Li

The Transformer

Transformer Block

Input: Set of vectors x
Output: Set of vectors y

Self-Attention is the only interaction between vectors

LayerNorm and MLP work on each vector independently

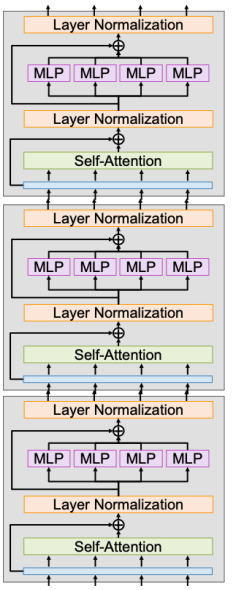
Highly scalable and parallelizable, most of the compute is just 6 matmuls:

4 from Self-Attention
2 from MLP

A Transformer is just a stack of identical Transformer blocks!

They have not changed much since 2017... but have gotten a lot bigger

Original: [Vaswani et al, 2017]
12 blocks, D=1024, H=16, N=512
213M params



Vaswani et al, "Attention is all you need," NeurIPS 2017

slide credit: Justin Johnson, Fei Fei Li

The Transformer

Transformer Block

Input: Set of vectors x
Output: Set of vectors y

Self-Attention is the only interaction between vectors

LayerNorm and MLP work on each vector independently

Highly scalable and parallelizable, most of the compute is just 6 matmuls:

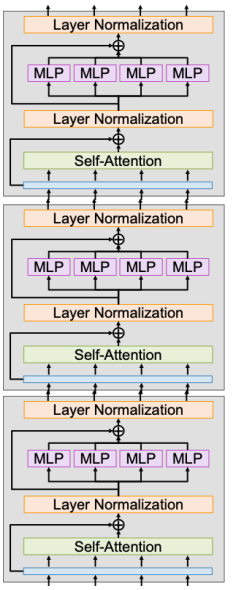
4 from Self-Attention
2 from MLP

A Transformer is just a stack of identical Transformer blocks!

They have not changed much since 2017... but have gotten a lot bigger

Original: [Vaswani et al, 2017]
12 blocks, D=1024, H=16, N=512
213M params

GPT-2: [Radford et al, 2019]
48 blocks, D=1600, H=25, N=1024
1.5B params



Vaswani et al, "Attention is all you need," NeurIPS 2017

slide credit: Justin Johnson, Fei Fei Li

The Transformer

Transformer Block

Input: Set of vectors x
Output: Set of vectors y

Self-Attention is the only interaction between vectors

LayerNorm and MLP work on each vector independently

Highly scalable and parallelizable, most of the compute is just 6 matmuls:

4 from Self-Attention
2 from MLP

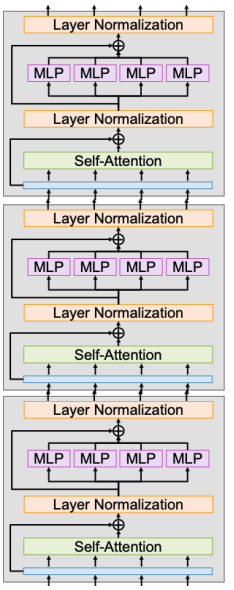
A Transformer is just a stack of identical Transformer blocks!

They have not changed much since 2017... but have gotten a lot bigger

Original: [Vaswani et al, 2017]
12 blocks, D=1024, H=16, N=512
213M params

GPT-2: [Radford et al, 2019]
48 blocks, D=1600, H=25, N=1024
1.5B params

GPT-3: [Brown et al, 2020]
96 blocks, D=12288, H=96, N=2048
175B params



Vaswani et al, "Attention is all you need," NeurIPS 2017

slide credit: Justin Johnson, Fei Fei Li

Vision Transformers (ViT)

Input image:
e.g. 224x224x3

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

slide credit: Justin Johnson, Fei Fei Li

Vision Transformers (ViT)

Input image:
e.g. 224x224x3

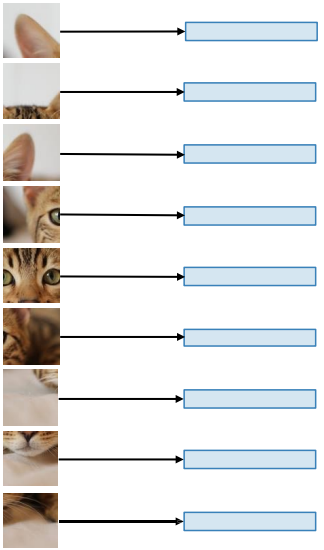
Break into patches
e.g. 16x16x3

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

slide credit: Justin Johnson, Fei Fei Li

Vision Transformers (ViT)

Input image:
e.g. 224x224x3



Break into patches
e.g. 16x16x3

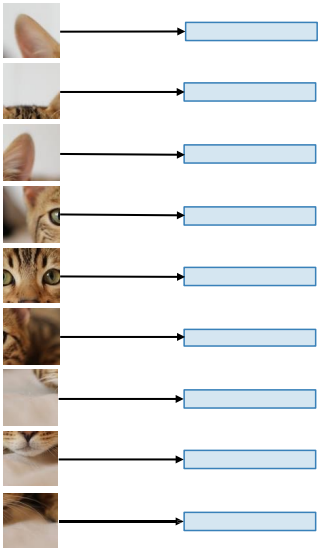
Flatten and apply a linear
transform 768 => D

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

slide credit: Justin Johnson, Fei Fei Li

Vision Transformers (ViT)

Input image:
e.g. 224x224x3



Break into patches
e.g. 16x16x3

Flatten and apply a linear transform 768 => D

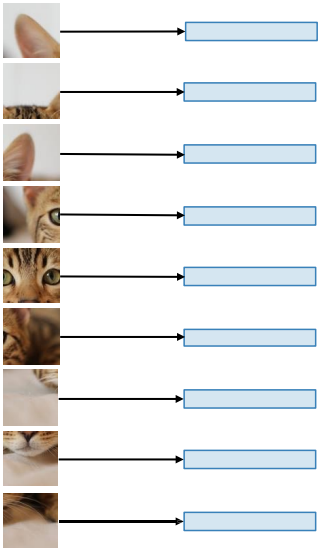
Q: Any other way to describe this operation?

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

slide credit: Justin Johnson, Fei Fei Li

Vision Transformers (ViT)

Input image:
e.g. 224x224x3



Break into patches
e.g. 16x16x3

Flatten and apply a linear transform 768 => D

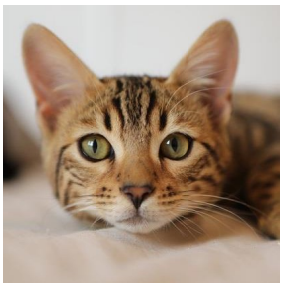
Q: Any other way to describe this operation?

A: 16x16 conv with stride 16, 3 input channels, D output channels

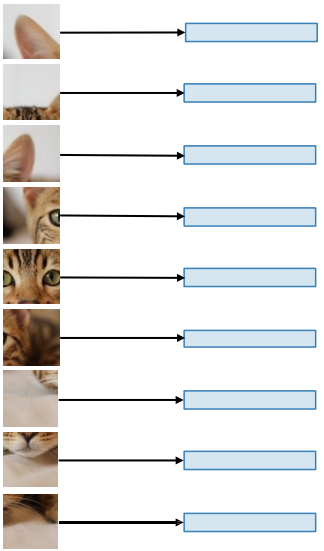
Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

slide credit: Justin Johnson, Fei Fei Li

Vision Transformers (ViT)

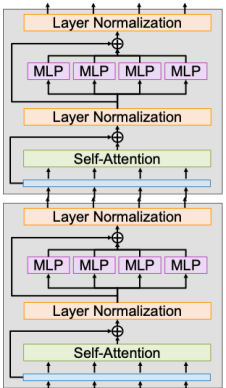


Input image:
e.g. 224x224x3



Break into patches
e.g. 16x16x3

Flatten and apply a linear transform 768 => D

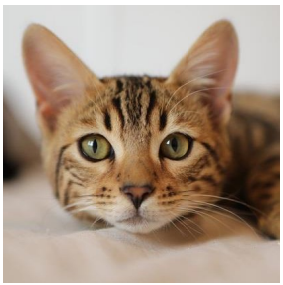


D-dim vector per patch are the input vectors to the Transformer

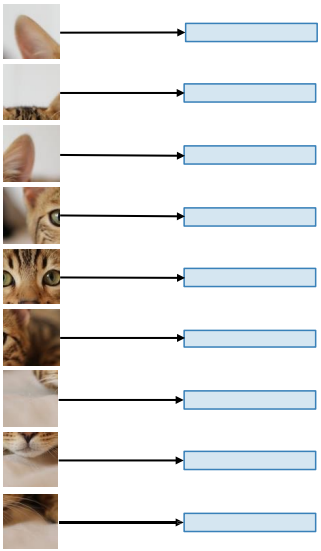
Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

slide credit: Justin Johnson, Fei Fei Li

Vision Transformers (ViT)

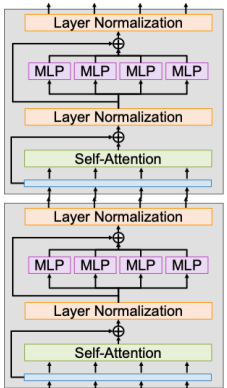


Input image:
e.g. 224x224x3



Break into patches
e.g. 16x16x3

Flatten and apply a linear transform 768 => D



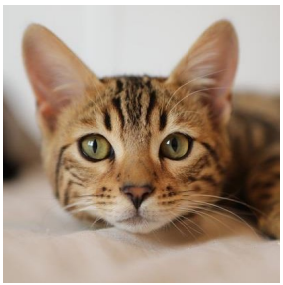
D-dim vector per patch are the input vectors to the Transformer

Use positional encoding to tell the transformer the 2D position of each patch

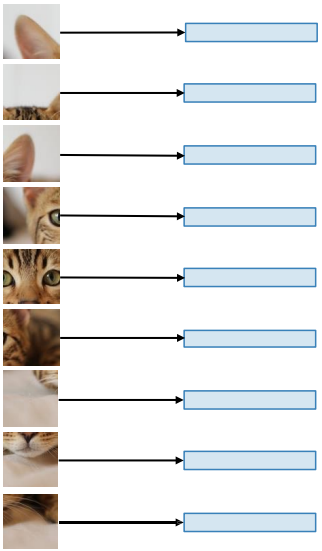
Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

slide credit: Justin Johnson, Fei Fei Li

Vision Transformers (ViT)

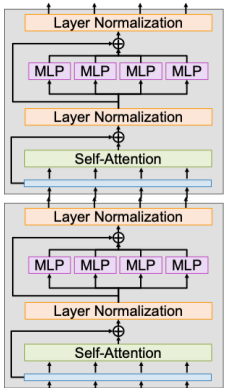


Input image:
e.g. 224x224x3



Break into patches
e.g. 16x16x3

Flatten and apply a linear transform 768 => D



D-dim vector per patch are the input vectors to the Transformer

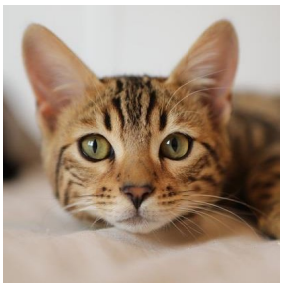
Don't use any masking; each image patch can look at all other image patches

Use positional encoding to tell the transformer the 2D position of each patch

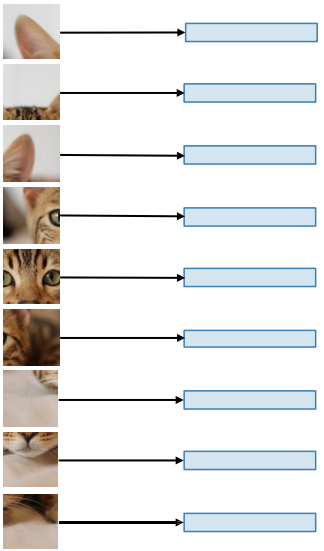
Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

slide credit: Justin Johnson, Fei Fei Li

Vision Transformers (ViT)

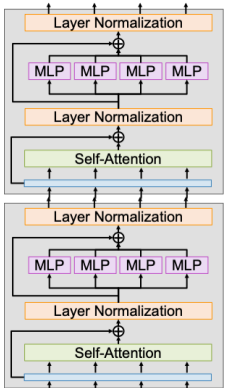


Input image:
e.g. 224x224x3



Break into patches
e.g. 16x16x3

Flatten and apply a linear transform 768 => D



D-dim vector per patch are the input vectors to the Transformer

Transformer gives an output vector per patch

Don't use any masking; each image patch can look at all other image patches

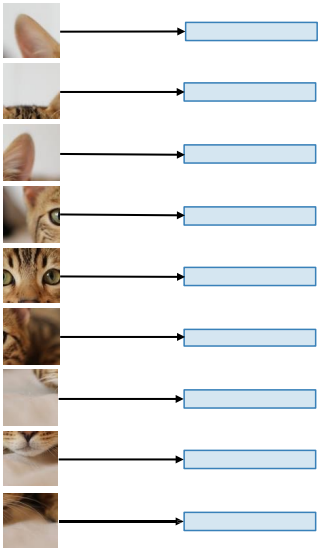
Use positional encoding to tell the transformer the 2D position of each patch

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

slide credit: Justin Johnson, Fei Fei Li

Vision Transformers (ViT)

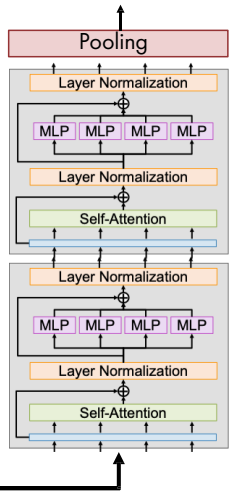
Input image:
e.g. 224x224x3



Break into patches
e.g. 16x16x3

Flatten and apply a linear transform 768 => D

Average pool NxD vectors to 1xD, apply a linear layer D=>C to predict class scores



D-dim vector per patch are the input vectors to the Transformer

Transformer gives an output vector per patch

Don't use any masking; each image patch can look at all other image patches

Use positional encoding to tell the transformer the 2D position of each patch

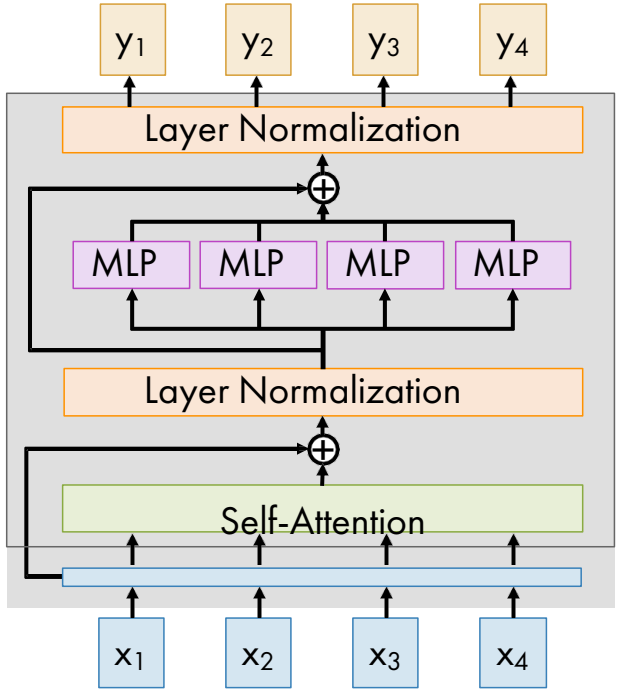
Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021

slide credit: Justin Johnson, Fei Fei Li

Tweaking Transformers

The Transformer architecture has not changed much since 2017.

But a few changes have become common:

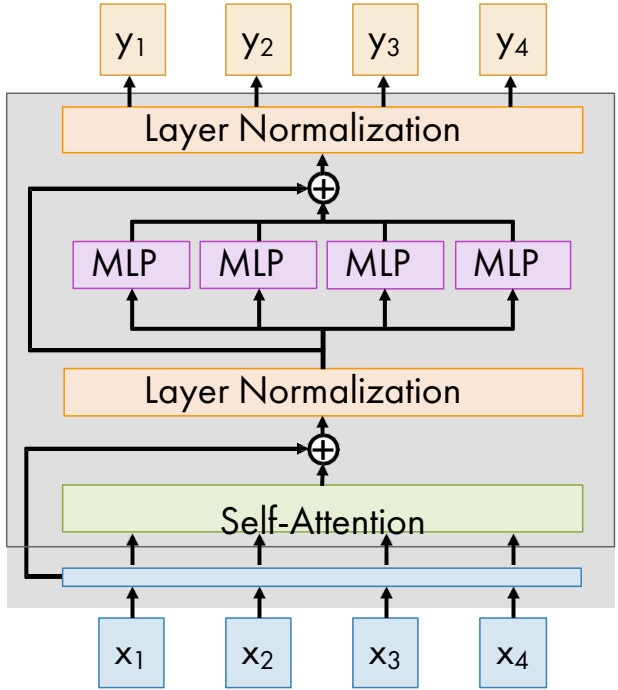


slide credit: Justin Johnson, Fei Fei Li

Pre-Norm Transformer

Layer normalization is outside the residual connections

Kind of weird, the model can't actually learn the identify function



Baevski & Auli, "Adaptive Input Representations for Neural Language Modeling", arXiv 2018

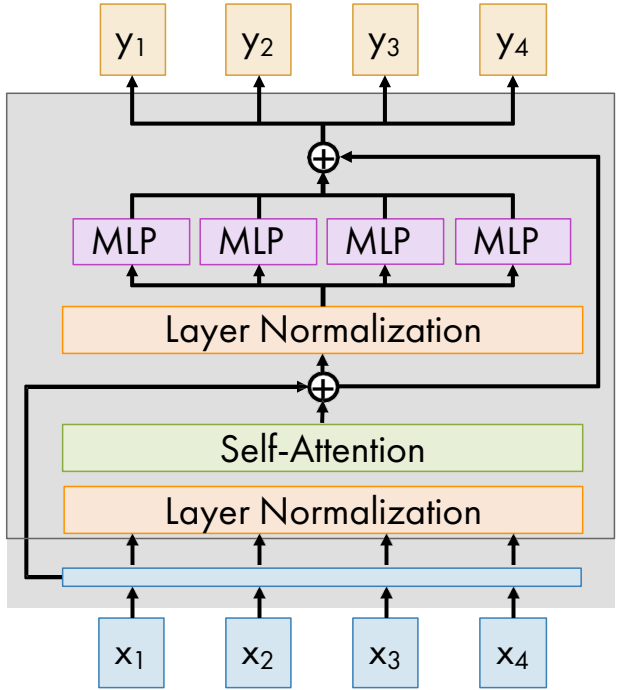
slide credit: Justin Johnson, Fei Fei Li

Pre-Norm Transformer

Layer normalization is outside the residual connections

Kind of weird, the model can't actually learn the identify function

Solution: Move layer normalization before the Self- Attention and MLP, inside the residual connections. Training is more stable.



Baevski & Auli, "Adaptive Input Representations for Neural Language Modeling", arXiv 2018

slide credit: Justin Johnson, Fei Fei Li

RMSNorm

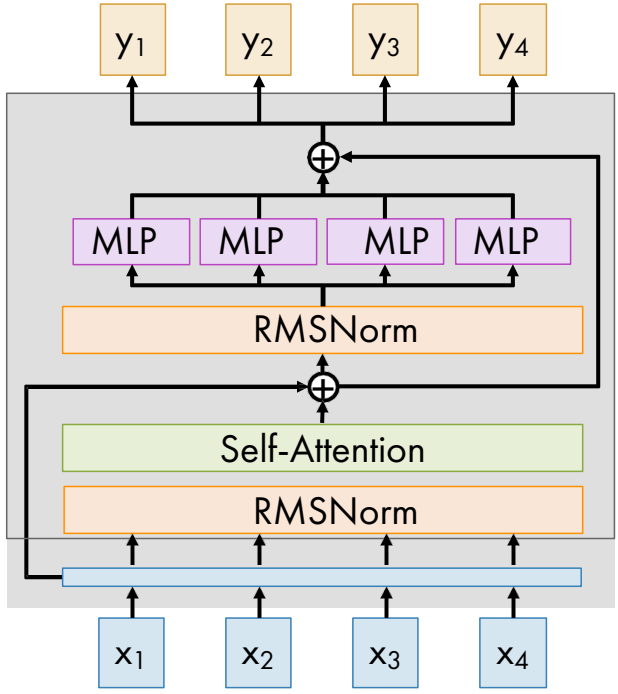
Replace Layer Normalization with Root-Mean-Square Normalization (RMSNorm)

Input: x [shape D]
Output: y [shape D]
Weight: γ [shape D]

$$y_i = \frac{x_i}{RMS(x)} * \gamma_i$$

$$RMS(x) = \sqrt{\varepsilon + \frac{1}{N}\sum_{i=1}^{N} x_i^2}$$

Training is a bit more stable



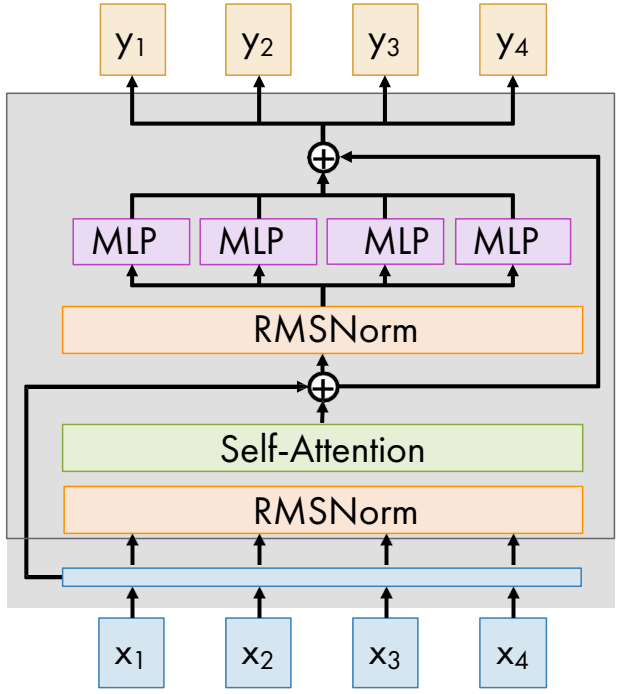
Zhang and Sennrich, "Root Mean Square Layer Normalization", NeurIPS 2019

slide credit: Justin Johnson, Fei Fei Li

SwiGLU MLP

Classic MLP:

Input: X [N x D]
Weights: W_1 [D x 4D]
W_2 [4D x D]
Output: $Y = \sigma(XW_1)W_2$ [N x D]



Shazeer, "GLU Variants Improve Transformers", 2020

slide credit: Justin Johnson, Fei Fei Li

SwiGLU MLP

Classic MLP:

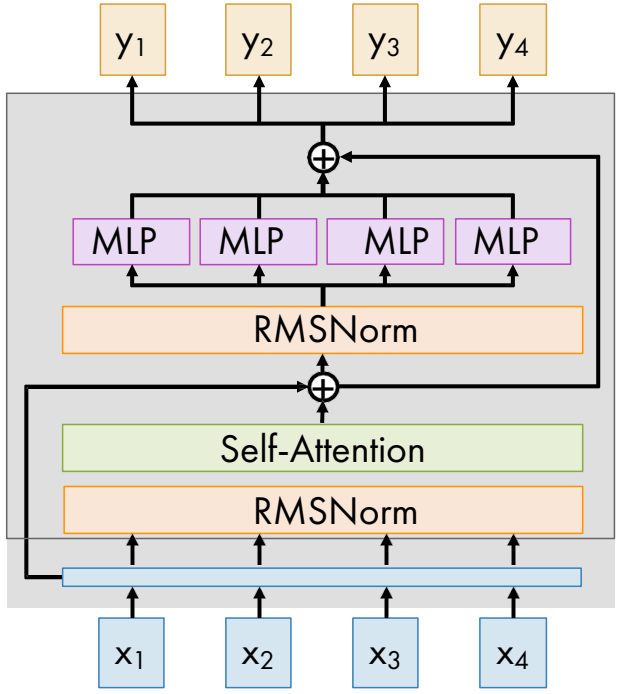
Input: X [N x D]
Weights: W_1 [D x 4D]
W_2 [4D x D]
Output: $Y = \sigma(XW_1)W_2$ [N x D]

SwiGLU MLP:

Input: X [N x D]
Weights: W_1 , W_2 [D x H]
W_3 [H x D]
Output:

$$Y = (\sigma(XW_1) \odot XW_2)W_3$$

Setting H = 8D/3 keeps same total params



Shazeer, "GLU Variants Improve Transformers", 2020

slide credit: Justin Johnson, Fei Fei Li

SwiGLU MLP

Classic MLP:

Input: X [N x D]
Weights: W_1 [D x 4D]
W_2 [4D x D]
Output: Y = $\sigma(XW_1)W_2$ [N x D]

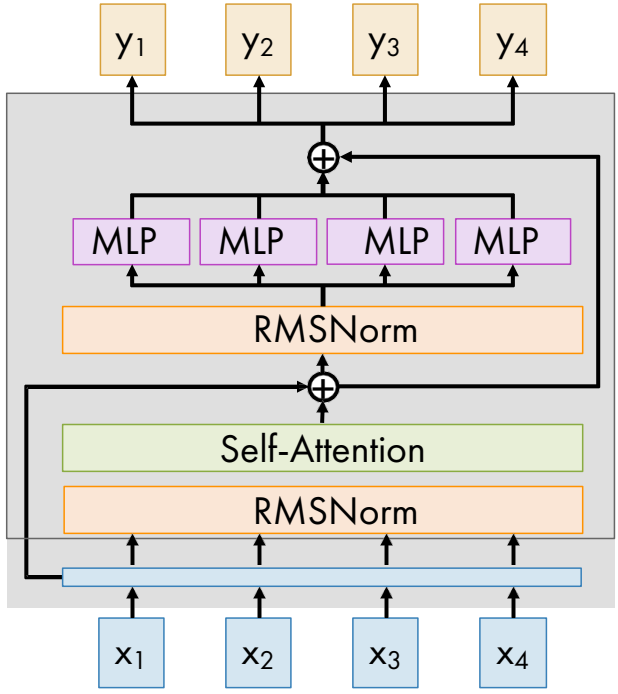
SwiGLU MLP:

Input: X [N x D]
Weights: W_1 , W_2 [D x H]
W_3 [H x D]
Output:

$$Y = (\sigma(XW_1) \odot XW_2)W_3$$

Setting H = 8D/3 keeps same total params

We offer no explanation as to why these architectures seem to work; we attribute their success, as all else, to divine benevolence.



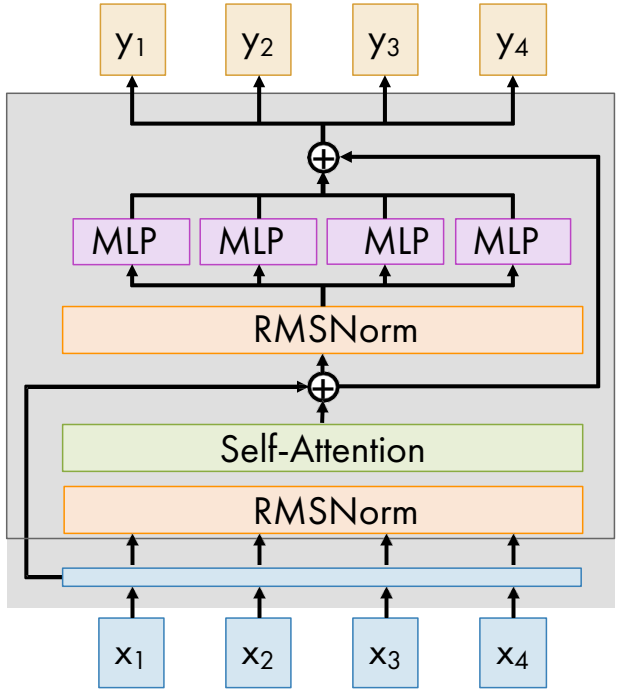
Shazeer, "GLU Variants Improve Transformers", 2020

slide credit: Justin Johnson, Fei Fei Li

Mixture of Experts (MoE)

Learn E separate sets of MLP weights in each block; each MLP is an expert

W_1: [D x 4D] => [E x D x 4D]
W_2: [4D x D] => [E x 4D x D]



Shazeer et al, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer", 2017

slide credit: Justin Johnson, Fei Fei Li

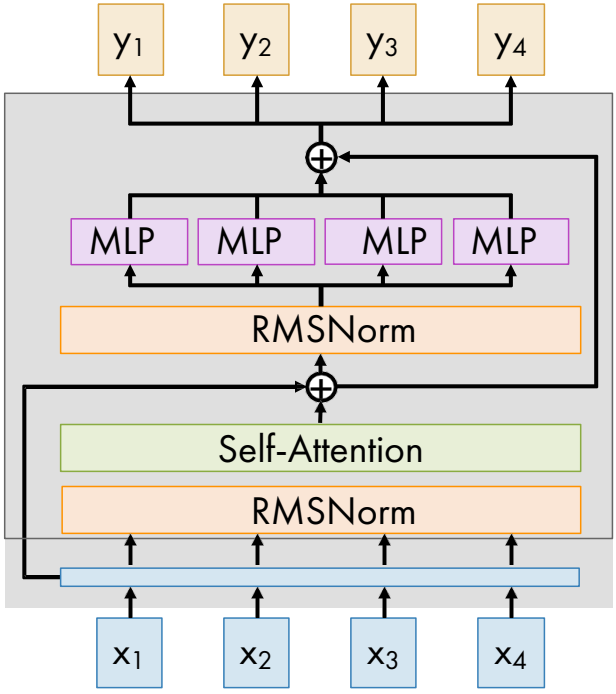
Mixture of Experts (MoE)

Learn E separate sets of MLP weights in each block; each MLP is an expert

W_1: [D x 4D] => [E x D x 4D]
W_2: [4D x D] => [E x 4D x D]

Each token gets routed to A < E of the experts. These are the active experts.

Increases params by E,
But only increases compute by A



Shazeer et al, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer", 2017

slide credit: Justin Johnson, Fei Fei Li

Mixture of Experts (MoE)

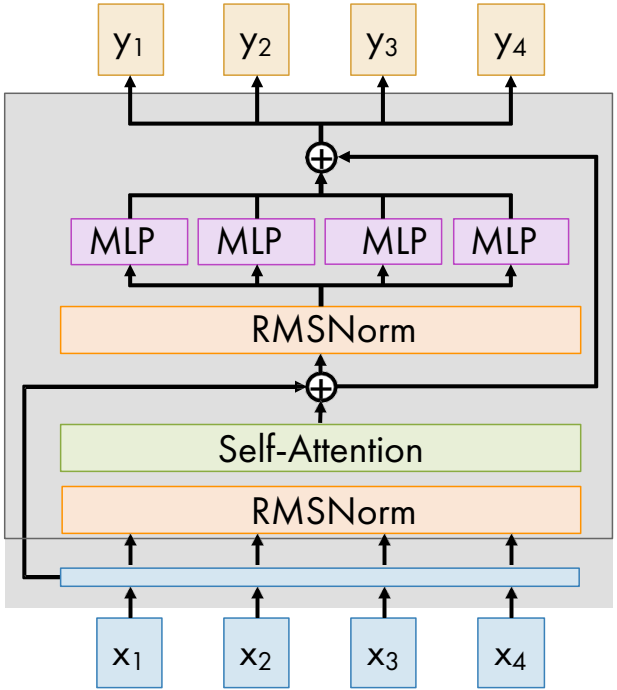
Learn E separate sets of MLP weights in each block; each MLP is an expert

W_1: [D x 4D] => [E x D x 4D]
W_2: [4D x D] => [E x 4D x D]

Each token gets routed to A < E of the experts. These are the active experts.

Increases params by E,
But only increases compute by A

All of the biggest LLMs today (e.g. GPT4o, GPT4.5, Claude 3.7, Gemini 2.5 Pro, etc) almost certainly use MoE and have > 1T params; but they don't publish details anymore



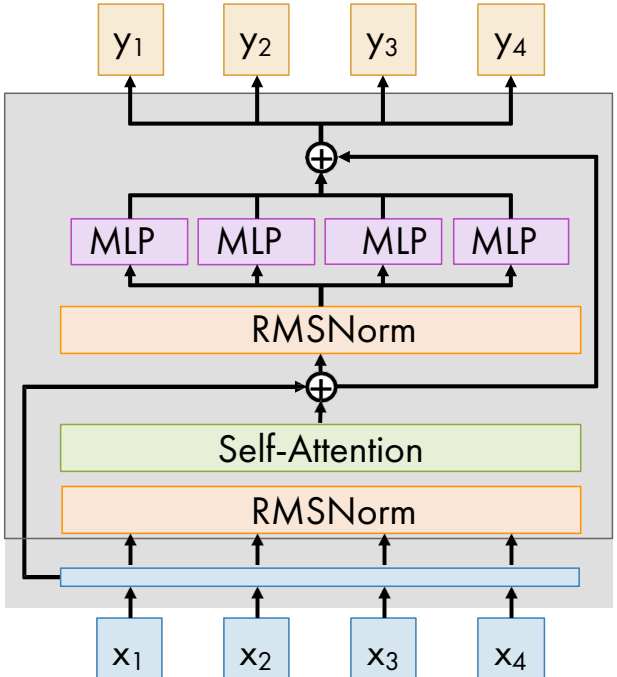
slide credit: Justin Johnson, Fei Fei Li

Tweaking Transformers

The Transformer architecture has not changed much since 2017.

But a few changes have become common:

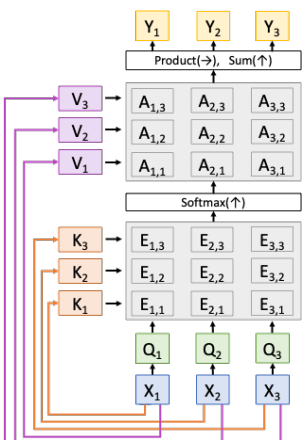
- Pre-Norm: Move normalization inside residual
- RMSNorm: Different normalization layer
- SwiGLU: Different MLP architecture
- Mixture of Experts (MoE): Learn E different MLPs, use A < E of them per token. Massively increase params, modest increase to compute cost.



slide credit: Justin Johnson, Fei Fei Li

Summary: Attention + Transformers

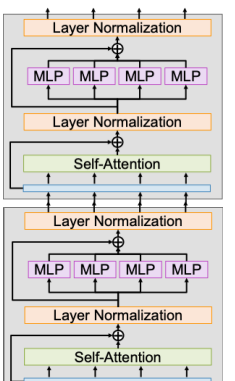
Attention: A new primitive that operates on sets of vectors



Transformers are the backbone of all large AI models today!

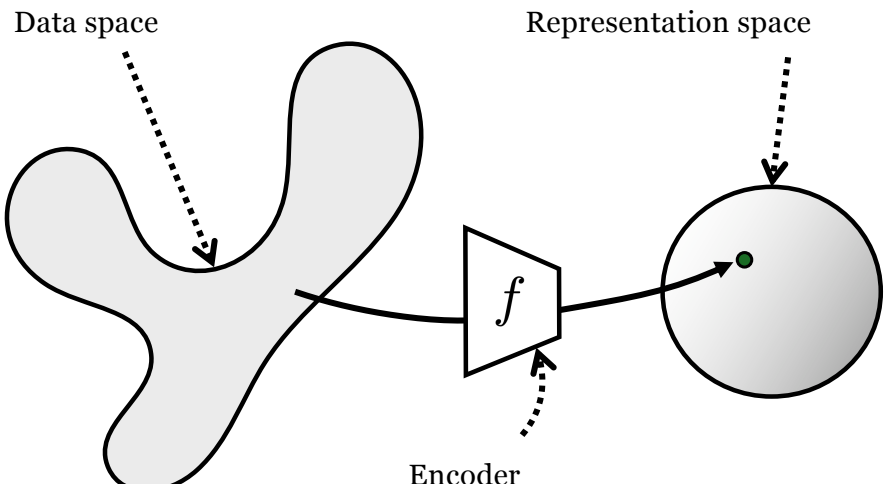
Used for language, vision, speech, ...

Transformer: A neural network architecture that uses attention everywhere

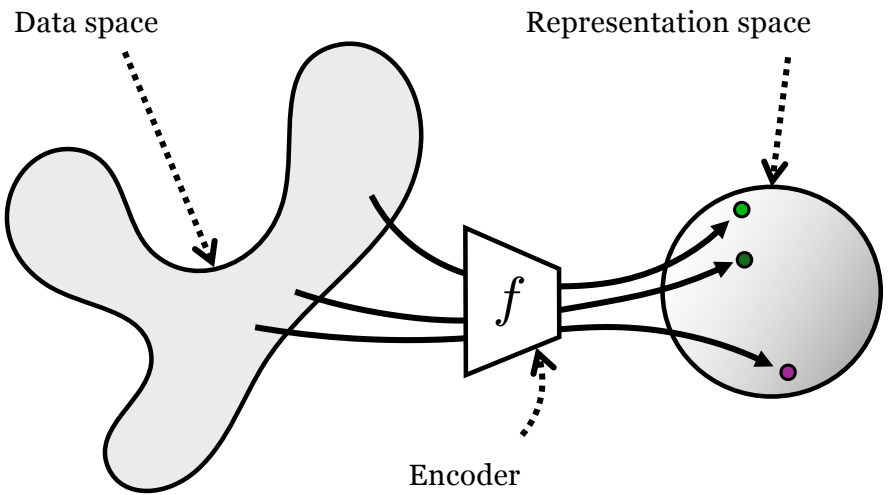


slide credit: Justin Johnson, Fei Fei Li

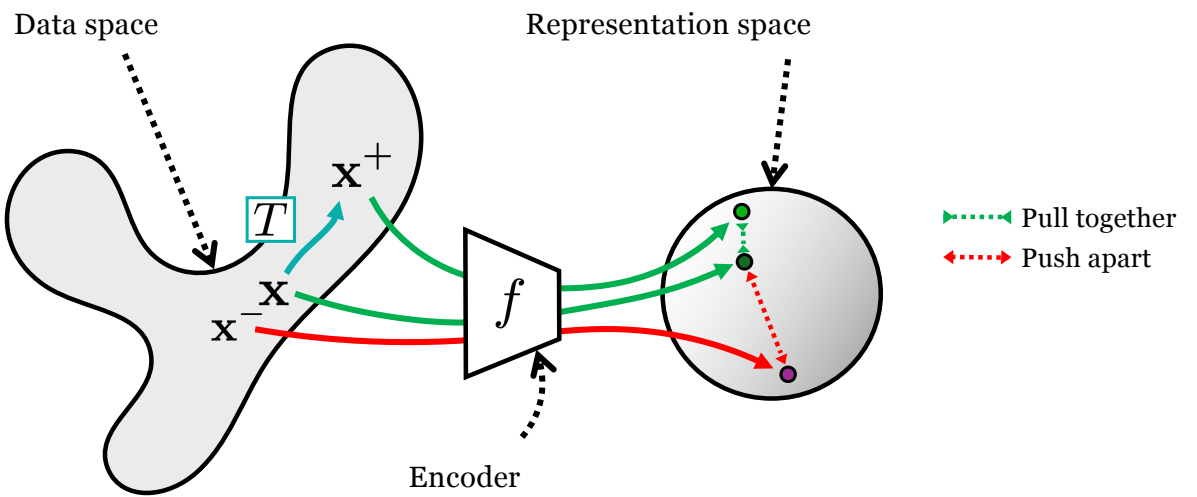
Contrastive Learning



Contrastive Learning

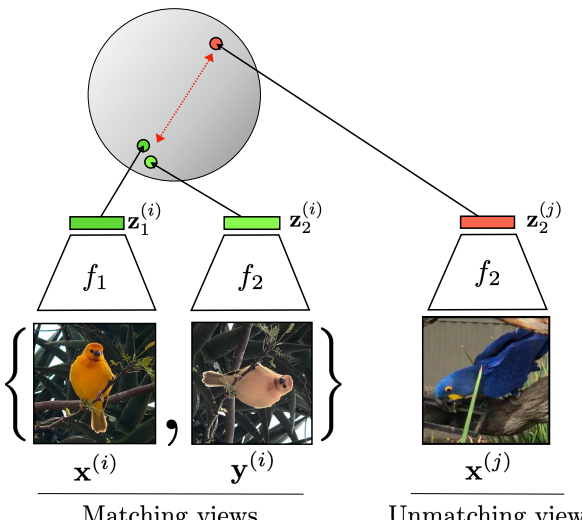


Contrastive Learning



Contrastive Learning

- **Data from the same view should have similar representations**
- **Other data should have distinct representations**



Data

$\{\mathbf{x}^{(i)}\}_{i=1}^{N}, T \longrightarrow$

Constrastive learning (transformations)

Objective

$\sum_{i,j} D(f(T(\mathbf{x}^{(i)})), f(\mathbf{x}^{(i)}))$ ⟷ Positive

$-D(f(\mathbf{x}^{(i)}), f(\mathbf{x}^{(j)}))$ ⟷ Negative $\longrightarrow f$

Hypothesis space

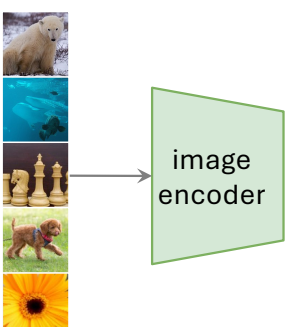
$f : \mathbb{R}^N \to \mathbb{R}^M$

Contrastive Learning – another point of view

- **Supervised** classification is Contrastive Learning

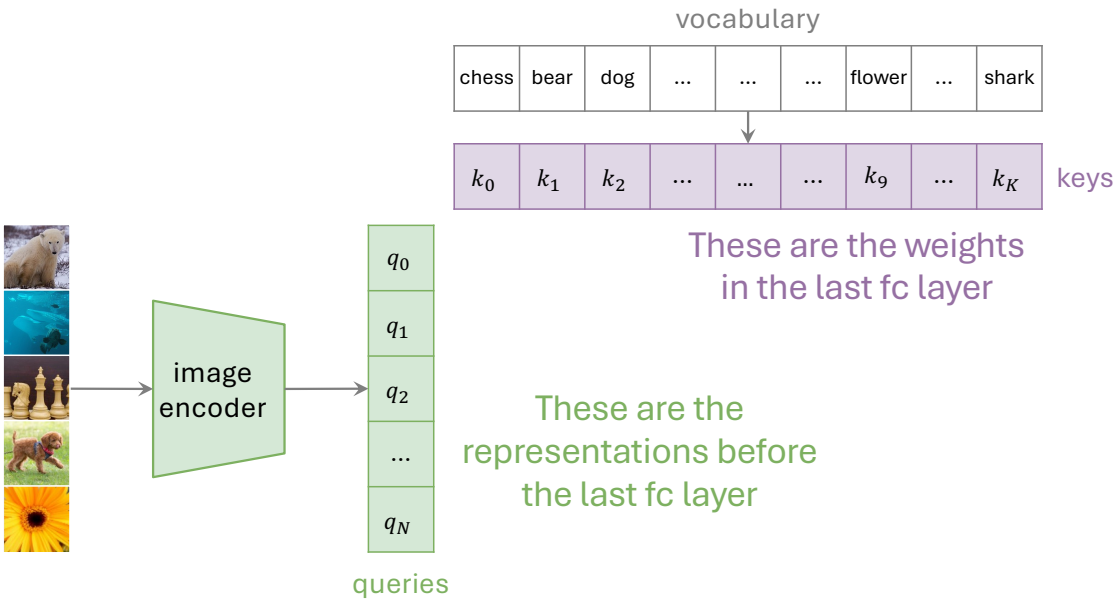
vocabulary

chess	bear	dog	...	...	...	flower	...	shark



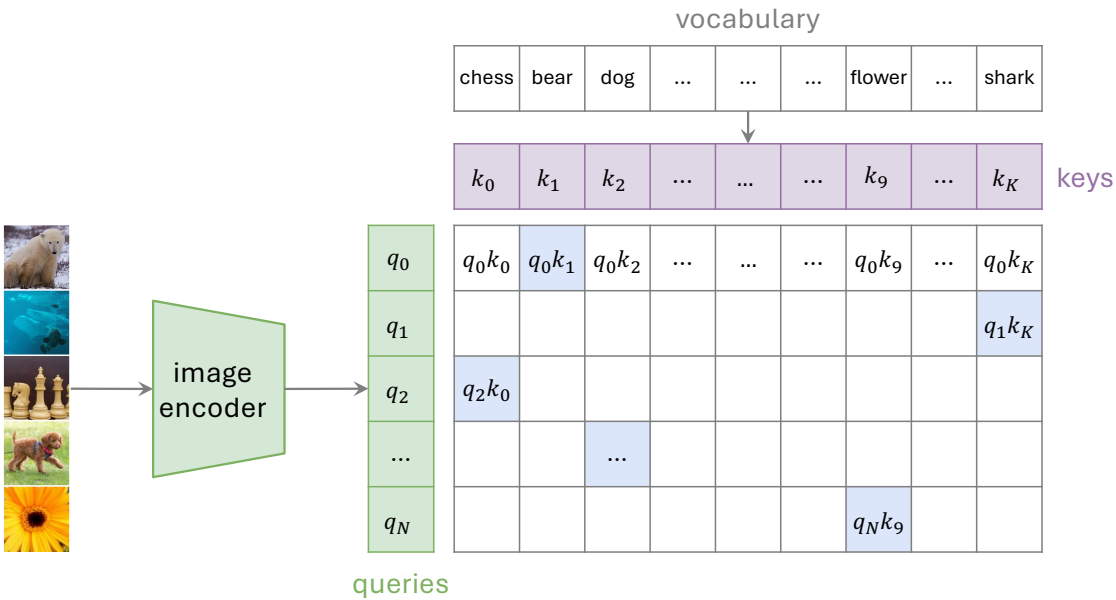
Contrastive Learning – another point of view

- **Supervised** classification is Contrastive Learning



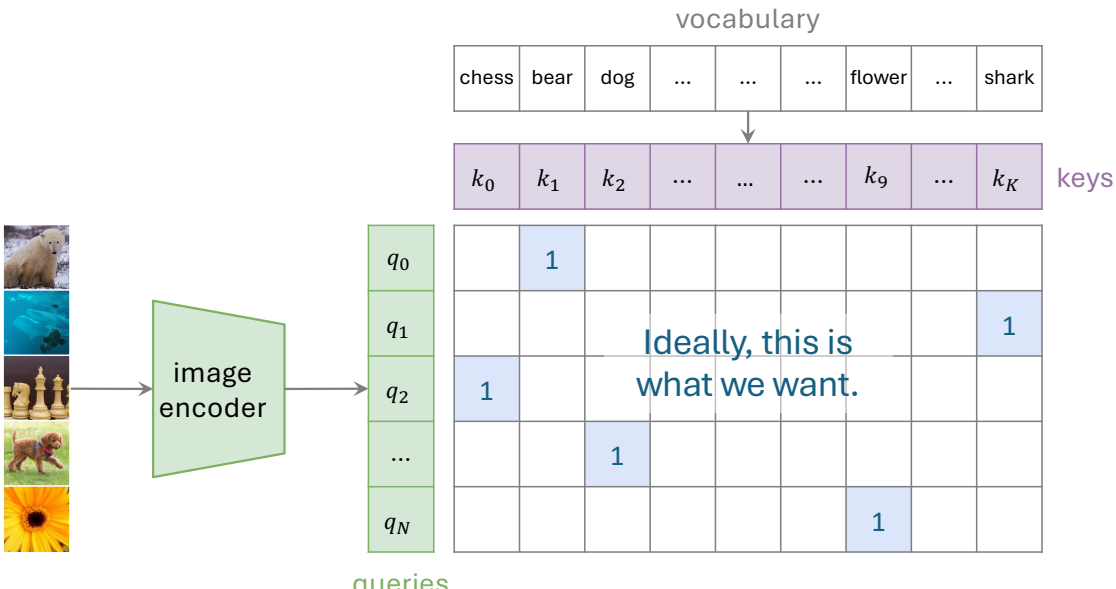
Contrastive Learning – another point of view

- **Supervised** classification is Contrastive Learning



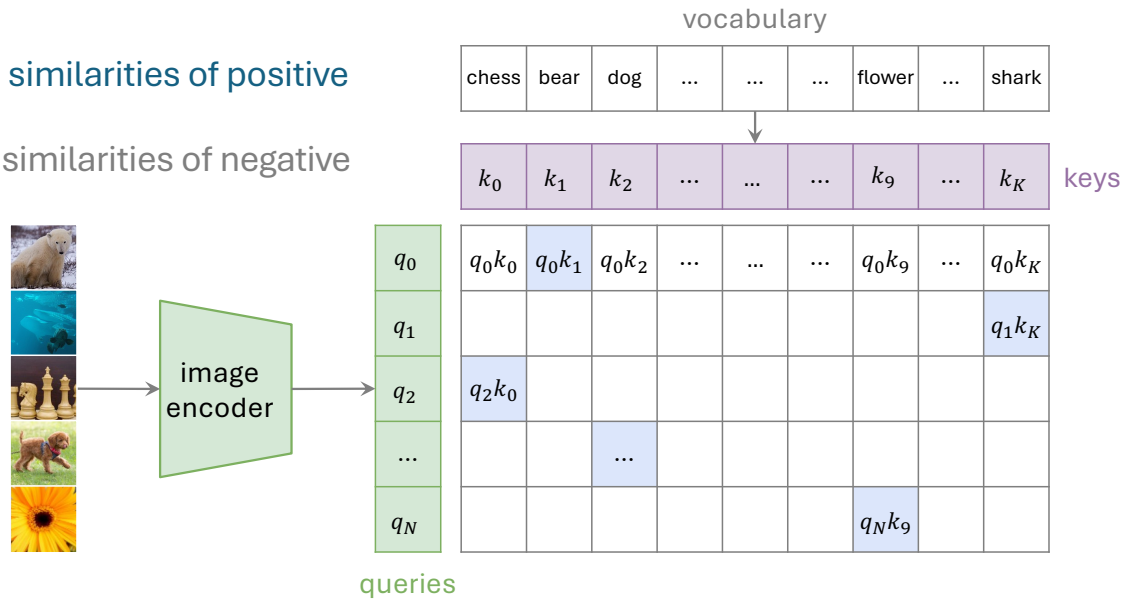
Contrastive Learning – another point of view

- **Supervised** classification is Contrastive Learning



Contrastive Learning – another point of view

- **Supervised** classification is Contrastive Learning
  - maximize similarities of positive pairs
  - minimize similarities of negative pairs

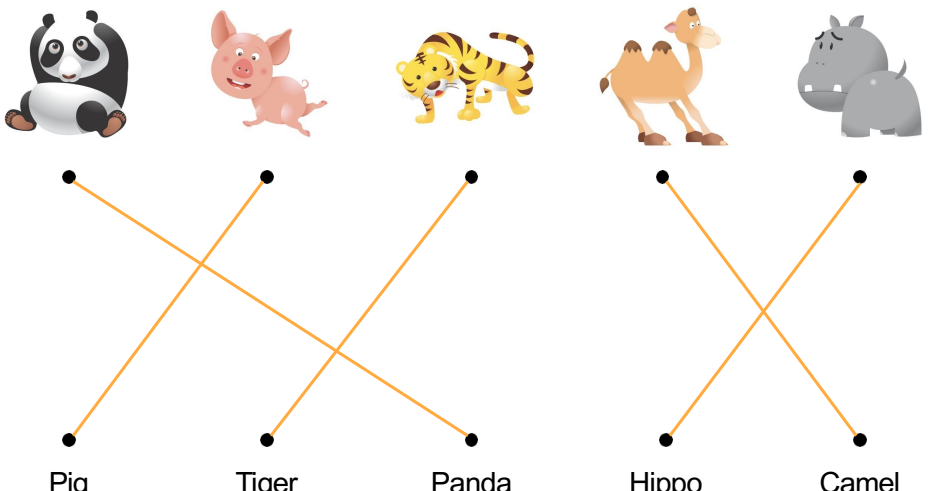


Supervised Classification

Pig Tiger Panda Hippo Camel

Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

Supervised Classification



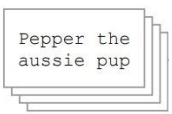
Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

CLIP: Contrastive Language-Image Pre-training

Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

CLIP: Contrastive Language-Image Pre-training

- Large-scale image-text paired dataset from the Internet



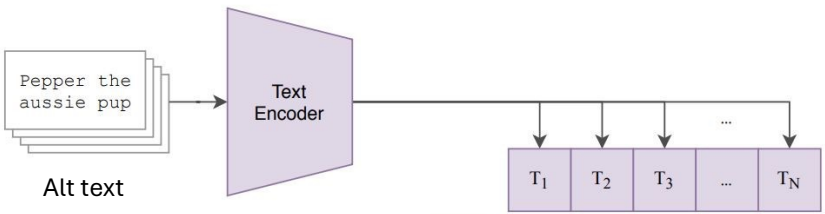
Alt text

Image

Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

CLIP: Contrastive Language-Image Pre-training

- Text encoder to extract representations from texts

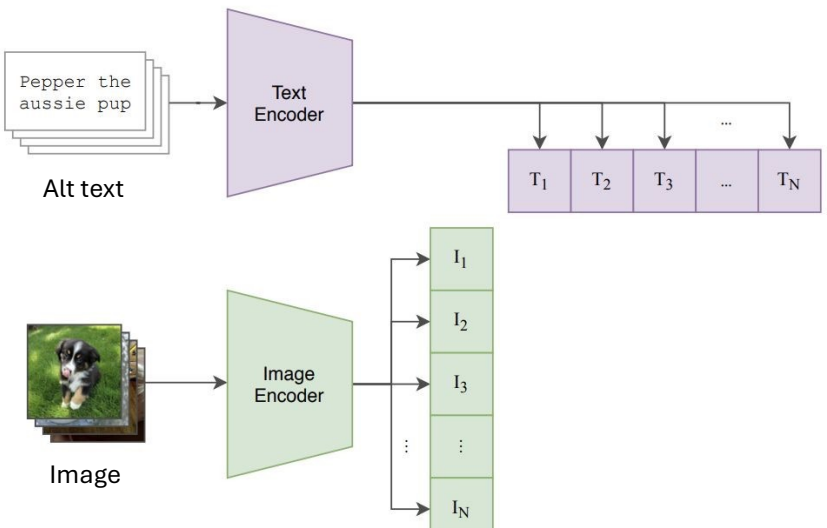


Image

Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

CLIP: Contrastive Language-Image Pre-training

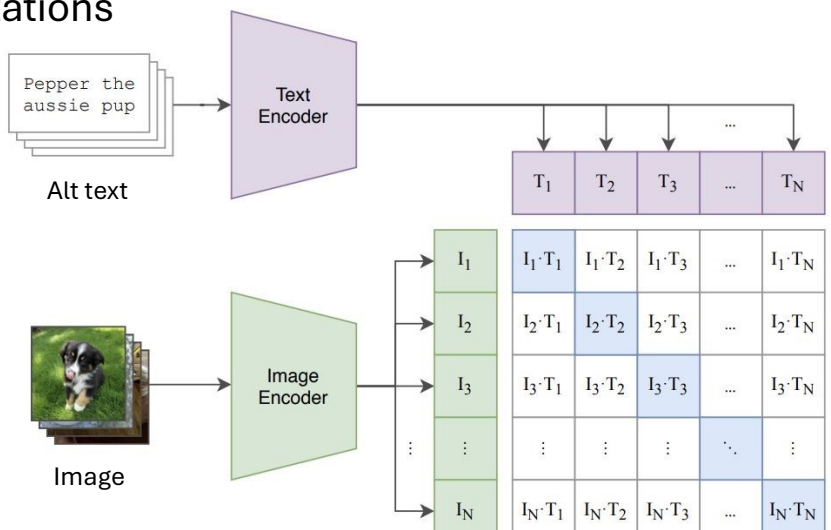
- Image encoder to extract representations from images



Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

CLIP: Contrastive Language-Image Pre-training

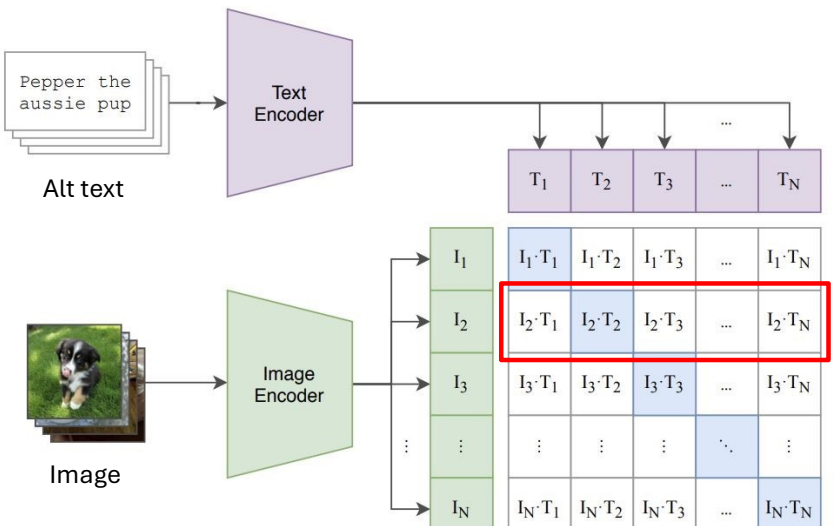
- Pairwise similarity matrix between image and text representations



Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

CLIP: Contrastive Language-Image Pre-training

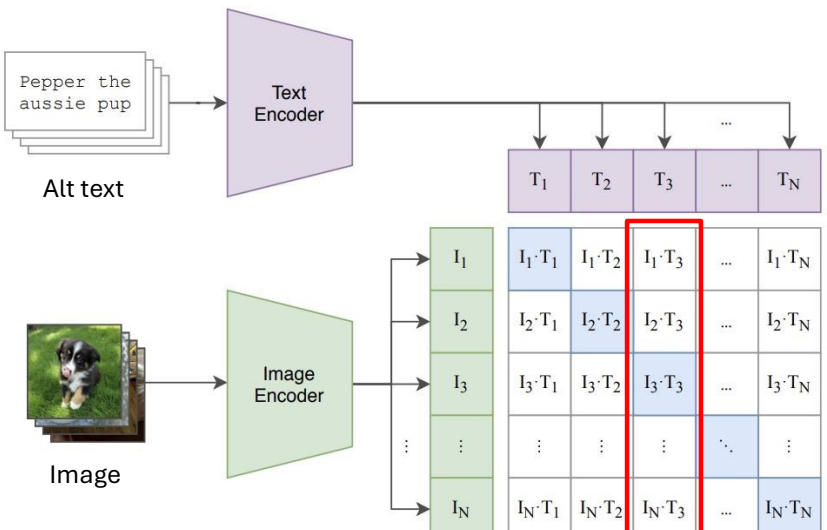
- For each image, compute similarity to all text representations



Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

CLIP: Contrastive Language-Image Pre-training

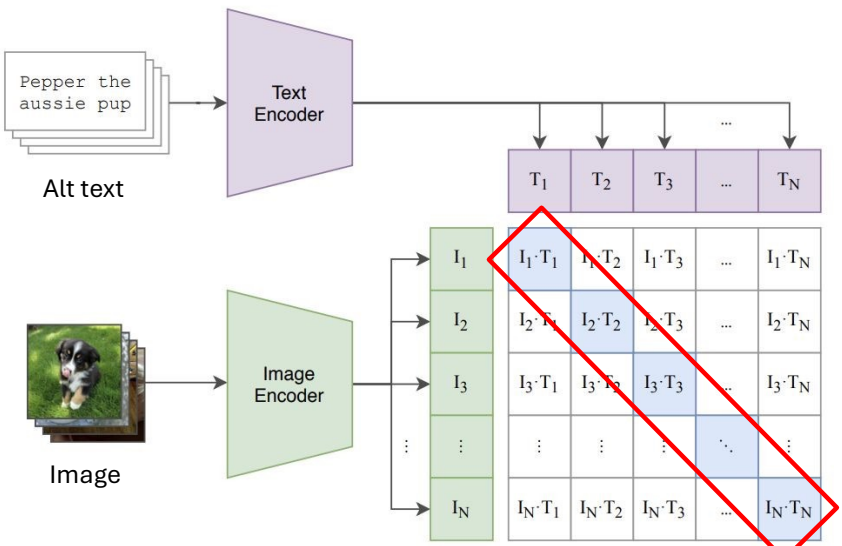
- For each text, compute similarity to all image representations



Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

CLIP: Contrastive Language-Image Pre-training

- Only the similarity between paired image-text should be high



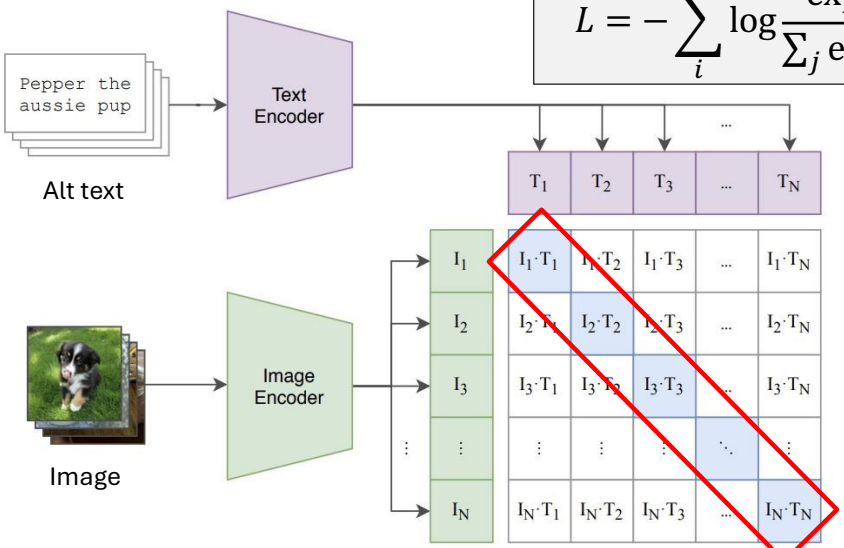
Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

CLIP: Contrastive Language-Image Pre-training

- Optimize standard InfoNCE loss

CLIP loss

$$L = -\sum_i \log \frac{\exp(\mathbf{I}_i \mathbf{T}_i / \tau)}{\sum_j \exp(\mathbf{I}_i \mathbf{T}_j / \tau)}$$



Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

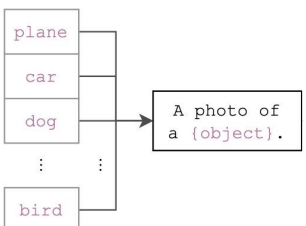
Zero-shot image classification

- CLIP can directly perform image classification without fine-tuning

Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

Zero-shot image classification

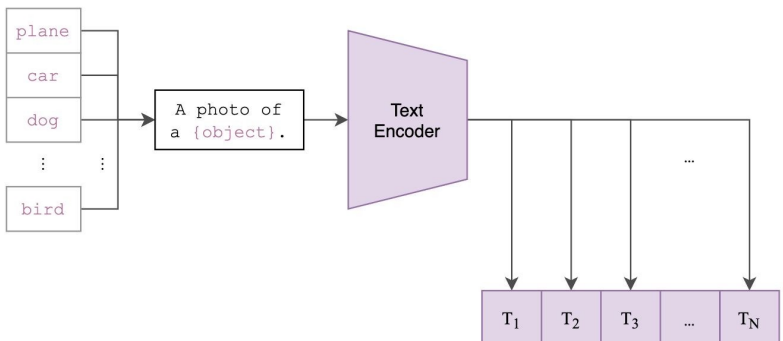
- CLIP can directly perform image classification without fine-tuning



Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

Zero-shot image classification

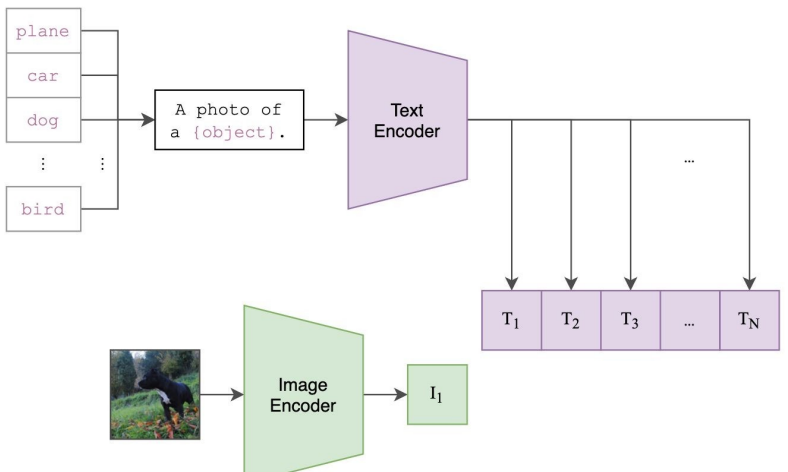
- CLIP can directly perform image classification without fine-tuning



Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

Zero-shot image classification

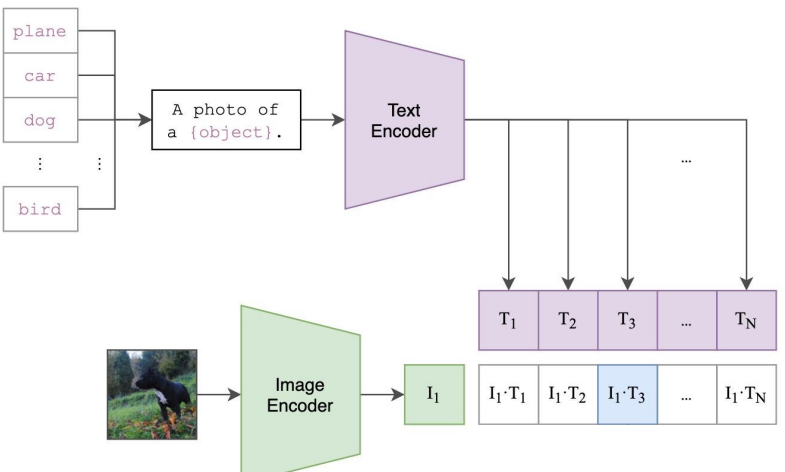
- CLIP can directly perform image classification without fine-tuning



Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

Zero-shot image classification

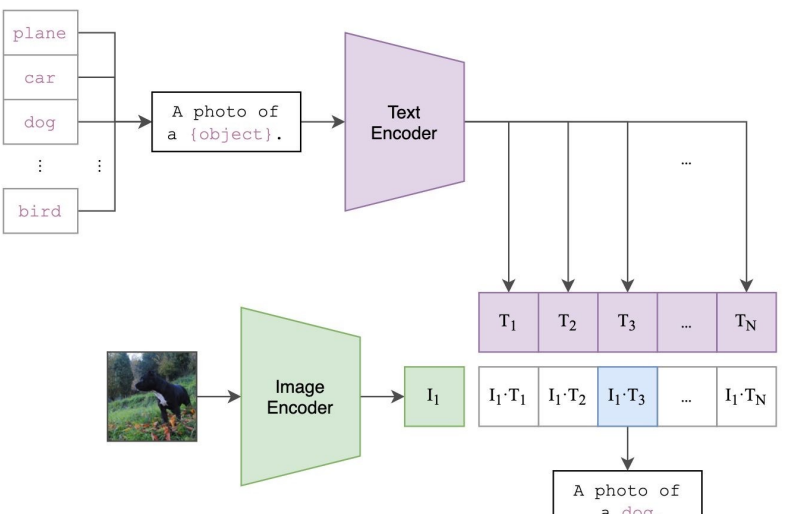
- CLIP can directly perform image classification without fine-tuning



Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

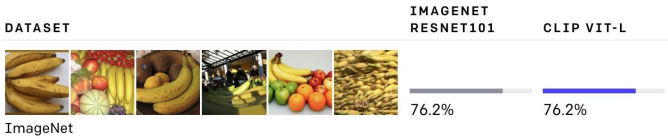
Zero-shot image classification

- CLIP can directly perform image classification without fine-tuning



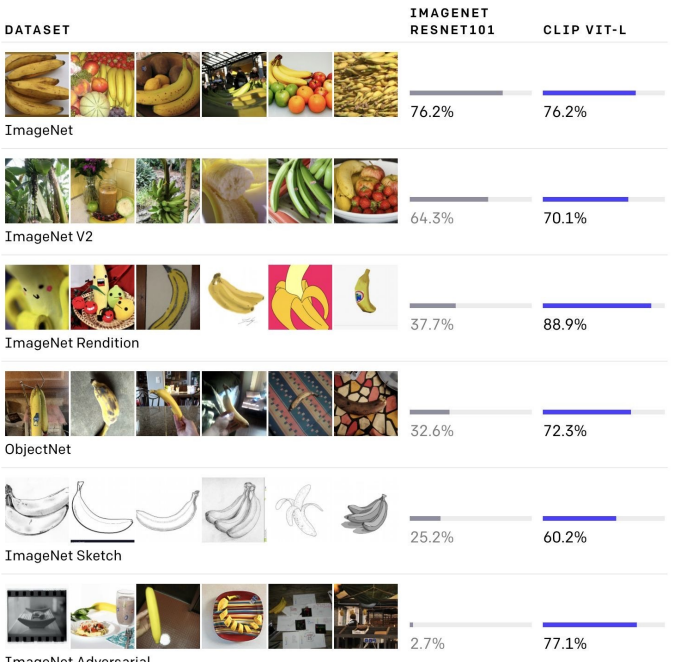
Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

Zero-shot CLIP is much more robust



Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

Zero-shot CLIP is much more robust



Radford et al., Learning Transferable Visual Models From Natural Language Supervision, ICML 2021

Summary

- motivation / why use different architectures?
- limitations of CNNs
- Attention
- Transformers

Takeaways

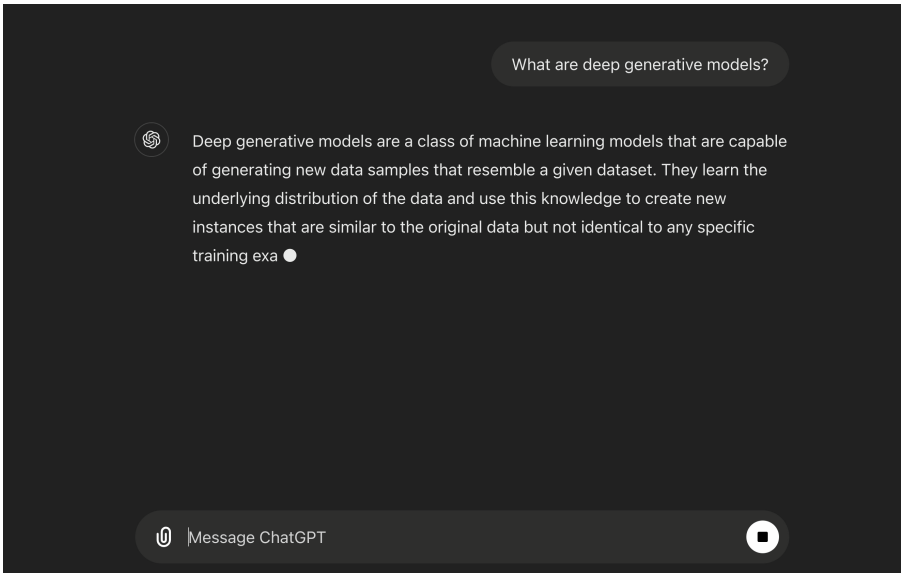
- why would we use different network architectures?
- what are limitations of CNNs?
- what is attention and how is it computed?
  - what are the compute/memory requirements?
- how does the transformer architecture work and what are the key innovations?
- How do vision transformers work?

Outline

- motivation / why use different architectures?
- limitations of CNNs
- attention
- transformers
- generative models

The “GenAI” Era

Chatbot and natural language conversation,



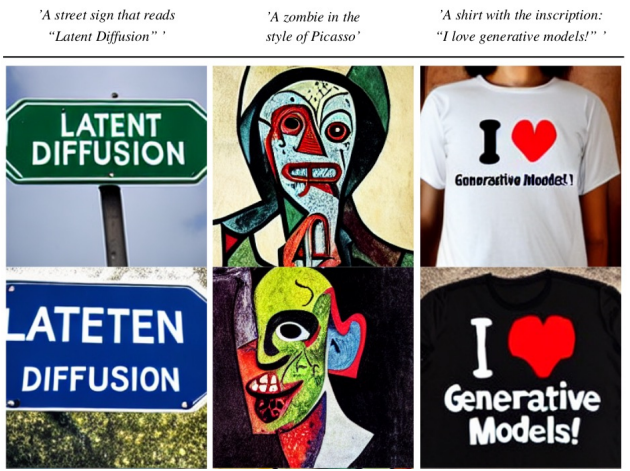
The “GenAI” Era

Class-to-image generation, 2019

Generated by BigGAN

The “GenAI” Era

Text-to-image generation, 2022



Generated by Latent Diffusion

The “GenAI” Era

Text-to-image generation, 2024

Generated by Stable Diffusion 3 Medium.
Prompt: teddy bear teaching a course, with "generative models" written on blackboard

The “GenAI” Era

Text-to-image generation, 2025

Getting started

Generated by GPT 4o

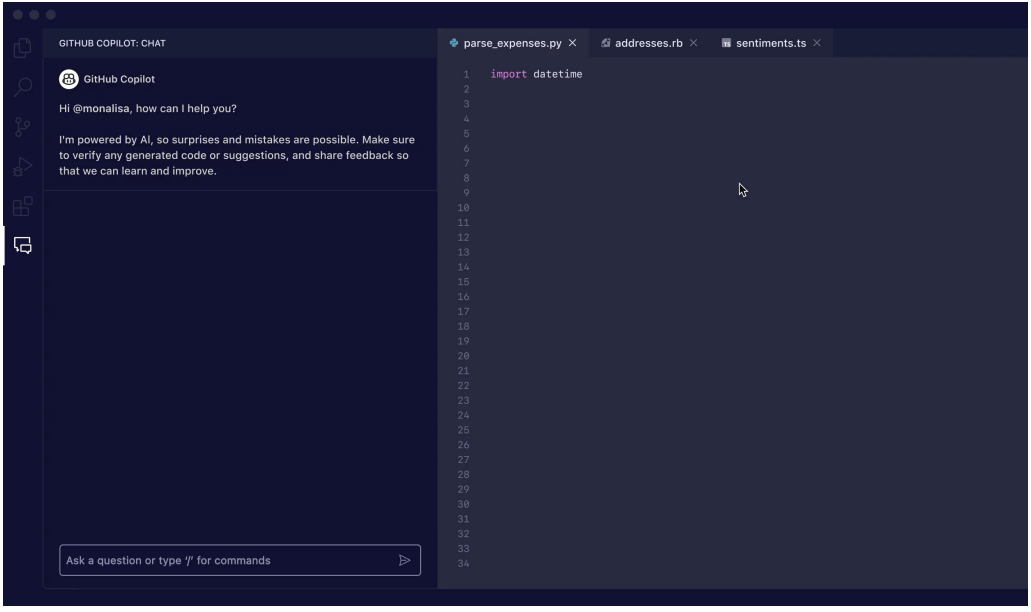
The “GenAI” Era

Text-to-video generation, 2024

Generated by Sora

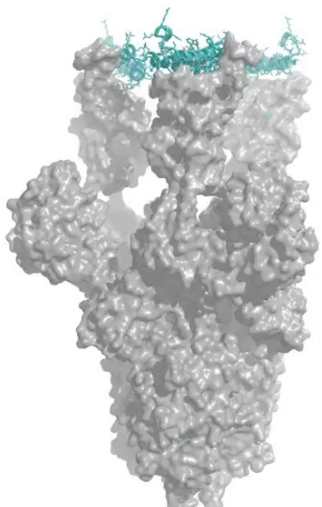
The “GenAI” Era

AI assistant for code generation



The “GenAI” Era

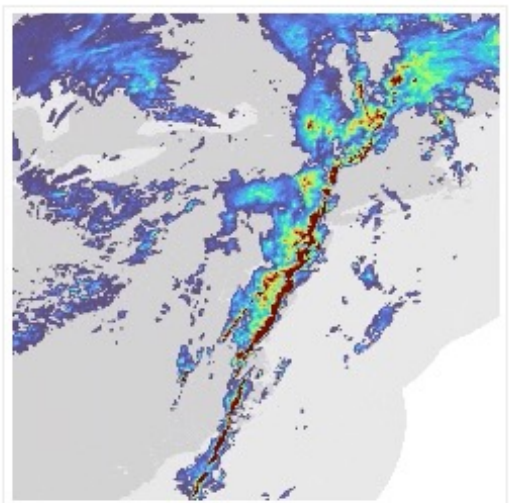
Protein design and generation



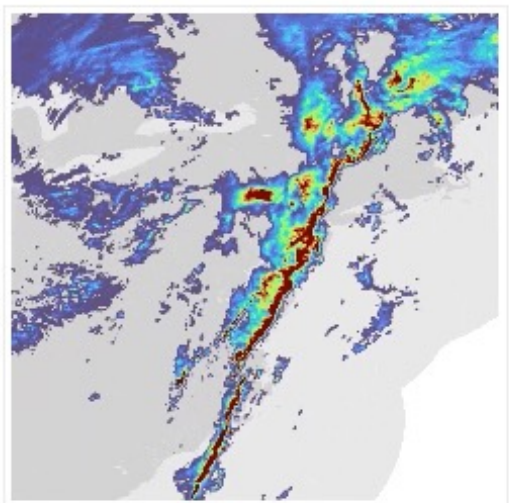
Watson, et al. De novo design of protein structure and function with RFdiffusion, Nature 2023

The "GenAI" Era

Weather forecasting



Target

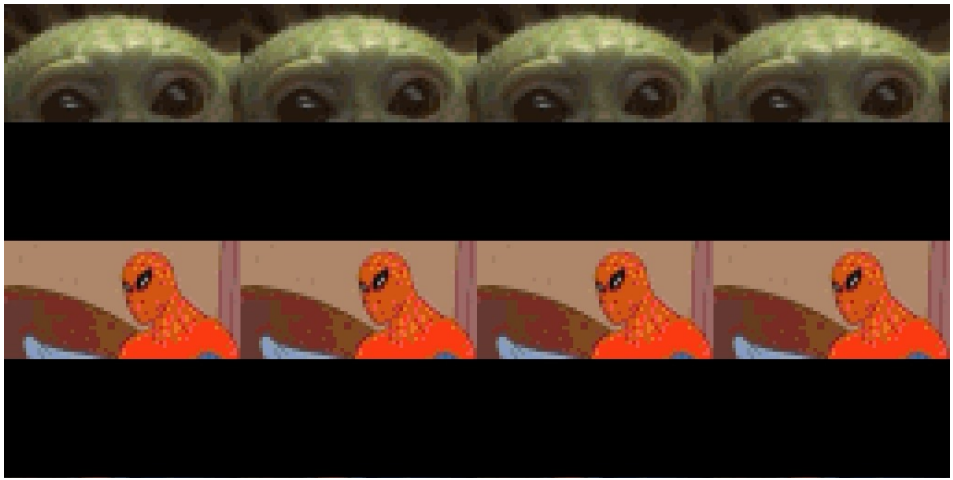


DGMR

Skilful precipitation nowcasting using deep generative models of radar, Nature 2021

Generative Models before the “GenAI” Era

2020, image GPT: Fill bottom part of the image



Chen et al. “Generative Pretraining from Pixels”, ICML 2020

Generative Models before the “GenAI” Era

2016, Context Encoder: Fill center part of the image

Pathak, et al., “Context Encoders: Feature Learning by Inpainting”, CVPR 2016

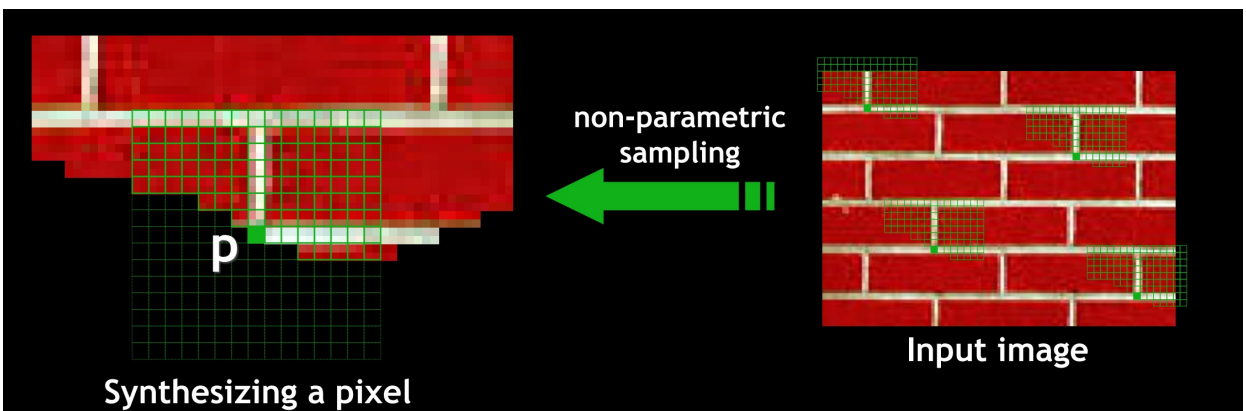
Generative Models before the “GenAI” Era

2009, PatchMatch: Photoshop’s Content-aware Fill

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, SIGGRAPH 2009

Generative Models before the "GenAI" Era

1999, the Efros-Leung algorithm for texture synthesis

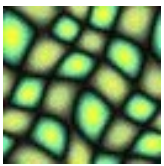


Texture Synthesis by Non-parametric Sampling, ICCV 1999

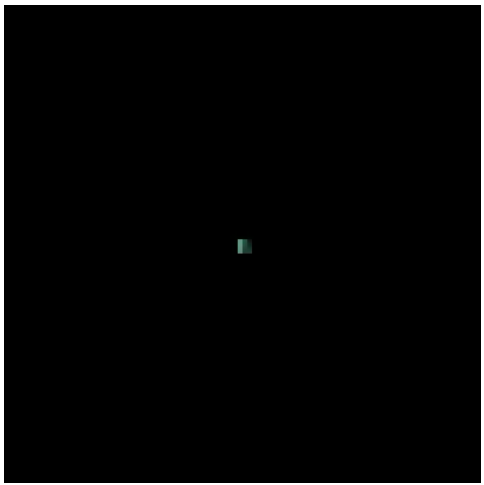
Generative Models before the "GenAI" Era

1999, the Efros-Leung algorithm for texture synthesis

In today's word: this is an **Autoregressive** model



exemplar



Texture Synthesis by Non-parametric Sampling, ICCV 1999

Overview (next time)

- Introduction
  - What are Generative Models?
  - What can Generative Models do?
  - Generative models and representation learning

- Generative Models: Concepts and Recent Frontiers
  - Variational Autoencoders (VAE)
  - Generative Adversarial Networks (GAN)
  - Autoregressive models
  - Diffusion models