# Invariance, equivariance, and inductive bias in deep learning

**Domenico Tortorella**

Computational Intelligence and Machine Learning (CIML)
Department of Computer Science, University of Pisa

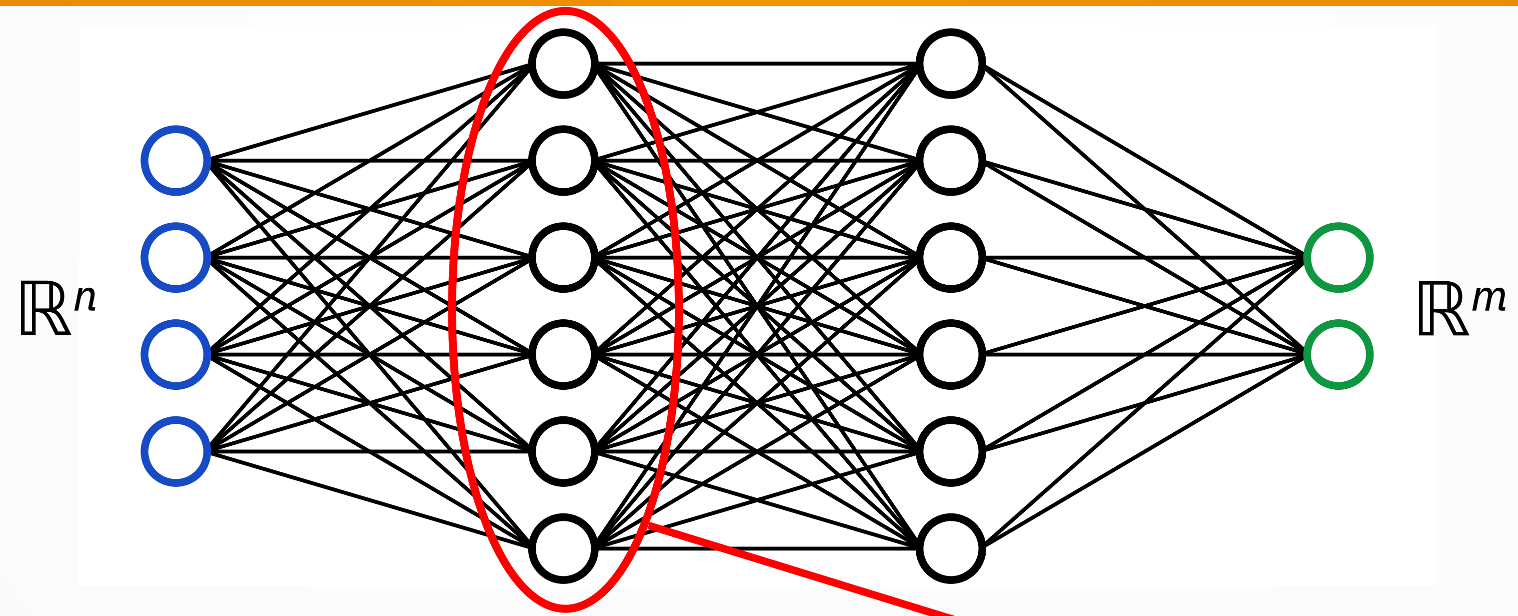
Università di Pisa
Computational Intelligence and Machine Learning Group

# Outline

- Introduction
- A familiar example: CNNs
- A primer on groups and symmetries
- Building invariant/equivariant neural networks
- Invariance/equivariance in action: Sets
- Invariance/equivariance in action: Graphs
- Wrap up

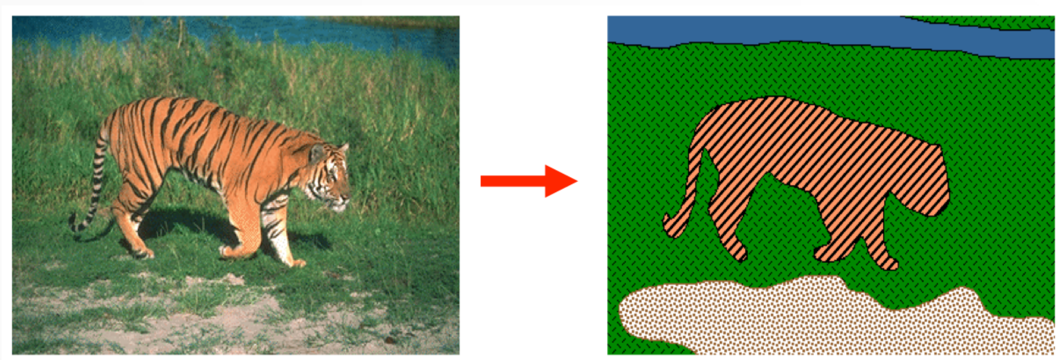
# A familiar example: CNNs

# Neural networks

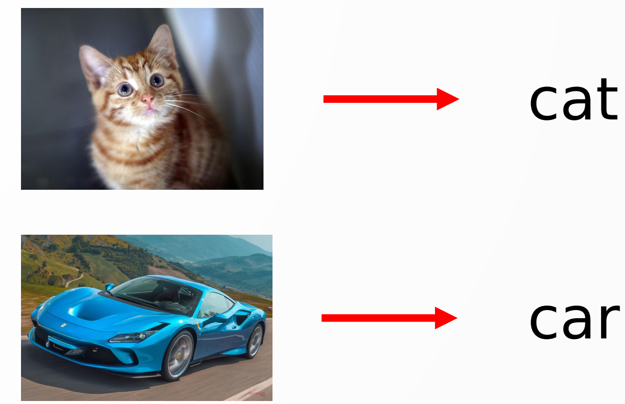



- Composition of simple functions $\mathbf{x} \mapsto \sigma(\mathbf{W}\,\mathbf{x} + \mathbf{b})$
- Trained by minimizing a loss over weights $\mathbf{W}$, $\mathbf{b}$

# Image recognition



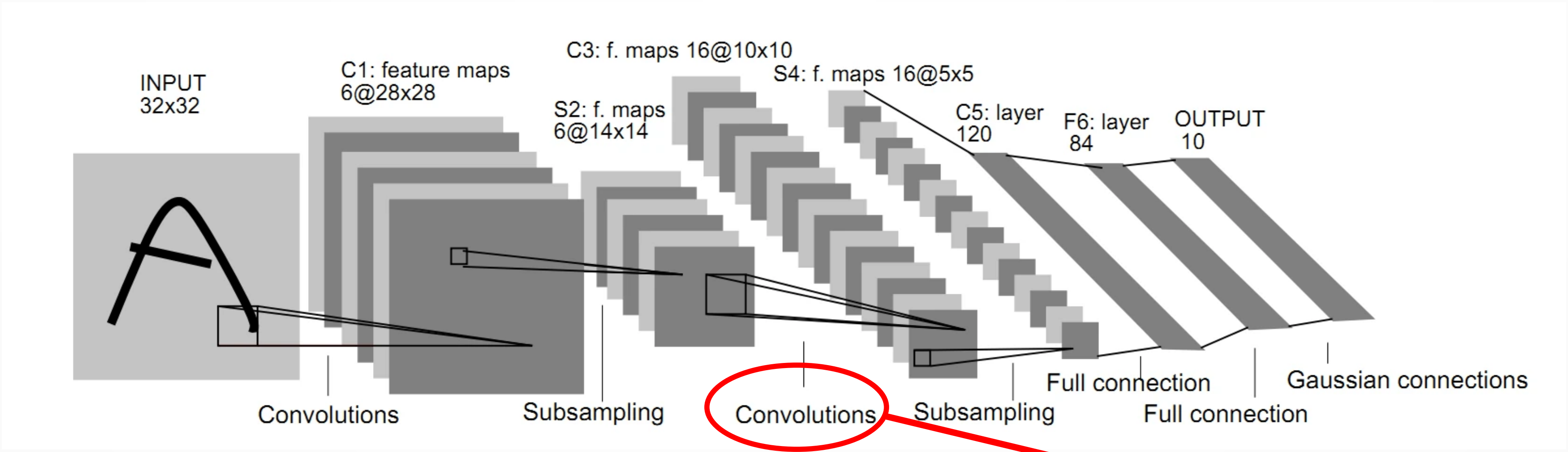
SEGMENTATION




CLASSIFICATION

- Images, instead of vectors
- Functions acting on tensors $\mathbb{R}^{n \times m \times d}$

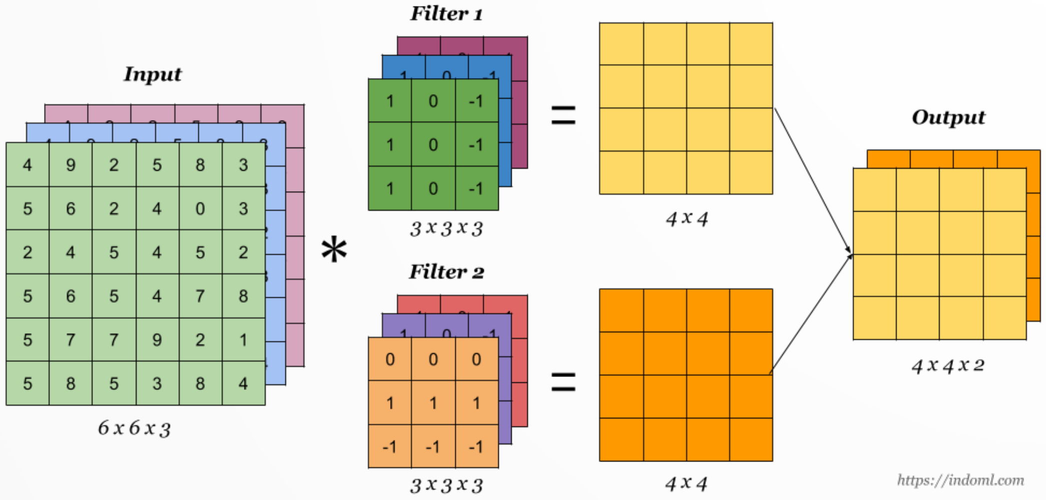
# Convolutional Neural Networks




LeNet-5

- Composition of simple functions $\mathbf{x} \mapsto \sigma( \boxed{\mathbf{W} \star \mathbf{x}} + \mathbf{b} )$
  - plus sub-sampling, optionally a final MLP
- Trained by minimizing a loss over weights **W**, **b**

# Convolution (2D)

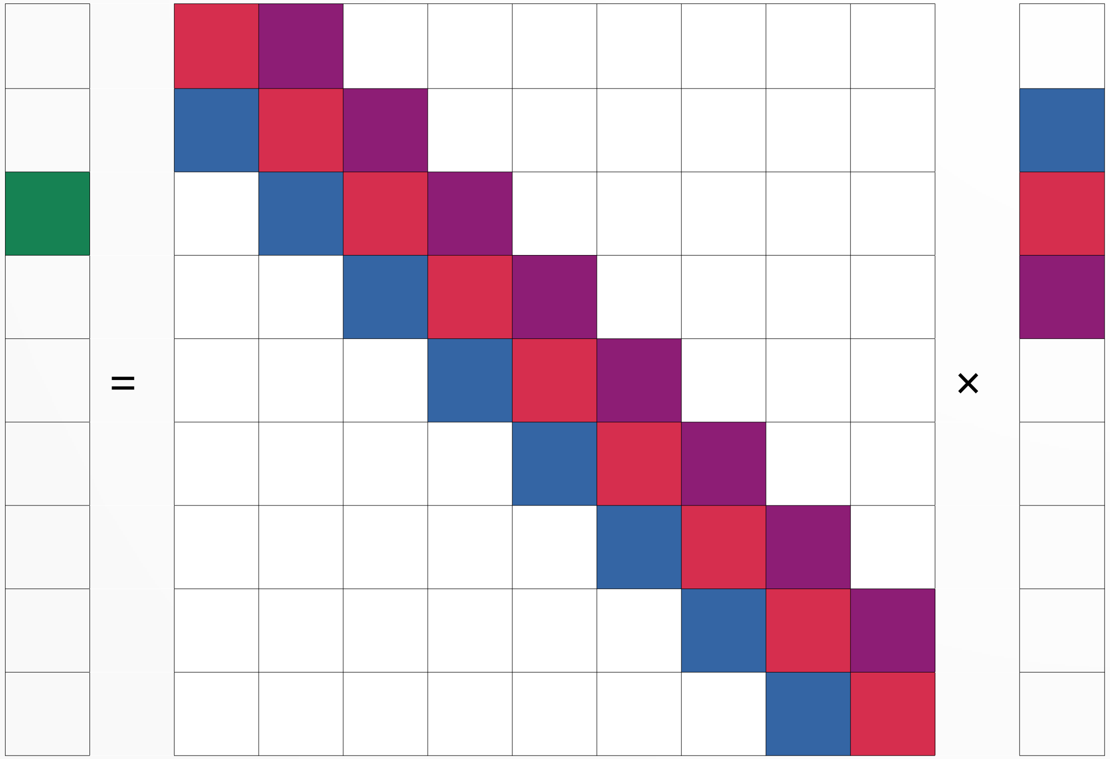



- Action of $\mathbf{W} \star \mathbf{x}$ on image

- Filter **W** acts locally in each pixel and its neighbourhood, sliding across image
- Formally,

$$\mathbf{x}'_{i,j} = \sum_{-k \leq i' \leq k} \sum_{-k \leq j' \leq k} \mathbf{W}_{i',j'} \, \mathbf{x}_{(i+i'),(j+j')}$$

# Convolution (1D)




- Convolution viewed as matrix product

- Convolution is in fact weight sharing
- A linear function

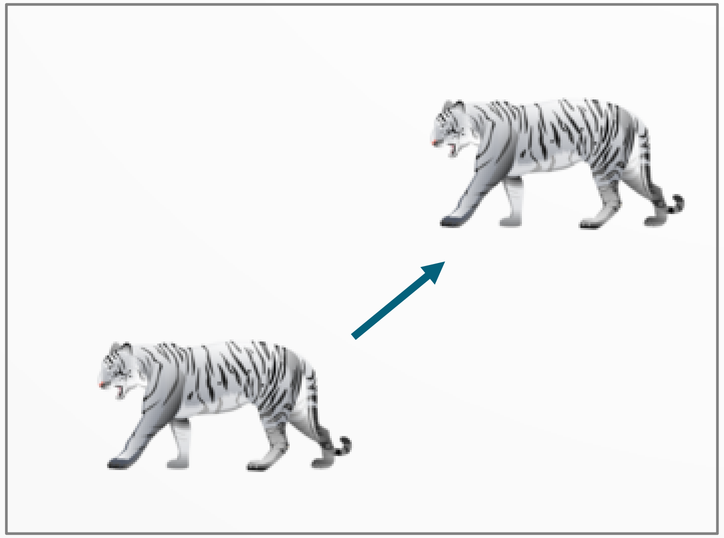
$$\mathbb{R}^{n \times m \times d} \rightarrow \mathbb{R}^{n' \times m' \times d'}$$

but with way less than $n \times m \times d \times n' \times m' \times d'$ parameters

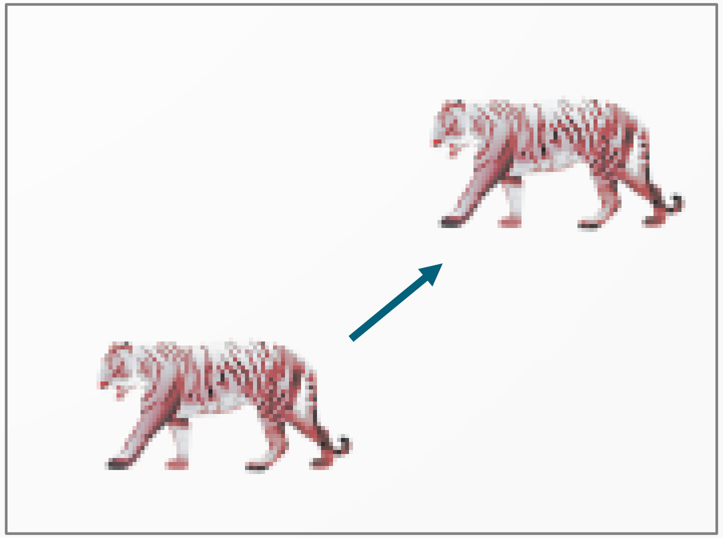
# Translation equivariance

- Why doing convolution?
  - Equivariance to image translations

$\mathbf{W} \star$



=



# Other equivariances

- What about other transformations?
  - Reflections
  - Rotations


- CNNs are **not** equivariant w.r.t. them
  - Do data augmentation (i.e. learn equivariance/invariance)
  - Change model to include this inductive bias! [G-CNN]

# A primer on groups and symmetries

# Groups

- A set $\mathcal{G}$ with a binary operation $\cdot$ satisfying
  - (Associativity) $\mathfrak{a}\cdot(\mathfrak{b}\cdot\mathfrak{c}) = (\mathfrak{a}\cdot\mathfrak{b})\cdot\mathfrak{c}$ for all $\mathfrak{a}, \mathfrak{b}, \mathfrak{c} \in \mathcal{G}$
  - (Identity) $\exists\, \mathfrak{e} \in \mathcal{G}$ such that $\mathfrak{e}\cdot\mathfrak{a} = \mathfrak{a}\cdot\mathfrak{e} = \mathfrak{a}$
  - (Inverse) $\exists\, \mathfrak{a}^{-1} \in \mathcal{G}$ such that $\mathfrak{a}^{-1}\cdot\mathfrak{a} = \mathfrak{a}\cdot\mathfrak{a}^{-1} = \mathfrak{e}$
  - (Closure) $\mathfrak{a}, \mathfrak{b} \in \mathcal{G} \Rightarrow \mathfrak{a}\cdot\mathfrak{b} \in \mathcal{G}$ [for sub-groups]
- Transformations on objects “behave” like a group

# Group actions

- How $x \in S$ is transformed by $\mathfrak{g} \in \mathcal{G}$, $\quad \mathbf{x} \mapsto \mathfrak{g}.\mathbf{x}$
- An action must satisfy
  - (Identity) $\mathfrak{e}.\mathbf{x} = \mathbf{x}$
  - (Compatibility) $\mathfrak{g}.(\mathfrak{h}.\mathbf{x}) = (\mathfrak{g}\cdot\mathfrak{h}).\mathbf{x}$

$\mathfrak{g} \quad . \quad \mathbf{x} \qquad\qquad \mathfrak{g}.\mathbf{x}$

# Invariance, equivariance

- W.r.t. a function $f$: $S \to S'$
- Invariance: transformations do not affect result

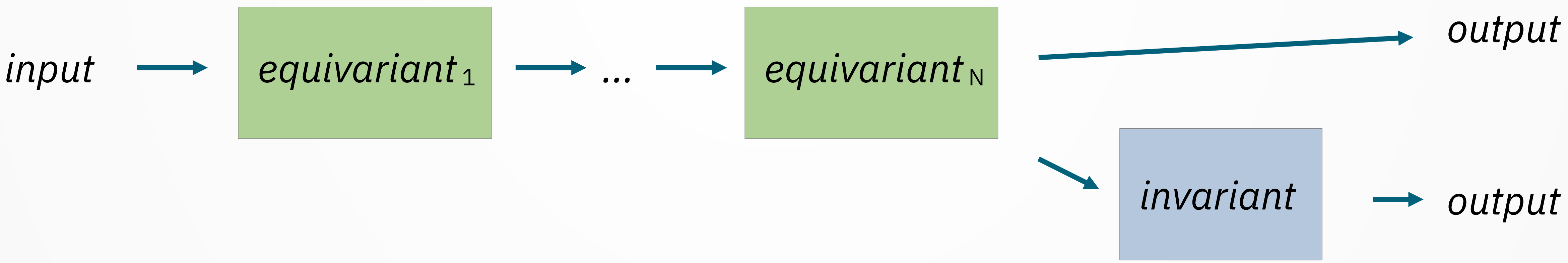
$$f(\mathfrak{g}.\mathbf{x}) = f(\mathbf{x})$$

- Equivariance: output transforms as input

$$f(\mathfrak{g}.\mathbf{x}) = \mathfrak{g}.f(\mathbf{x})$$

- Such functions preserve “symmetries” in data

# Back to neural networks




- Composition of equivariant layers
  - Optionally, a final invariant layer
  - Point-wise activations σ(·), and channel-wise functions (e.g. MLPs) are invariant/equivariant

# Invariance/equivariance in action: Sets

# Tasks on (multi-)sets

- Element classification/regression
  - $\{x_1, x_2, \ldots, x_n\} \rightarrow \{y_1, y_2, \ldots, y_n\}$
- Set classification/regression
  - $\{x_1, x_2, \ldots, x_n\} \rightarrow y$
- Early approaches used RNN
  - What about element order?

# Permutation group

- Group $\mathcal{S}_n$ of all bijective functions $\{1..n\} \to \{1..n\}$
  - Group operation is function composition
- A $\pi \in \mathcal{S}_n$ acts by swapping elements in a sequence
  - $\pi.\langle x_1, x_2, x_3, x_4, x_5 \rangle = \langle x_1, x_3, x_4, x_2, x_5 \rangle$
- A neural network for sets should be invariant/ equivariant w.r.t. permutations

# Neural nets for sets

- Two linear equivariant operations
  - Identity
  - Sum of all elements

$$\begin{bmatrix} \ \\ \ \\ \ \\ \ \\ \ \end{bmatrix} = \boldsymbol{\alpha} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} \ \\ \ \\ \ \\ \ \\ \ \end{bmatrix} + \boldsymbol{\beta} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \ \\ \ \\ \ \\ \ \\ \ \end{bmatrix}$$

- Only one linear invariant function
  - Sum of all elements (aggregation)

$$\begin{bmatrix} \ \end{bmatrix} = \boldsymbol{\alpha} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} \ \\ \ \\ \ \\ \ \\ \ \end{bmatrix}$$

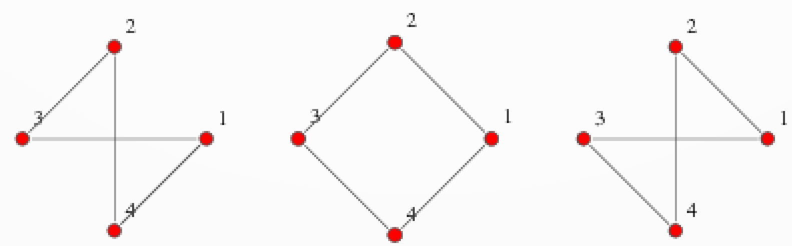
# Deep sets

- DeepSets combines equivariant functions in layers, plus invariant in final (also, MLP)
  - Multi-channel, like CNNs
- Other permutation-invariant functions can be used to perform aggregation (max, mean, ...)
  - but only sum is linear

# Invariance/equivariance in action: Graphs

# Graphs

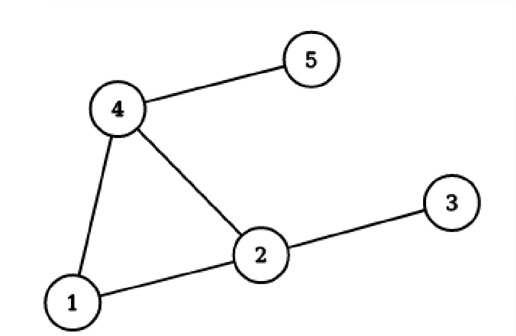
- A set of vertices $V$ and a set of edges $E \subseteq V \times V$
- Vertices and edges may have labels
- Edges can be represented by an adjacency matrix



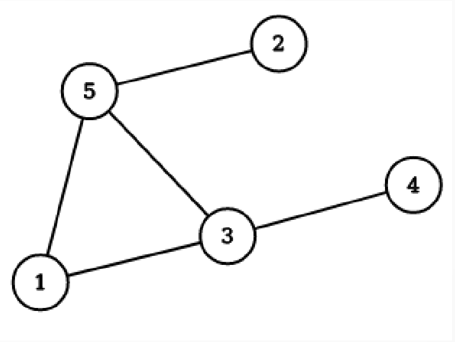
$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

# Graph isomorphism

- Changing vertex "numbering" does not change graphs




=




- But the adjacency matrix *does change*!

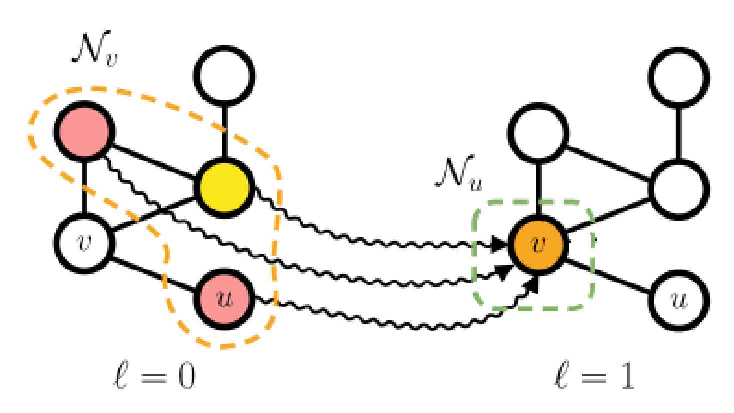
| | | | | |
|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |

$\neq$

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 |

# Graph convolution networks

- Layers act locally on vertex neighbourhood
  - Reordering vertices do not affect encoding




- At most as representative as 2-WL isomorphism test
  - Higher order equivariant networks are provably more representative (*k*-order ⇒ *k*-WL)

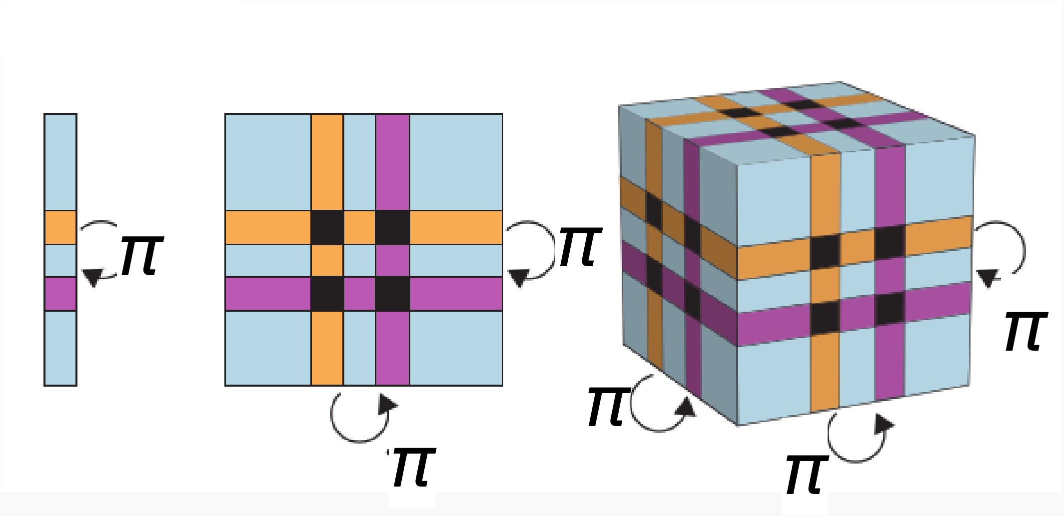
# Message-passing networks

- Message-passing layers consisting of
  - Neighborhood aggregation: $\boldsymbol{a}_v^{(l)} = \textsc{Aggregate}^{(l)}(\{\boldsymbol{h}_u^{(l-1)} \mid u \in \mathcal{N}(v)\})$
  - Context combination: $\boldsymbol{h}_u^{(l)} = \textsc{Combine}^{(l)}(\boldsymbol{h}_u^{(l-1)}, \boldsymbol{a}_v^{(l)})$
- The two steps can also be implemented by a single function
- A final readout layer to encode the graph: $\boldsymbol{h}_G = \textsc{Readout}(\{\boldsymbol{h}_v^{(L)} \mid v \in V\})$
- According to the different choice of the three functions, we can have different Graph Convolutional Networks

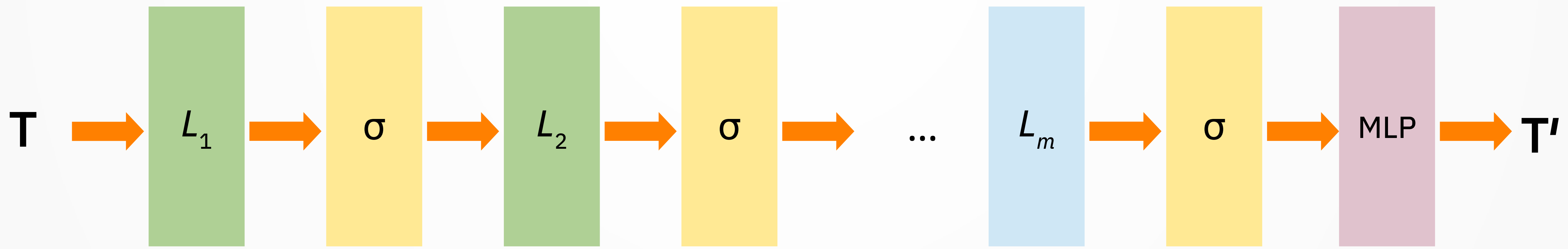
# Invariant and equivariant layers

- Given a permutation group $\mathcal{S}_n$, a linear function
  - $L: \mathbb{R}^{n^k} \to \mathbb{R}^{n^{k'}}$ is ***equivariant*** iff $L(\pi.\mathbf{x}) = \pi.L(\mathbf{x})$ for all $\pi \in \mathcal{S}_n$, $\mathbf{x} \in \mathbb{R}^{n^k}$
  - $L: \mathbb{R}^{n^k} \to \mathbb{R}$ is ***invariant*** iff $L(\pi.\mathbf{x}) = L(\mathbf{x})$ for all $\pi \in \mathcal{S}_n$, $\mathbf{x} \in \mathbb{R}^{n^k}$
- Specifically, we consider the symmetric group $\mathcal{S}_n$, i.e. the group of all permutations of 1..*n*, acting on *k*-order tensors in the following way:

$$(\pi \,.\, \mathbf{x})_{i_1,\,\ldots,\,i_k} = \mathbf{x}_{\pi^{-1}(i_1),\,\ldots,\,\pi^{-1}(i_k)}$$



# Invariant and equivariant nets




- Take as input a tensor encoding of the graph and its vertex/edge features
  - e.g., the adjacency matrix (i.e. 2-tensor) **A**, a 3-tensor $\mathbf{T}_{ii:} = \boldsymbol{l}_i$, a 3-tensor $\mathbf{T}_{ij:} = \boldsymbol{l}_{ij}$, their composition, etc.
- Composition of equivariant layers and point-wise non-linearities (can be of different orders), with an invariant final layer for invariant networks (optionally followed by a MLP)
- Produce vertex-level representations (equivariant network), or graph-level representations (invariant network)

# Basis for inv't and equiv't layers

- Since $L$ is a linear operator, it can be expressed as a linear combination of basis operators
- Invariant and equivariant operators can be represented as $\mathbb{R}^{n^{k'} \times n^k}$ and $\mathbb{R}^{1 \times n^k}$ matrices acting on $\mathbb{R}^{n^k}$ vectors with the standard matrix product
- Invariance/equivariance ⇒ weight sharing, way less than $n^k / n^{k+k'}$ parameters:
  - invariance → $\mathbf{L}\,\mathbf{x} = \mathbf{L}\,(\pi.\mathbf{x})$ ⇒ $\mathbf{L}_{\boldsymbol{i}} = \mathbf{L}_{\pi(\boldsymbol{i})}$ for all $\boldsymbol{i}$, $\pi$
  - equivariance → $\pi.(\mathbf{L}\,\mathbf{x}) = \mathbf{L}\,(\pi.\mathbf{x})$ ⇒ $\mathbf{L}_{\boldsymbol{i},\pi^{-1}(\boldsymbol{j})} = \mathbf{L}_{\pi(\boldsymbol{i}),\boldsymbol{j}}$ for all $\boldsymbol{ij}$, $\pi$

# Basis for inv’t and equiv’t layers

- Consider multi-indices $\boldsymbol{i} = (i_1, \ldots, i_k) \in \{1..n\}^k$, with $\pi(\boldsymbol{i}) = (\pi(i_1), \ldots, \pi(i_k))$ for all permutations $\pi \in \mathcal{S}_n$
- Define the equivalence relation $\boldsymbol{a} \sim \boldsymbol{b}$ iff $a_i = a_j \Leftrightarrow b_i = b_j$, i.e. they have the same equality patterns: $(1,2,1) \sim (3,4,3)$
- The equivalence classes $\gamma \in \{1..n\}^k/\sim$ are closed under group $\mathcal{S}_n$ actions: $\boldsymbol{a} \sim \pi(\boldsymbol{a})$ for all $\pi \in \mathcal{S}_n$, $\boldsymbol{a} \in \{1..n\}^k$
- An orthogonal basis is defined as $\mathbf{B}_{\boldsymbol{a}}^{(\gamma)} = [\![\boldsymbol{a} \in \gamma]\!]$, with dimension $\mathbb{b}(k)$, i.e. the number of different partitions of a $k$-element set
- Biases can be added (invariant/equivariant $\mathbf{C}_{\boldsymbol{a}}^{(\gamma)}$ constants), and a $d$-dimensional feature channel can be appended

# Wrap-up

# Conclusions

- Data has **symmetries**
  - e.g. isometries don't change object category
- Traditionally, learnt by training with data augmentation
- ***Include symmetries in inductive bias by using equivariant neural networks***
  - Reduce parameters, data

# References


- Michael M. Bronstein, Joan Bruna, Taco Cohen, Petar Veličković (2021). **Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges.** Retrieved from https://arxiv.org/abs/2104.13478
- Ian Goodfellow, Yoshua Bengio, Aaron Courville (2015). **Deep learning.** MIT Press. Available at http://www.deeplearningbook.org
- Cohen, T. S., & Welling, M. (2016). **Group Equivariant Convolutional Networks.** In Proceedings of the 33rd International Conference on Machine Learning (Vol. 48, pp. 2990–2999). Retrieved from http://proceedings.mlr.press/v48/cohenc16.html
- Tai, K. S., Bailis, P., & Valiant, G. (2019). **Equivariant transformer networks.** In Proceedings of the 36th International Conference on Machine Learning (Vol. 97, pp. 6086–6095). Retrieved from http://proceedings.mlr.press/v97/tai19a.html

# References


- Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2019). **How Powerful are Graph Neural Networks?** Proceedings of the International Conference on Learning Representations (2019). Retrieved from http://arxiv.org/abs/1810.00826
- Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., & Borgwardt, K. M. (2011). **Weisfeiler-Lehman Graph Kernels**. The Journal of Machine Learning Research, 12, 2539–2561. https://doi.org/10.1.1.232.1510
- Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., & Smola, A. J. (2017). **Deep Sets**. Advances in Neural Information Processing Systems 30 (NIPS 2017) (pp. 3391–3401). Retrieved from http://arxiv.org/abs/1703.06114
- Wagstaff, E., Fuchs, F. B., Engelcke, M., Posner, I., & Osborne, M. (2019). **On the Limitations of Representing Functions on Sets**. Proceedings of the 36th International Conference on Machine Learning (pp. 6487–6494). Retrieved from http://arxiv.org/abs/1901.09006

# References


- Maron, H., Ben-Hamu, H., Shamir, N., & Lipman, Y. (2019a). **Invariant and Equivariant Graph Networks**. Proceedings of the International Conference on Learning Representations (2019). Retrieved from http://arxiv.org/abs/1812.09902
- Maron, H., Fetaya, E., Segol, N., & Lipman, Y. (2019b). **On the Universality of Invariant Networks**. Proceedings of the 36th International Conference on Machine Learning (pp. 4363–4371). Retrieved from http://arxiv.org/abs/1901.09342
- Maron, H., Ben-Hamu, H., Serviansky, H., & Lipman, Y. (2019c). **Provably Powerful Graph Networks**. Retrieved from http://arxiv.org/abs/1905.11136
- Micheli, A. (2009). **Neural Network for Graphs: A Contextual Constructive Approach**. IEEE Transactions on Neural Networks, 20(3), 498–511. https://doi.org/10.1109/TNN.2008.2010350