JOHANNES KEPLER
UNIVERSITY LINZ

Introduction to graph neural networks with applications to binding site identification

Günter Klambauer
ELLIS program Director (ML for molecule discovery)
Artificial Intelligence in Life Sciences group
LIT AI Lab & Institute for Machine Learning

JOHANNES KEPLER
UNIVERSITY LINZ
Altenberger Strasse 69
4040 Linz, Austria
jku.at

JOHANNES KEPLER UNIVERSITY LINZ

Overview

- Part 1:
  - Recap on Deep Learning and multi-layer perceptrons
  - Intro to Geometric Deep Learning and Graph Neural Networks (GNNs)
  - General applications
- Part 2:
  - EGNNs for binding site identification

Graph neural networks and chemistry

- Graph neural networks (GNN) are a versatile technique in cheminformatics and computer-aided drug discovery (CADD)
  - **Bioactivity and property prediction** and QSAR
  - Forward- and retro**synthesis** prediction
  - Molecular **encoders**: encode a molecule as a vector of features
  - Basis for **generative models**
  - **Representation learning** tasks on macromolecules

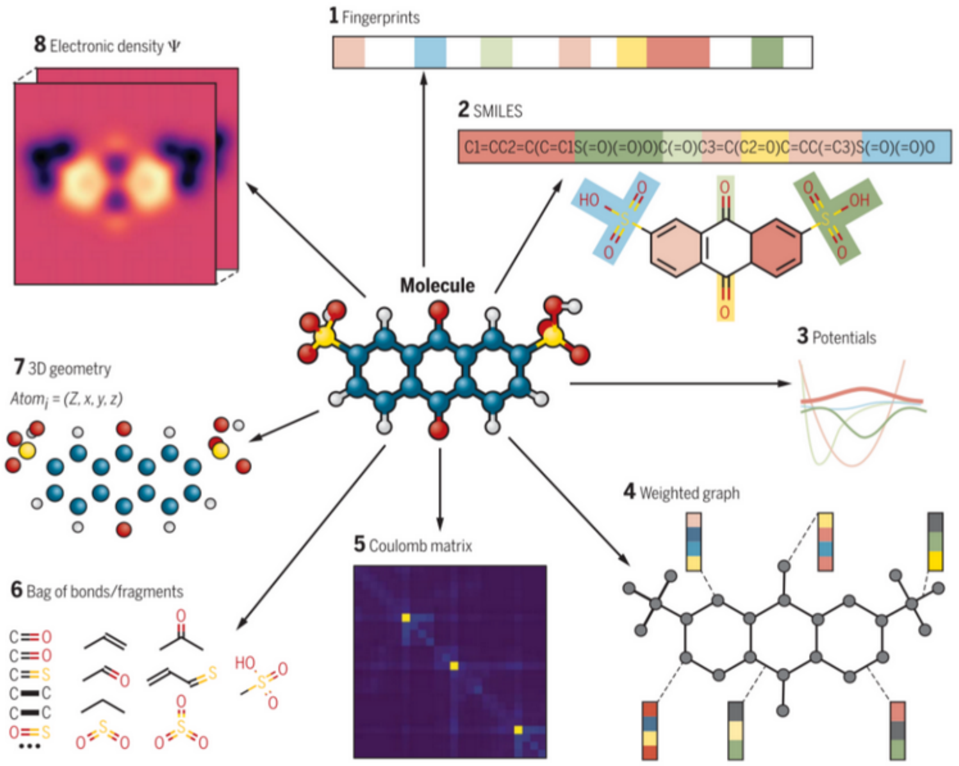
Graph neural networks and chemistry

- WARNING!

  Graphs are only one possible representation of molecules

  - Some ML researchers falsely think that the molecular graph is the "correct" representation WRONG!
  - Some ML researchers think that GNNs are the "correct" way to do learning tasks on molecules – WRONG!

- BUT: GNNs are very **versatile**...



Recap: Deep Learning and Multi-layer Perceptrons (MLPs)

JOHANNES KEPLER UNIVERSITY LINZ

Notation, Deep Learning and Empirical Risk Minimization

- We have access to a labeled dataset

$$\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, \boldsymbol{y}^{(N)})\}$$

- A model function with adjustable parameters $\boldsymbol{w}$

$$\hat{\boldsymbol{y}} = g(\boldsymbol{x}; \boldsymbol{w})$$

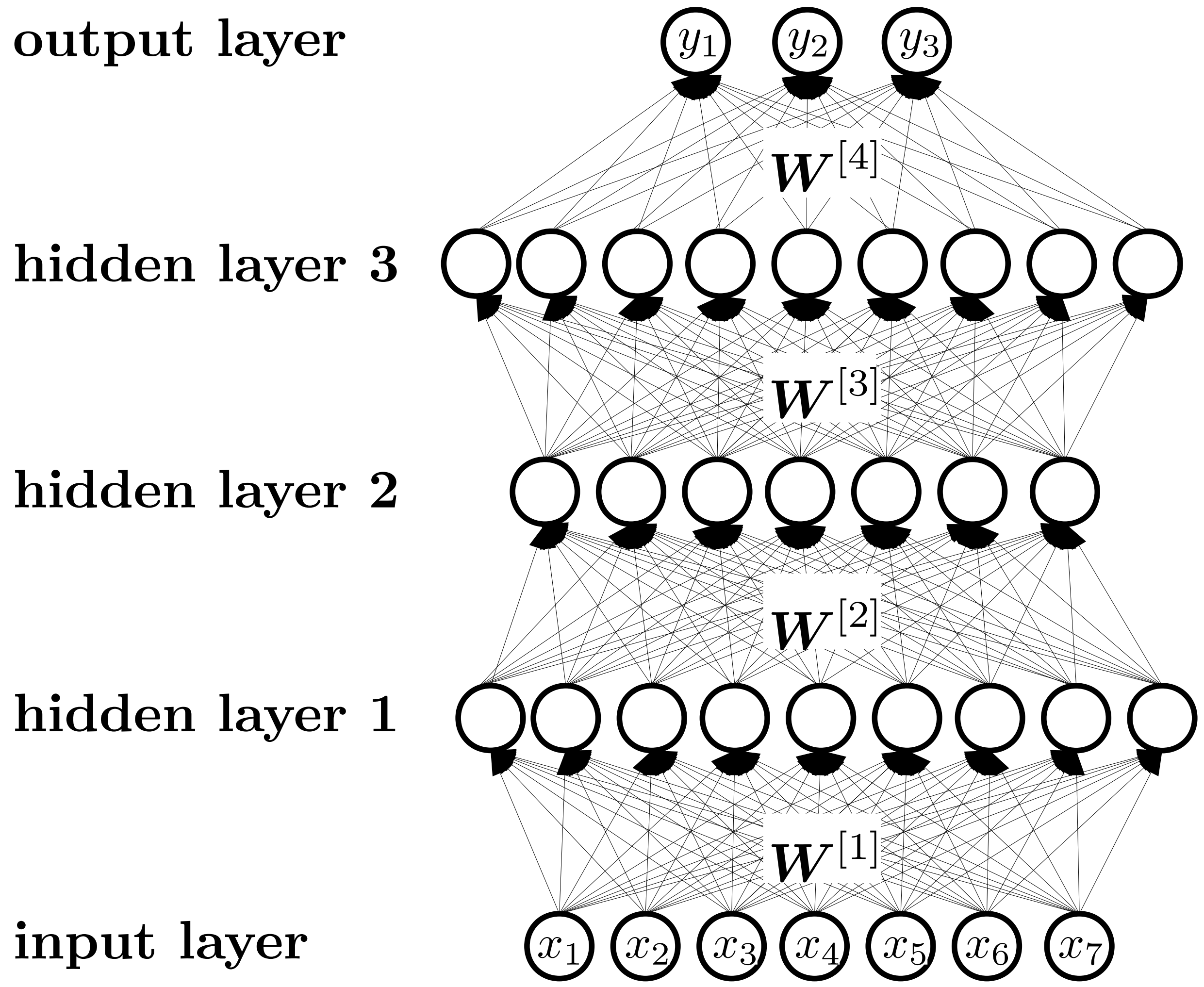
- We will adapt the parameters to minimize the empirical risk

$$\mathrm{R}_{\mathrm{emp}}(\boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} L(\boldsymbol{y}^{(n)}, g(\boldsymbol{x}^{(n)}; \boldsymbol{w}))$$

- Using gradient descent

$$\boldsymbol{w}^{\mathrm{new}} = \boldsymbol{w}^{\mathrm{old}} - \eta \nabla_{\boldsymbol{w}} \mathrm{R}_{\mathrm{emp}}(\boldsymbol{w})|_{\boldsymbol{w}_{\mathrm{old}}}$$

Structure of a deep neural network (DNN)



$$\hat{y} = \sigma(\underbrace{\boldsymbol{W}^{[4]}\ \boldsymbol{a}^{[3]}}_{=\boldsymbol{s}^{[4]}})$$

$$\boldsymbol{a}^{[3]} = f(\underbrace{\boldsymbol{W}^{[3]}\ \boldsymbol{a}^{[2]}}_{=\boldsymbol{s}^{[3]}})$$

$$\boldsymbol{a}^{[2]} = f(\underbrace{\boldsymbol{W}^{[2]}\ \boldsymbol{a}^{[1]}}_{=\boldsymbol{s}^{[2]}})$$

$$\boldsymbol{a}^{[1]} = f(\underbrace{\boldsymbol{W}^{[1]}\ \boldsymbol{x}}_{\boldsymbol{s}^{[1]}})$$

$$\boldsymbol{x}$$

DNNs: notation

- $\boldsymbol{x}$: input for layer 0; equivalent to $\boldsymbol{a}^{[0]}$
- $\boldsymbol{W}^{[l]}$: weight matrix connecting layer $l-1$ and layer l
- $\boldsymbol{s}^{[l]}$: pre-activations of layer l
- $\boldsymbol{a}^{[l]}$: activations of layer l
- f: activation function that is applied element-wise to a vector.

JOHANNES KEPLER UNIVERSITY LINZ

DNNs: Forward pass

- Activations from one layer to the next layer:

$$\boldsymbol{a}^{[l]} = f(\boldsymbol{s}^{[l]}) = f(\boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]})$$

- Full function:

$$\hat{\boldsymbol{y}} = g(\boldsymbol{x}; \boldsymbol{W}^{[1]}, \ldots, \boldsymbol{W}^{[L]}) = \sigma(\boldsymbol{W}^{[L]}(\ldots f(\boldsymbol{W}^{[2]} f(\boldsymbol{W}^{[1]}\boldsymbol{x}))\ldots)$$

- With bias units

$$\boldsymbol{a}^{[l]} = f(\boldsymbol{s}^{[l]}) = f(\boldsymbol{W}^{[l]}\boldsymbol{a}^{[l-1]} + \boldsymbol{b}^{[l]}).$$

JOHANNES KEPLER UNIVERSITY LINZ

Why non-linearities?

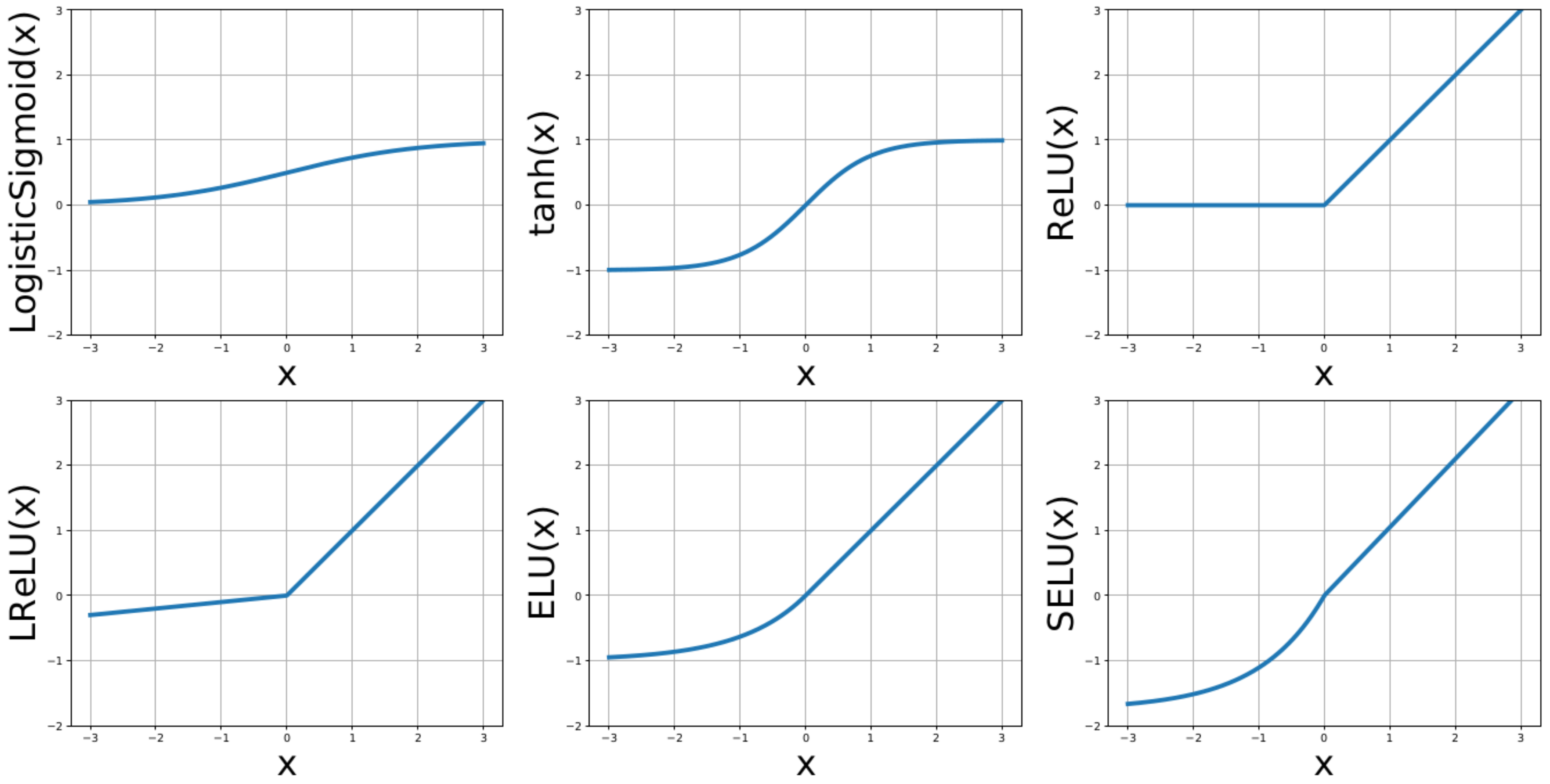
- With non-linearity:

$$g(\boldsymbol{x}; \boldsymbol{W}^{[1]}, \dots, \boldsymbol{W}^{[L]}) = \sigma(\boldsymbol{W}^{[L]}(\dots f(\boldsymbol{W}^{[2]} f(\boldsymbol{W}^{[1]}\boldsymbol{x}))\dots))$$

- Without non-linearity:

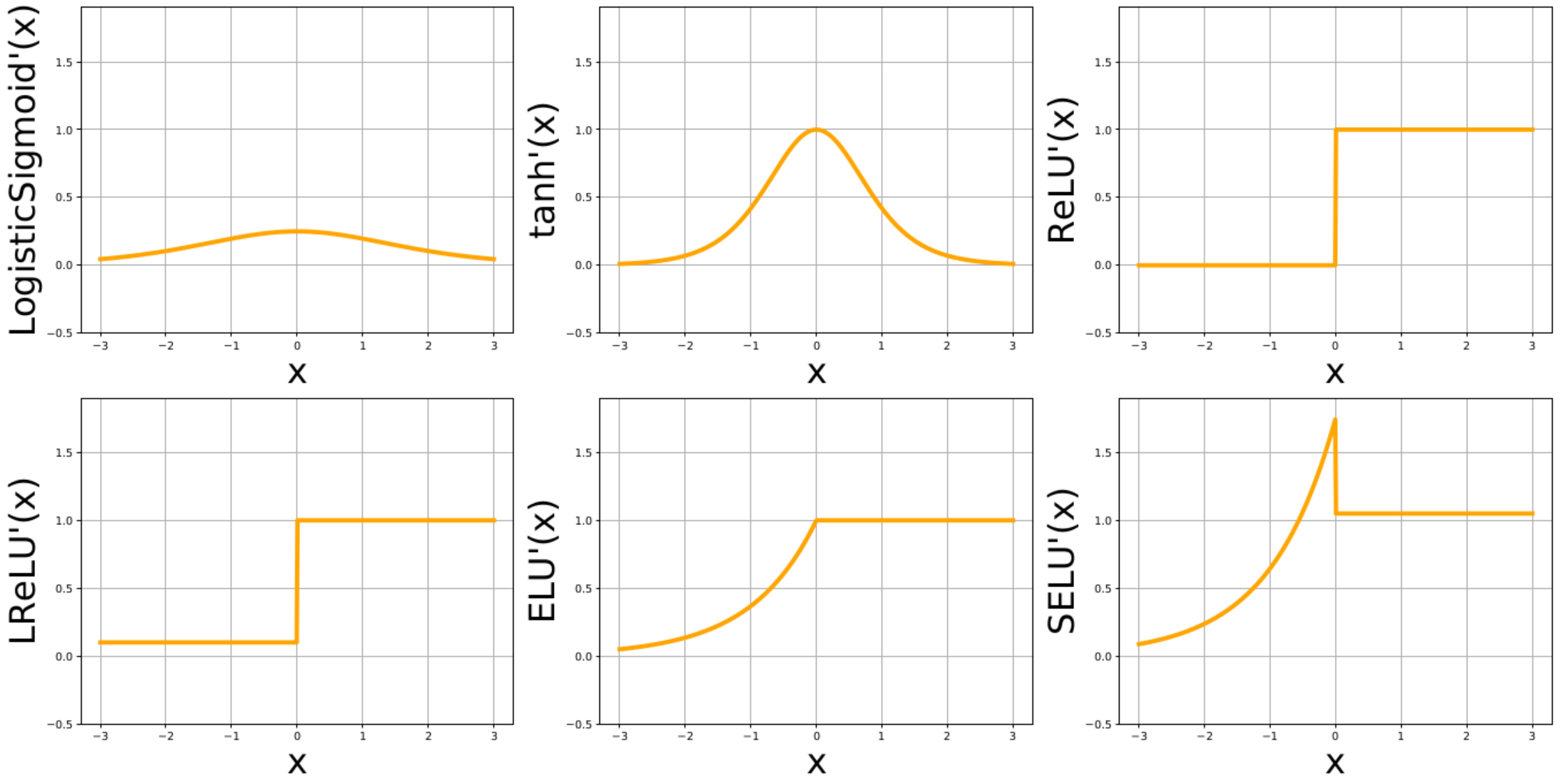
$$g(\boldsymbol{x}; \boldsymbol{W}^{[1]}, \dots, \boldsymbol{W}^{[L]}) = \sigma(\underbrace{\boldsymbol{W}^{[L]} \dots \boldsymbol{W}^{[1]}}_{=\hat{\boldsymbol{W}}}\boldsymbol{x})$$

A note on activation functions

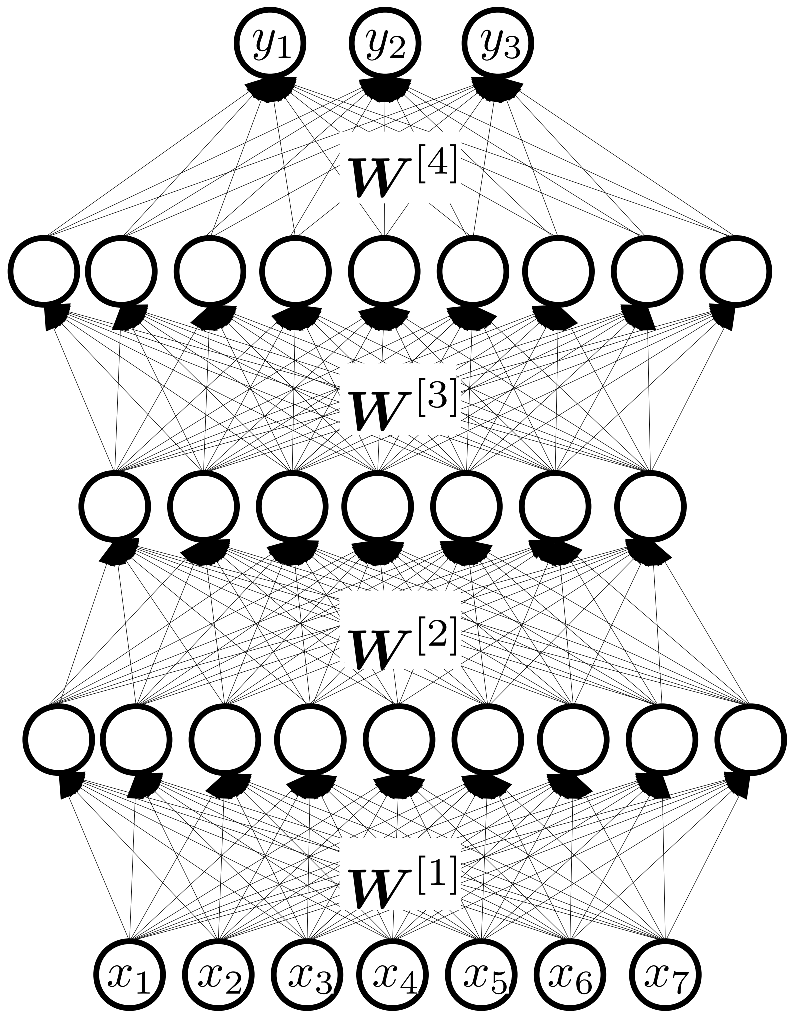


JOHANNES KEPLER
UNIVERSITY LINZ

A note on activation functions: derivatives



DNN: Forward pass



$$\hat{y} = \sigma(\underbrace{\boldsymbol{W}^{[4]}\ \boldsymbol{a}^{[3]}}_{=\boldsymbol{s}^{[4]}})$$

$$\boldsymbol{a}^{[3]} = f(\underbrace{\boldsymbol{W}^{[3]}\ \boldsymbol{a}^{[2]}}_{=\boldsymbol{s}^{[3]}})$$

$$\boldsymbol{a}^{[2]} = f(\underbrace{\boldsymbol{W}^{[2]}\ \boldsymbol{a}^{[1]}}_{=\boldsymbol{s}^{[2]}})$$

$$\boldsymbol{a}^{[1]} = f(\underbrace{\boldsymbol{W}^{[1]}\ \boldsymbol{x}}_{\boldsymbol{s}^{[1]}})$$

$$\boldsymbol{x}$$

JOHANNES KEPLER UNIVERSITY LINZ

Backpropagation in DNNs

- Without loss of generality, we derive with respect to a weight in the first layer:

$$\frac{\partial}{w_{ij}^{[1]}} L(\boldsymbol{y}, g(\boldsymbol{x}; \boldsymbol{W}^{[1]}, \ldots, \boldsymbol{W}^{[L]})) = \underbrace{\frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{s}^{[L]}}}_{=:A} \cdot \ldots \cdot \underbrace{\frac{\partial \boldsymbol{s}^{[l]}}{\partial \boldsymbol{s}^{[l-1]}}}_{=:B} \cdot \ldots \cdot \underbrace{\frac{\partial \boldsymbol{s}^{[1]}}{\partial w_{ij}^{[1]}}}_{=:C}$$

- Now we treat the expressions A, B and C.

Backpropagation in DNNs

- We first define:

$$\boldsymbol{\delta}^{[l]} := \frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{s}^{[l]}}$$

JOHANNES KEPLER UNIVERSITY LINZ

Backpropagation in DNNs

- Without loss of generality, we derive with respect to a weight in the first layer:

$$\frac{\partial}{w_{ij}^{[1]}} L(\boldsymbol{y}, g(\boldsymbol{x}; \boldsymbol{W}^{[1]}, \ldots, \boldsymbol{W}^{[L]})) = \underbrace{\frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{s}^{[L]}}}_{=:A} \cdot \ldots \cdot \underbrace{\frac{\partial \boldsymbol{s}^{[l]}}{\partial \boldsymbol{s}^{[l-1]}}}_{=:B} \cdot \ldots \cdot \underbrace{\frac{\partial \boldsymbol{s}^{[1]}}{\partial w_{ij}^{[1]}}}_{=:C}$$

- Now we treat the expressions A, B and C.

Backpropagation in DNNs

- Without loss of generality, we derive with respect to a weight in the first layer:

$$\frac{\partial}{w_{ij}^{[1]}} L(\boldsymbol{y}, g(\boldsymbol{x}; \boldsymbol{W}^{[1]}, \ldots, \boldsymbol{W}^{[L]})) = \underbrace{\frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{s}^{[L]}}}_{=:A} \cdot \ldots \cdot \underbrace{\frac{\partial \boldsymbol{s}^{[l]}}{\partial \boldsymbol{s}^{[l-1]}}}_{=:B} \cdot \ldots \cdot \underbrace{\frac{\partial \boldsymbol{s}^{[1]}}{\partial w_{ij}^{[1]}}}_{=:C}$$

- A $\frac{\partial L(\boldsymbol{y},\hat{\boldsymbol{y}})}{\partial \hat{\boldsymbol{y}}} \frac{\hat{\boldsymbol{y}}}{\partial \boldsymbol{s}^{[L]}}$: Depends on the choice of loss function and activation. If canonical links are chosen, this can result in

$$\boldsymbol{\delta}^{[L]} = (\hat{\boldsymbol{y}} - \boldsymbol{y})^T$$

JOHANNES KEPLER UNIVERSITY LINZ

Backpropagation in DNNs

- Without loss of generality, we derive with respect to a weight in the first layer:

$$\frac{\partial}{w_{ij}^{[1]}} L(\boldsymbol{y}, g(\boldsymbol{x}; \boldsymbol{W}^{[1]}, \ldots, \boldsymbol{W}^{[L]})) = \underbrace{\frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{s}^{[L]}}}_{=:A} \cdot \ldots \cdot \underbrace{\frac{\partial \boldsymbol{s}^{[l]}}{\partial \boldsymbol{s}^{[l-1]}}}_{=:B} \cdot \ldots \cdot \underbrace{\frac{\partial \boldsymbol{s}^{[1]}}{\partial w_{ij}^{[1]}}}_{=:C}$$

- B $\frac{\partial \boldsymbol{s}^{[l]}}{\partial \boldsymbol{s}^{[l-1]}}$:

$$\boldsymbol{s}^{[l]} = \boldsymbol{W}^{[l]} f(\boldsymbol{s}^{[l-1]})$$

$$\frac{\partial \boldsymbol{s}^{[l]}}{\partial \boldsymbol{s}^{[l-1]}} = \boldsymbol{W}^{[l]} \operatorname{diag}\left(f'(\boldsymbol{s}^{[l-1]})\right)$$

JOHANNES KEPLER UNIVERSITY LINZ

Backpropagation in DNNs

- Without loss of generality, we derive with respect to a weight in the first layer:

$$\frac{\partial}{w_{ij}^{[1]}} L(\boldsymbol{y}, g(\boldsymbol{x}; \boldsymbol{W}^{[1]}, \dots, \boldsymbol{W}^{[L]})) = \underbrace{\frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{s}^{[L]}}}_{=:A} \cdot \dots \cdot \underbrace{\frac{\partial \boldsymbol{s}^{[l]}}{\partial \boldsymbol{s}^{[l-1]}}}_{=:B} \cdot \dots \cdot \underbrace{\frac{\partial \boldsymbol{s}^{[1]}}{\partial w_{ij}^{[1]}}}_{=:C}$$

- *Recursive formula* for deltas:

$$\boldsymbol{\delta}^{[l-1]} = \frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{s}^{[l]}} \frac{\partial \boldsymbol{s}^{[l]}}{\partial \boldsymbol{s}^{[l-1]}} = \boldsymbol{\delta}^{[l]} \underbrace{\boldsymbol{W}^{[l]} \operatorname{diag}\left(f'(\boldsymbol{s}^{[l-1]}) \right)}_{\boldsymbol{J}^{[l]}}$$

JOHANNES KEPLER UNIVERSITY LINZ

Backpropagation in DNNs

- Without loss of generality, we derive with respect to a weight in the first layer:

$$\frac{\partial}{w_{ij}^{[1]}} L(\boldsymbol{y}, g(\boldsymbol{x}; \boldsymbol{W}^{[1]}, \ldots, \boldsymbol{W}^{[L]})) = \underbrace{\frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{s}^{[L]}}}_{=:A} \cdot \ldots \cdot \underbrace{\frac{\partial \boldsymbol{s}^{[l]}}{\partial \boldsymbol{s}^{[l-1]}}}_{=:B} \cdot \ldots \cdot \underbrace{\frac{\partial \boldsymbol{s}^{[1]}}{\partial w_{ij}^{[1]}}}_{=:C}$$

- C: $\frac{\partial \boldsymbol{s}^{[1]}}{\partial w_{ij}^{[1]}} = \frac{\partial}{\partial w_{ij}^{[1]}} \boldsymbol{W}^{[1]} \boldsymbol{a}^{[0]} = \begin{pmatrix} 0 \\ \vdots \\ a_j^{[0]} \\ \vdots \\ 0 \end{pmatrix},$

  where the non-zero entry $a_j^{[0]}$ is at the i-th position.

Backpropagation in DNNs

- Overall we find

$$\frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial w_{ij}^{[l]}} = \frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{s}^{[l]}} \frac{\partial \boldsymbol{s}^{[l]}}{\partial w_{ij}^{[l]}} = (\boldsymbol{\delta}^{[l]})_i (\boldsymbol{a}^{[l-1]})_j$$

or conveniently:

$$\frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{W}^{[l]}} = \boldsymbol{\delta}^{[l]} \boldsymbol{a}^{[l-1]^T}.$$

Summary of backprop

- Calculate deltas at output units $\boldsymbol{\delta}^{[L]}$
- Backpropagate deltas through network using

$$\boldsymbol{\delta}^{[l-1]} = \frac{\partial L(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial \boldsymbol{s}^{[l]}} \frac{\partial \boldsymbol{s}^{[l]}}{\partial \boldsymbol{s}^{[l-1]}} = \boldsymbol{\delta}^{[l]} \underbrace{\boldsymbol{W}^{[l]} \operatorname{diag}\left(f'(\boldsymbol{s}^{[l-1]})\right)}_{\boldsymbol{J}^{[l]}}$$

- Calculate weight update

$$\Delta \boldsymbol{W}^{[l]} = -\eta \boldsymbol{\delta}^{[l]} (\boldsymbol{a}^{[l-1]})^T$$

The vanishing gradient problem

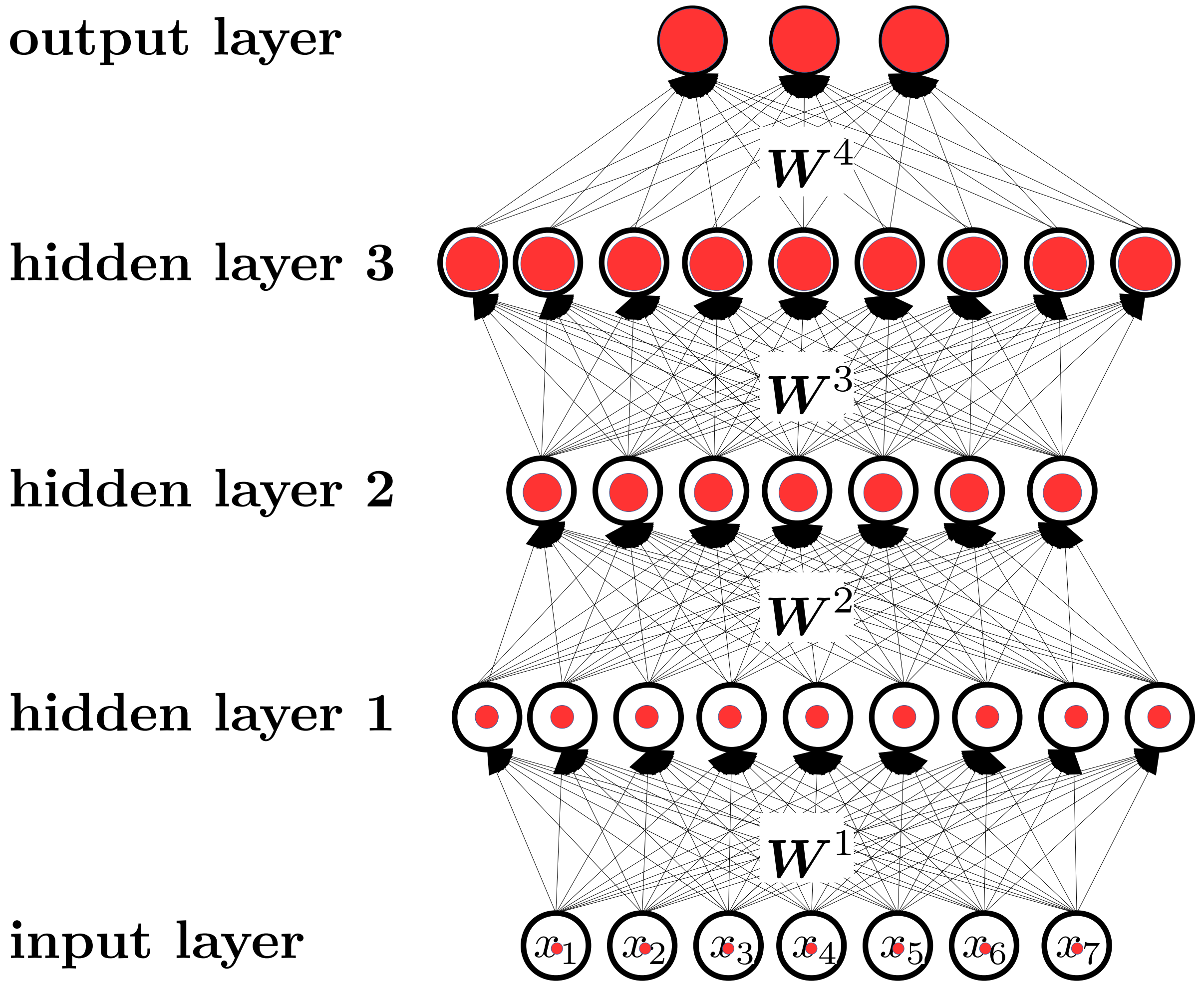
- We know from above:

$$\boldsymbol{\delta}^{[l-1]} = \boldsymbol{\delta}^{[l]} \underbrace{\boldsymbol{W}^{[l]} \operatorname{diag}\left(f'(\boldsymbol{s}^{[l-1]})\right)}_{\boldsymbol{J}^{[l]}} = \boldsymbol{\delta}^{[l]} \boldsymbol{J}^{[l]}$$

- Thus we have: $\|\boldsymbol{J}^{[l]}\| \leq k \leq 1$

$$\|\boldsymbol{\delta}^{[1]}\| = \|\boldsymbol{\delta}^{[L]} \prod_{l=L}^{2} \boldsymbol{J}^{[l]}\| \leq k^{L-1} \|\boldsymbol{\delta}^{[L]}\| \quad \Rightarrow \|\boldsymbol{\delta}^{[1]}\| \approx 0$$

The vanishing gradient problem



$$\boldsymbol{\delta}^{[L]}$$

$$\boldsymbol{\delta}^{[L-1]} = \boldsymbol{\delta}^{[L]} J^{[L]}$$

$$\boldsymbol{\delta}^{[L-2]} = \boldsymbol{\delta}^{[L-1]} J^{[L-1]}$$

$$\boldsymbol{\delta}^{[L-3]} = \boldsymbol{\delta}^{[L-2]} J^{[L-2]}$$

$$\vdots$$

$$\boldsymbol{\delta}^{[0]} = \boldsymbol{\delta}^{[1]} J^{[1]}$$

Online, stochastic and batch training

- Online learning/training: single sample from training set is propagated through network and then a parameter update is performed.

$$\boldsymbol{w}^{\mathrm{new}} = \boldsymbol{w}^{\mathrm{old}} - \eta \nabla_{\boldsymbol{w}} R_{\mathrm{emp}}(y, \boldsymbol{x}, \boldsymbol{w})\big|_{\boldsymbol{w}^{\mathrm{old}}}$$

- Full-batch learning/training: all samples from training set are propagated through network and then a parameter update is performed.

$$\boldsymbol{w}^{\mathrm{new}} = \boldsymbol{w}^{\mathrm{old}} - \eta \nabla_{\boldsymbol{w}} R_{\mathrm{emp}}(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{w})\big|_{\boldsymbol{w}^{\mathrm{old}}}$$

- Stochastic learning/training: a small subset of samples from the training set are propagetd through network and then a parameter update is performed

$$\boldsymbol{w}^{\mathrm{new}} = \boldsymbol{w}^{\mathrm{old}} - \eta \tilde{\nabla}_{\boldsymbol{w}} R_{\mathrm{emp}}(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{w})\big|_{\boldsymbol{w}^{\mathrm{old}}}$$

Approx. the full-batch gradient with a random subsample (“mini batch”):

$$\nabla_{\boldsymbol{w}} R_{\mathrm{emp}}(\boldsymbol{y}, \boldsymbol{X}, \boldsymbol{w}) = \frac{1}{N} \sum_{n=1}^{N} \nabla_{\boldsymbol{w}} L(y^n, g(\boldsymbol{x}^n, \boldsymbol{w})) \approx \frac{1}{B} \sum_{b=1}^{B} \nabla_{\boldsymbol{w}} L(y^{n_b}, g(\boldsymbol{x}^{n_b}, \boldsymbol{w}))$$

JOHANNES KEPLER UNIVERSITY LINZ

Deep feed-forward neural networks

- Same principle as MLPs
  - Layers of interconnected neurons
  - More layers; more neurons
  - Different activation functions

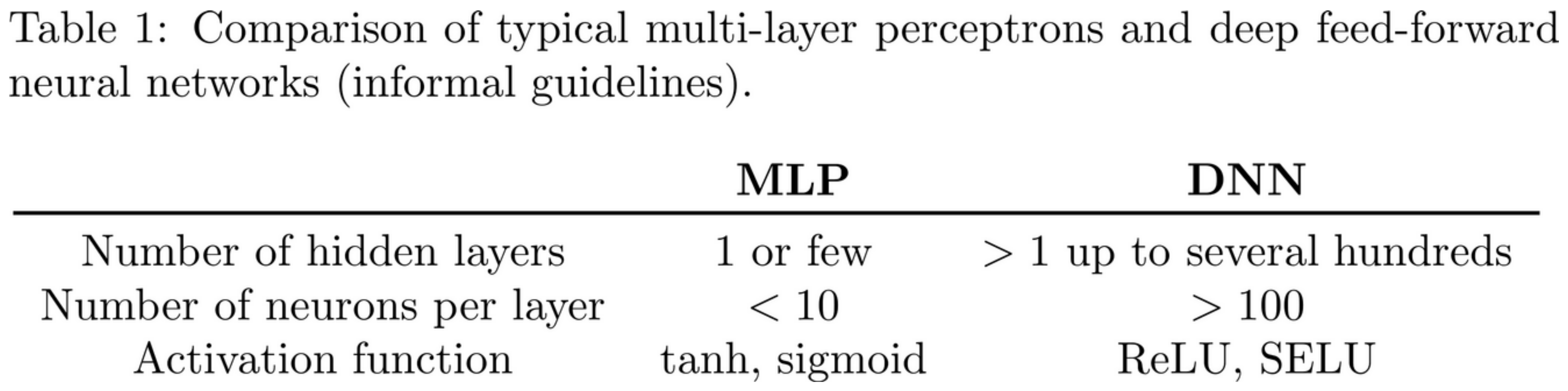


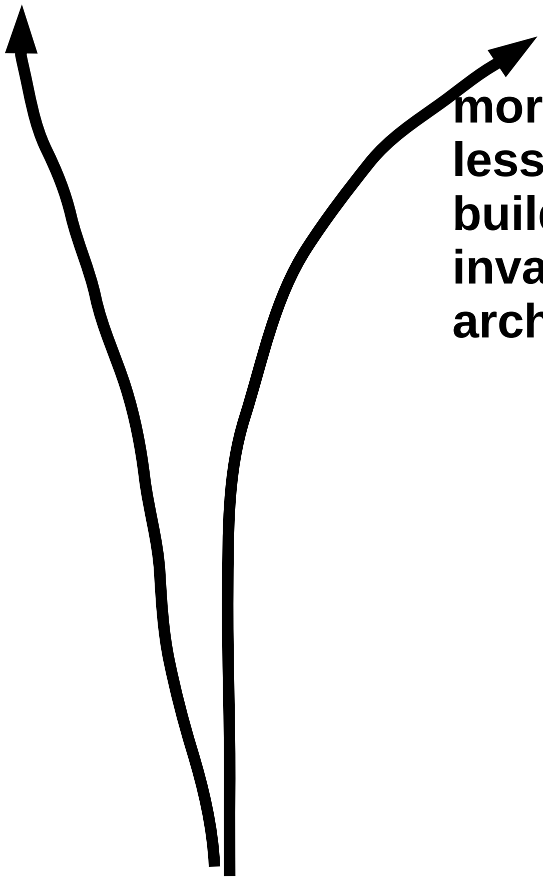
Table 1: Comparison of typical multi-layer perceptrons and deep feed-forward neural networks (informal guidelines).

	MLP	**DNN**
Number of hidden layers	1 or few	>1 up to several hundreds
Number of neurons per layer	<10	>100
Activation function	tanh, sigmoid	ReLU, SELU

JOHANNES KEPLER UNIVERSITY LINZ

Geometric Deep Learning and GNNs

Currently two streams in ML



GEOMETRIC DL

less inductive biases,
more data,
learn equivariances and
invariances from data

Vision
transformer,
MLPMixer,
ConvNext,
CLIP, ...

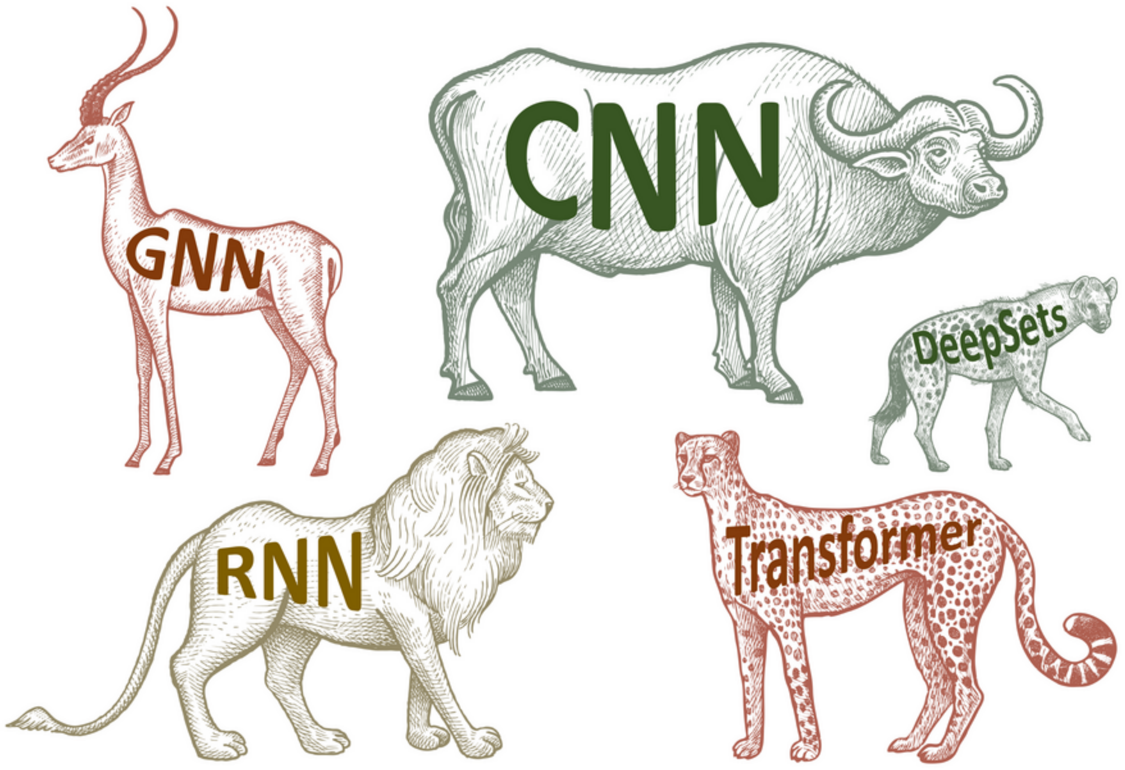
more inductive biases,
less data,
build equivariances and
invariances into
architectures

Graph neural
networks,
Message-passing
networks,
spherical CNN

JOHANNES KEPLER
UNIVERSITY LINZ

Geometric Deep Learning

- Attempt at a unified view on Deep Learning architectures
  - From the perspective of symmetry properties
- Comparison to different geometries in mathematics



Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric deep learning: going beyond euclidean data. IEEE Signal Processing Magazine, 34(4), 18-42.

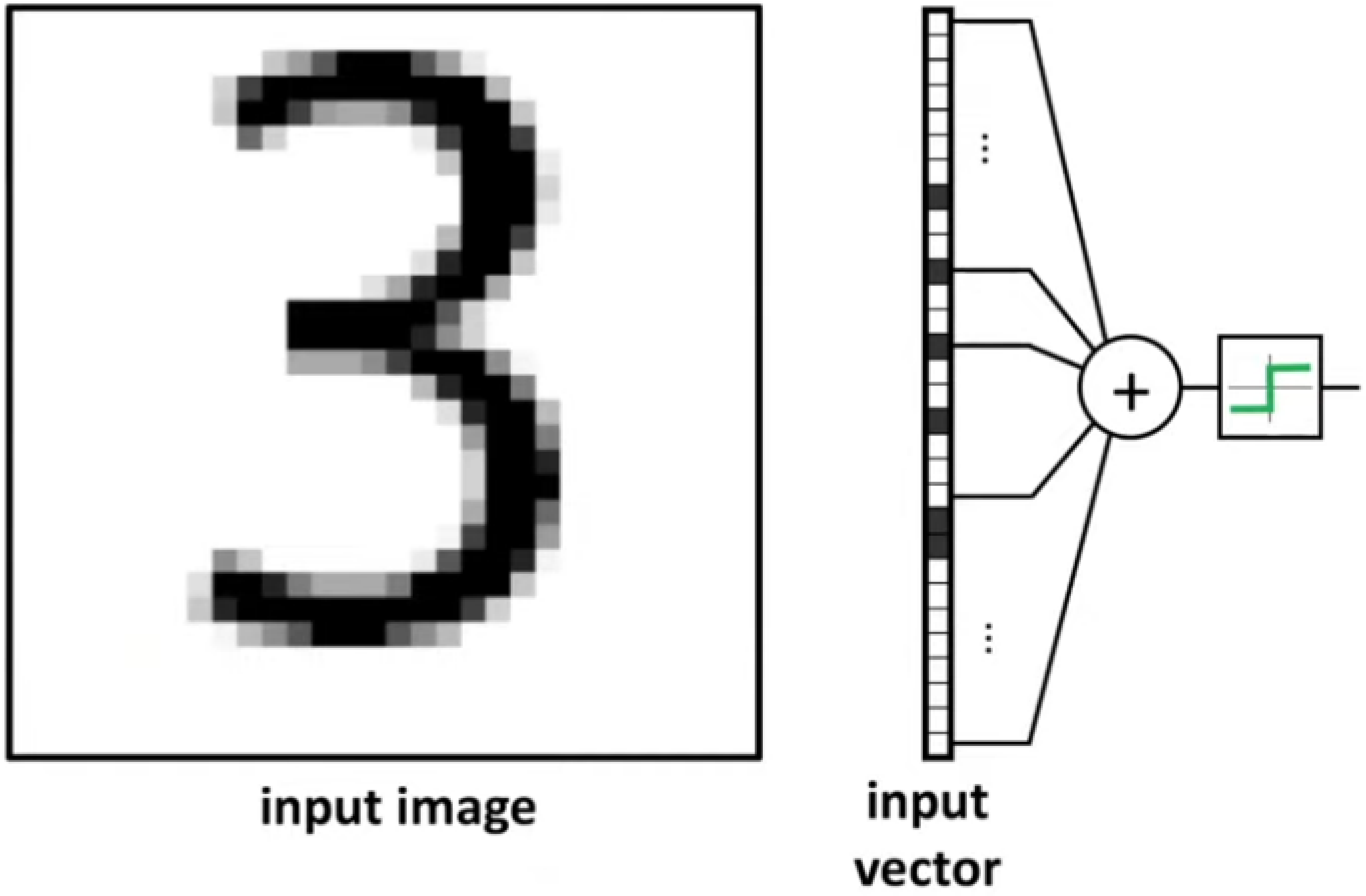
Inductive biases

- *“Biases and initial knowledge are at the heart of the ability to generalize beyond observed data*” – Tom Mitchell
- “*No generalization without inductive bias*” – Max Welling, Amsterdam, April 20 2019

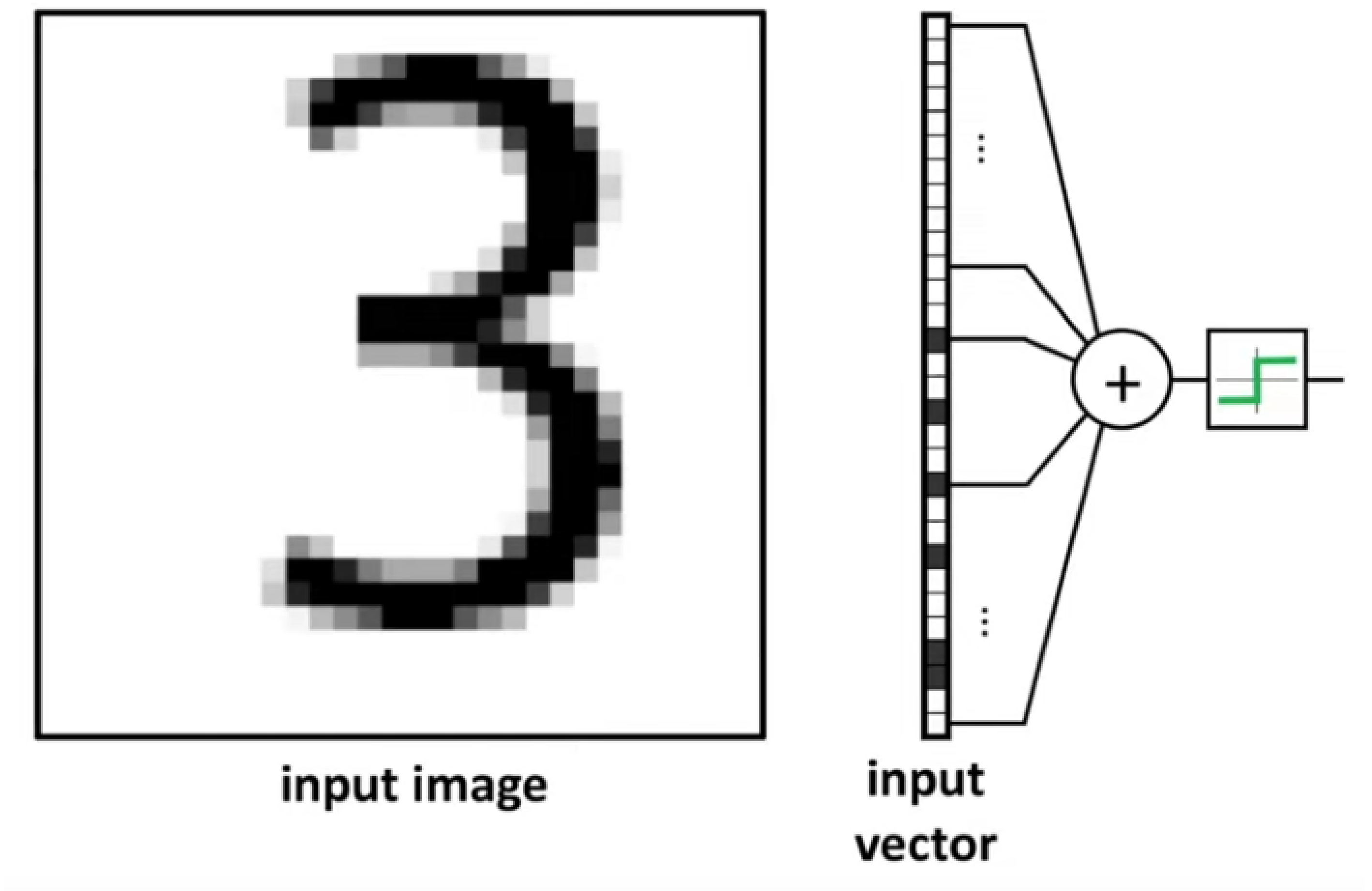
- Equip neural networks with biases that lead to models that learn to generalize well
- What could be such properties?

Mitchell, T. M. (1980). The need for biases in learning generalizations (pp. 184-191). New Jersey: Department of Computer Science, Laboratory for Computer Science Research, Rutgers Univ..

Symmetries in computer vision tasks



Symmetries in computer vision tasks



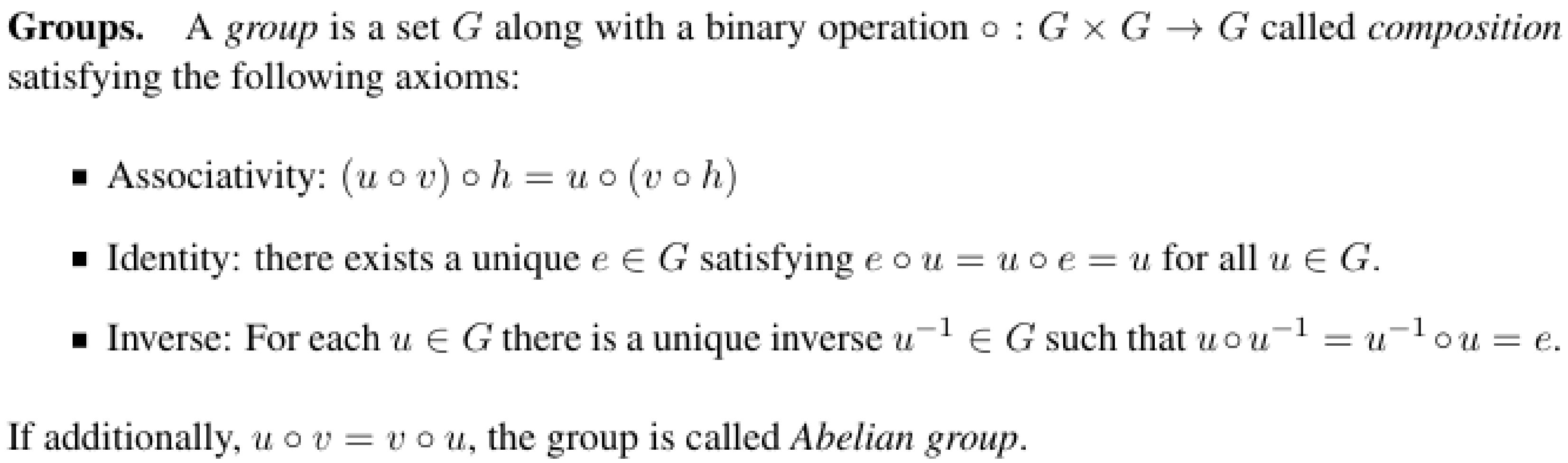
- Must learn invariance from data

Geometric priors, symmetries, and invariances

- Intuitive: a symmetry of an object or system is a transformation that leaves a certain property of said object or system unchanged or *invariant*.
- Geometric prior: knowledge about the structure of the object
- Transformations may be smooth, continuous, or discrete
- Everywhere in ML tasks
- Set of symmetries satisfies a number of properties
  - Combination of symmetries
  - Invertible
  - In mathematics known as *group*

Symmetries and groups

- Let u and v be two symmetries



Groups. A *group* is a set G along with a binary operation $\circ : G \times G \to G$ called *composition* satisfying the following axioms:

- Associativity: $(u \circ v) \circ h = u \circ (v \circ h)$
- Identity: there exists a unique $e \in G$ satisfying $e \circ u = u \circ e = u$ for all $u \in G$.
- Inverse: For each $u \in G$ there is a unique inverse $u^{-1} \in G$ such that $u \circ u^{-1} = u^{-1} \circ u = e$.

If additionally, $u \circ v = v \circ u$, the group is called *Abelian group*.

- Example (computer vision): shifting an object on an image

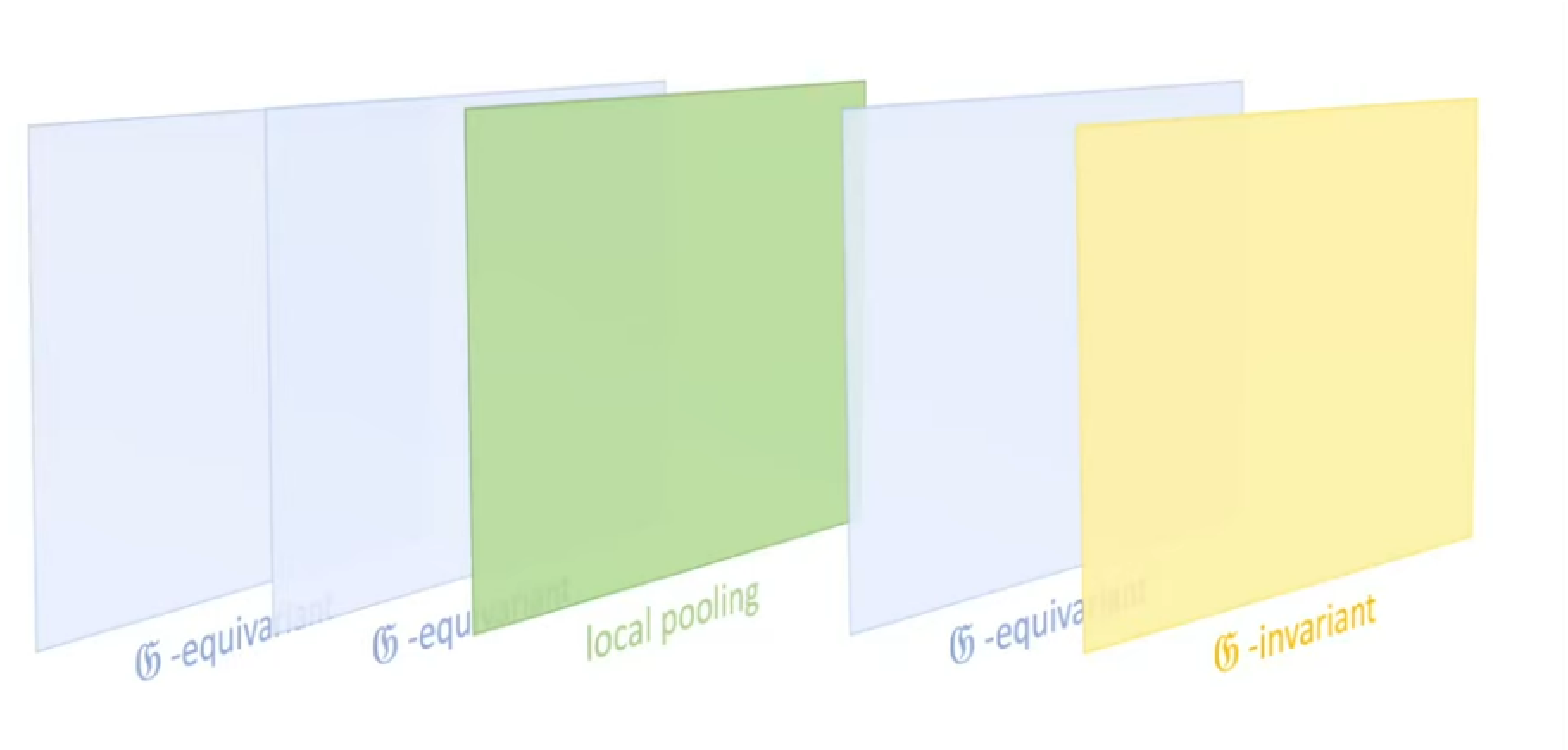
Invariance, invariant functions

Definition 2 *A function* $g : \mathcal{X} \to \mathcal{Y}$ *is* invariant with respect to G *if* $g(u(\boldsymbol{x})) = g(\boldsymbol{x})$, *where* $u \in G$ *is an operation of the group* G*. In other words, the output is unaffected by the group action on the input.*

- Invariances built in some Deep Learning operations
  - We had already encountered some
    - Max-pooling
    - Mean-pooling
    - Sum-pooling
  - Could be extended to other invariances
    - e.g. invariances to shifting objects

JOHANNES KEPLER UNIVERSITY LINZ

Geometric DL blueprint



- Not only CNNs fall under this blueprint!

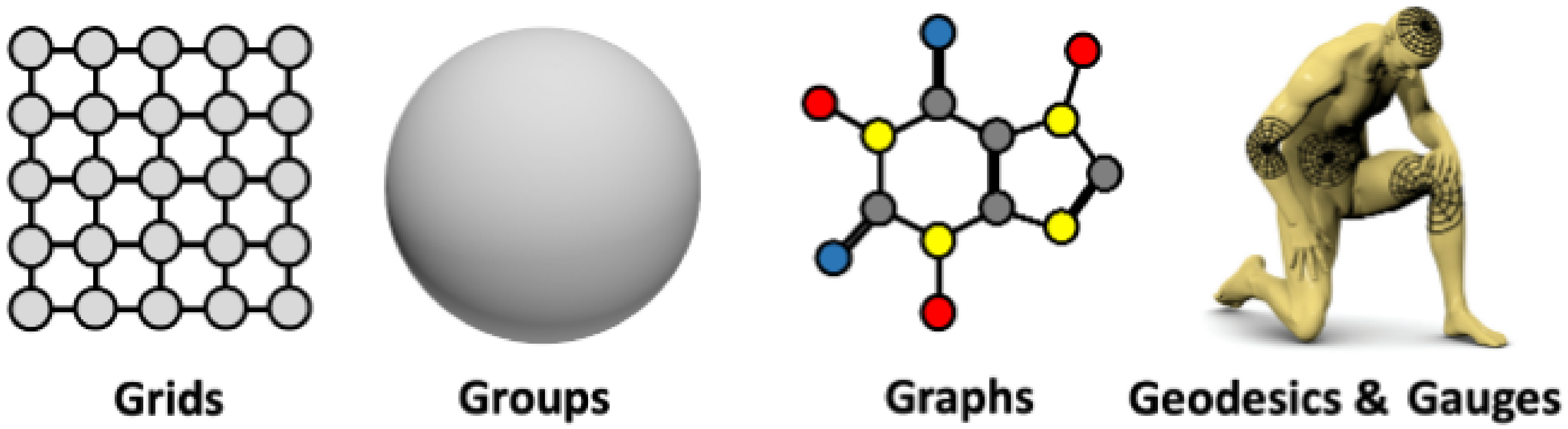
JOHANNES KEPLER UNIVERSITY LINZ

Equivariance, equivariant function

Definition 3 *A function* $g : \mathcal{X} \rightarrow \mathcal{Y}$ *is* equivariant with respect to G *if* $g(u(\boldsymbol{x})) = u(g(\boldsymbol{x}))$*, where* $u \in G$ *is an operation of the group* G*. In other words, the operation affects the input and output in the same way.*

- Convolutional layers are shift-equivariant:
  - If object is shifted, the resulting feature map activations are also shifted

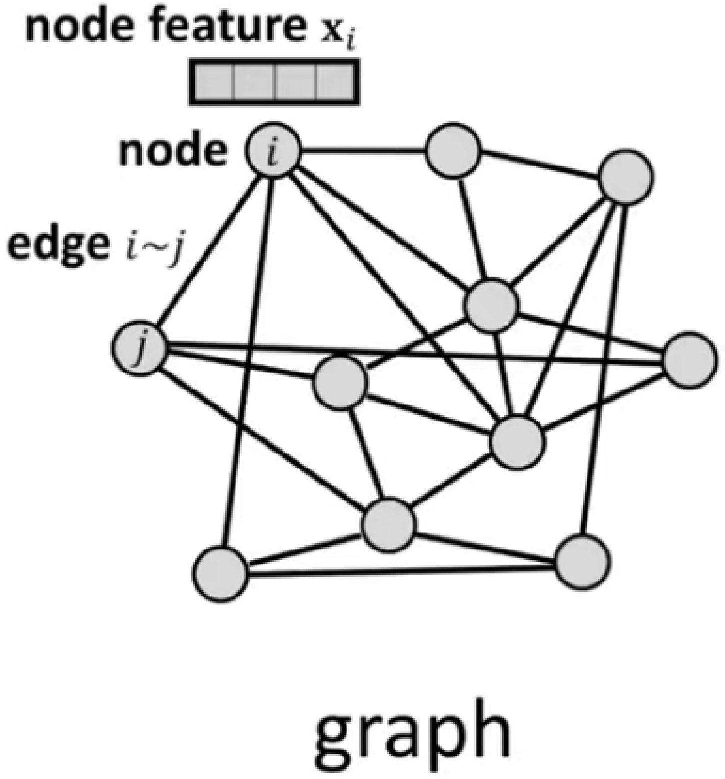
The 4G or 5G of Geometric DL



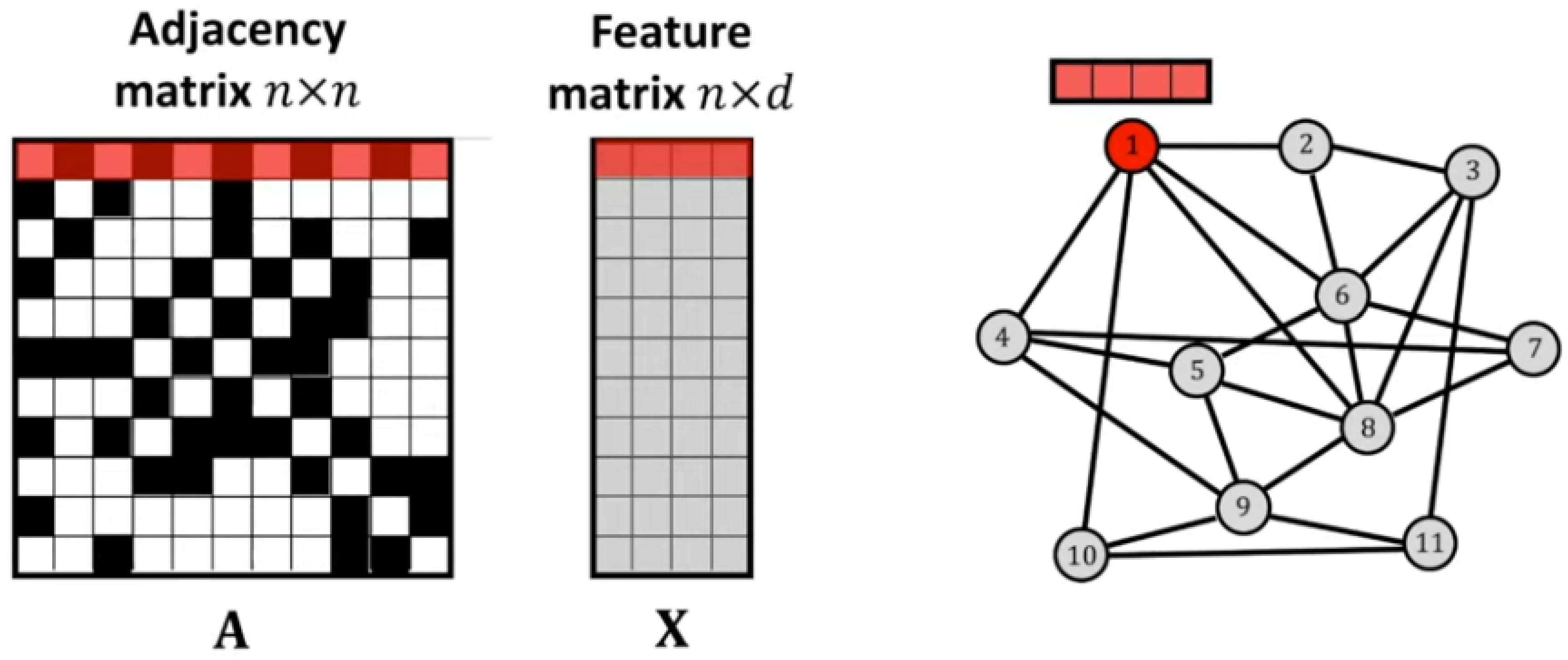
- Structure of domain
- Particular invariances and equivariances for each domain

Graph neural networks

- Main area of Geometric Deep Learning graphs
  - Graphs are very general;
    - images could be understood as graphs
      - pixels are nodes, adjacent pixels have a connection (edge)
    - Time-series could be understood as graphs
      - Directed graph, time-point is node, connected to next time-point
    - Sets could be understood as graphs
      - Element of set is a node, all elements of set are connected
- No canonical order of nodes



Graphs: ML description



Note: Notation for graphs

Note that there are different notations for graphs. Often the mathematical notation in terms of sets of nodes and edges is used (see Subsection 16.3.2). In machine learning, another notation is frequently used: a node i has the initial representation $\boldsymbol{x}$ and all I nodes together form $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}^I\}$. The edges are written in the adjacency matrix $\boldsymbol{A}$ and so a single graph is a pair $(\boldsymbol{X}, \boldsymbol{A})$. A training set of graphs can be denoted as $\{(\boldsymbol{X}^{(1)}, \boldsymbol{A}^{(1)}), \ldots, (\boldsymbol{X}^{(n)}, \boldsymbol{A}^{(n)}), \ldots, (\boldsymbol{X}^{(N)}, \boldsymbol{A}^{(N)})\}$.

JOHANNES KEPLER UNIVERSITY LINZ

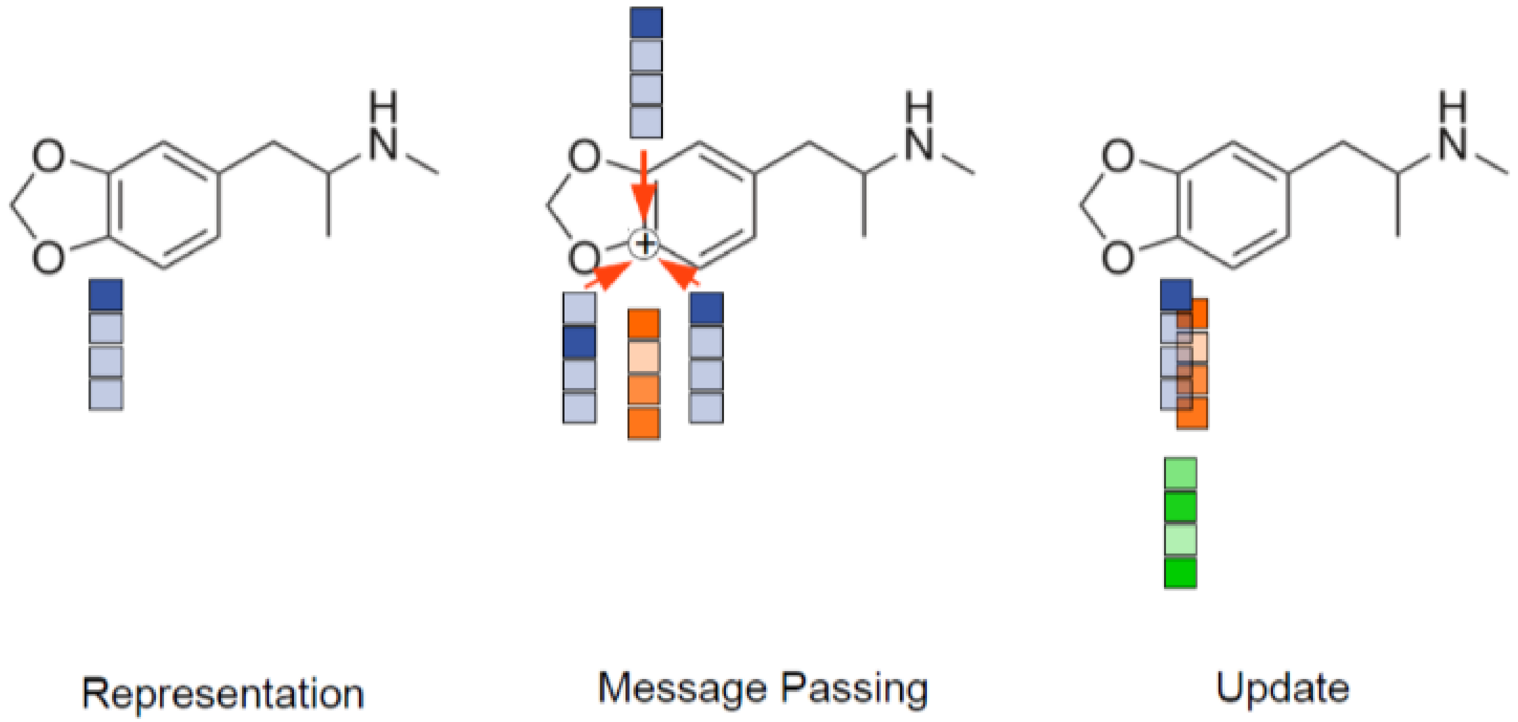
Graphs: Math description

A graph G consist of a set of *nodes* (vertices) $\mathcal{V}$ and set of edges $\mathcal{E} \subseteq \{\{i, j\} \mid (i, j) \in \mathcal{V}^2 \wedge i \neq j\}$. Both nodes $i \in \mathcal{V}$ and edges $\{i, j\} \in E$ can have labels (labelled graphs). We will use $\boldsymbol{x}_i$ to denote a feature vector, or representation, of the node i, and $\boldsymbol{e}_{ij}$ to denote the feature vector, or representation, of the edge $\{i, j\}$. The degree of a vertex of a graph $\deg(i)$ is the number of edges that are incident to the vertex. For a simple graph with vertex set $\mathcal{V}$, the *adjacency matrix* is a square $|V| \times |V|$ matrix A such that its element $A_{i,j}$ is one when there is an edge from vertex i to vertex j, and zero when there is no edge. In the graph neural network methods, both nodes and edges can have hidden represenations $\boldsymbol{h}_i$ and $\boldsymbol{h}_{ij}$, respectively.

Machine learning on graphs

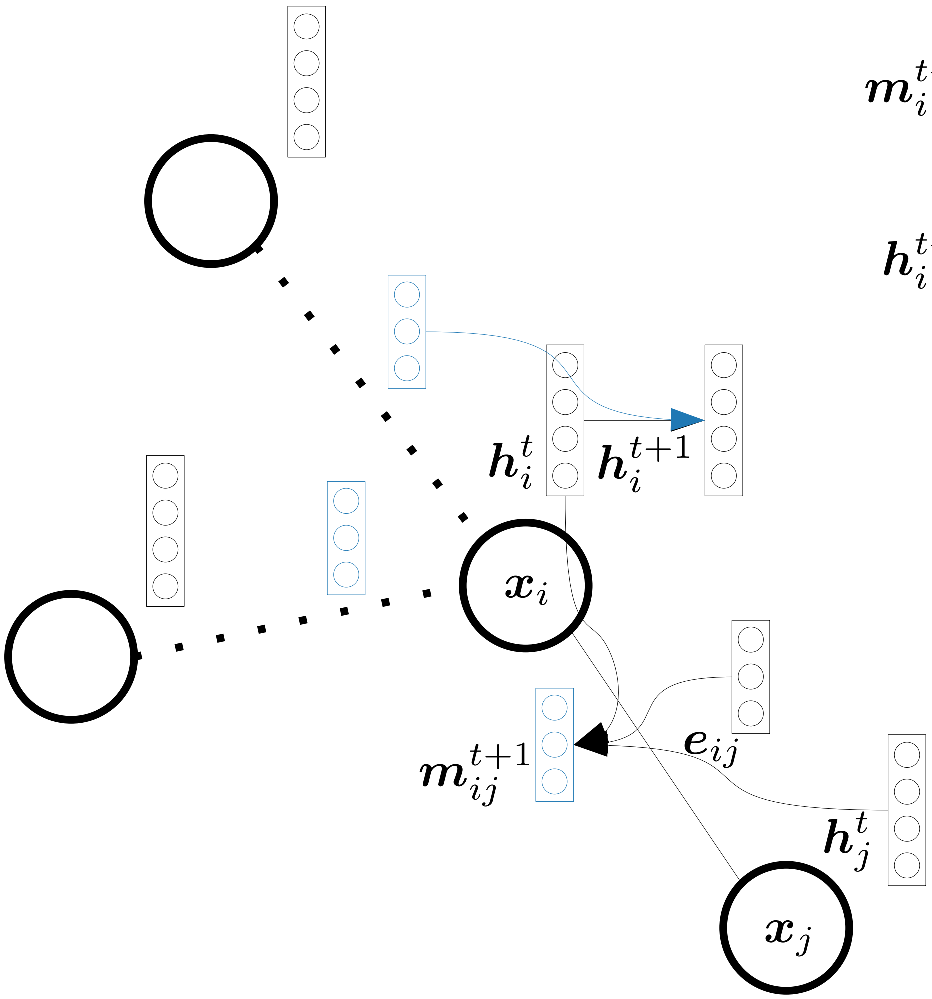
- Input objects for a machine learning method
  - E.g. social networks, molecules
  - Node-level classification: label per node has to be predicted
    - Example: social network; predict where a person likes to go on holidays
  - Graph-level classification: label per graph has to be predicted
    - Example: drug discovery; predict whether a molecule is toxic
- Desired invariances/equivariances
  - Representation of graph nodes equivariant
  - Prediction on whole graph: invariant to permutation of nodes

Machine learning on graphs: Message-passing framework



JOHANNES KEPLER UNIVERSITY LINZ

Machine learning on graphs: Message-passing framework



$$m_i^{t+1} = \sum_{j \in N(j)} \underbrace{\phi(h_i^t, h_j^t, e_{ij})}_{m_{ij}^{t+1}}$$

$$h_i^{t+1} = \psi(h_i^t, m_i^{t+1})$$

JOHANNES KEPLER UNIVERSITY LINZ

Message-passing neural networks

- Two main steps and two multi-layer perceptrons (MLP):

$$\boldsymbol{m}_i^{t+1} = \sum_{j \in N(j)} \underbrace{\phi(\boldsymbol{h}_i^t, \boldsymbol{h}_j^t, \boldsymbol{e}_{ij})}_{\boldsymbol{m}_{ij}^{t+1}}$$

$$\boldsymbol{h}_i^{t+1} = \psi(\boldsymbol{h}_i^t, \boldsymbol{m}_i^{t+1})$$

message generation function

$\phi(.,.,.)$ MLP

update function

$\psi(.,.)$ MLP

- i, j: Indices of nodes in a graph
- $\boldsymbol{h}_i^t$: representation of node i after t message passing phases
- $\boldsymbol{h}_i^0 = \boldsymbol{x}_i$: initial representation are the node features (e.g. atom typ

JOHANNES KEPLER UNIVERSITY LINZ

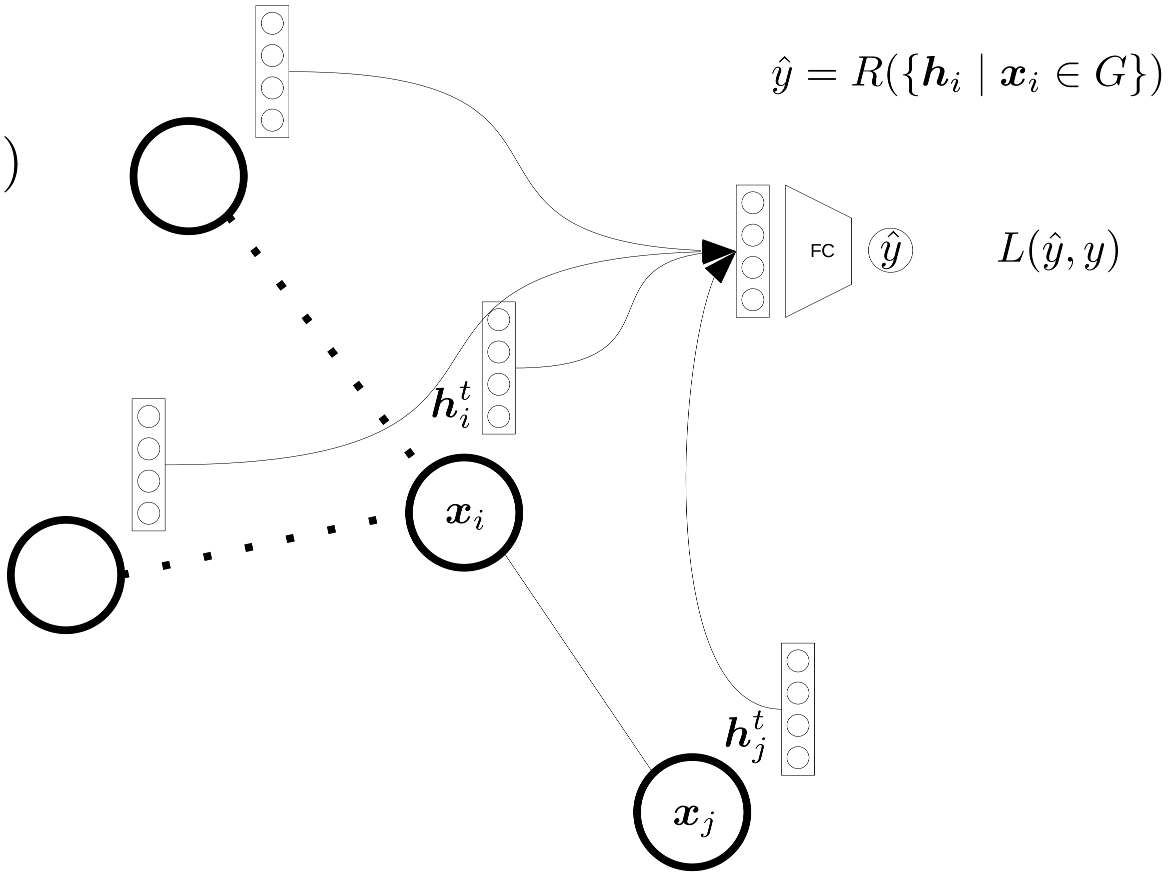
Read-out phase for graph-level classification

- Readout function

$$\hat{y} = R(\{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_N\})$$

- Permutation invariant
- Example:

$$\hat{\boldsymbol{y}} = \boldsymbol{w}^T \left(\frac{1}{|G|} \sum_{\nu \in G} \boldsymbol{h}_\nu \right)$$



E(n)-equivariant neural nets (EGNN) Satorras et al.

Font type!!

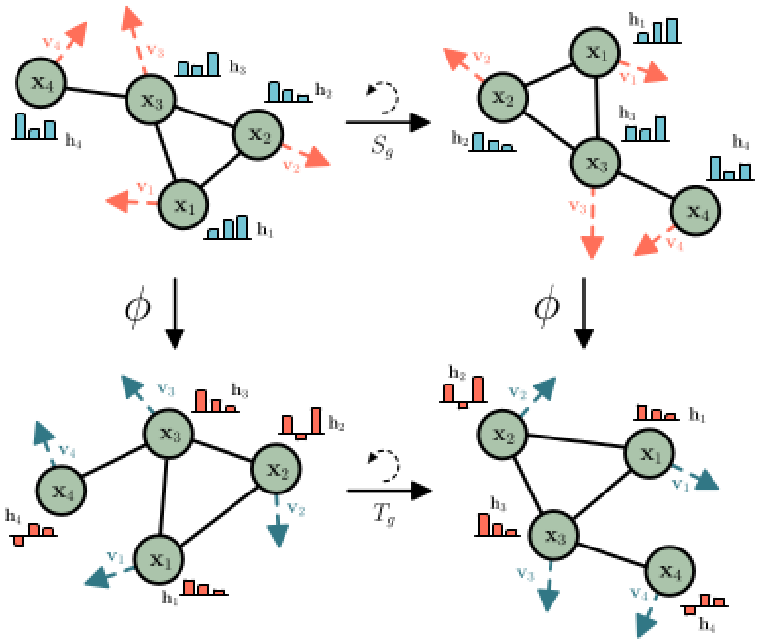
- Coordinate embeddings $\mathbf{x}_i^l$
- Node embeddings $\boldsymbol{h}^t$
- Message passing

$$\boldsymbol{m}_{ij} = \phi(\boldsymbol{h}_i^t, \boldsymbol{h}_j^t, ||\mathbf{x}_i^t - \mathbf{x}_j^t||^2, \boldsymbol{e}_{ij})$$

$$\boldsymbol{m}_i = \sum_{j \in N(i)} \boldsymbol{m}_{ij}$$

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + C \sum_{\mu \neq \nu} (\mathbf{x}_i^t - \mathbf{x}_j^t) \phi_1(\boldsymbol{m}_{ij})$$

$$\boldsymbol{h}_i^{t+1} = \psi_2(\boldsymbol{h}_i^t, \boldsymbol{m}_i)$$



- Properties: rotation-, and translation equivariance (layer) and invariance (whole network); **E(n) equivariant**

JKU JOHANNES KEPLER UNIVERSITY LINZ

Proof sketch of equivariance

- We first note that the messages are invariant under roto-translations, $\mathbf{R}\mathbf{x}_i + \mathbf{t} \quad \forall i$

$$\boldsymbol{m}_{ij} = \boldsymbol{\phi}(\boldsymbol{h}_i^t, \boldsymbol{h}_j^t, ||\mathbf{R}\mathbf{x}_i^t + \mathbf{t} - (\mathbf{R}\mathbf{x}_j^t + \mathbf{t})||^2, \boldsymbol{e}_{ij}) = \boldsymbol{\phi}(\boldsymbol{h}_i^t, \boldsymbol{h}_j^t, ||\mathbf{x}_i^t - \mathbf{x}_j^t||^2, \boldsymbol{e}_{ij})$$

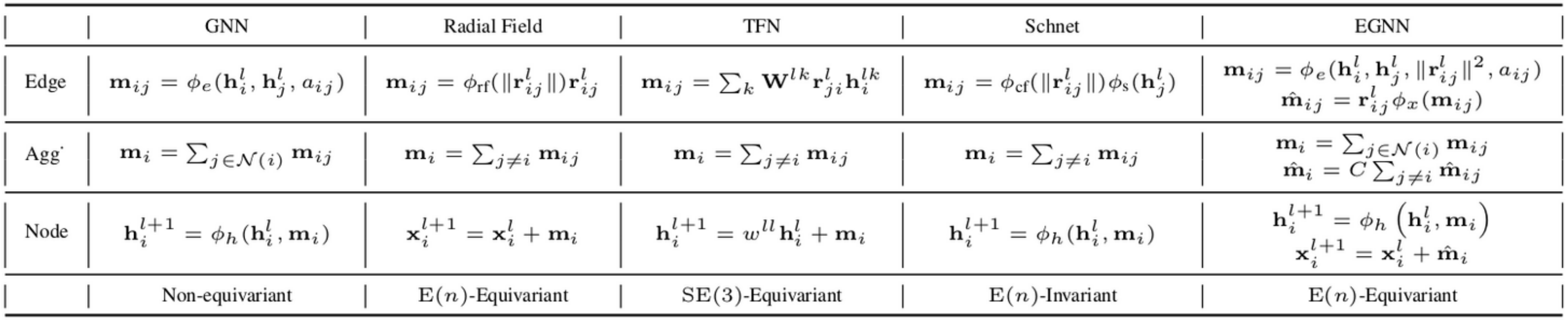
because
$$\begin{aligned} \|\mathbf{R}\mathbf{x}_i^t + \mathbf{t} - [\mathbf{R}\mathbf{x}_j^t + \mathbf{t}]\|^2 &= \|\mathbf{R}\mathbf{x}_i^t - \mathbf{R}\mathbf{x}_j^t\|^2 \\ &= (\mathbf{x}_i^t - \mathbf{x}_j^t)^\top \mathbf{R}^\top \mathbf{R} (\mathbf{x}_i^t - \mathbf{x}_j^t) \\ &= (\mathbf{x}_i^t - \mathbf{x}_j^t)^\top \mathbf{I} (\mathbf{x}_i^t - \mathbf{x}_j^t) \\ &= \|\mathbf{x}_i^t - \mathbf{x}_j^t\|^2 \end{aligned}$$

- Then, we find by re-organizing the terms, that the coordinate embeddings

$$\mathbf{R}\mathbf{x}_i^t + \mathbf{t} + C \sum_{\mu \neq \nu} (\mathbf{R}\mathbf{x}_i^t + \mathbf{t} - (\mathbf{R}\mathbf{x}_j^t + \mathbf{t}))\boldsymbol{\phi}_1(\boldsymbol{m}_{ij}) = \ldots = \mathbf{R}\mathbf{x}_i^{t+1} + \mathbf{t}$$

JOHANNES KEPLER UNIVERSITY LINZ

Overview of MPNNs and their associated groups



	GNN	Radial Field	TFN	Schnet	EGNN
Edge	$\mathbf{m}_{ij} = \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l, a_{ij})$	$\mathbf{m}_{ij} = \phi_{\text{rf}}(\lVert\mathbf{r}_{ij}^l\rVert)\mathbf{r}_{ij}^l$	$\mathbf{m}_{ij} = \sum_k \mathbf{W}^{lk}\mathbf{r}_{ji}^l\mathbf{h}_i^{lk}$	$\mathbf{m}_{ij} = \phi_{\text{cf}}(\lVert\mathbf{r}_{ij}^l\rVert)\phi_{\text{s}}(\mathbf{h}_j^l)$	$\mathbf{m}_{ij} = \phi_e(\mathbf{h}_i^l, \mathbf{h}_j^l, \lVert\mathbf{r}_{ij}^l\rVert^2, a_{ij})$ $\hat{\mathbf{m}}_{ij} = \mathbf{r}_{ij}^l\phi_x(\mathbf{m}_{ij})$
Agg.	$\mathbf{m}_i = \sum_{j\in\mathcal{N}(i)} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\neq i} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\neq i} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\neq i} \mathbf{m}_{ij}$	$\mathbf{m}_i = \sum_{j\in\mathcal{N}(i)} \mathbf{m}_{ij}$ $\hat{\mathbf{m}}_i = C\sum_{j\neq i} \hat{\mathbf{m}}_{ij}$
Node	$\mathbf{h}_i^{l+1} = \phi_h(\mathbf{h}_i^l, \mathbf{m}_i)$	$\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \mathbf{m}_i$	$\mathbf{h}_i^{l+1} = w^{ll}\mathbf{h}_i^l + \mathbf{m}_i$	$\mathbf{h}_i^{l+1} = \phi_h(\mathbf{h}_i^l, \mathbf{m}_i)$	$\mathbf{h}_i^{l+1} = \phi_h\left(\mathbf{h}_i^l, \mathbf{m}_i\right)$ $\mathbf{x}_i^{l+1} = \mathbf{x}_i^l + \hat{\mathbf{m}}_i$
	Non-equivariant	$\mathrm{E}(n)$-Equivariant	$\mathrm{SE}(3)$-Equivariant	$\mathrm{E}(n)$-Invariant	$\mathrm{E}(n)$-Equivariant

Methods using the Graph Laplacian

- Can also be formulated as message-passing neural networks
- Reminder: adjacency matrix $\boldsymbol{A}$ (contains neighbourhood)
- Degree matrix: D (contains degree of nodes, diagonal)
- Graph Laplacian: $\boldsymbol{L} = \boldsymbol{I}_n - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$
- **Kipf & Welling (2016):**
  message passing and node update in one

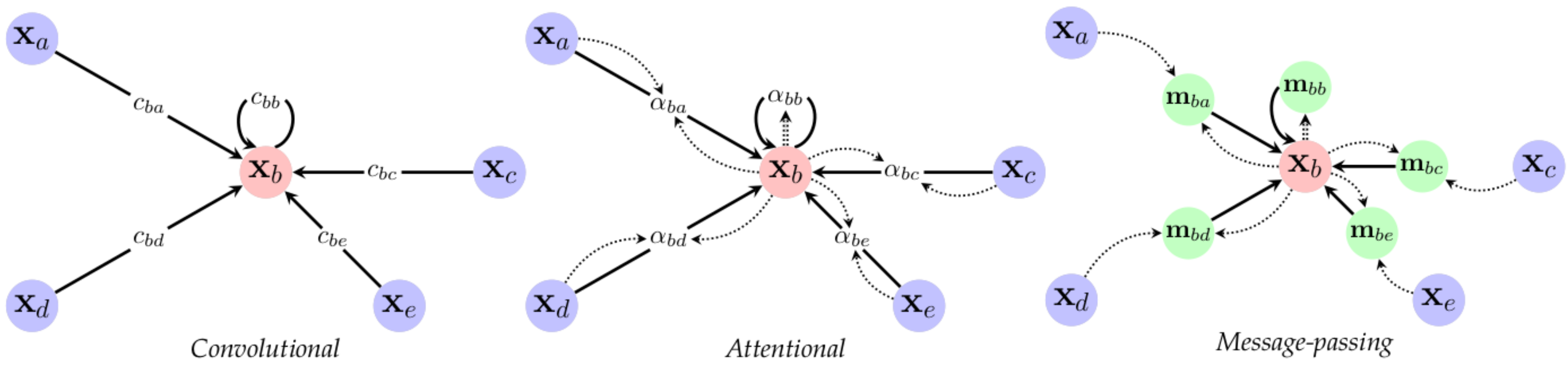
$$\boldsymbol{H}^{l+1} = \sigma\left(\tilde{\boldsymbol{D}}^{-1/2}\tilde{\boldsymbol{A}}\tilde{\boldsymbol{D}}^{-1/2}\boldsymbol{H}^{l}\boldsymbol{W}^{l}\right)$$

  Makes connection with transformers obvious

$$\boldsymbol{H}^{l+1} = \underbrace{\boldsymbol{A}}_{\text{attention}} \underbrace{\boldsymbol{H}^{l}\boldsymbol{W}_{V}^{l}}_{\boldsymbol{V}}$$

JOHANNES KEPLER UNIVERSITY LINZ

Analogy to convolutional networks and transformers



- Comparison of exchange of information in CNNs, Transformers, and MPNNs

JOHANNES KEPLER UNIVERSITY LINZ

Problems with learning MPNNs (without much details; informal)

- **Limited expressivity:** limited ability to distinguish non-isomorpic graphs (see later); → Weisfeiler-Lehman test
- **Oversmoothing:** representations of nodes become more and more similar over message-passing steps
- **Oversquashing:** influence of one node on another node depends decreases exponentially with length of shortest path (something like the “vanishing gradient problem” for GNNs)
- **Under-reaching:** Information from one node is necessary for correctly classifying another node, but there are insufficient message passing steps to transfer the information

Aggregation functions

- Up to now, we have used sum as aggregation function

$$\boldsymbol{m}_i^{t+1} = \sum_{j \in N(j)} \phi(\boldsymbol{h}_i^t, \boldsymbol{h}_j^t, \boldsymbol{e}_{ij})$$

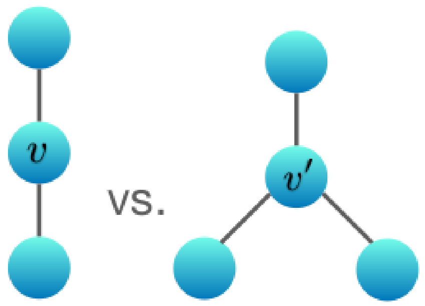
- This is good for expressivity (see later), but

$\frac{1}{N}\sum_{j=1}^{N}$	$\max_{j=1}^{N}$	$\sum_{j=1}^{N}$	$\frac{1}{\sqrt{N}}\sum_{j=1}^{N}$
mean aggregation (MEAN)	max aggregation (MAX)	sum aggregation (SUM)	variance-preserving aggregation (VPA)
✗ expressivity	✗ expressivity	✓ expressivity	✓ expressivity
○ signal propagation	○ signal propagation	✗ signal propagation	✓ signal propagation

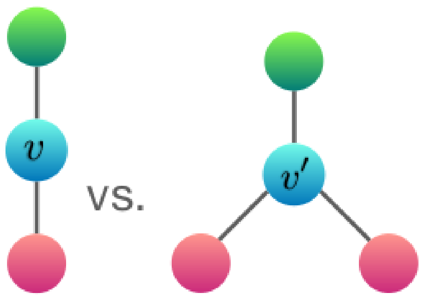
Schneckenreiter, L., Freinschlag, R., Sestak, F., Brandstetter, J., Klambauer, G., & Mayr, A. (2024). GNN-VPA: A Variance-Preserving Aggregation Strategy for Graph Neural Networks. International Conference on Learning Representations (tiny papers track) arXiv preprint arXiv:2403.04747.

Problems with GNNs

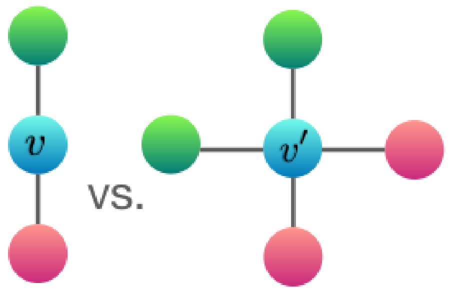
- Limited ability to discriminate different graphs
  - Aggregation function leads to problems (see Fig.)
- Graph isomorphism problem
  - NP hard
- Weisfeiler-Lehman test



(a) Mean and Max both fail



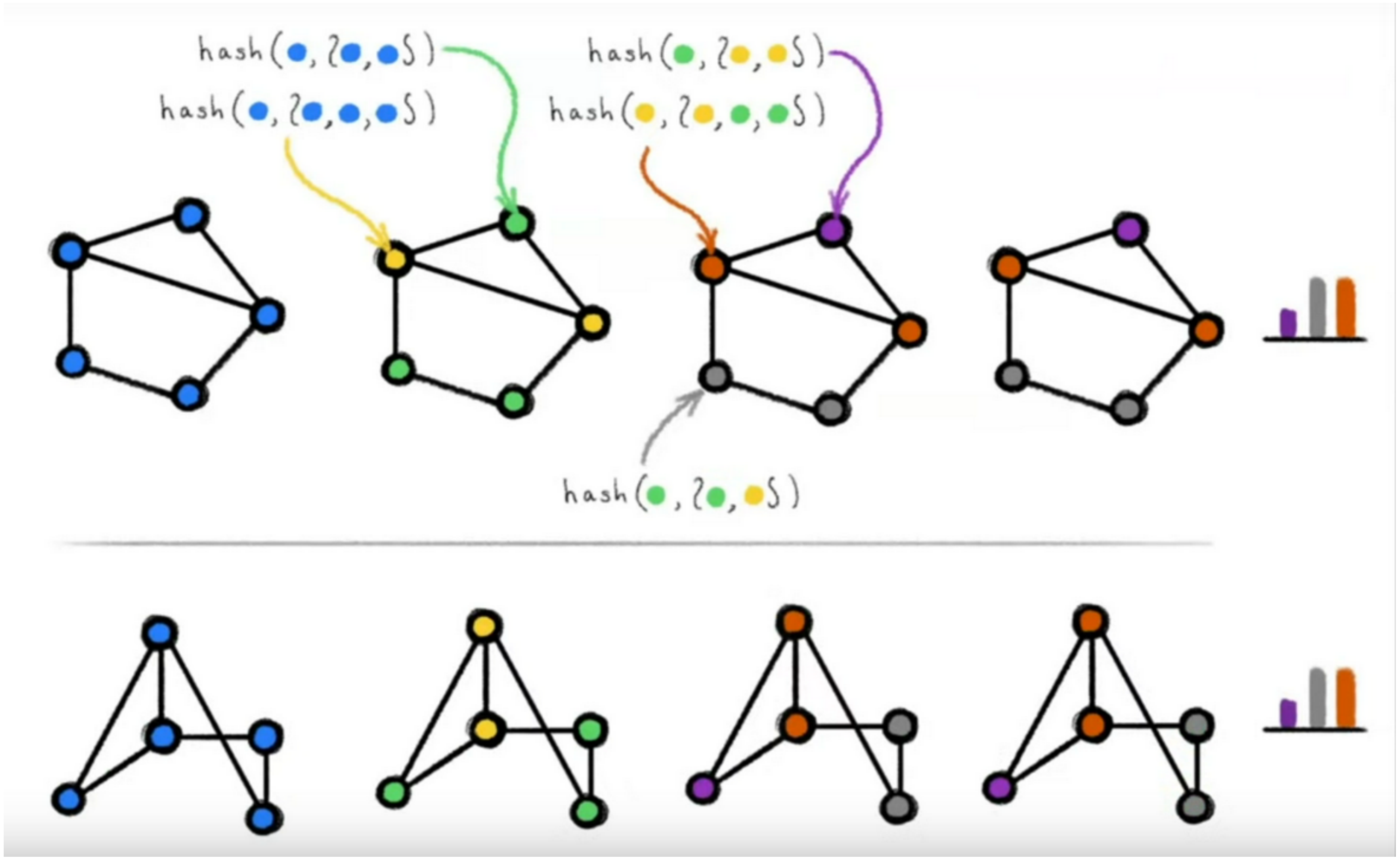
(b) Max fails



(c) Mean and Max both fail

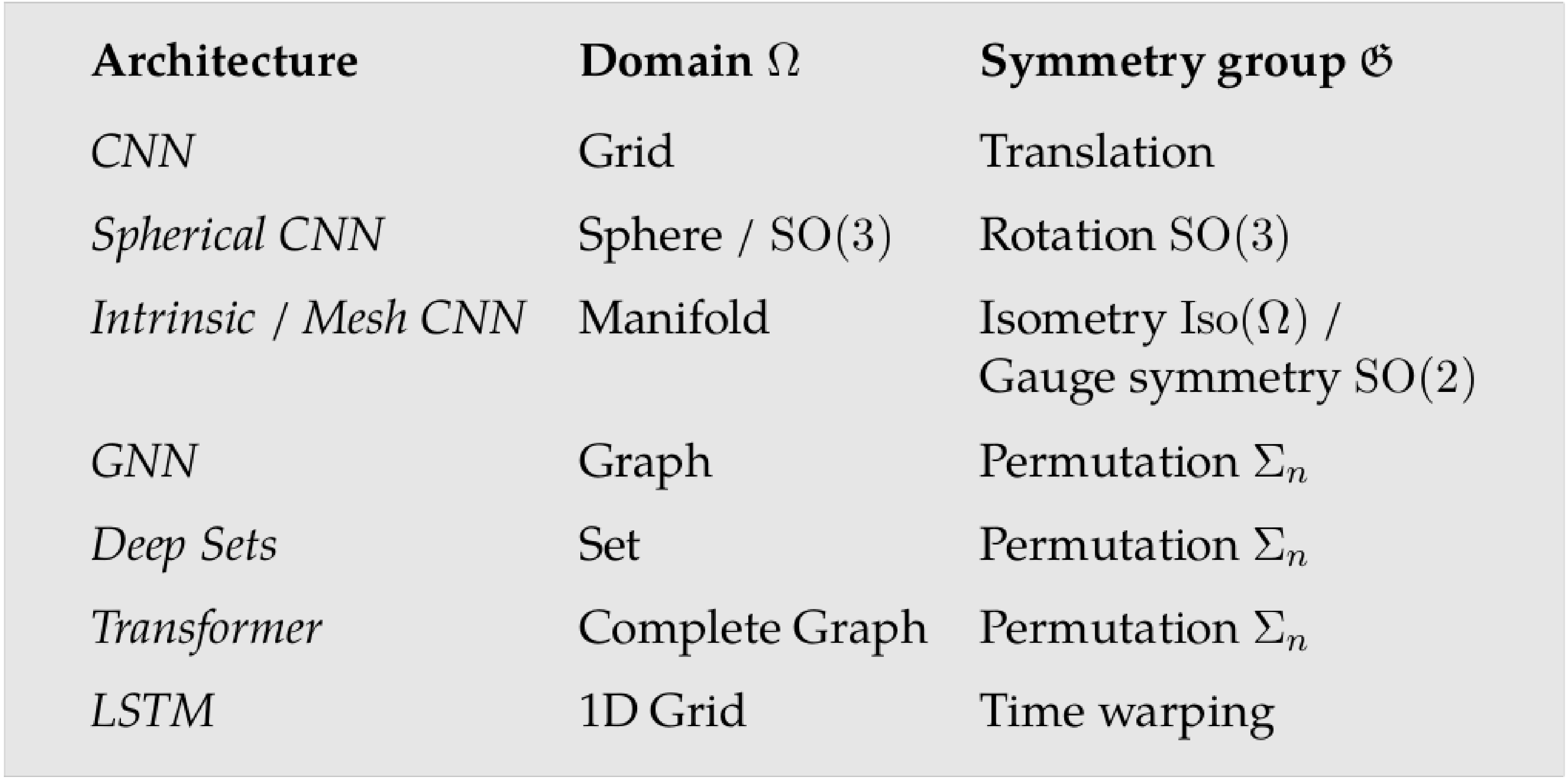
Xu, K., Hu, W., Leskovec, J., & Jegelka, S. (2018). How powerful are graph neural networks?. arXiv preprint arXiv:1810.00826.

Weisfeiler-Lehman test: necessary condition for graph isomorphism



JOHANNES KEPLER UNIVERSITY LINZ

Geometric DL: a short overview



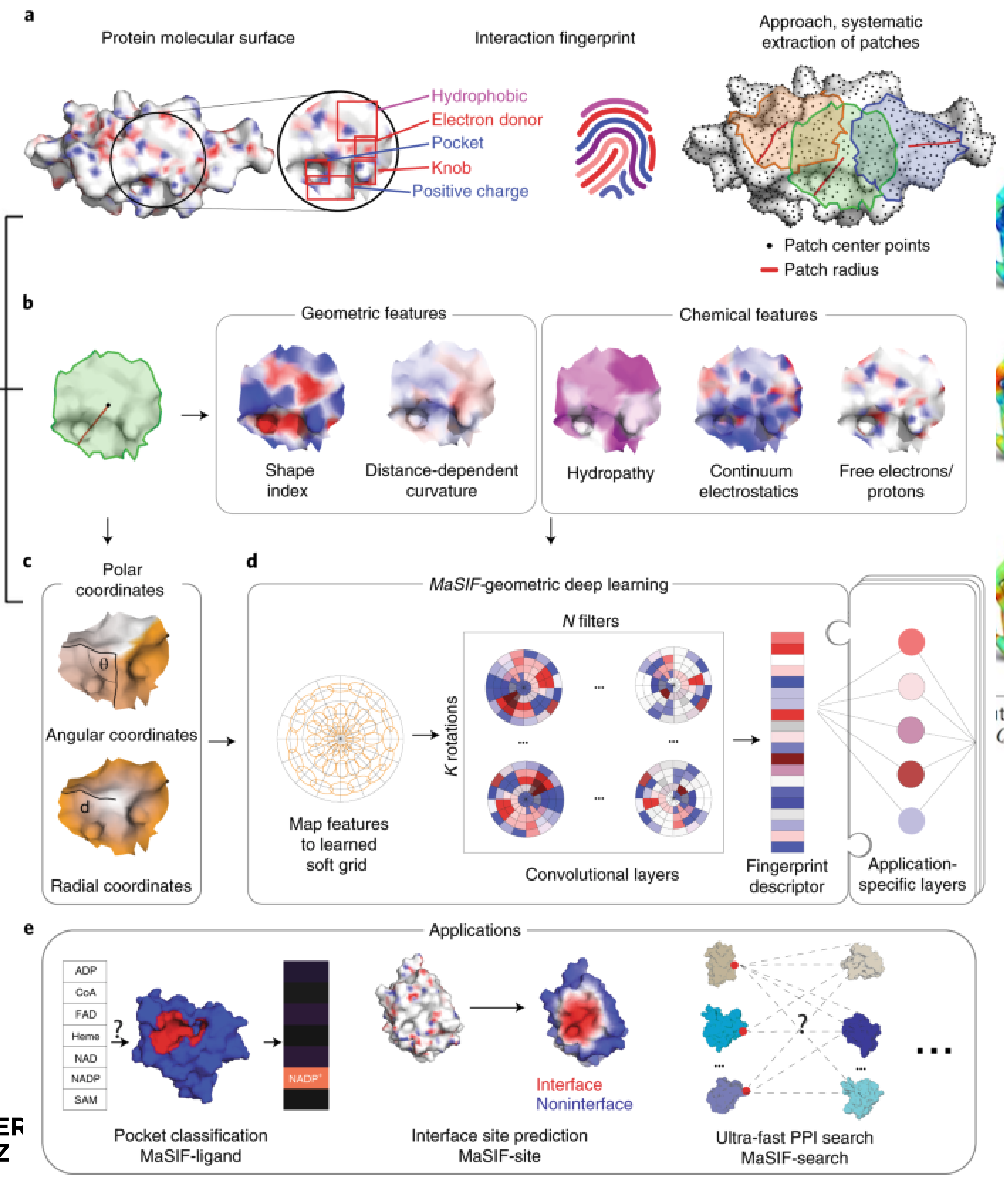
Architecture	**Domain Ω**	**Symmetry group $\mathfrak{G}$**
CNN	Grid	Translation
Spherical CNN	Sphere / SO(3)	Rotation SO(3)
Intrinsic / Mesh CNN	Manifold	Isometry Iso(Ω) / Gauge symmetry SO(2)
GNN	Graph	Permutation Σ_n
Deep Sets	Set	Permutation Σ_n
Transformer	Complete Graph	Permutation Σ_n
LSTM	1D Grid	Time warping

JOHANNES KEPLER UNIVERSITY LINZ

Applications

- Chemistry and Drug Design
- Protein biology: 3D conformations of proteins are point clouds; SE(3) transformer (roto-translation invariant)
- Recommender Systems and social networks
- Traffic forecasting
- Object recognition
- Game computing
- Natural language processing
- Healthcare
- Particle physics and astrophysics

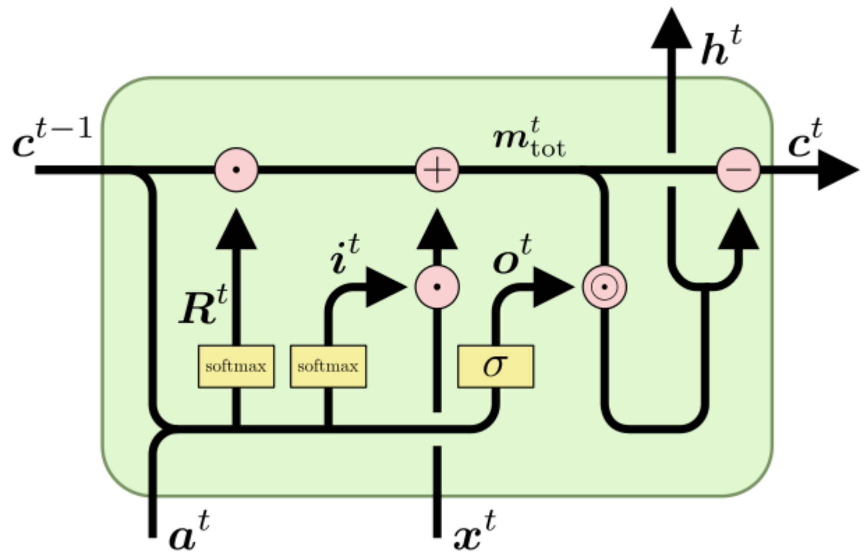
Applications

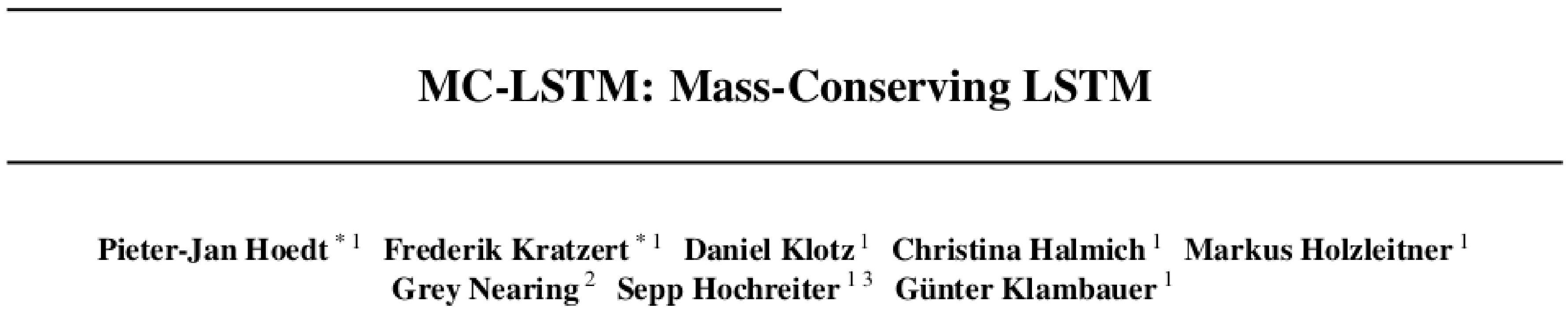


JOHANNES KEPLER UNIVERSITY LINZ

Recent works at IML on Geometric Deep Learning

- Mass-conserving LSTM
- **Equivariance** with respect to adding mass on input

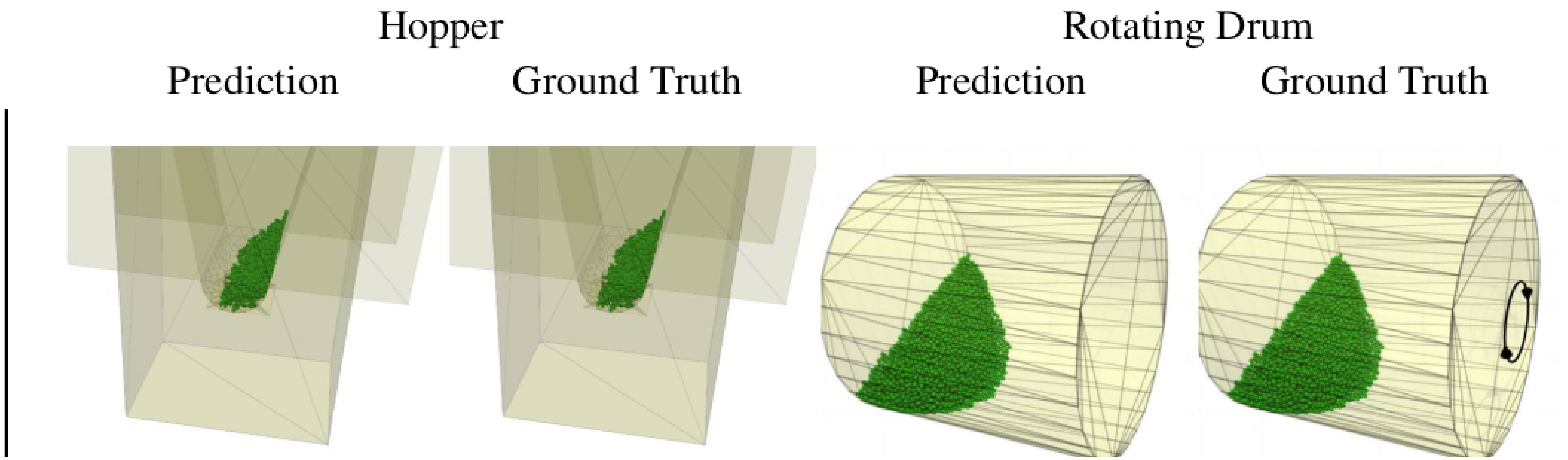




MC-LSTM: Mass-Conserving LSTM

Pieter-Jan Hoedt [*1] Frederik Kratzert [*1] Daniel Klotz [1] Christina Halmich [1] Markus Holzleitner [1]
Grey Nearing [2] Sepp Hochreiter [1 3] Günter Klambauer [1]

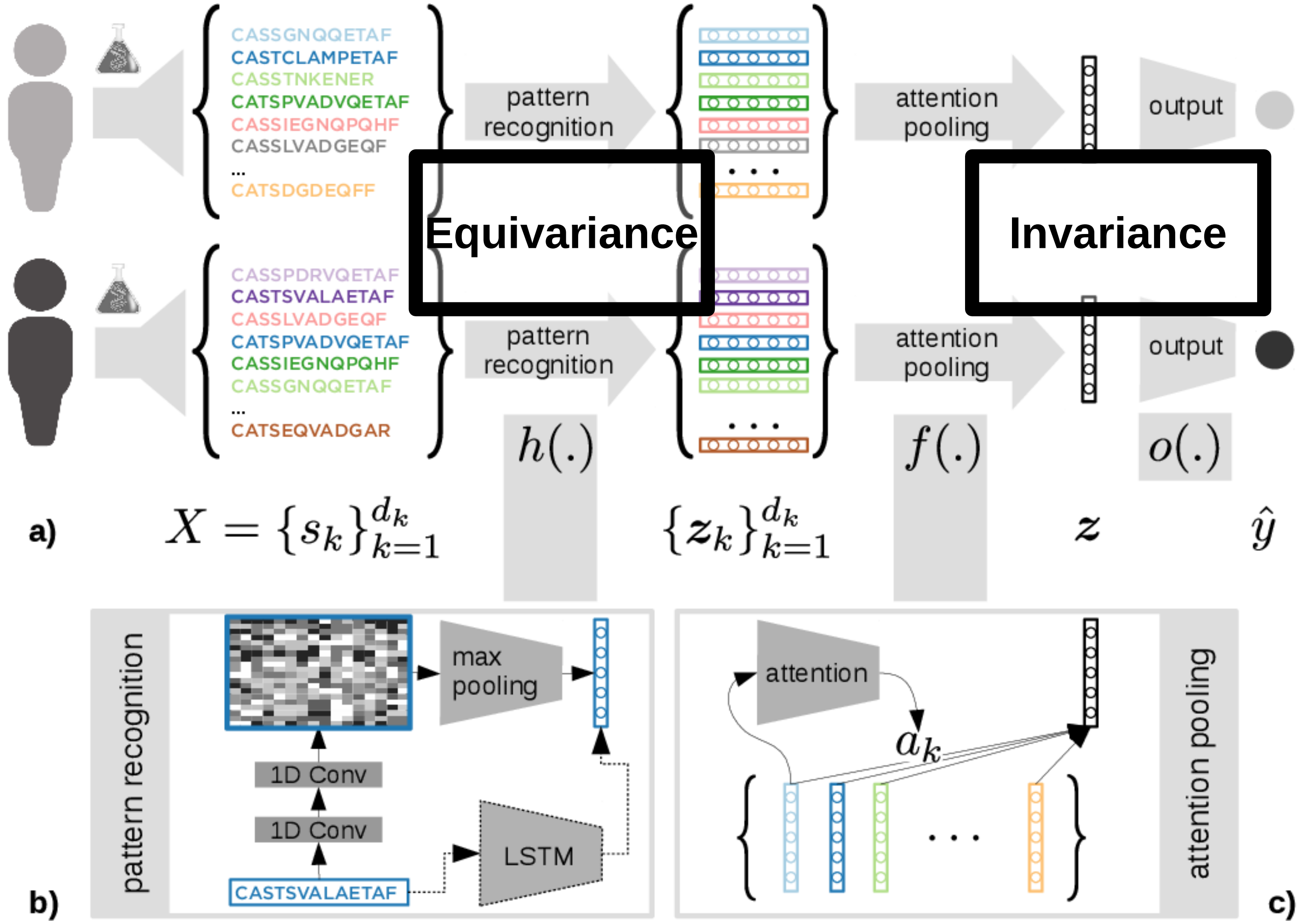
Recent works at IML on Geometric Deep Learning



LEARNING 3D GRANULAR FLOW SIMULATIONS

Andreas Mayr[*] **Sebastian Lehner**[*] **Arno Mayrhofer**[†] **Christoph Kloss**[†]
Sepp Hochreiter[*,‡] **Johannes Brandstetter**[*,§]
[*] ELLIS Unit Linz, LIT AI Lab, Institute for Machine Learning,
Johannes Kepler University Linz, Austria

Recent works at IML: Immune repertoire classification



Widrich, M., Schäfl, B., Pavlović, M., Ramsauer, H., Gruber, L., Holzleitner, M., ... & Klambauer, G. (2020). Modern hopfield networks and attention for immune repertoire classification. Advances in Neural Information Processing Systems, 33, 18832-18845.

Summary

- GNNs and Geometric Deep Learning are approaches to look at Deep Learning architectures from the perspective of symmetries, equivariances and invariances
- Particular equivariances and invariances with respect to transformations are desired, lead to certain architectures, such as CNN
- Graph neural networks, and message-passing networks operate on graphs;
  - Connections to transformers and convolutions
- Active research areas and applications