Deep Learning for Vision (DLV)
Classification - part II

Denis Coquenet

2024-2025

Goals of this course

Knowledge

- What is attention, how it works
- What is Transformer:
  - main components: multi-head self/cross attention, positional encoding, encoder vs decoder
  - advantages and drawbacks

Skills and know-how

- Choose between fully-connected, convolution and attention layers given context
- Choose/adapt architecture according to constraints
- Propose ways to deal with few labeled data given context

Table of contents

The Transformer was proposed for Neural Machine Translation (NMT) task

This is known as a sequence-to-sequence problem.
Input: a sequence of tokens. Output: a sequence of tokens

Let's assume we want to translate French sentences to English.
We are working at character level (*i.e.*, each token corresponds to a character).
We use a simple character set common for both English and French and made up of 26 letters.

Propose a way to encode each character on a vector of d dimensions.
This is known as input embedding.

1. if $d \geq 26$
2. if $d < 26$

Exercise on embedding: correction

Character	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

1. if $d \geq 26$: one-hot encoding
   ➤ Example with $d = 27$

A	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Z	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

2. if $d < 26$:
   ➤ Learn a matrix of trainable weights $\boldsymbol{E}_{\text{char}} \in \mathbb{R}^{26 \times d}$
   (one character encoded by one row)

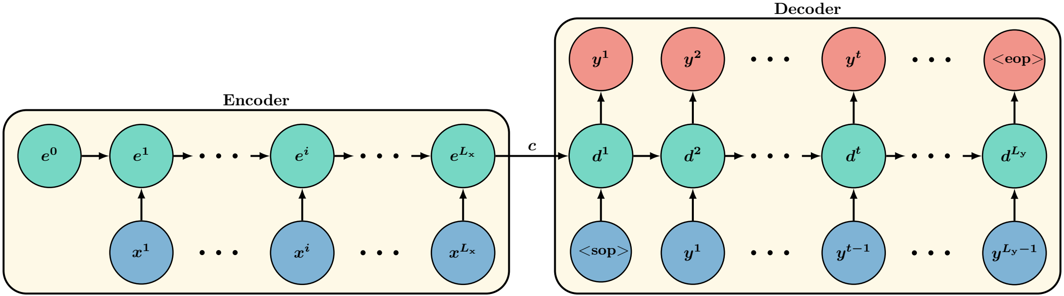
➤ In practice, second option is generally used

Neural Machine Translation (NMT)

A sequence-to-sequence task

$\boldsymbol{x}$: a sequence of input tokens representing some text.

$\boldsymbol{y}$: a sequence of output tokens representing the translated text.



For instance, encoder and decoder can be LSTM/GRU.

Limitation

- The whole input sequence is compressed into a fixed-size context vector c.

Idea: attention mechanism

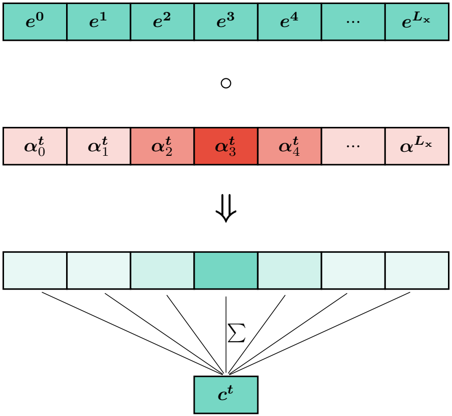
From a static context vector:

$$\boldsymbol{c} = f(\boldsymbol{x})$$

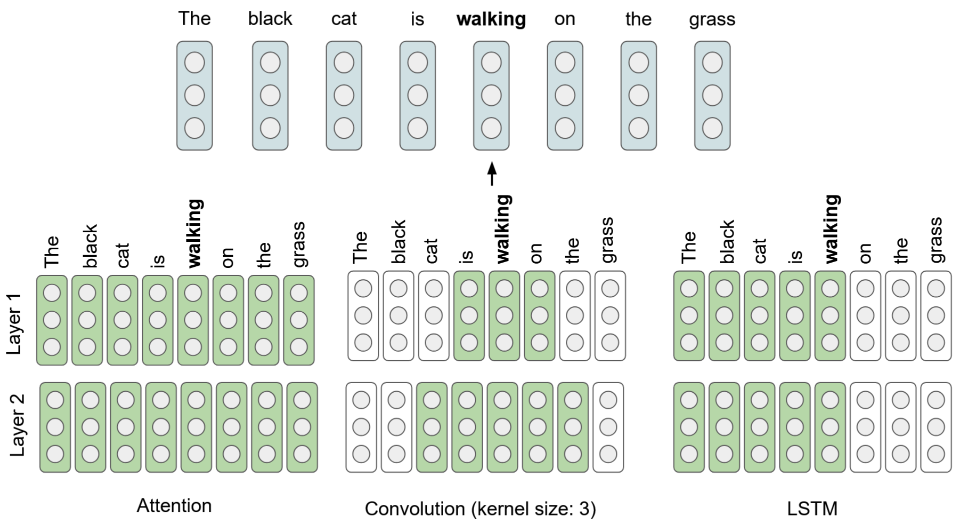
to a dynamic context vector:

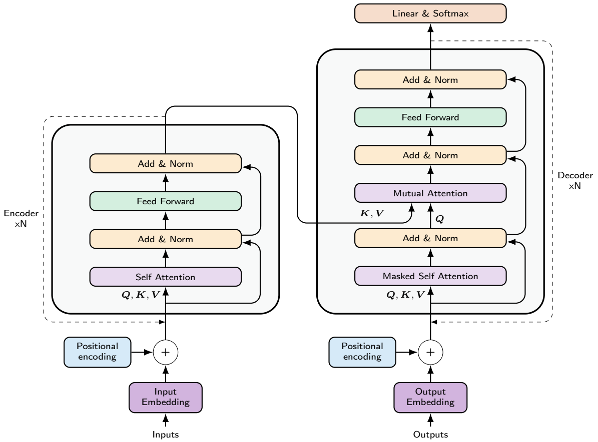
$$\boldsymbol{c}^t = \sum_{i=1}^{L_x} \alpha_i^t \boldsymbol{e}^i$$

with: $\sum_i \alpha_i = 1$.



Context modeling comparison





Two main blocks:

- Transformer encoder
- Transformer decoder

One-shot encoding

$$\boldsymbol{f} = \text{encoder}(\boldsymbol{x} + \text{PE})$$

Iterative decoding

$$\boldsymbol{c}^t = \text{decoder}(\boldsymbol{f}, \{\hat{\boldsymbol{y}}^0, ..., \hat{\boldsymbol{y}}^{t-1}\} + \text{PE})$$

$$\hat{\boldsymbol{y}}^t = \arg\max(\boldsymbol{W}\boldsymbol{c}^t)$$

➤ Mainly relies on the Query-Key-Value attention paradigm

Query-Key-Value paradigm

Concept

Goal: retrieve the useful information among a collection of data

Inputs: a *query* and a set of data

Approach

- Compute a similarity score between the query and all the data
- Generate an answer based on the most relevant data

➤ Two representations of the data are used: *keys* for comparison with the query, and *values* for the answer

Analogy with library search engine

Query: keywords; Keys: book titles; Values: book contents

Query-Key-Value paradigm

Issue

Gradient propagation for hard attention (select only the most relevant item)
➤ Soft attention (weighted sum of items)

Scaled Dot-Product Attention

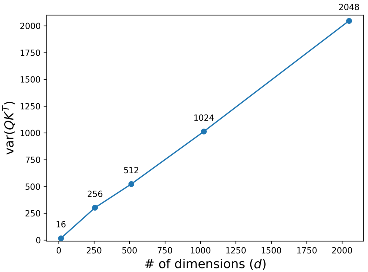
$\boldsymbol{q} \in \mathbb{R}^d$: a query vector
$\boldsymbol{K} \in \mathbb{R}^{L \times d}$: a matrix gathering key vectors for each available item
$\boldsymbol{V} \in \mathbb{R}^{L \times d}$: a matrix gathering values vectors for each available item

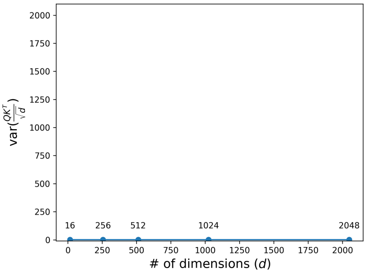
$$\mathrm{Attention}(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \underbrace{\mathrm{softmax}\left(\frac{\boldsymbol{q}\boldsymbol{K}^T}{\sqrt{d}}\right)}_{\alpha} \boldsymbol{V} = \boldsymbol{c} \quad (\in \mathbb{R}^d)$$

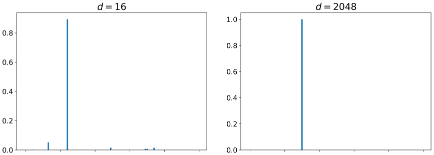
➤ Dot product as distance function

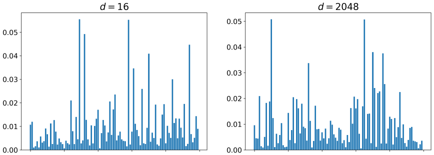
Why scaling by $\frac{1}{\sqrt{d}}$?

For 1 query and 10,000 keys (from normal distribution):









- Queries, Keys and Values are from the same source

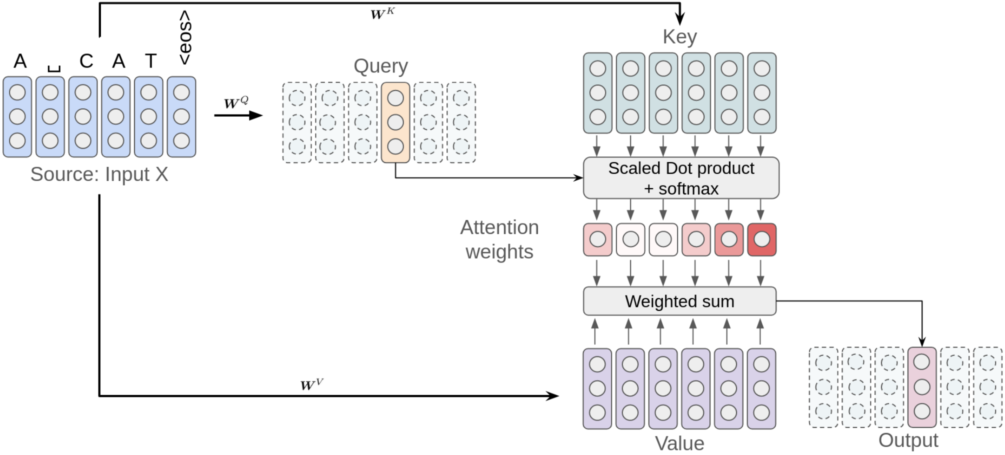
Query-key-value for feature extraction

$\boldsymbol{X}$: the latent representation of an input sequence.

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q (\in \mathbb{R}^{L_x \times d})$$
$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K (\in \mathbb{R}^{L_x \times d})$$
$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V (\in \mathbb{R}^{L_x \times d})$$

$$\boldsymbol{Y} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d}}\right)\boldsymbol{V}$$

$\boldsymbol{W}^Q, \boldsymbol{W}^K \text{and} \boldsymbol{W}^V$ are matrices of trainable weights (fully-connected layers)



- ➤ Output can be computed in parallel through matrix multiplications
- ➤ Output length = Query length (= Key length = Value length)

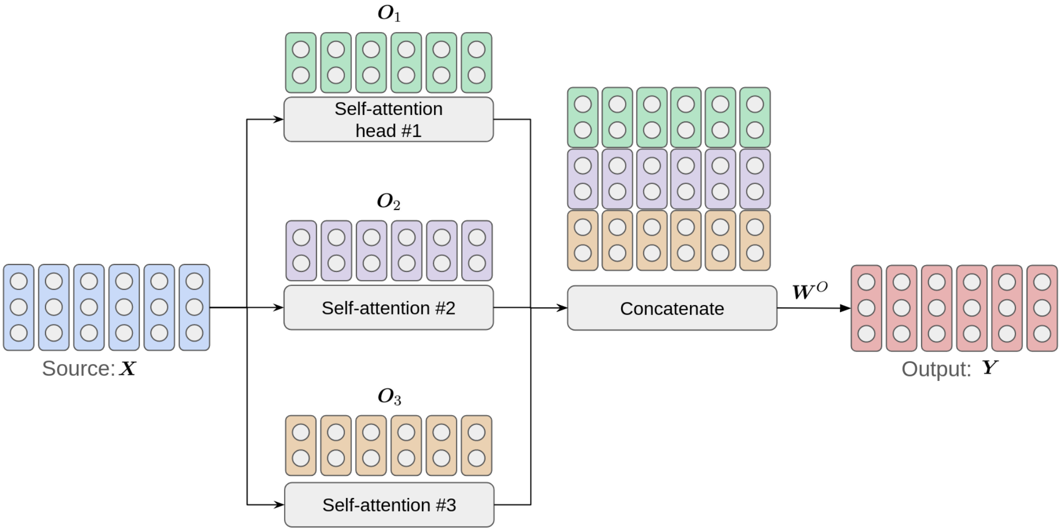
h self-attention in parallel

$$\boldsymbol{Q}_1 = \boldsymbol{X}\boldsymbol{W}_1^Q \quad \dots \quad \boldsymbol{Q}_h = \boldsymbol{X}\boldsymbol{W}_h^Q$$
$$\boldsymbol{K}_1 = \boldsymbol{X}\boldsymbol{W}_1^K \quad \dots \quad \boldsymbol{K}_h = \boldsymbol{X}\boldsymbol{W}_h^K$$
$$\boldsymbol{V}_1 = \boldsymbol{X}\boldsymbol{W}_1^V \quad \dots \quad \boldsymbol{V}_h = \boldsymbol{X}\boldsymbol{W}_h^V$$

$$\boldsymbol{O}_1 = \mathrm{softmax}\left(\frac{\boldsymbol{Q}_1\boldsymbol{K}_1^T}{\sqrt{d}}\right)\boldsymbol{V}_1 \quad \dots \quad \boldsymbol{O}_h = \mathrm{softmax}\left(\frac{\boldsymbol{Q}_h\boldsymbol{K}_h^T}{\sqrt{d}}\right)\boldsymbol{V}_h$$

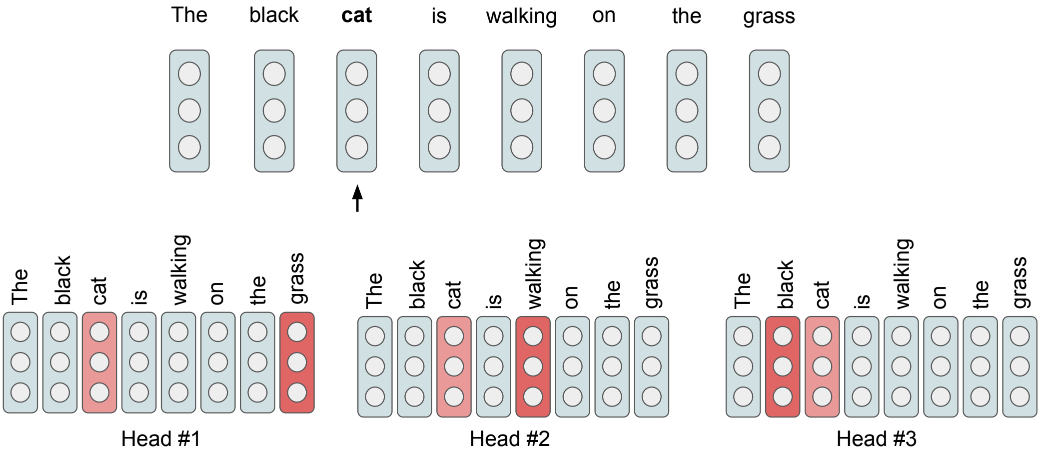
$$\boldsymbol{Y} = \mathrm{concat}(\boldsymbol{O}_1, \dots, \boldsymbol{O}_h)\boldsymbol{W}^O$$

Multi-head Self-Attention (MSA)



Intuition

Each head is specialized for a specific query (*e.g.*, action, location, description)



Issue with Multi-head Self Attention (MSA)

Self-attention is invariant to input permutation:
$\mathrm{MSA}([\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3]) = [\boldsymbol{y}_1, \boldsymbol{y}_2, \boldsymbol{y}_3] \Rightarrow \mathrm{MSA}([\boldsymbol{x}_2, \boldsymbol{x}_1, \boldsymbol{x}_3]) = [\boldsymbol{y}_2, \boldsymbol{y}_1, \boldsymbol{y}_3]$

The position information is lost whereas it is crucial

- ➤ A word/sentence is a sequence of characters, not a set of characters

Goal

Inject positional information to preserve the sequentiality

Solution

Additive positional encoding:

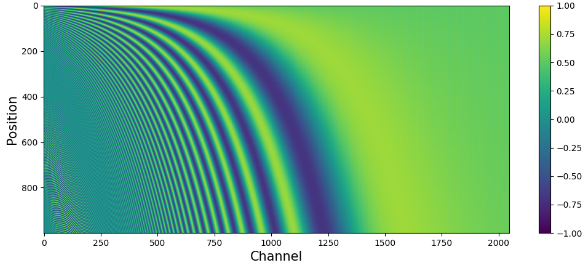
- ➤ Input of transformer encoder: $\boldsymbol{I} = \boldsymbol{X} + \mathrm{PE}$

A combination of sines and cosines

$\text{PE}(p, k)$: k^{th} value of positional embedding vector for position p.

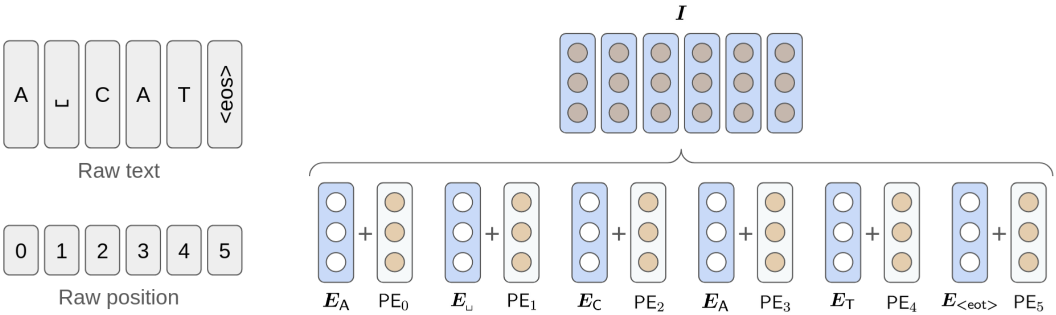
$$\text{PE}(p, 2k) = \sin(w_k \cdot p)$$
$$\text{PE}(p, 2k+1) = \cos(w_k \cdot p)$$

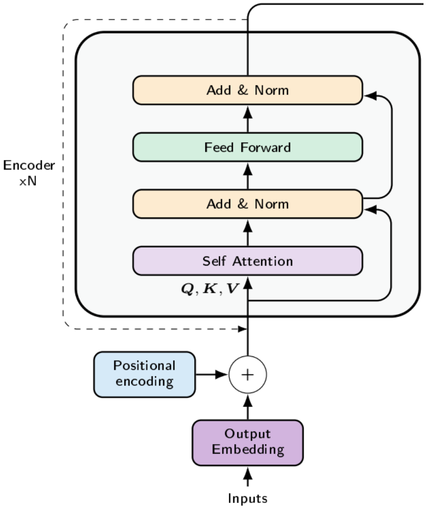
$\forall k \in [0, d/2]$, with $w_k = 1/10000^{2k/d}$



Advantages

- Does not depend on the input length
- Encoding unique for each position
- Periodicity of sine/cosine could help to adapt to unseen position
- Bonus: no additional trainable parameters

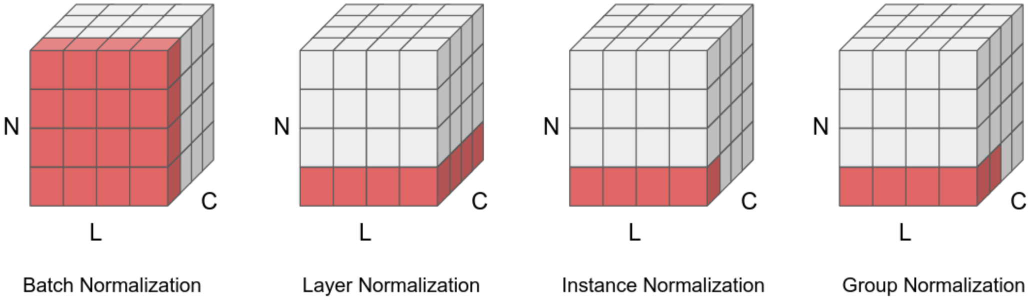




A stack of 6 transformer encoder layers

- Multi-head Self Attention
  - ➤ Global context modeling
- Add: residual connections
  - ➤ Multi-scale representation
  - ➤ Reinforce identity
- Norm: layer normalization
  - ➤ Stabilize training (range of values)
- Feed Forward: 2 fully-connected layers
  - ➤ Local projection

- Normalization approaches differs on how data is segmented to perform normalization



$$\hat{x} = \frac{x - \mu_x}{\sqrt{\sigma_x^2 + \epsilon}}$$

N: batch dimension (one sample)
L: length dimension (one character, or pixel)
C: channel dimension (one feature)

Transformer encoder layer

Input sequence of L token embedding: $\boldsymbol{X}$ of shape $L \times d$

Express as a function of L, d and n, the number of parameters, the number of floating point operations and the output shape (biases are ignored)

1. for a succession of two fully-connected layers: $\boldsymbol{Y} = (\boldsymbol{X}\boldsymbol{W}^1)\boldsymbol{W}^2$
   (applied on the channel axis, same weights for all tokens)
   The first one is made up of n neurons, the second one of d neurons
2. for the simplified self-attention layer: $\boldsymbol{Y} = (\boldsymbol{X}\boldsymbol{W}^{\mathsf{Q}})(\boldsymbol{X}\boldsymbol{W}^{\mathsf{K}})^T(\boldsymbol{X}\boldsymbol{W}^{\mathsf{V}})\boldsymbol{W}^{\mathsf{O}}$
   We keep d dimensions through the attention process

In both cases, compute those metrics for $d = 1024$, $n = 2048$ and

1. $L = 50$
2. $L = 784$
3. $L = 1,048,576$

Input sequence of L token embedding: $\boldsymbol{X}$ of shape $L \times d$

1. for a succession of two fully-connected layers: $\boldsymbol{Y} = (\boldsymbol{X}\boldsymbol{W}^1)\boldsymbol{W}^2$
   FC1: from $L \times d$ to $L \times n$ ($\boldsymbol{W}^1 \in \mathbb{R}^{d \times n}$); $2dnL$ FLOPs
   FC2: from $L \times n$ to $L \times d$ ($\boldsymbol{W}^2 \in \mathbb{R}^{n \times d}$); $2ndL$ FLOPs
   ➤ $2nd$ parameters, $4ndL$ FLOPs
2. for the simplified self-attention layer: $\boldsymbol{Y} = (\boldsymbol{X}\boldsymbol{W}^{\mathsf{Q}})(\boldsymbol{X}\boldsymbol{W}^{\mathsf{K}})^T(\boldsymbol{X}\boldsymbol{W}^{\mathsf{V}})\boldsymbol{W}^{\mathsf{O}}$
   $\boldsymbol{W}^{\mathsf{Q}} \in \mathbb{R}^{d \times d}, \boldsymbol{W}^{\mathsf{K}} \in \mathbb{R}^{d \times d}, \boldsymbol{W}^{\mathsf{V}} \in \mathbb{R}^{d \times d}, \boldsymbol{W}^{\mathsf{O}} \in \mathbb{R}^{d \times d} \rightarrow 4d^2$ parameters
   $\boldsymbol{X}\boldsymbol{W}^{\mathsf{i}}$: from $L \times d$ to $L \times d$; $2d^2L$ FLOPs ($\times 3$)
   $\boldsymbol{S} = (\boldsymbol{X}\boldsymbol{W}^{\mathsf{Q}})(\boldsymbol{X}\boldsymbol{W}^{\mathsf{K}})^T$: output $L \times L$; $2dL^2$ FLOPs
   $\boldsymbol{O} = \boldsymbol{S}(\boldsymbol{X}\boldsymbol{W}^{\mathsf{V}})$: output $L \times d$; $2dL^2$ FLOPs
   $\boldsymbol{Y} = \boldsymbol{O}\boldsymbol{W}^{\mathsf{O}}$: output $L \times d$; $2d^2L$ FLOPs
   ➤ $8d^2L + 4dL^2$ FLOPs

Correction

Input sequence of L token embedding: $\boldsymbol{X}$ of shape $L \times d$

1. for a succession of two fully-connected layers: $\boldsymbol{Y} = (\boldsymbol{X}\boldsymbol{W}^1)\boldsymbol{W}^2$
   ➤ $2nd$ parameters, $4ndL$ FLOPs
2. for the simplified self-attention layer: $\boldsymbol{Y} = (\boldsymbol{X}\boldsymbol{W}^{\mathsf{Q}})(\boldsymbol{X}\boldsymbol{W}^{\mathsf{K}})^T(\boldsymbol{X}\boldsymbol{W}^{\mathsf{V}})\boldsymbol{W}^{\mathsf{O}}$
   ➤$4d^2$ parameters, $8d^2L + 4dL^2$ FLOPs

	2 fully-connected		self attention	
	#params	# FLOPs	#params	# FLOPs
$L = 50$ (a sentence)	4.2 M	419 MFLOPs	4.2 M	430 MFLOPs
$L = 784$ (28×28 image)	4.2 M	6.6 GFLOPs	4.2 M	9.1 GFLOPs
$L = 1{,}048{,}576$ (1024×1024 image)	4.2 M	8.8 TFLOPs	4.2 M	4.5 PFLOPs

Remarks

1. Convolution with 1,024 kernels 3×3 and stride 1×1
   ➤ 9.4 M parameters, 19.8 TFLOPs (but never used on full image with as many filters)
2. A100 GPU (80GB, 2020): 312 TFLOPs/second on float32

Mutual attention (also known as cross attention)

- Queries from target domain, and keys and values from source domain

General case

Inputs: $\boldsymbol{X}$: the features from the encoded input sequence (source domain),
$\tilde{\boldsymbol{z}}$: the query sequence (*e.g.*, a question)
$\boldsymbol{z}$: the query vector ($\boldsymbol{z} = g(\tilde{\boldsymbol{z}})$).

$$\boldsymbol{q} = \boldsymbol{z}\boldsymbol{W}^Q (\in \mathbb{R}^d)$$
$$\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K (\in \mathbb{R}^{L_x \times d})$$
$$\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V (\in \mathbb{R}^{L_x \times d})$$

$$\boldsymbol{c} = \mathrm{Attention}(\boldsymbol{q}, \boldsymbol{K}, \boldsymbol{V}) = \underbrace{\text{softmax}\left(\frac{\boldsymbol{q}\boldsymbol{K}^T}{\sqrt{d}}\right)}_{\alpha} \boldsymbol{V}$$

Prediction: $\hat{\boldsymbol{y}} = h(\boldsymbol{c})$, with g and h parametric functions

Mutual attention (also known as cross attention)

- Queries from target domain, and keys and values from source domain

The iterative decoding process case

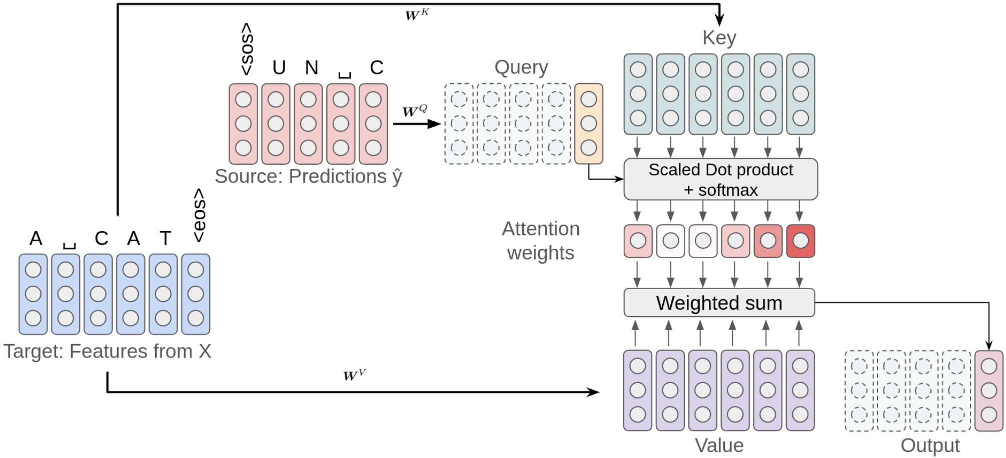
Inputs: $\boldsymbol{X}$: the features from the encoded input sequence (source domain),
$\tilde{\boldsymbol{z}} = \hat{\boldsymbol{y}}^{0:t-1}$: an ouput sequence (target domain)
$\boldsymbol{z}^t$: a query vector at iteration t ($\boldsymbol{z}^t = g(\hat{\boldsymbol{y}}^0, ..., \hat{\boldsymbol{y}}^{t-1})$).

$$\begin{aligned}
\boldsymbol{q}^t &= \boldsymbol{z}^t \boldsymbol{W}^Q (\in \mathbb{R}^d) \\
\boldsymbol{K} &= \boldsymbol{X}\boldsymbol{W}^K (\in \mathbb{R}^{L_x \times d}) \\
\boldsymbol{V} &= \boldsymbol{X}\boldsymbol{W}^V (\in \mathbb{R}^{L_x \times d})
\end{aligned}$$

$$\boldsymbol{c}^t = \mathrm{Attention}(\boldsymbol{q}^t, \boldsymbol{K}, \boldsymbol{V}) = \underbrace{\text{softmax}\left(\frac{\boldsymbol{q}^t \boldsymbol{K}^T}{\sqrt{d}}\right)}_{\alpha^t} \boldsymbol{V}$$

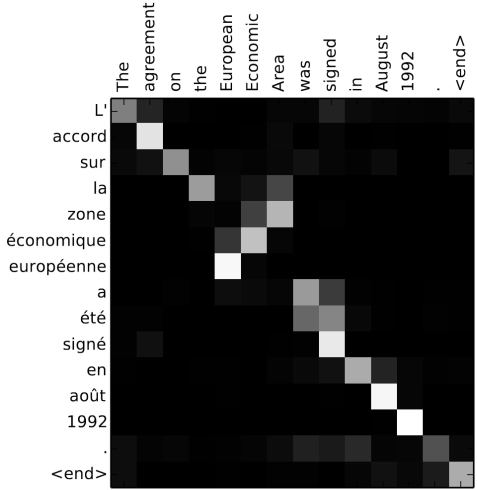
New prediction: $\hat{\boldsymbol{y}}^t = h(\boldsymbol{c}^t)$, with g and h parametric functions

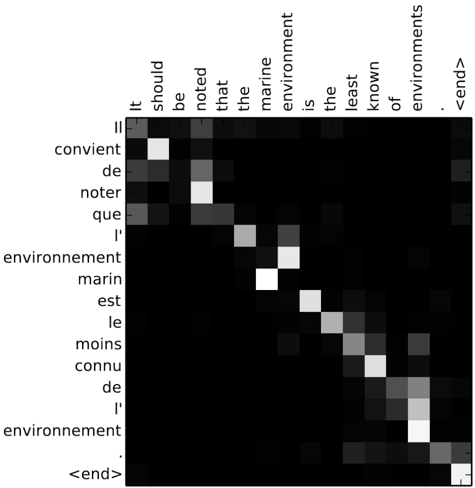
Mutual attention



➤ Output length = Query length ($\neq$ Key length = Value length)

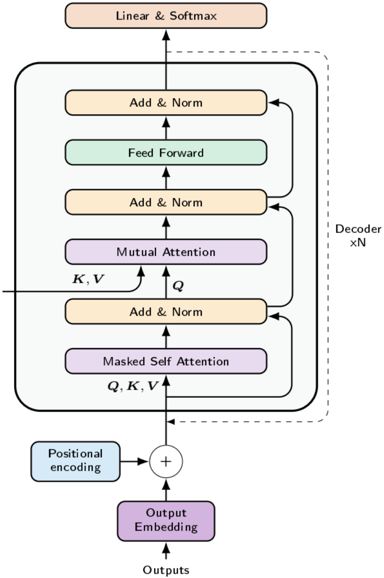
Mutual attention: attention map





Images from [2]

➤ Size: target sequence length $\times$ source sequence length

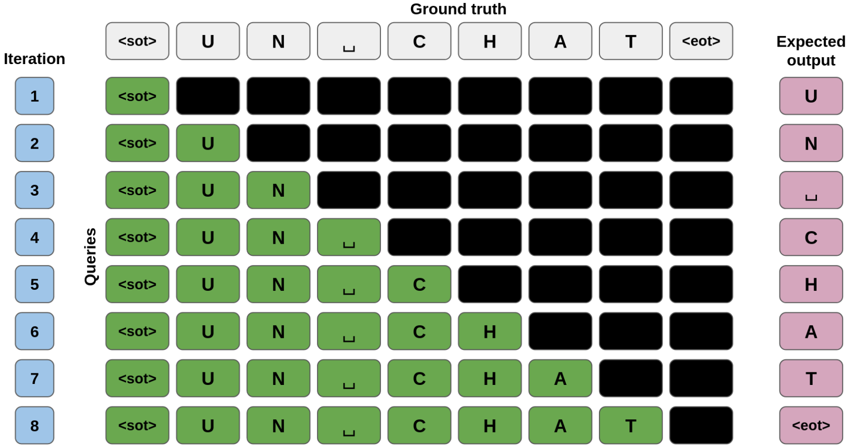


A stack of 6 transformer decoder layers

- Multi-head Masked Self Attention
  - ➤ Global query modeling
- Add: residual connections
  - ➤ Multi-scale representation
  - ➤ Reinforce identity
- Norm: layer normalization
  - ➤ Stabilize training (range of values)
- Multi-head mutual attention
  - ➤ Look for relevant information from source
- Feed Forward: 2 fully-connected layers
  - ➤ Local projection

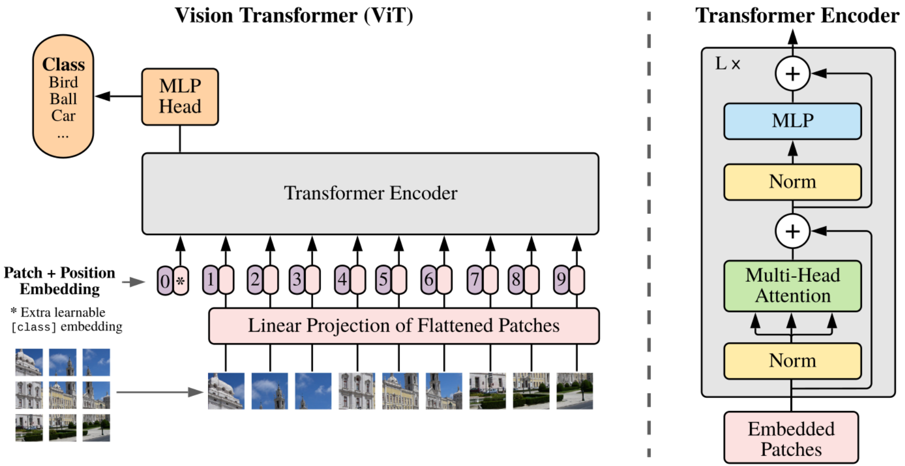
Masked self attention and teacher forcing

➤ Parallelize decoding process at training time using ground truth



Conclusion

- A modular approach: expressivity can be enhanced by adding more encoder/decoder layers
  - ➤ But slower training/inference and more required data
- Self attention: a way to model global context without sequential computations (at each layer, each token can attend to all the other tokens)
  - ➤ But a quadratic complexity
- Parameter-free additive positional encoding
  - ➤ But the relation between positions must be learned through training
- Defined for Natural Language Processing: SOTA architecture
  - ➤ How to adapt it for computer vision?



Input image: 224×224, patch size: $16 \times 16 \Rightarrow 196$ patches

Architecture	Top-1 accuracy (%) on ImageNet	# params (M)
ResNet-50	79.26	26
ResNet-101	80.13	45
ResNet-152	80.62	60
ViT-B/16	77.91	86
ViT-B/32	73.38	86
ViT-L/16	76.53	307
ViT-L/32	71.16	307

Attention-based architectures require more data to exploit their full potential:

- ➤ Spatial relations between patches must be learned from scratch
- ➤ What to do when there is not enough labeled data?

Table of contents

Create artificial examples

- Data augmentation
- Synthetic data

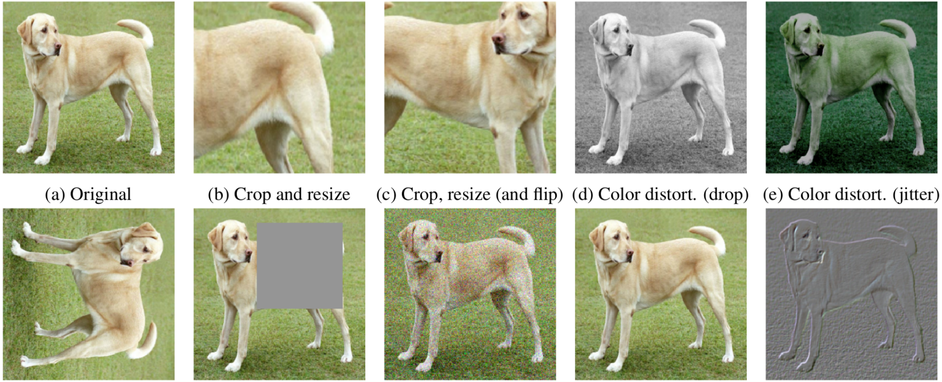
Benefit from other annotated datasets

- Transfer learning
- Fine-tuning

Benefit from unlabeled data

- Semi-supervised learning
- Self-supervised learning

- Deformation function applied on labeled data



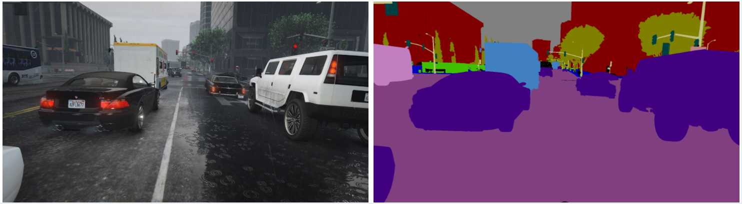
(a) Original (b) Crop and resize (c) Crop, resize (and flip) (d) Color distort. (drop) (e) Color distort. (jitter)

(f) Rotate $\{90°, 180°, 270°\}$ (g) Cutout (h) Gaussian noise (i) Gaussian blur (j) Sobel filtering

⚠Must preserve the information of interest (should not modify the associated ground truth)

➤ Create new couples (input, ground truth) from scratch

Video games as simulation engines



GTA V for semantic segmentation [4]

Digital fonts for handwritten text recognition

Idea

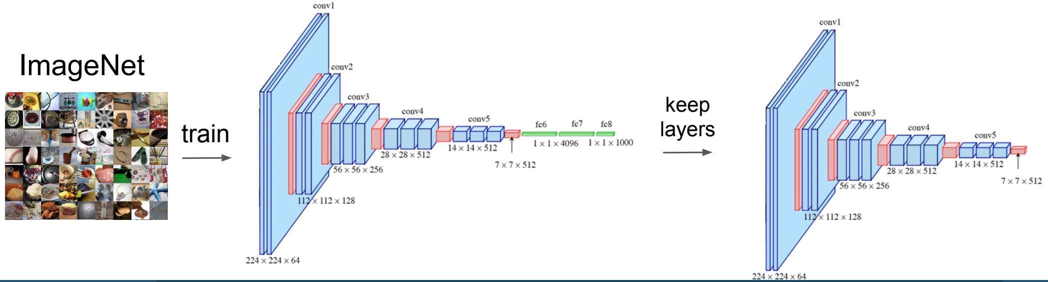
The knowledge learned from task A can be helpful for task B

➤ Benefit from annotated data of another task/dataset

First step: pre-training

Train on a source task/dataset

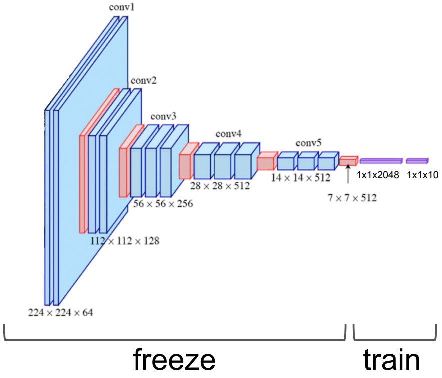
e.g., classification on ImageNet



Adaptation to target dataset

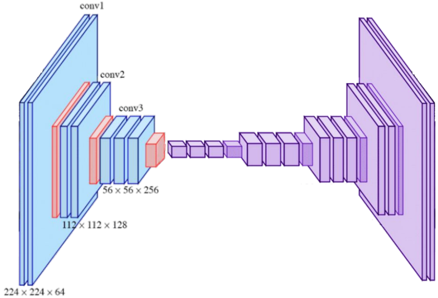
e.g., classification on CIFAR 10

- ➤ Classes different: change projection head
- ➤ Semantically similar: preserve top layers



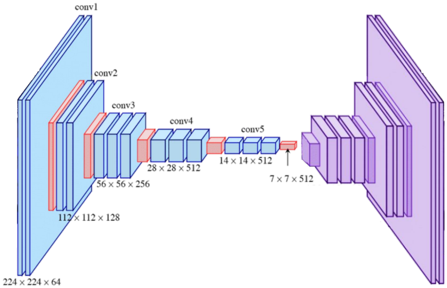
Adaptation to target task

e.g., segmentation



freeze | train

semantically far
much data



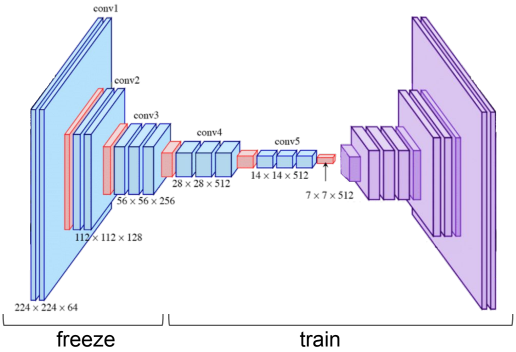
freeze | train

semantically close
few data

If enough target data: fine-tuning

Additional training step

➤ Training the transfered part of the architecture (either completely or top layers only), with low learning rate



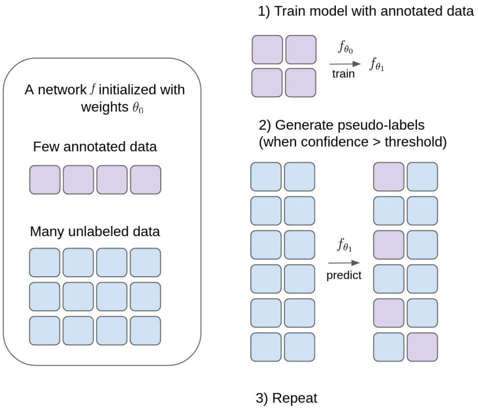
Getting back to ViT performance

Performance on ImageNet (fine-tuning included)

Architecture	Top-1 accuracy (%) on ImageNet	
	Pre-trained on ImageNet	Pre-trained on JFT-300M
ResNet-50	79.26	✗
ResNet-101	80.13	✗
ResNet-152	80.62	✗
ViT-B/16	77.91	84.15
ViT-B/32	73.38	80.73
ViT-L/16	76.53	87.12
ViT-L/32	71.16	84.37
ViT-H/14	✗	**88.04**

Idea: use prediction as ground truth

- Use trained model to generate pseudo-labels from unlabeled data
- Further train the model with both annotated data and pseudo-labels



Idea: Learn from unlabeled data

Two main approaches

Generative architecture

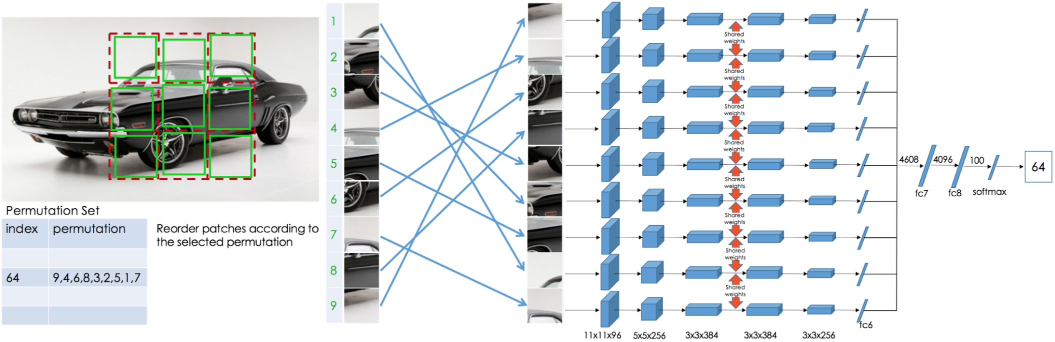
The goal is to solve a pretext task

- Solve Jigsaw puzzles
- Reconstruct the input image:
  (Variational) Auto-encoders, Masked auto-encoders

Joint Embedding Architecture

- Contrastive learning: SimCLR
  Learning from positive and negative samples
- Non-contrastive learning: DINO
  Learning from different representations of the same input

➤ Find the right permutation: classification formulation



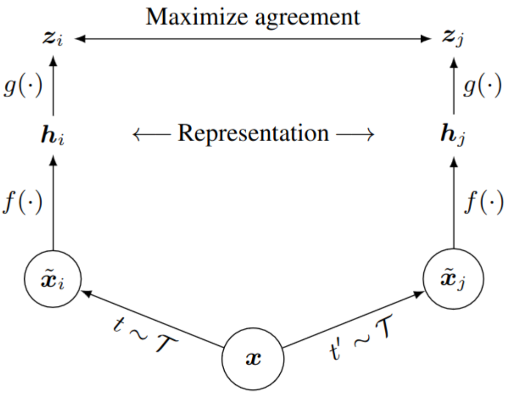
Contrastive learning: SimCLR (2020) [6]

➤ Idea: generate two views for each input (= positive pair). Bring latent representations of positives closer and keep latent representations of negatives away

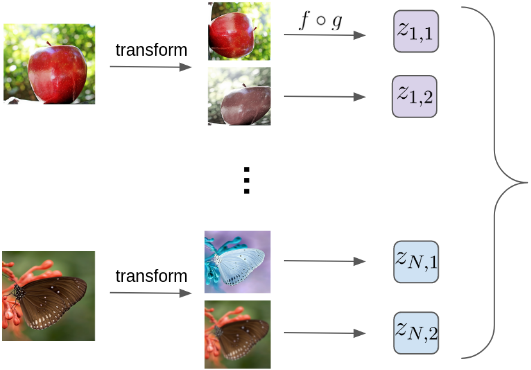
➤ Siamese networks: same architecture with same weights used on different inputs while comparing outputs

Algorithm 1 SimCLR's main learning algorithm.

input: batch size N, constant τ, structure of f, g, $\mathcal{T}$.
for sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
  for all $k \in \{1, \ldots, N\}$ **do**
    draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
    # the first augmentation
    $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
    $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
    # the second augmentation
    $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
    $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
    $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
  end for
  for all $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
    $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
  end for
  define $\ell(i,j)$ **as** $\ell(i,j) = -\log \frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(s_{i,k}/\tau)}$
  $\mathcal{L} = \frac{1}{2N} \sum_{k=1}^{N} [\ell(2k-1, 2k) + \ell(2k, 2k-1)]$
  update networks f and g to minimize $\mathcal{L}$
end for
return encoder network $f(\cdot)$, and throw away $g(\cdot)$



- Find the matching pairs
- Cosine similarity as distance metric



	$z_{1,1}$	$z_{1,2}$	⋯	$z_{N,1}$	$z_{N,2}$
$z_{1,1}$	X	1	⋯	0	0
$z_{1,2}$	1	X	⋯	0	0
⋮	⋮	⋮	X	⋮	⋮
$z_{N,1}$	0	0	⋯	X	1
$z_{N,2}$	0	0	⋯	1	X

- Efficiency depends on mini-batch size

Classification performance (Top-1 accuracy) with ResNet architecture

	Food	CIFAR10	CIFAR100	Birdsnap	SUN397	Cars	Aircraft	VOC2007	DTD	Pets	Caltech-101	Flowers
Linear evaluation:												
SimCLR (ours)	68.4	90.6	71.6	37.4	58.8	50.3	50.3	80.5	**74.5**	83.6	90.3	91.2
Supervised	**72.3**	**93.6**	**78.3**	**53.7**	**61.9**	**66.7**	**61.0**	**82.8**	**74.9**	**91.5**	**94.5**	**94.7**
Fine-tuned:												
SimCLR (ours)	**88.2**	**97.7**	**85.9**	**75.9**	63.5	91.3	**88.1**	84.1	**73.2**	89.2	92.1	97.0
Supervised	**88.3**	**97.5**	**86.4**	**75.8**	**64.3**	**92.1**	86.0	**85.0**	**74.6**	**92.1**	**93.3**	**97.6**
Random init	86.9	95.9	80.2	**76.1**	53.6	91.4	85.9	67.3	64.8	81.5	72.6	92.0

Pre-training on ImageNet

Backbone either frozen (linear evaluation=transfer learning) or trained (fine-tuning)

➤ Self-distillation through teacher-student paradigm

Algorithm 1 DINO PyTorch pseudocode w/o multi-crop.

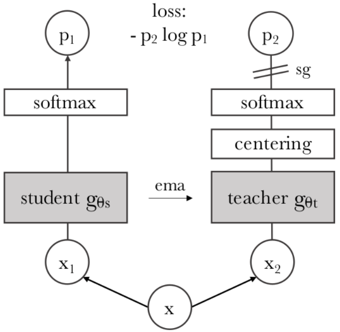
```
# gs, gt: student and teacher networks
# C: center (K)
# tps, tpt: student and teacher temperatures
# l, m: network and center momentum rates
gt.params = gs.params
for x in loader: # load a minibatch x with n samples
    x1, x2 = augment(x), augment(x) # random views

    s1, s2 = gs(x1), gs(x2) # student output n-by-K
    t1, t2 = gt(x1), gt(x2) # teacher output n-by-K

    loss = H(t1, s2)/2 + H(t2, s1)/2
    loss.backward() # back-propagate

    # student, teacher and center updates
    update(gs) # SGD
    gt.params = l*gt.params + (1-l)*gs.params
    C = m*C + (1-m)*cat([t1, t2]).mean(dim=0)

def H(t, s):
    t = t.detach() # stop gradient
    s = softmax(s / tps, dim=1)
    t = softmax((t - C) / tpt, dim=1) # center + sharpen
    return - (t * log(s)).sum(dim=1).mean()
```



x_1, x_2 are two augmentations of an image x (crop vs whole image).
Student trained to output same distribution as teacher with cross-entropy loss (H)
Stop-gradient (sg): only the student is trained through gradient descent.
Exponential moving average (ema): $\theta_t = \lambda\theta_t + (1-\lambda)\theta_s \quad \lambda \in [0.996, 1]$

DINO (self-DIstillation with NO labels, 2021) [7]

➤ Keywords

Knowledge distillation / teacher-student paradigm

Transfering knowledge from one network to another (generally for compression purposes).
e.g., a small network (student) is trained to mimick the output of a bigger network (teacher).

Self-distillation

Teacher and student share the same architecture (but not the same weights: these are not siamese network)
Teacher weights are not updated through gradient descent but with EMA from student weights.

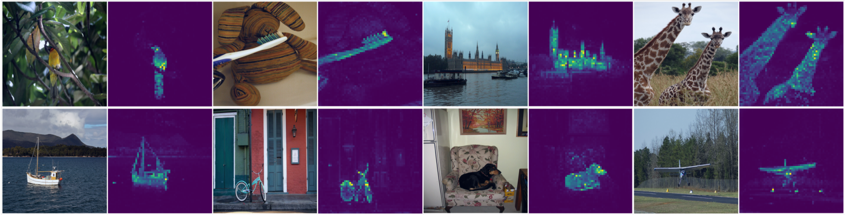
Failing modes

One dimension dominates, no matter the input:

- Centering: $t \leftarrow t - C$ with $C \leftarrow mC + (1-m)\frac{1}{B}\sum_{i=1}^{B} g_{\theta_t(x_i)}$

Uniform distribution, no matter the input:

- Sharpening: $t \leftarrow t/\tau_t$



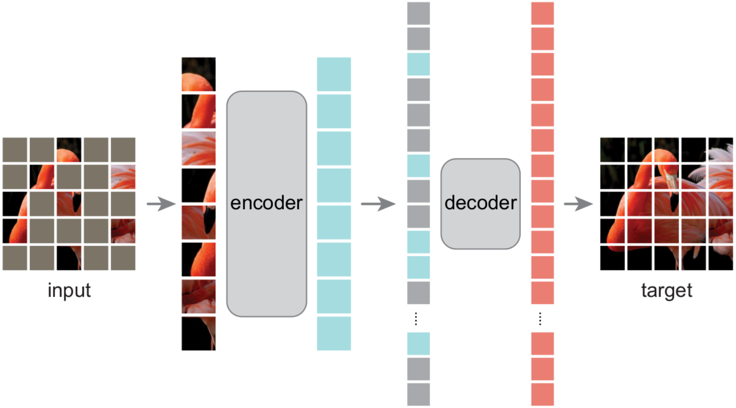
Attention maps: self-attention for CLS token

Classification performance (Top-1 accuracy) with ViT architecture

	Cifar_{10}	Cifar_{100}	INat_{18}	INat_{19}	Flwrs	Cars	INet
ViT-S/16							
Sup. [69]	**99.0**	89.5	70.7	76.6	98.2	92.1	79.9
DINO	**99.0**	**90.5**	**72.0**	**78.2**	**98.5**	**93.0**	**81.5**
ViT-B/16							
Sup. [69]	99.0	90.8	**73.2**	77.7	98.4	92.1	81.8
DINO	**99.1**	**91.7**	72.6	**78.6**	**98.8**	**93.0**	**82.8**

Pre-training on ImageNet

- Reconstruct the missing parts (pixel-level MSE loss)

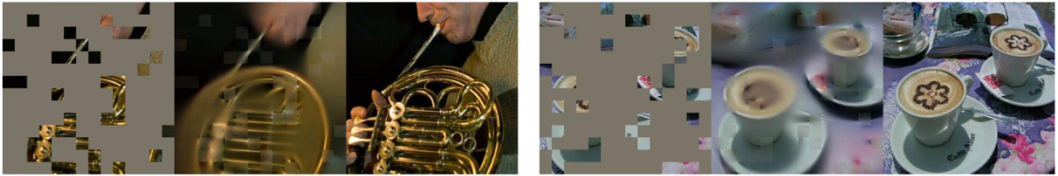


- Vision Transformer architecture

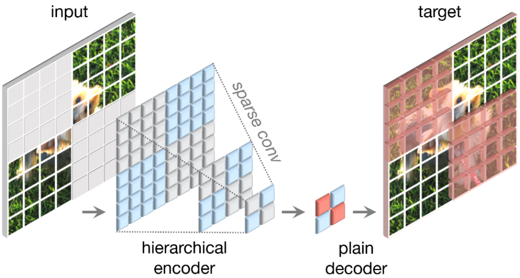
Masked auto-encoder (2022) [8]

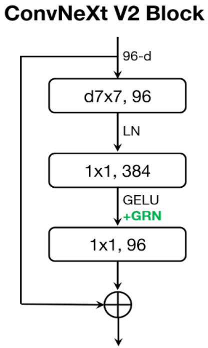
➤ Classification performance (Top-1 accuracy) with ViT architectures, pre-trained and evaluated on ImageNet

Approach	ViT-B	ViT-L	ViT-H	ViT-H_{488}
Supervised	82.3	82.6	83.1	✗
DINO	82.8	✗	✗	✗
MAE	83.6	85.9	86.9	**87.8**



Left: masked input, Middle: reconstructed image, Right: ground truth





- ➤ 88.9% top-1 accuracy on ImageNet (659M params)

Be careful with benchmarks V2

Architecture is not everything, be careful with benchmarks!

- Training time, number of epochs/iterations, mini-batch size
- Weight initialization, optimizer, initial learning rate, learning rate scheduler
- Dropout, normalization, activation functions
- Pre-processing, post-processing
- **Pre-training, transfer learning, data augmentation, synthetic data**

➤ Really difficult to fairly compare approaches

Conclusion

Architectures

- Deeper and deeper → more efficient but requires more data
- Powerful attention mechanism → requires even more data

Data is crucial

- There are many ways to deal with few labeled data
- But still an active field of research (*e.g.*, medical data)

Classification

- Performance constantly increasing but still not perfect
- Task limited (one instance per image, no location information)

➤ Next time: Practical Session!

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30 (NIPS)*. 2017, pp. 5998–6008.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *3rd International Conference on Learning Representations*. 2015.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations*. 2021.

[4] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. "Playing for Data: Ground Truth from Computer Games". In: *14th European Conference on Computer Vision*. Vol. 9906. 2016, pp. 102–118.

[5] Mehdi Noroozi and Paolo Favaro. "Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles". In: *14th European Conferenceon Computer Vision*. Vol. 9910. 2016, pp. 69–84.

[6] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. "A Simple Framework for Contrastive Learning of Visual Representations". In: *Proceedings of the 37th International Conference on Machine Learning*. Vol. 119. Proceedings of Machine Learning Research. 2020, pp. 1597–1607.

[7] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. "Emerging Properties in Self-Supervised Vision Transformers". In: *International Conference on Computer Vision*. 2021, pp. 9630–9640.

[8] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross B. Girshick. "Masked Autoencoders Are Scalable Vision Learners". In: *Conference on Computer Vision and Pattern Recognition*. 2022, pp. 15979–15988.

[9] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. "ConvNeXt V2: Co-designing and Scaling ConvNets with Masked Autoencoders". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, 2023, pp. 16133–16142.