



Machine Learning for discrete optimization: Graph Neural Networks, generalization under shifts, and loss functions

Stefanie Jegelka
MIT

based on joint work with

Keyulu Xu, Ching-Yao Chuang, Nikolaos Karalias, Joshua Robinson, Andreas Loukas, Jingling Li, Mozhi Zhang, Simon S. Du, Ken-ichi Kawarabayashi

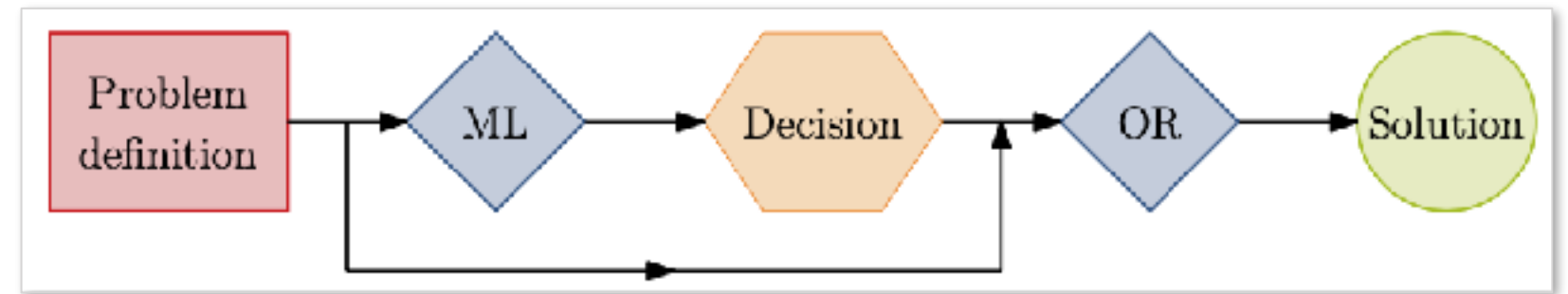
Learning and Algorithms

- Predicting good configurations / solvers

(Leyton-Brown et al 2002, Hutter et al 2011, Gupta et al 2015, Balcan et al 2017..)

- Predicting problem parameters

(Amos & Kolter 2017, Wilder et al 2019, Donti et al 2019, Elmachoub & Grigas 2020, Mandi & Guns 2020, Berthet et a...)



Learning and Algorithms

- Predicting good configurations / solvers

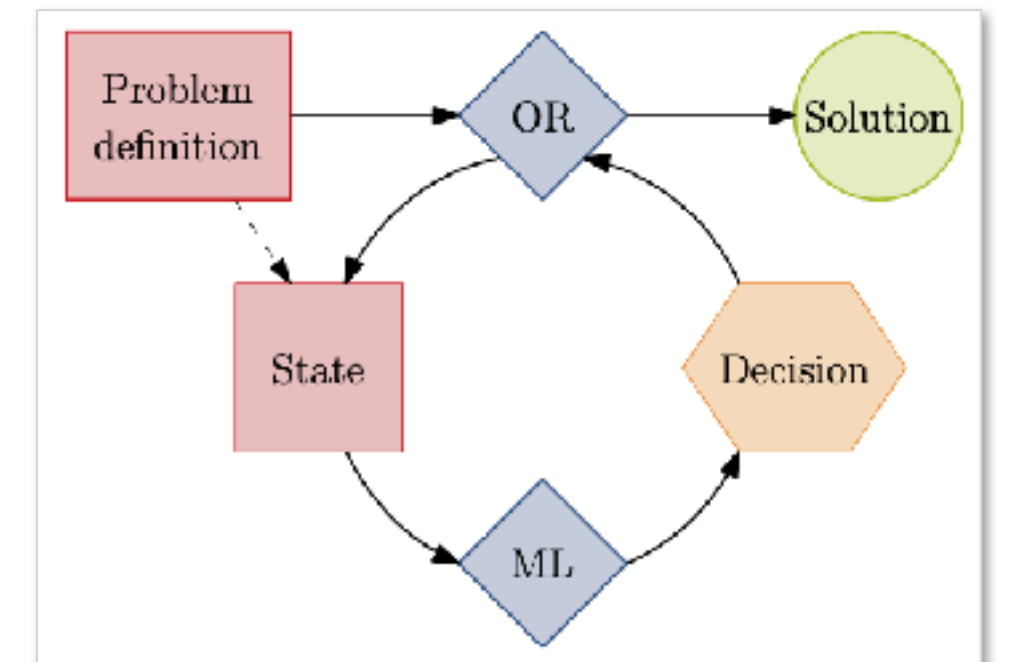
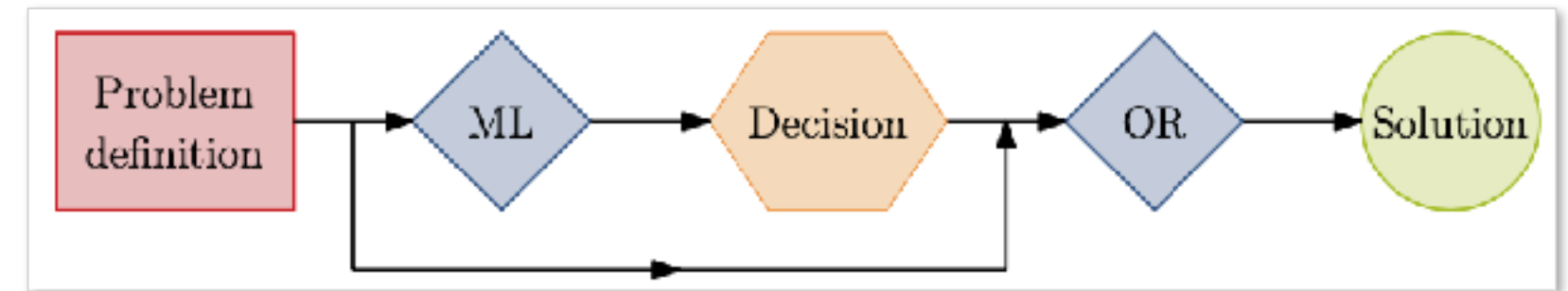
(Leyton-Brown et al 2002, Hutter et al 2011, Gupta et al 2015, Balcan et al 2017..)

- Predicting problem parameters

(Amos & Kolter 2017, Wilder et al 2019, Donti et al 2019, Elmachoub & Grigas 2020, Mandi & Guns 2020, Berthet et a...)

- Machine learning oracles within a fixed algorithm (online algorithms, branch-and-bound,...)

(Kraska et 2018, Balcan et al 2018, Hsu et al 2019, Gasse et al 2019, Dong et al 2020..)



Learning and Algorithms

- Predicting good configurations / solvers

(Leyton-Brown et al 2002, Hutter et al 2011, Gupta et al 2015, Balcan et al 2017..)

- Predicting problem parameters

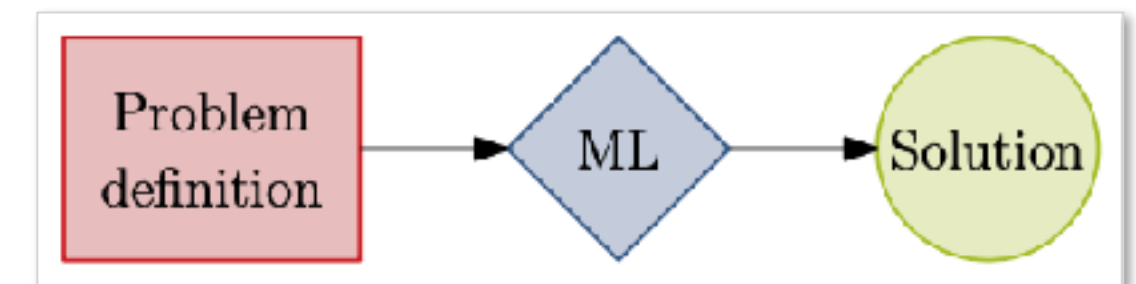
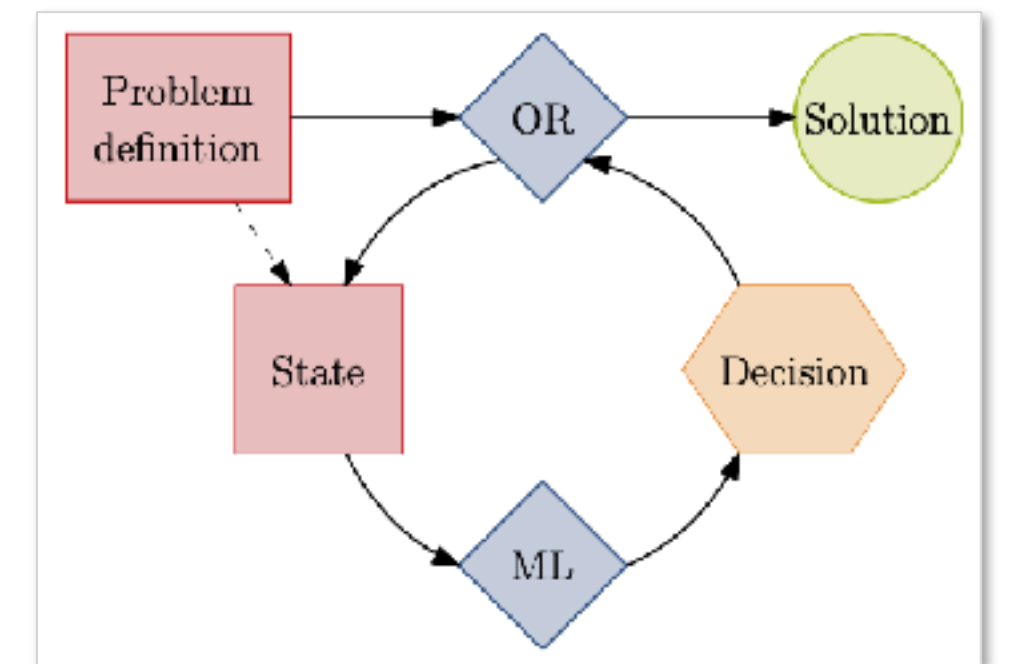
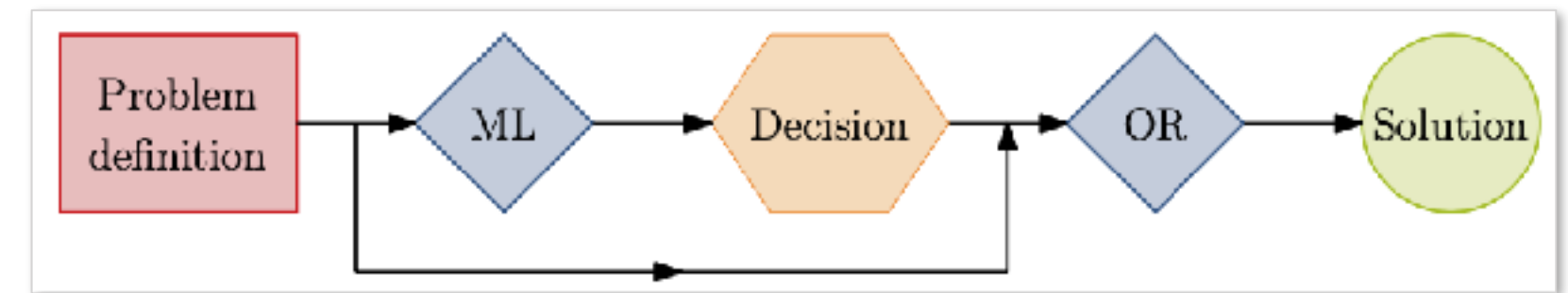
(Amos & Kolter 2017, Wilder et al 2019, Donti et al 2019, Elmachtoub & Grigas 2020, Mandi & Guns 2020, Berthet et a...)

- Machine learning oracles within a fixed algorithm (online algorithms, branch-and-bound,...)

(Kraska et al 2018, Balcan et al 2018, Hsu et al 2019, Gasse et al 2019, Dong et al 2020..)

- Learning a full algorithm, Algorithm implemented as a neural network

(Graves et al 2014, Kaiser & Sutskever 2015, Kurach et al 2015, Kool & Welling 2018, Veličković et al 2020, Karalias & Loukas 2020, Kotary et al 2021, Schuetz et al 2022,...)



Why learn an algorithm?

- Adapt to specific input distributions (without analytic form)
- Helping abstraction of “natural” inputs, e.g., infer costs/weights
(Veličković & Blundell 2021, Lombardi & Milano 2018)
- Bypass expensive exact algorithm
- Discover new heuristics/algorithms, augment expert knowledge
(Xhonneux et al 2021, Veličković et al 2020)
- Many other ML “reasoning” tasks are algorithmic in nature...

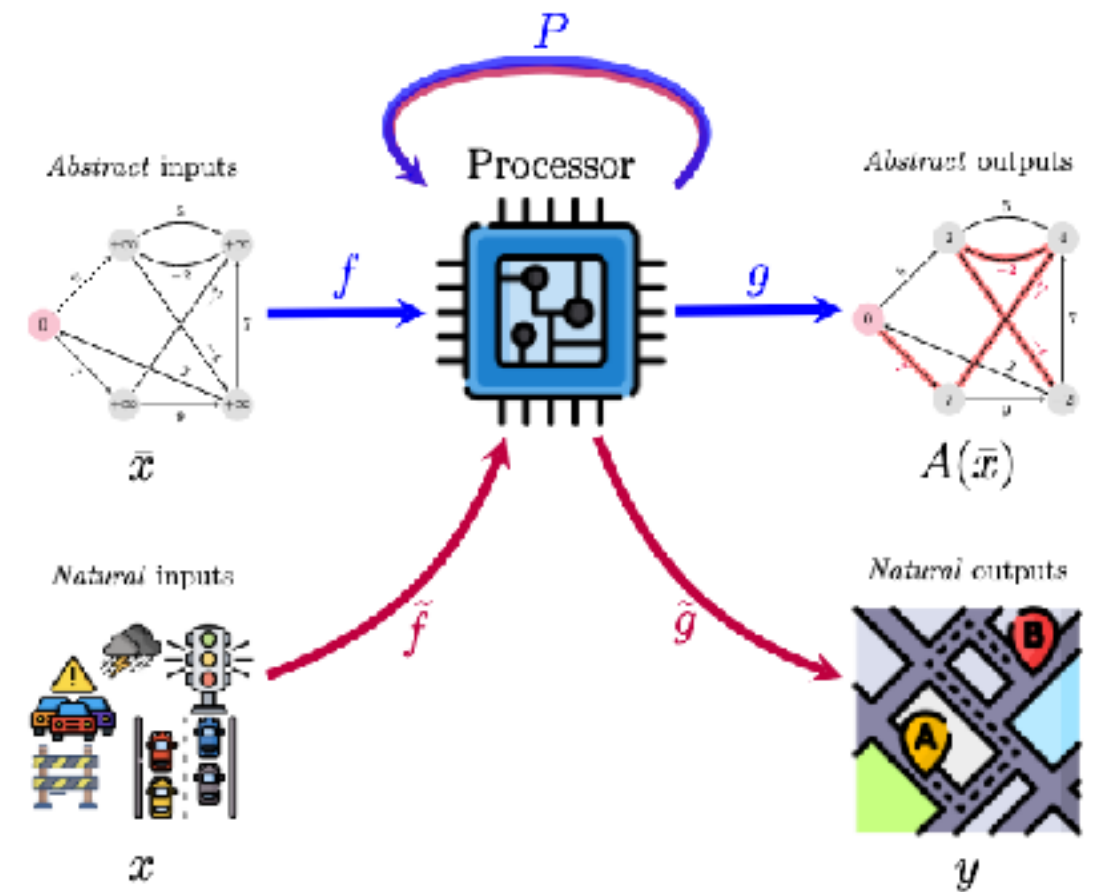


Figure: Veličković & Blundell. Neural Algorithmic Reasoning. Patterns, 2021

Today:
Some thoughts on learning (for) algorithms

Disclaimer

- Learning for algorithms is still a nascent field
(“Although most of the approaches [...] are still at an exploratory level of deployment [...], we strongly believe that this is just the beginning of a new era for combinatorial optimization algorithms.” (Bengio et al 2020))
- Not the “latest & greatest” here...
- ... but **some ideas to better understand ML models’ behavior**

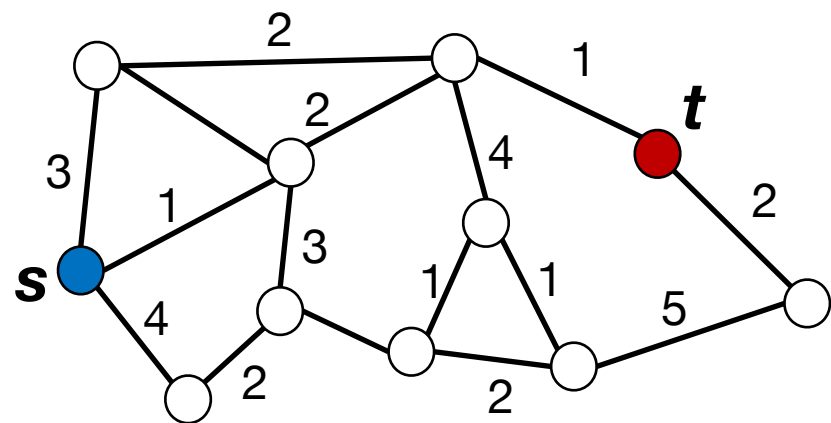
How to do learning for algorithms?

$$\min_x c^\top x$$

$$Ax \leq b$$

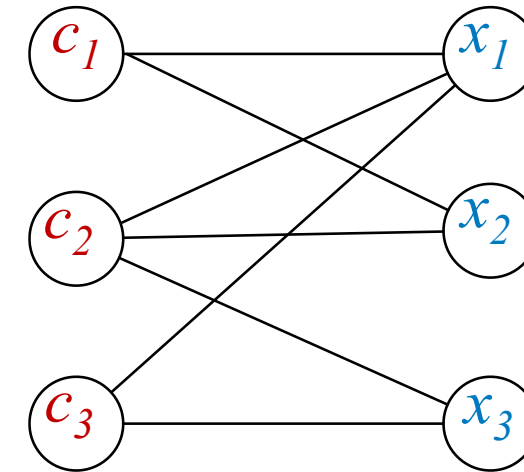
$$l \leq x \leq u$$

$$x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}$$



How to do learning for algorithms?

$$\begin{aligned} \min_x & c^\top x \\ & Ax \leq b \\ & l \leq x \leq u \\ & x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \end{aligned}$$



constraints

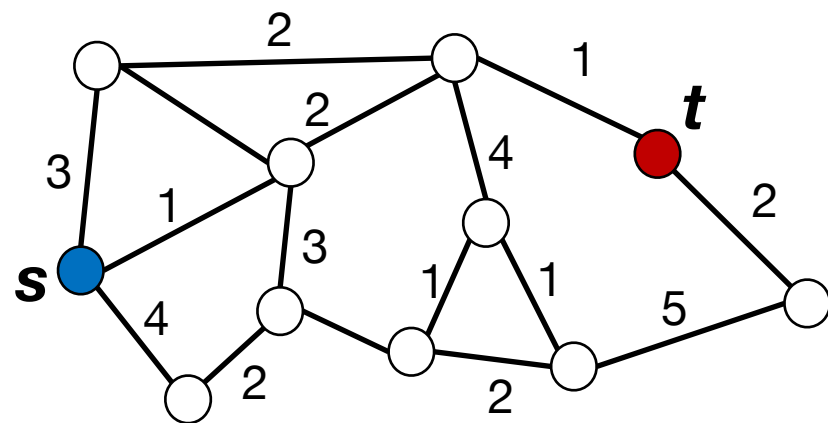
variables

(Gasse et al 2019)

clauses

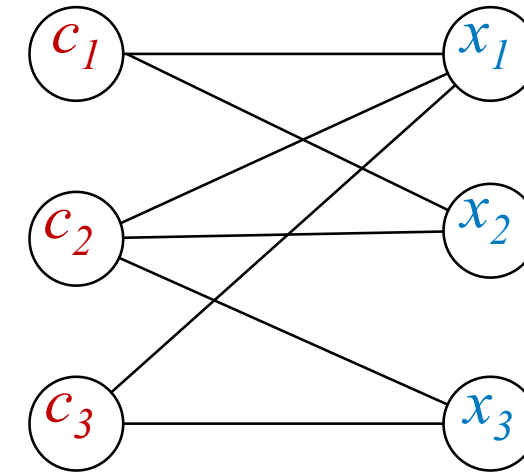
variables

(Selsam et al 2018)



How to do learning for algorithms?

$$\begin{aligned} \min_x & c^\top x \\ & Ax \leq b \\ & l \leq x \leq u \\ & x \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \end{aligned}$$



constraints

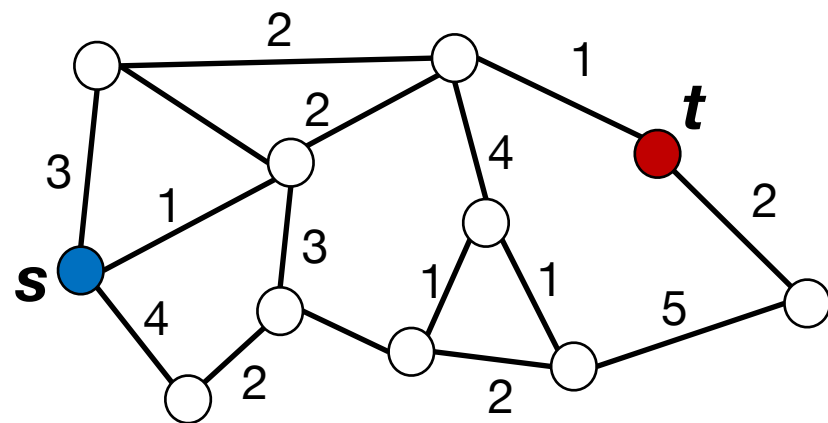
variables

(Gasse et al 2019)

clauses

variables

(Selsam et al 2018)

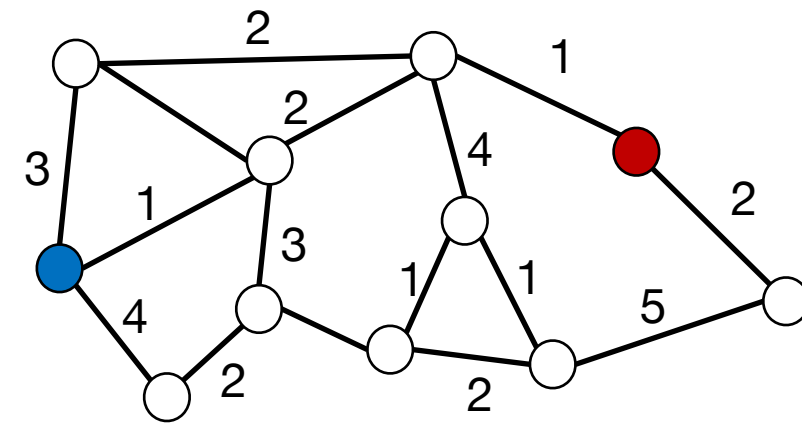


**Focus here:
Input is a graph
with attributes**

Setup: learning task

“graph regression”

- **Input** x : graph (or set) with attributes (optimization instance)
- Desired **output** $g(x)$: optimal value
- **Training data**: $\{(x^{(i)}, g(x^{(i)}))\}_{i=1}^N$ with $x^{(i)} \sim P$ $\leftarrow P$ unknown

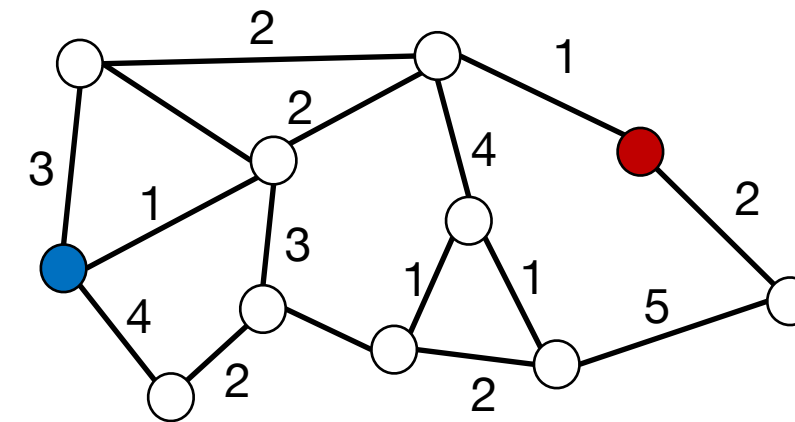


$$g(x) = 4$$

Setup: learning task

“graph regression”

- **Input** x : graph (or set) with attributes (optimization instance)
- Desired **output** $g(x)$: optimal value
- **Training data**: $\{(x^{(i)}, g(x^{(i)}))\}_{i=1}^N$ with $x^{(i)} \sim P$ $\leftarrow P$ unknown
- **Goal**: find a function $f \in \mathcal{F}$ that *generalizes* (= low “risk”)



$$g(x) = 4$$

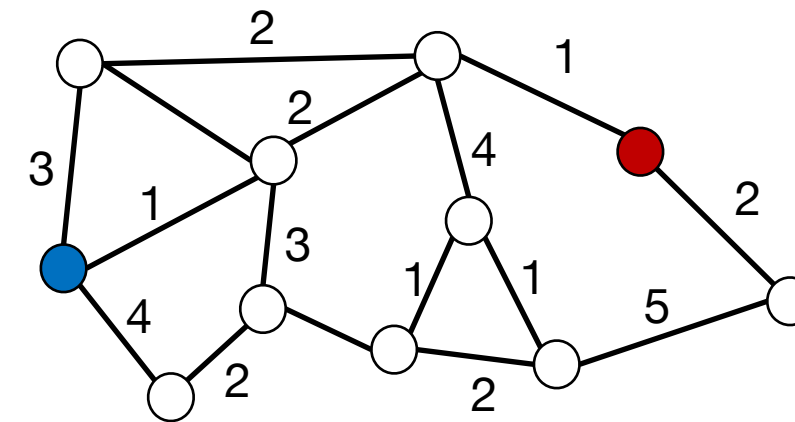
$$\mathbb{E}_{x \sim P} [\ell(f(x), g(x))]]$$

Setup: learning task

“graph regression”

- **Input** x : graph (or set) with attributes (optimization instance)
- Desired **output** $g(x)$: optimal value
- **Training data**: $\{(x^{(i)}, g(x^{(i)}))\}_{i=1}^N$ with $x^{(i)} \sim P$ $\leftarrow P$ unknown
- **Goal**: find a function $f \in \mathcal{F}$ that *generalizes* (= low “risk”)

$$\mathbb{E}_{x \sim P} [\ell(f(x), g(x))]]$$

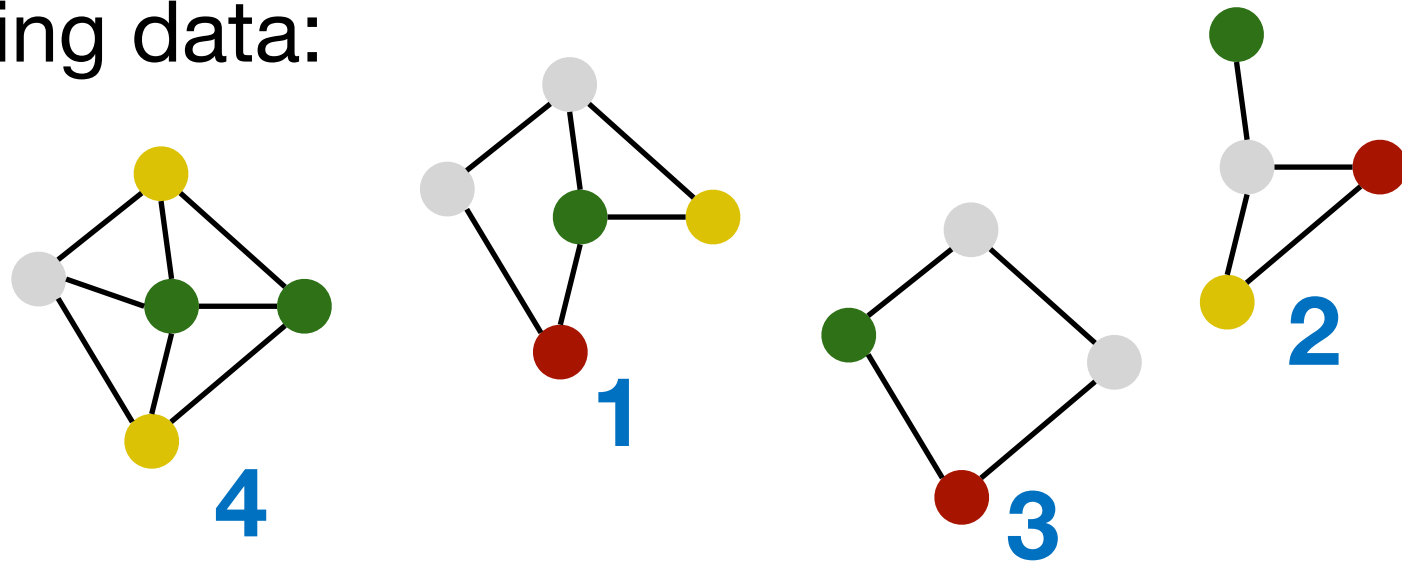


$$g(x) = 4$$

- Other aspects:**
- learning setting
 - loss
 - constraints

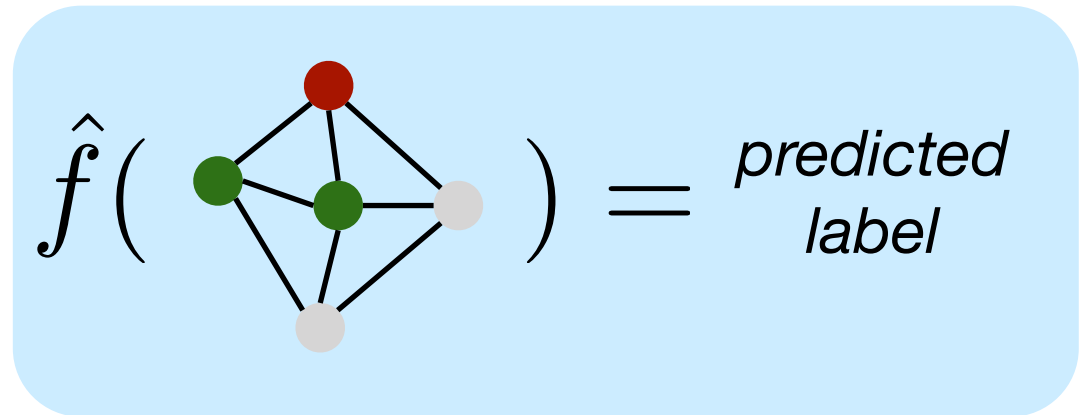
Learning

Training data:



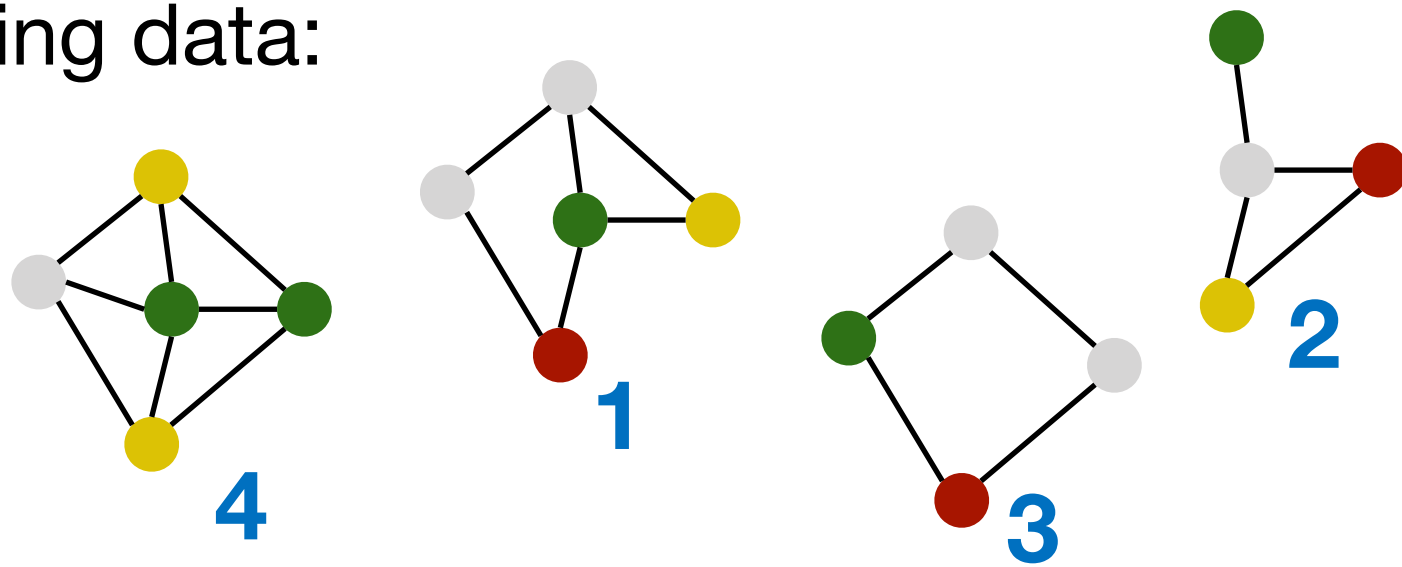
$$\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^N \ell(f(x_i), g(x_i))$$

“Learn” function



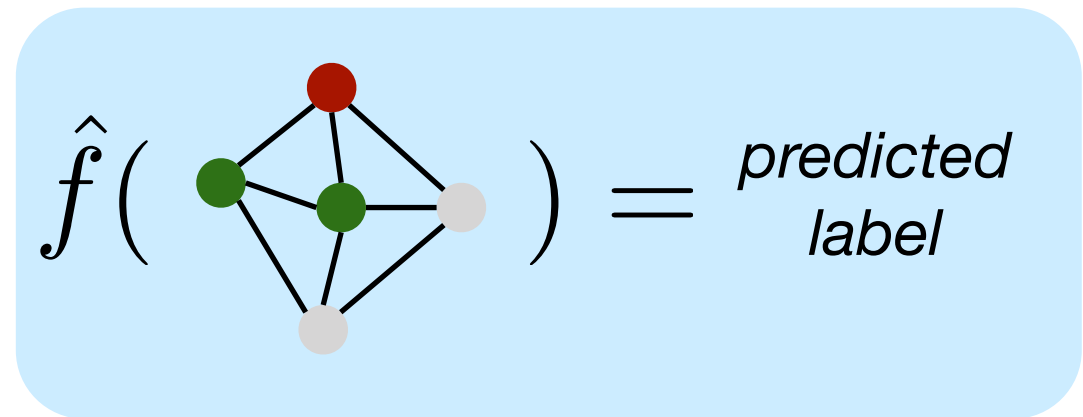
Learning

Training data:

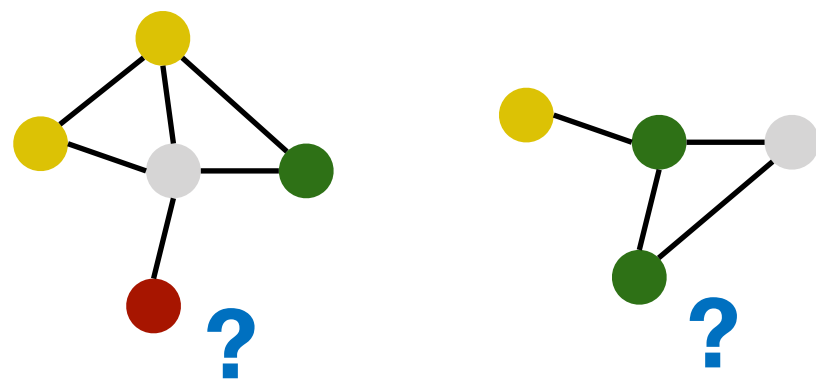


$$\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^N \ell(f(x_i), g(x_i))$$

“Learn” function



Test data:

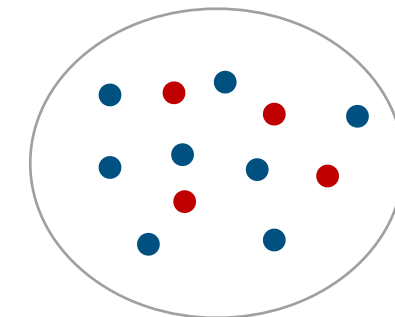


Evaluation: generalization

$$\mathbb{E}_{x \sim P} [\ell(f(x), g(x))]]$$

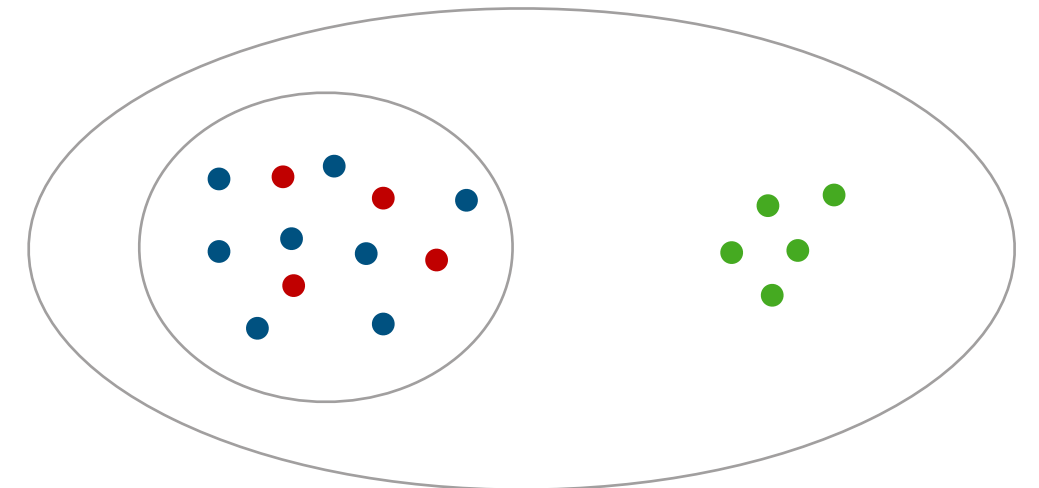
Outline

- Graph Neural Networks
- To what kinds of instances will my model generalize?
Prediction under distribution shifts
 - **stability**: measuring **data** shifts appropriately
 - **large shifts**: understanding **model** behavior by decomposition
- Beyond regression: **extending set functions** as **loss functions** for neural networks



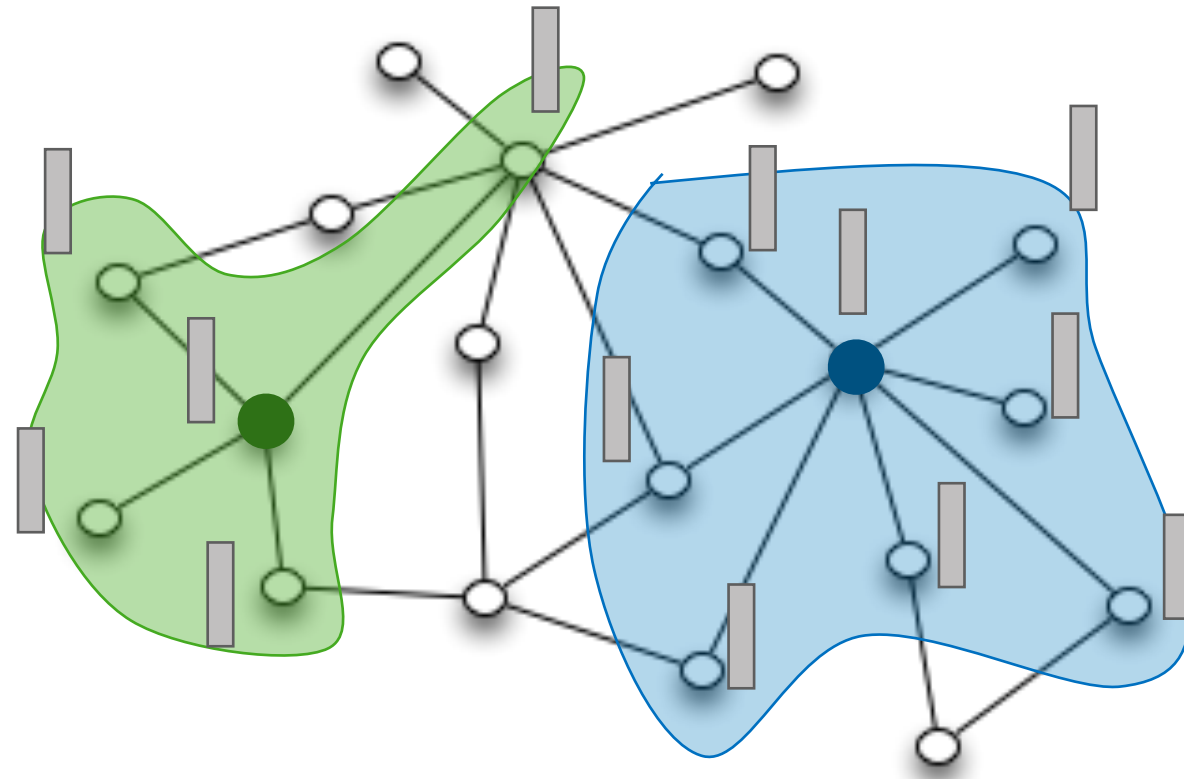
Outline

- Graph Neural Networks
- To what kinds of instances will my model generalize?
Prediction under distribution shifts
 - **stability**: measuring **data** shifts appropriately
 - **large shifts**: understanding **model** behavior by decomposition
- Beyond regression: **extending set functions** as **loss functions** for neural networks



(Message passing) Graph neural networks

Input: graph G with node attributes $x_v \in \mathbb{R}^{d_0}$, edge attributes $w(u, v) \in \mathbb{R}^{d'}$

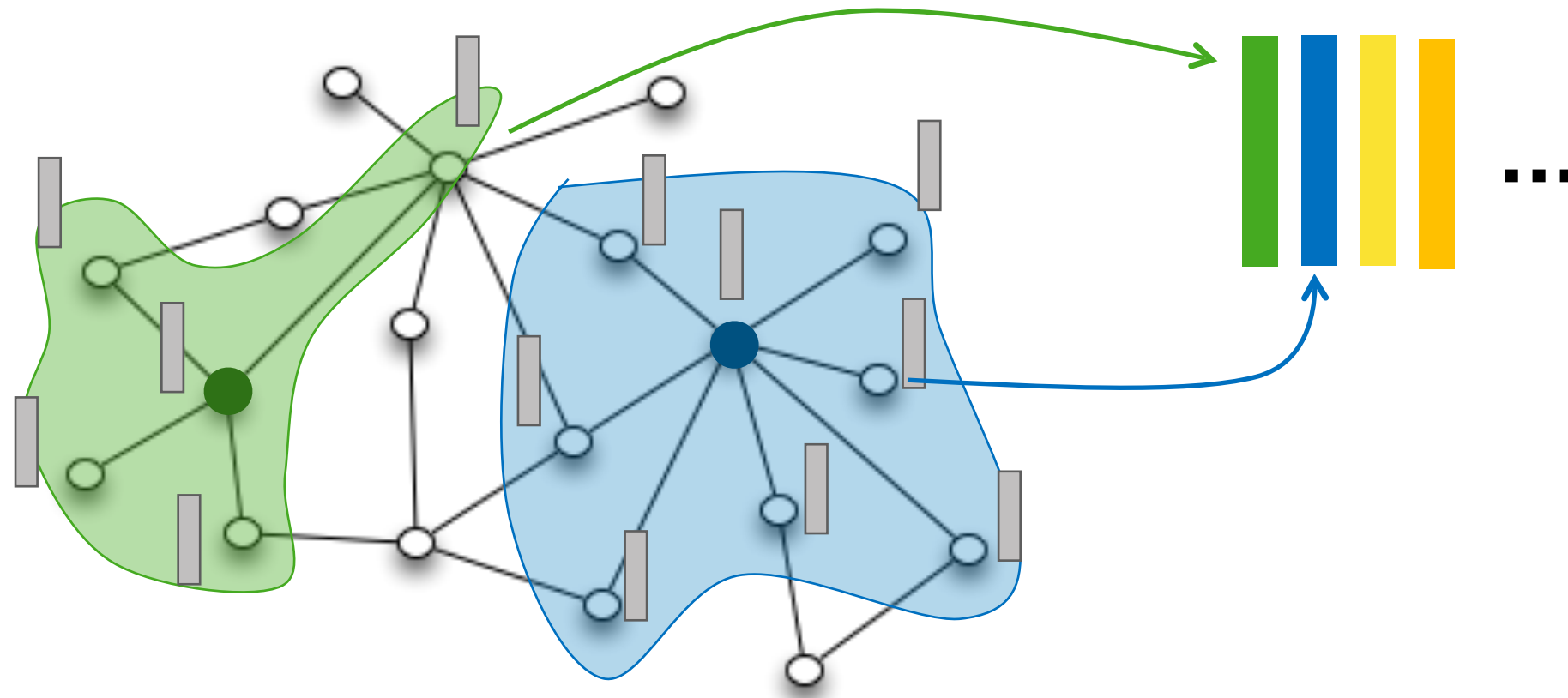


Idea:

1. Encode each node (node's neighborhood): *node embedding*
2. Aggregate set of node embeddings into a *graph embedding*

(Message passing) Graph neural networks

Input: graph G with node attributes $x_v \in \mathbb{R}^{d_0}$, edge attributes $w(u, v) \in \mathbb{R}^{d'}$

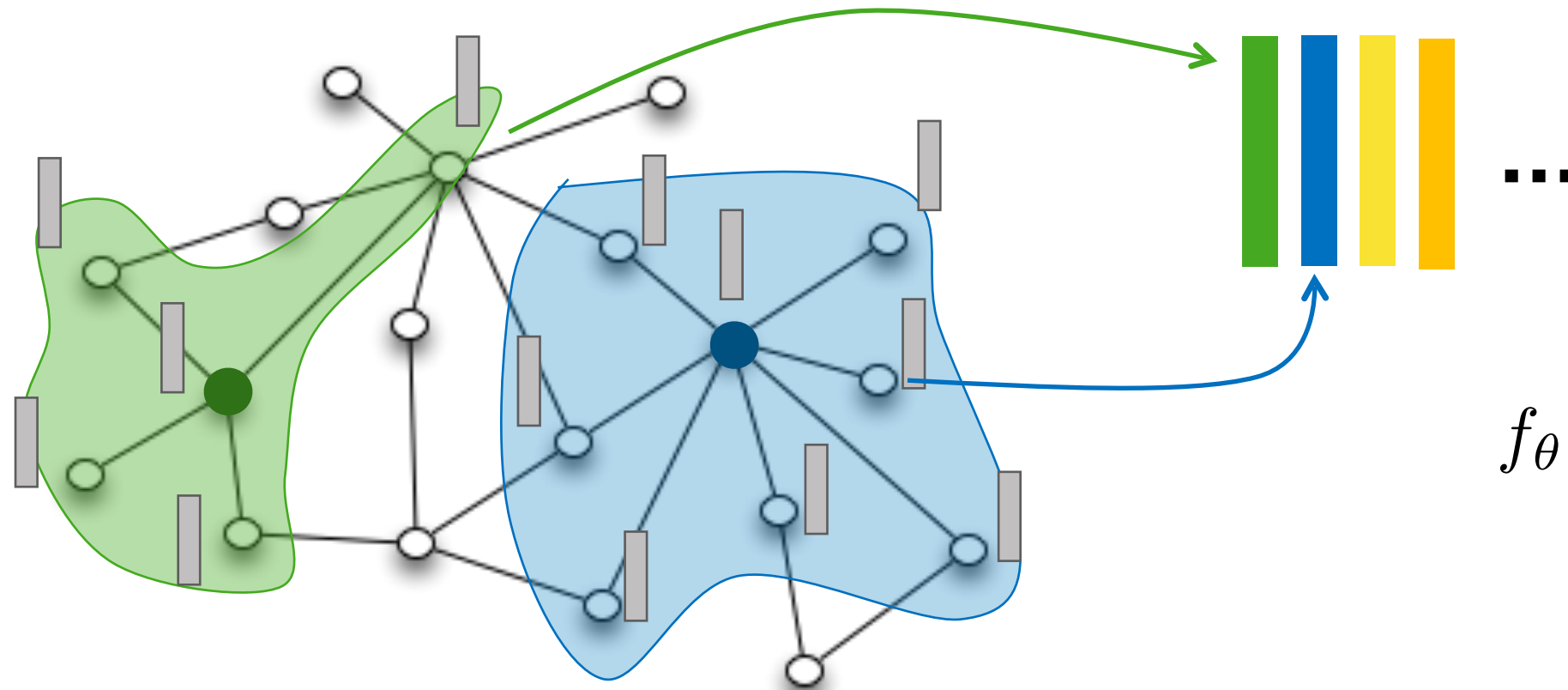


Idea:

1. Encode each node (node's neighborhood): *node embedding*
2. Aggregate set of node embeddings into a *graph embedding*

(Message passing) Graph neural networks

Input: graph G with node attributes $x_v \in \mathbb{R}^{d_0}$, edge attributes $w(u, v) \in \mathbb{R}^{d'}$



$$f_{\theta}(G) = f_{\text{Read}}(\{h_v \mid v \in V\})$$

$$\in \mathbb{R}^{d_{\text{out}}}$$

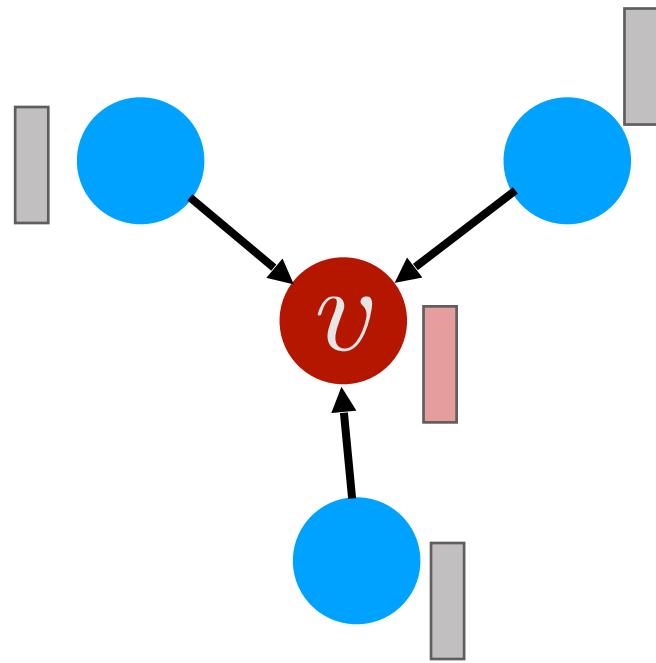
Idea:

1. Encode each node (node's neighborhood): *node embedding*
2. Aggregate set of node embeddings into a *graph embedding*

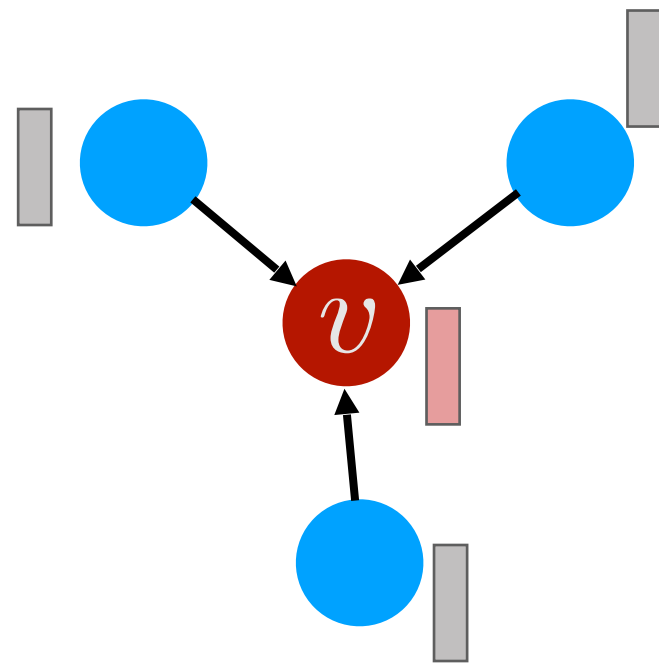
Node embedding: message passing

$$h_v^{(0)} = x_v, \forall v \in V$$

$$h_v^{(t)} \in \mathbb{R}^{d_t}$$



Node embedding: message passing



$$h_v^{(0)} = x_v, \forall v \in V$$

$$h_v^{(t)} \in \mathbb{R}^{d_t}$$

In each round k :

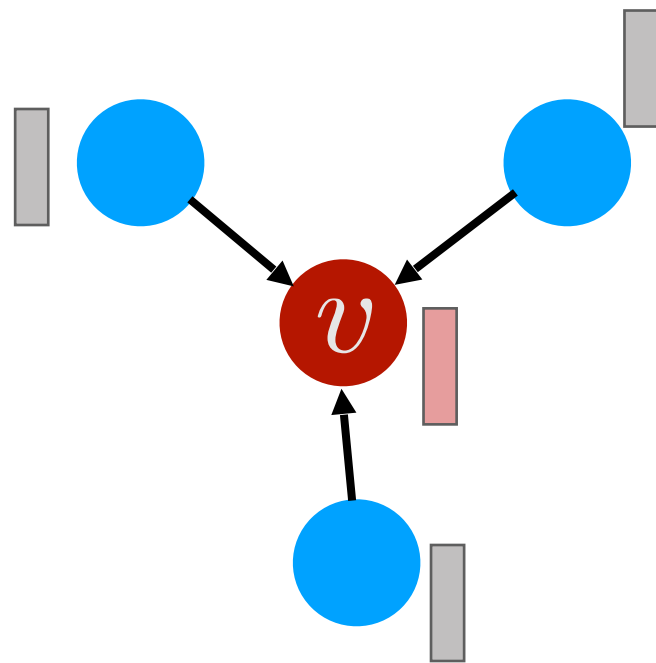
Aggregate over neighbors

$$m_{\mathcal{N}(v)}^{(k)} = \text{AGGREGATE}^{(k)} \left(\left\{ \left\{ h_u^{(k-1)} : u \in \mathcal{N}(v) \right\} \right\} \right)$$

feature description
of node u in round $k-1$

Update: Combine with current node

Node embedding: message passing



$$h_v^{(0)} = x_v, \forall v \in V$$

$$h_v^{(t)} \in \mathbb{R}^{d_t}$$

In each round k :

Aggregate over neighbors

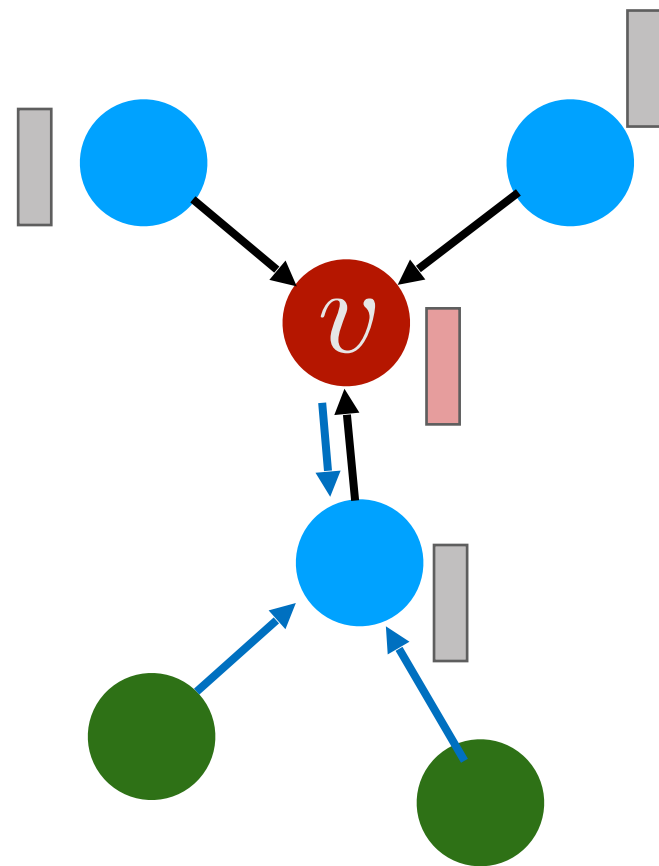
$$m_{\mathcal{N}(v)}^{(k)} = \text{AGGREGATE}^{(k)} \left(\left\{ \left\{ h_u^{(k-1)} : u \in \mathcal{N}(v) \right\} \right\} \right)$$

feature description
of node u in round $k-1$

Update: Combine with current node

$$h_v^{(k)} = \text{COMBINE}^{(k)} \left(h_v^{(k-1)}, m_{\mathcal{N}(v)}^{(k)} \right)$$

Node embedding: message passing



$$h_v^{(0)} = x_v, \forall v \in V$$

$$h_v^{(t)} \in \mathbb{R}^{d_t}$$

In each round k :

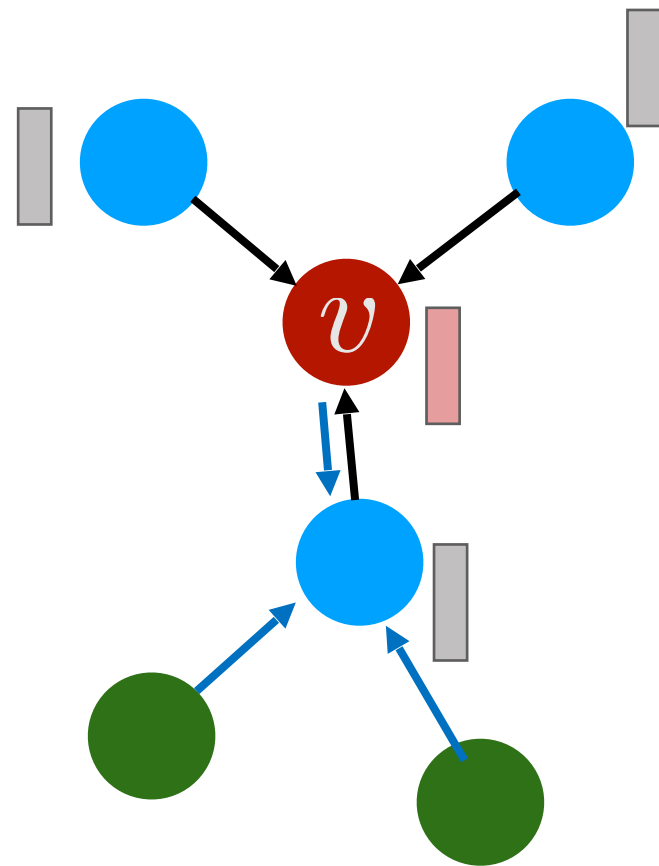
Aggregate over neighbors

$$m_{\mathcal{N}(v)}^{(k)} = \text{AGGREGATE}^{(k)} \left(\left\{ \left\{ h_u^{(k-1)} : u \in \mathcal{N}(v) \right\} \right\} \right)$$

Update: Combine with current node

$$h_v^{(k)} = \text{COMBINE}^{(k)} \left(h_v^{(k-1)}, m_{\mathcal{N}(v)}^{(k)} \right)$$

Node embedding: message passing

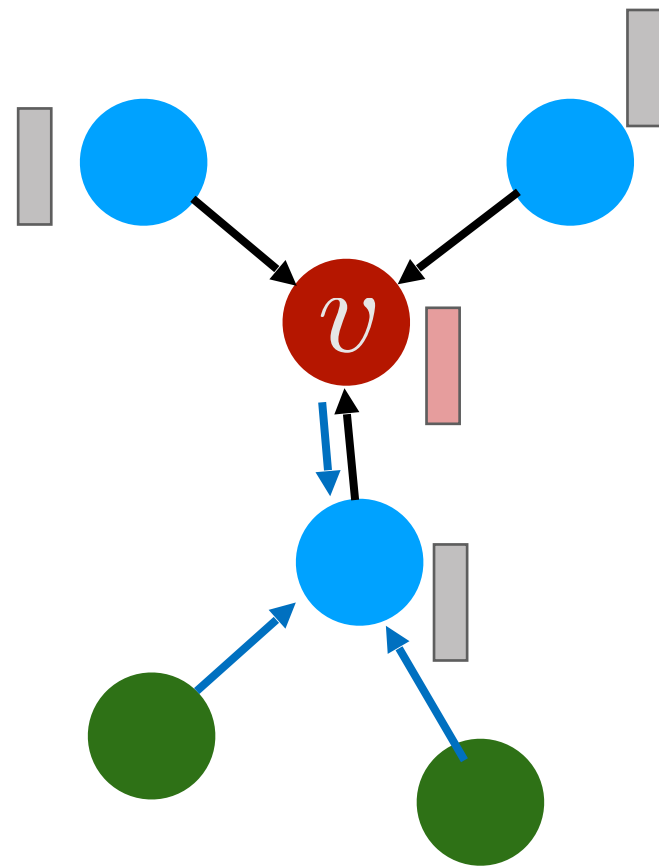


In each round k :
Aggregate over neighbors

feature description
of node u in round $k-1$

$$m_{\mathcal{N}(v)}^{(k)} = \sum_{u \in \mathcal{N}(v)} \text{MLP}^{(k)}(\mathbf{h}_u^{(k-1)}, \mathbf{h}_v^{(k-1)}, \mathbf{w}_{(v,u)})$$

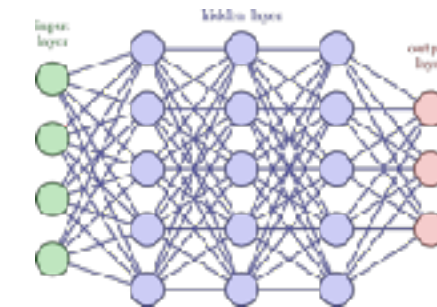
Node embedding: message passing



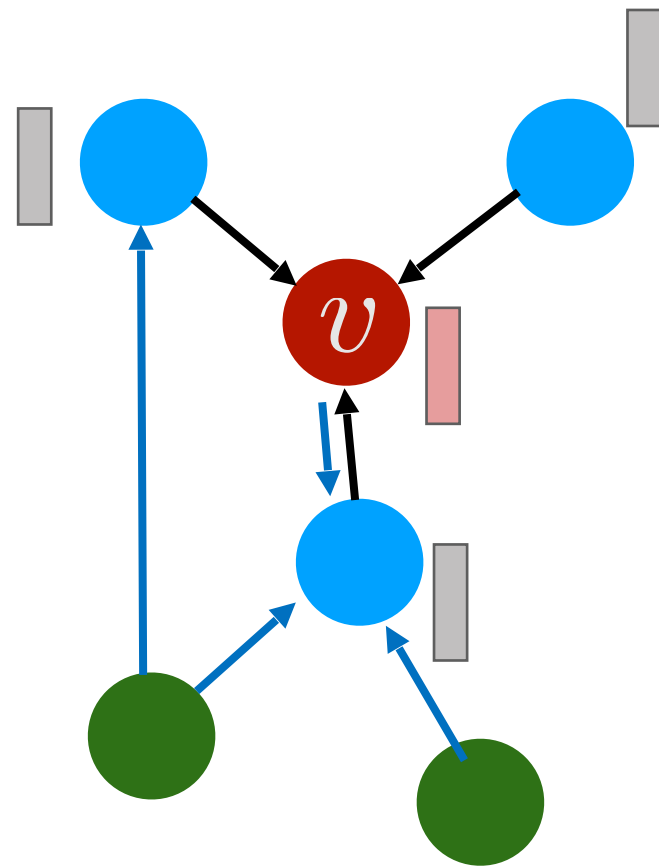
In each round k :
Aggregate over neighbors

$$m_{\mathcal{N}(v)}^{(k)} = \sum_{u \in \mathcal{N}(v)} \text{MLP}^{(k)}(h_u^{(k-1)}, h_v^{(k-1)}, \mathbf{w}_{(v,u)})$$

feature description
of node u in round $k-1$



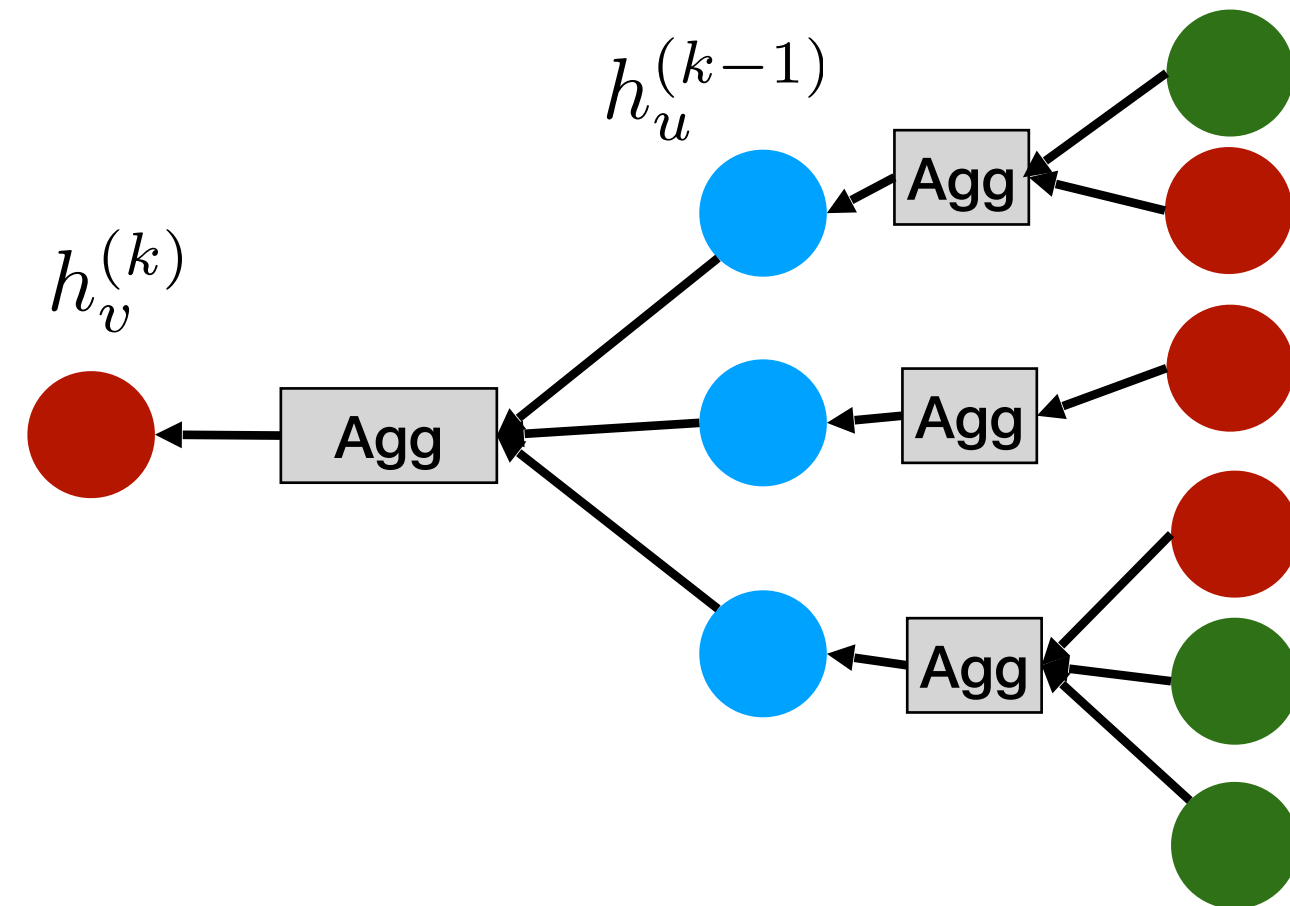
Message passing unrolled



In each round k :

Aggregate over neighbors and update representation

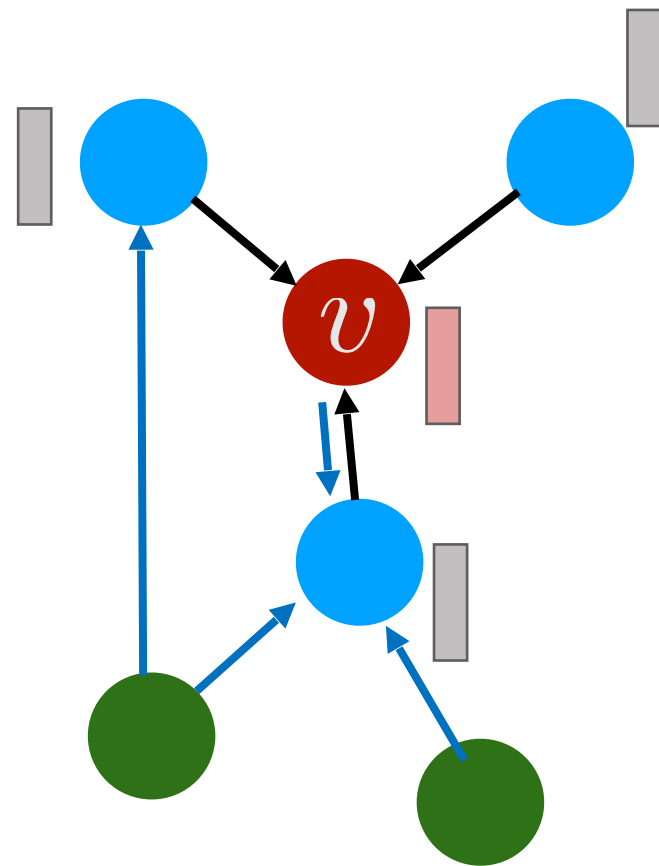
“Unrolled”: **computation tree**



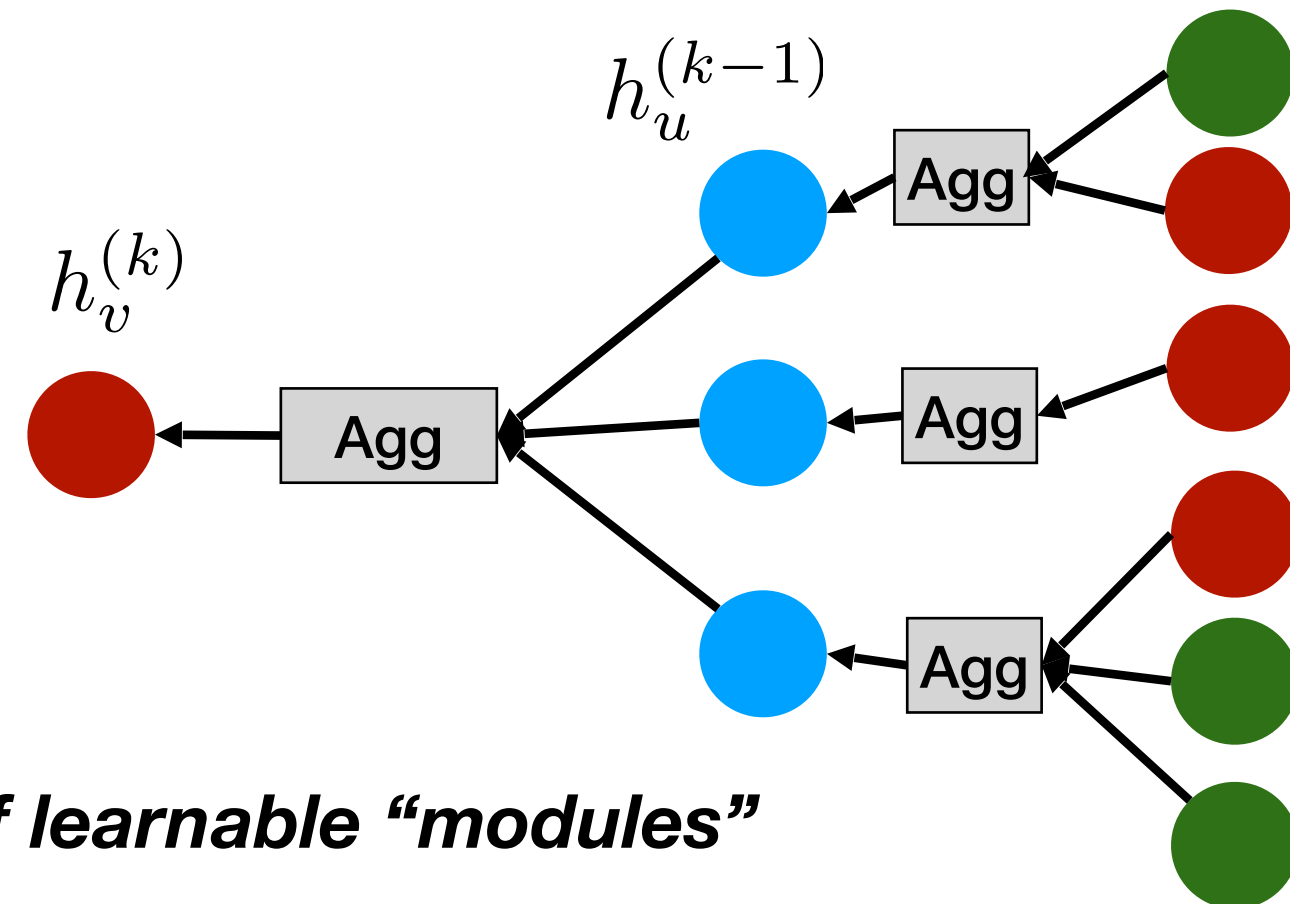
Message passing unrolled

In each round k :

Aggregate over neighbors and update representation



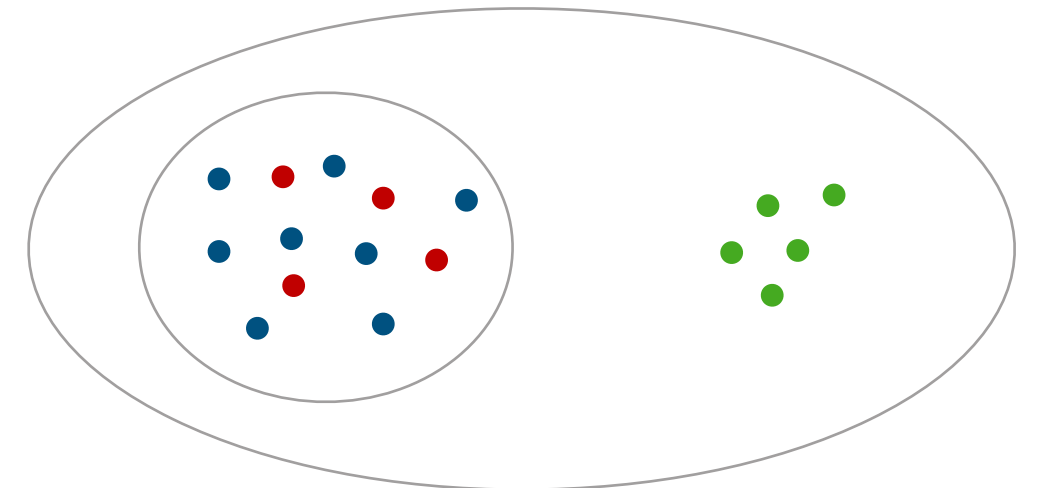
“Unrolled”: **computation tree**



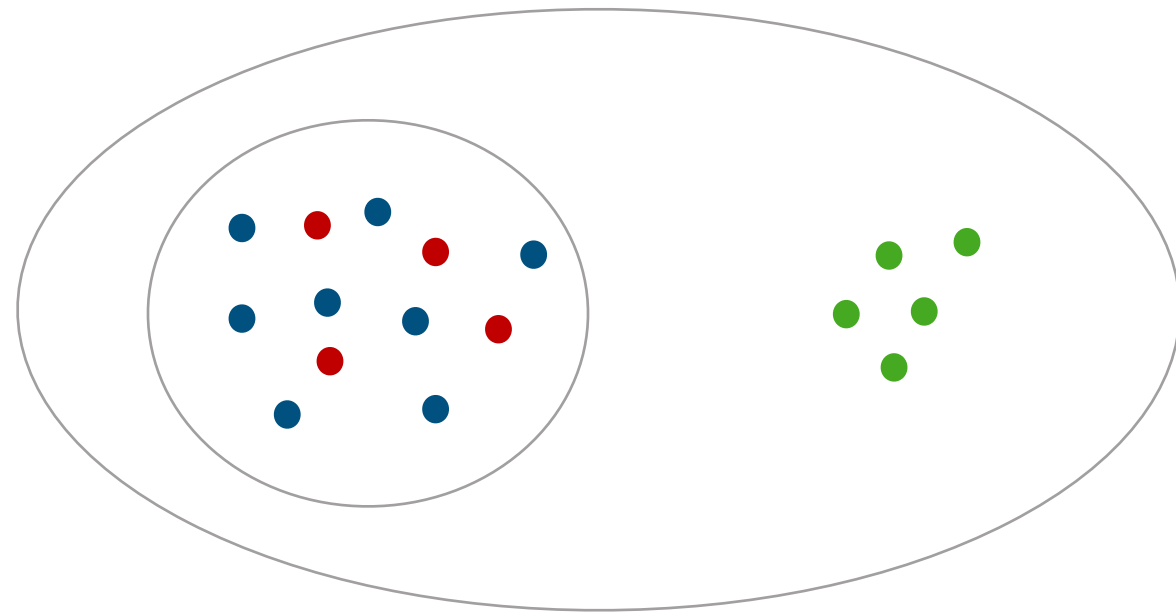
Structured arrangement of learnable “modules”

Outline

- Graph Neural Networks
- **To what kinds of instances will my model generalize?**
Prediction under distribution shifts
 - **stability**: measuring **data** shifts appropriately
 - **large shifts**: understanding **model** behavior by decomposition
- Beyond regression: **extending set functions** as **loss functions** for neural networks

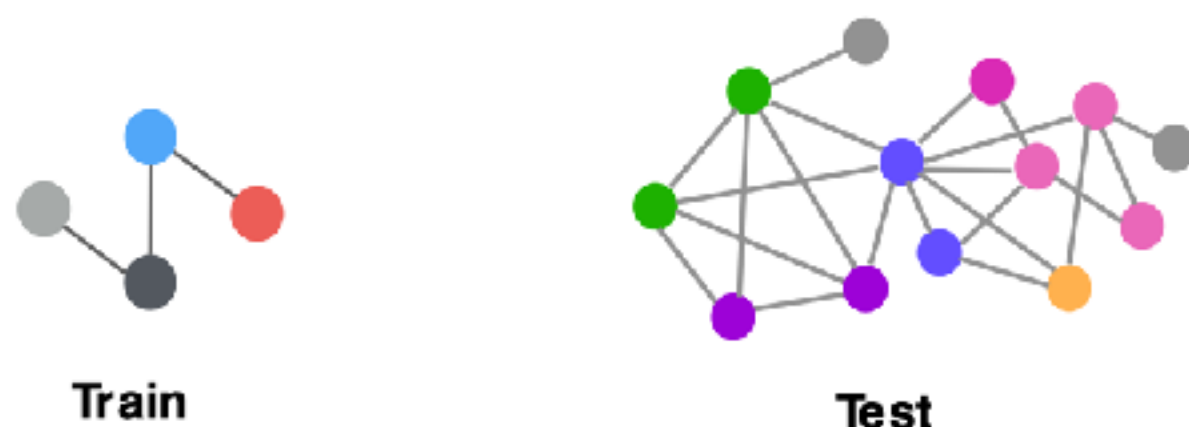


Graph predictions and distribution shifts



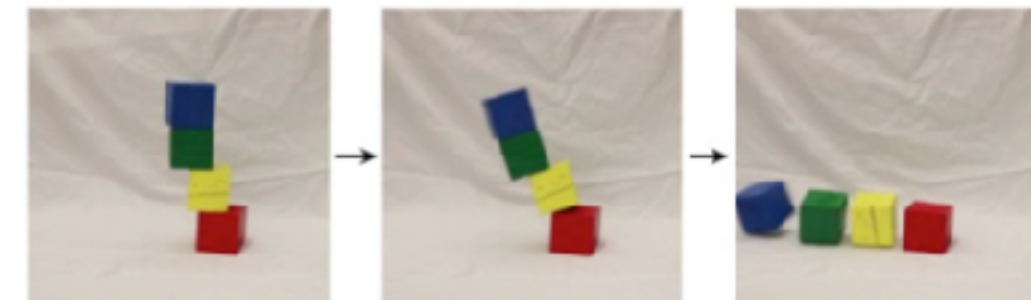
$$\mathbb{E}_{x \sim Q} [\ell(f_{\theta}(x), g(x))]$$

$\text{support}(Q) \supset \text{support}(\text{training dist})$



different graph size,
graph structure, edge weights, ...

(Battaglia et al 2018, Dai et al 2018, Velickovic et al 2020)



Physical reasoning
different position, mass, number of objects

Big picture: when may extrapolation “work”?

1) Data distributions in training and test are sufficiently similar

same distribution of computation trees (message passing GNNs) (*Yehudai-Fetaya-Meirom-Chechik-Maron 21*)

shared underlying structure (spectral GNNs) (*Levie et al 2019, Ruiz et al 2020*)

--- or ... ---

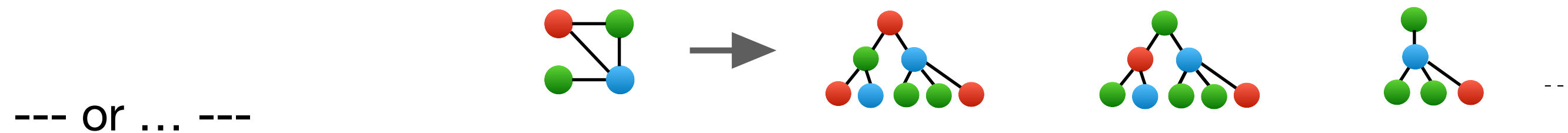


Big picture: when may extrapolation “work”?

1) Data distributions in training and test are sufficiently similar

same distribution of computation trees (message passing GNNs) (*Yehudai-Fetaya-Meirom-Chechik-Maron 21*)

shared underlying structure (spectral GNNs) (*Levie et al 2019, Ruiz et al 2020*)

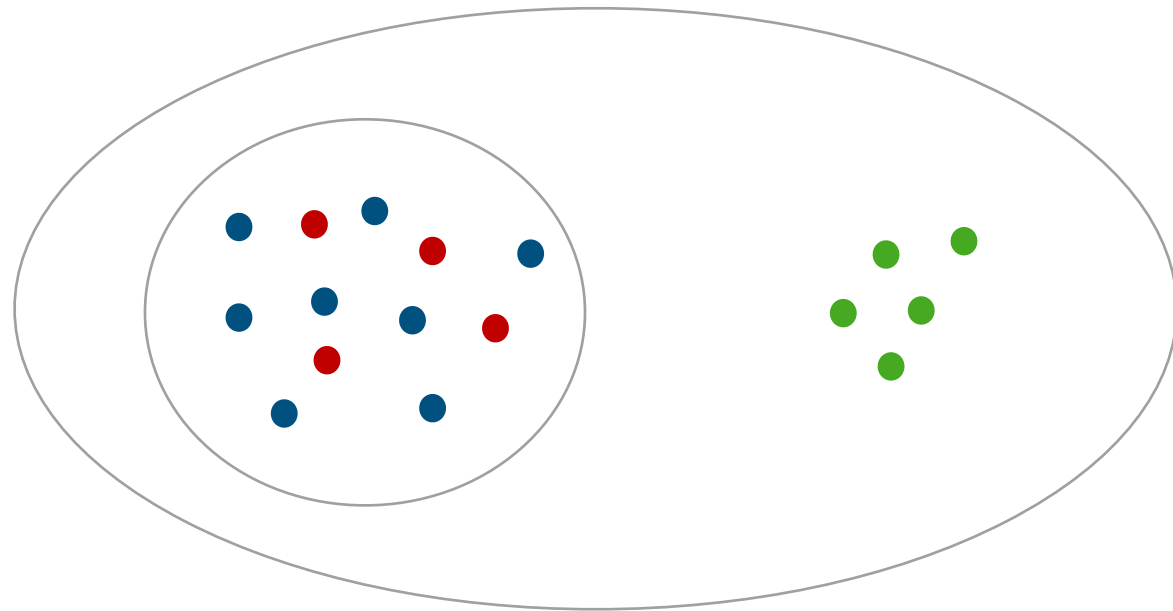


2) Understand what the model “learns”, and work around that: restrict the model via prior knowledge

(*Xu-Zhang-Li-Du-Kawarabayashi-Jegelka 21*)

Neural network structure, optimization algorithm, data geometry

Graph predictions and distribution shifts



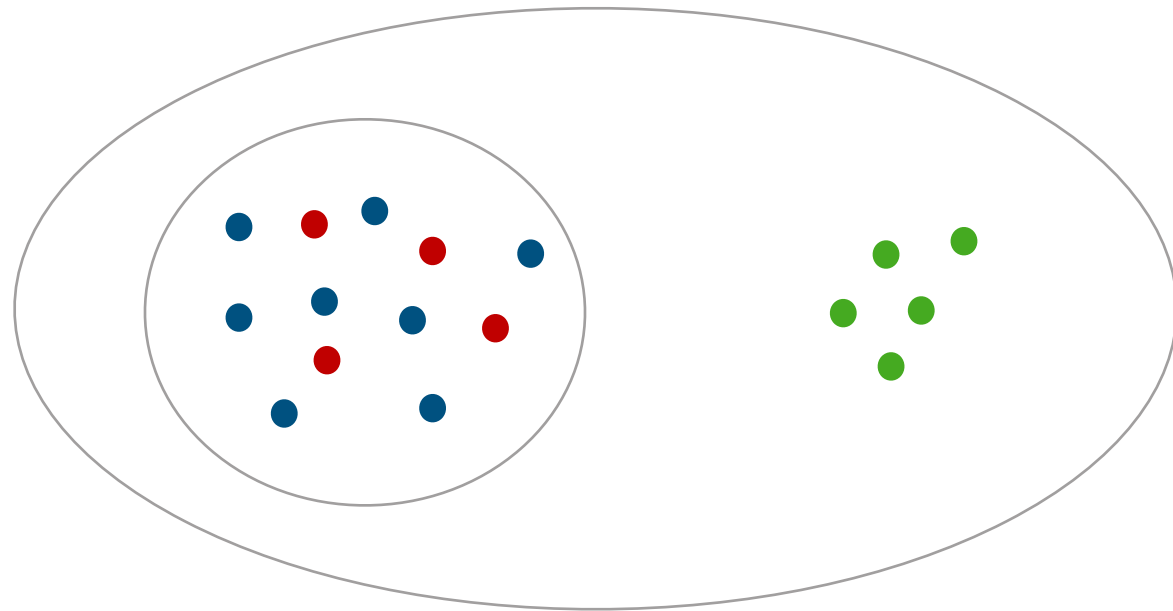
$$\mathbb{E}_{x \sim \mathcal{Q}} [\ell(f_{\theta}(x), g(x))]$$

$\text{support}(\mathcal{Q}) \supset \text{support}(\text{training dist})$

- **Worst-case scenario:** arbitrary predictions on unseen computation trees

Theorem (Yehudai et al 2021): Let \mathcal{P} and \mathcal{Q} be finitely supported distributions on graphs, and $\mathcal{P}^t, \mathcal{Q}^t$ the distribution of computation trees at depth t . If any graph in \mathcal{Q} contains a tree in $\mathcal{Q}^t \setminus \mathcal{P}^t$, then there is a GNN with depth at most $t + 3$ that perfectly solves the task on \mathcal{P} but has arbitrarily large error on all graphs from \mathcal{Q} .

Graph predictions and distribution shifts



$$\mathbb{E}_{x \sim Q} [\ell(f_\theta(x), g(x))]$$

$\text{support}(Q) \supset \text{support}(\text{training dist})$

- **Worst-case scenario:** arbitrary predictions on unseen computation trees

Theorem (Yehudai et al 2021): Let \mathcal{P} and \mathcal{Q} be finitely supported distributions on graphs, and $\mathcal{P}^t, \mathcal{Q}^t$ the distribution of computation trees at depth t . If any graph in \mathcal{Q} contains a tree in $\mathcal{Q}^t \setminus \mathcal{P}^t$, then there is a GNN with depth at most $t + 3$ that perfectly solves the task on \mathcal{P} but has arbitrarily large error on all graphs from \mathcal{Q} .

Smother degrading performance with appropriate metric on graphs?

Stability and measuring perturbations: Tree mover's distance

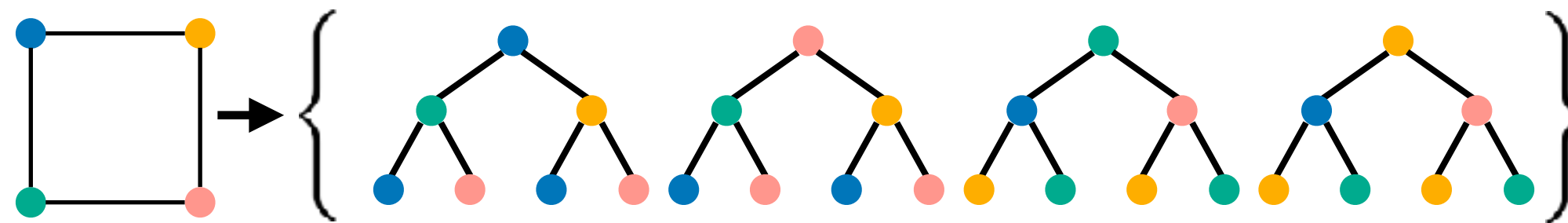
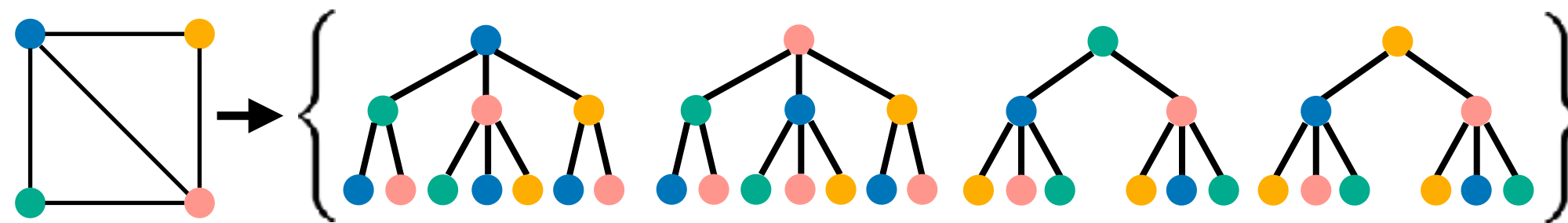
C. Chuang, S. Jegelka. Tree Mover's Distance: Bridging Graph Metrics and Stability of Graph Neural Networks. NeurIPS, 2022

An appropriate metric?

- Metric should capture Lipschitz/stability properties of GNNs, including invariances

An appropriate metric?

- Metric should capture Lipschitz/stability properties of GNNs, including invariances



- Message passing GNN compares sets of computation trees (subtree patterns)
- Idea: **Optimal transport distance**

Tree mover's distance

- Earth mover's distance between sets in Euclidean space:

$$\min_{\gamma \in \Gamma(X, Y)} \sum_{i, j=1}^{n, m} d(x_i, y_j) \cdot \gamma_{ij}$$

Distance between
vectors

$$\Gamma(X, Y) = \{ \gamma \in \mathbb{R}_+^{n \times m} \mid \gamma \mathbf{1}_m = \mathbf{1}_n; \gamma^\top \mathbf{1}_n = \mathbf{1}_m \}$$

Tree mover's distance

- Earth mover's distance between sets in Euclidean space:

$$\min_{\gamma \in \Gamma(X, Y)} \sum_{i, j=1}^{n, m} d(x_i, y_j) \cdot \gamma_{ij}$$

Distance between
vectors

$$\Gamma(X, Y) = \{ \gamma \in \mathbb{R}_+^{n \times m} \mid \gamma \mathbf{1}_m = \mathbf{1}_n; \gamma^\top \mathbf{1}_n = \mathbf{1}_m \}$$

- Tree mover's distance: x_i, x_j are trees.
Distance between trees?



Tree Distance via Hierarchical OT

$$\begin{aligned}
 \text{TD} \left(\begin{array}{c} \text{Tree Distance} \\ \text{Diagram 1} \end{array}, \begin{array}{c} \text{Diagram 2} \end{array} \right) &= \begin{array}{c} \text{Root Difference} \\ \|\bullet - \bullet\| \end{array} + \text{OT} \left(\begin{array}{c} \text{Subtree Difference} \\ \left\{ \begin{array}{c} \text{Diagram 3} \\ \text{Diagram 4} \end{array} \right\}, \left\{ \begin{array}{c} \text{Diagram 5} \\ \text{Blank Tree} \end{array} \right\} \end{array} \right) \\
 &= \begin{array}{c} \|\bullet - \bullet\| \end{array} + \text{TD} \left(\begin{array}{c} \text{Diagram 3} \\ \text{Diagram 4} \end{array}, \begin{array}{c} \text{Diagram 5} \end{array} \right) + \text{TD} \left(\begin{array}{c} \text{Diagram 4} \\ \text{Blank Tree} \end{array}, \begin{array}{c} \text{Blank Tree} \end{array} \right)
 \end{aligned}$$

$$\text{TD}_w(T_a, T_b) := \begin{cases} \|\mathbf{x}_{r_a} - \mathbf{x}_{r_b}\| + w(L) \cdot \text{OT}_{\text{TD}_w}(\rho(\mathcal{T}_{r_a}, \mathcal{T}_{r_b})) & \text{if } L > 1 \\ \|\mathbf{x}_{r_a} - \mathbf{x}_{r_b}\| & \text{otherwise,} \end{cases}$$

Recursive!

where $L = \max(\text{Depth}(T_a), \text{Depth}(T_b))$ and $w : \mathbb{N} \rightarrow \mathbb{R}^+$ is a depth-dependent weighting function.

Properties & implications of Tree mover's distance

- pseudo-metric: distinguishes the same graphs as the color refinement / Weisfeiler-Leman algorithm and GNNs, but graded: same “invariances”!

Properties & implications of Tree mover's distance

- pseudo-metric: distinguishes the same graphs as the color refinement / Weisfeiler-Leman algorithm and GNNs, but graded: same “invariances”!
- relation to **stability of GNNs**: Lipschitz constant of GNN (GIN)

$$\|h(G_a) - h(G_b)\| \leq \prod_{l=1}^{L+1} K_{\phi}^{(l)} \cdot \text{TMD}_w^{L+1}(G_a, G_b).$$

Properties & implications of Tree mover's distance

- pseudo-metric: distinguishes the same graphs as the color refinement / Weisfeiler-Leman algorithm and GNNs, but graded: same “invariances”!
- relation to **stability of GNNs**: Lipschitz constant of GNN (GIN)

$$\|h(G_a) - h(G_b)\| \leq \prod_{l=1}^{L+1} K_{\phi}^{(l)} \cdot \text{TMD}_w^{L+1}(G_a, G_b)$$

- use TMD in **cross-domain generalization** bound (*Shen et al, 2018*):

$$R_T(h) \leq R_S(h) + 2\text{Lip}(h) \cdot \mathcal{W}_1(p_S, p_T) + \text{small value}$$

Target Risk *Source Risk* *Domain Discrepancy*

Properties & implications of Tree mover's distance

- pseudo-metric: distinguishes the same graphs as the color refinement / Weisfeiler-Leman algorithm and GNNs, but graded: same “invariances”!
- relation to **stability of GNNs**: Lipschitz constant of GNN (GIN)

$$\|h(G_a) - h(G_b)\| \leq \prod_{l=1}^{L+1} K_{\phi}^{(l)} \cdot \text{TMD}_w^{L+1}(G_a, G_b)$$


- use TMD in **cross-domain generalization** bound (*Shen et al, 2018*):

$$R_T(h) \leq R_S(h) + 2\text{Lip}(h) \cdot \mathcal{W}_1(p_S, p_T) + \text{small value}$$

*Target
Risk*

*Source
Risk*

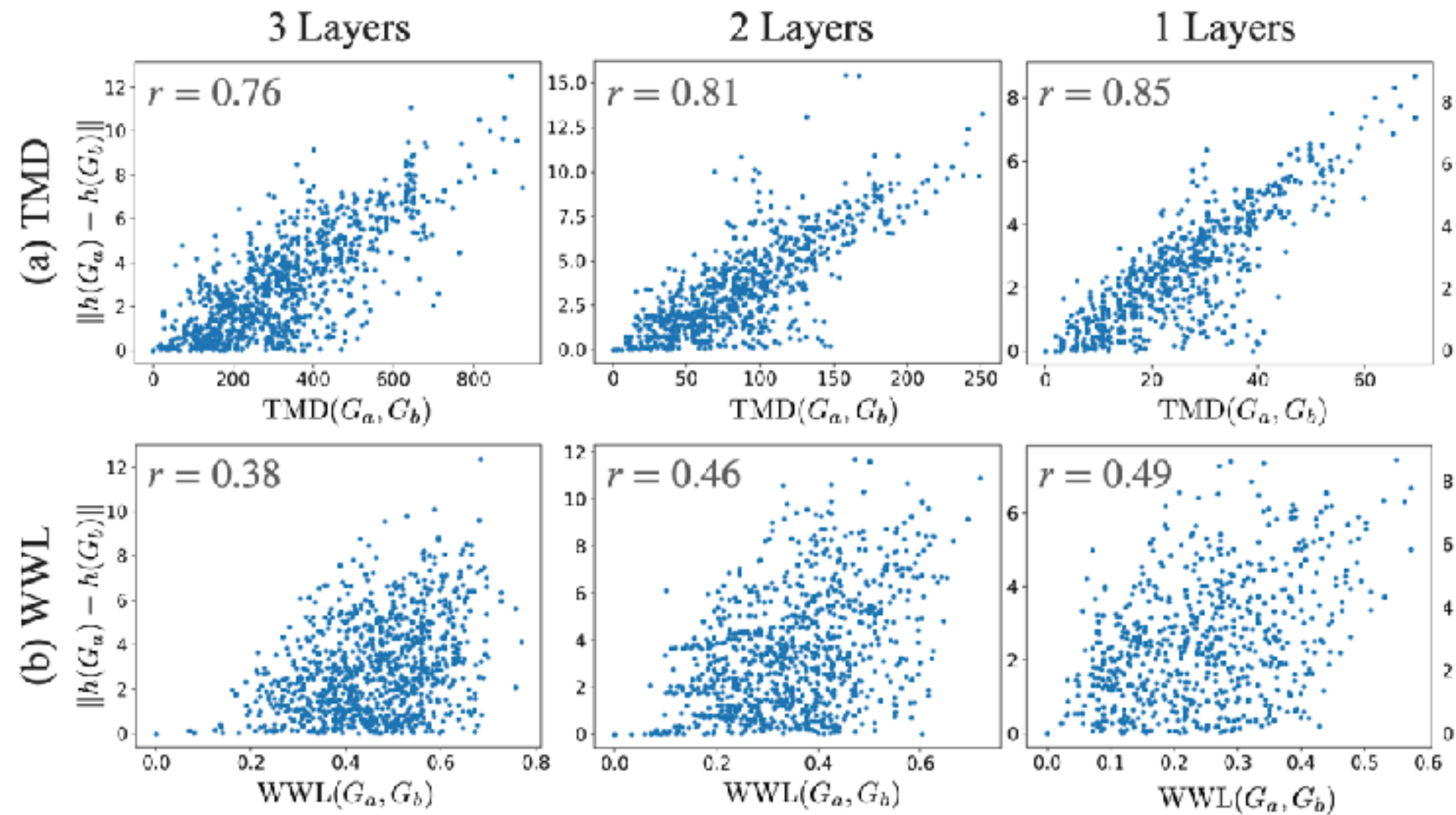
*Domain
Discrepancy*


$$\inf_{\pi \in \Pi(p_S, p_T)} \int \text{TMD}_w^{L+1}(G_a, G_b) d\pi(G_a, G_b)$$

Empirically

- comparison with Wasserstein Weisfeiler-Leman metric (*Togninalli et al 2019*)

1. **stability:**

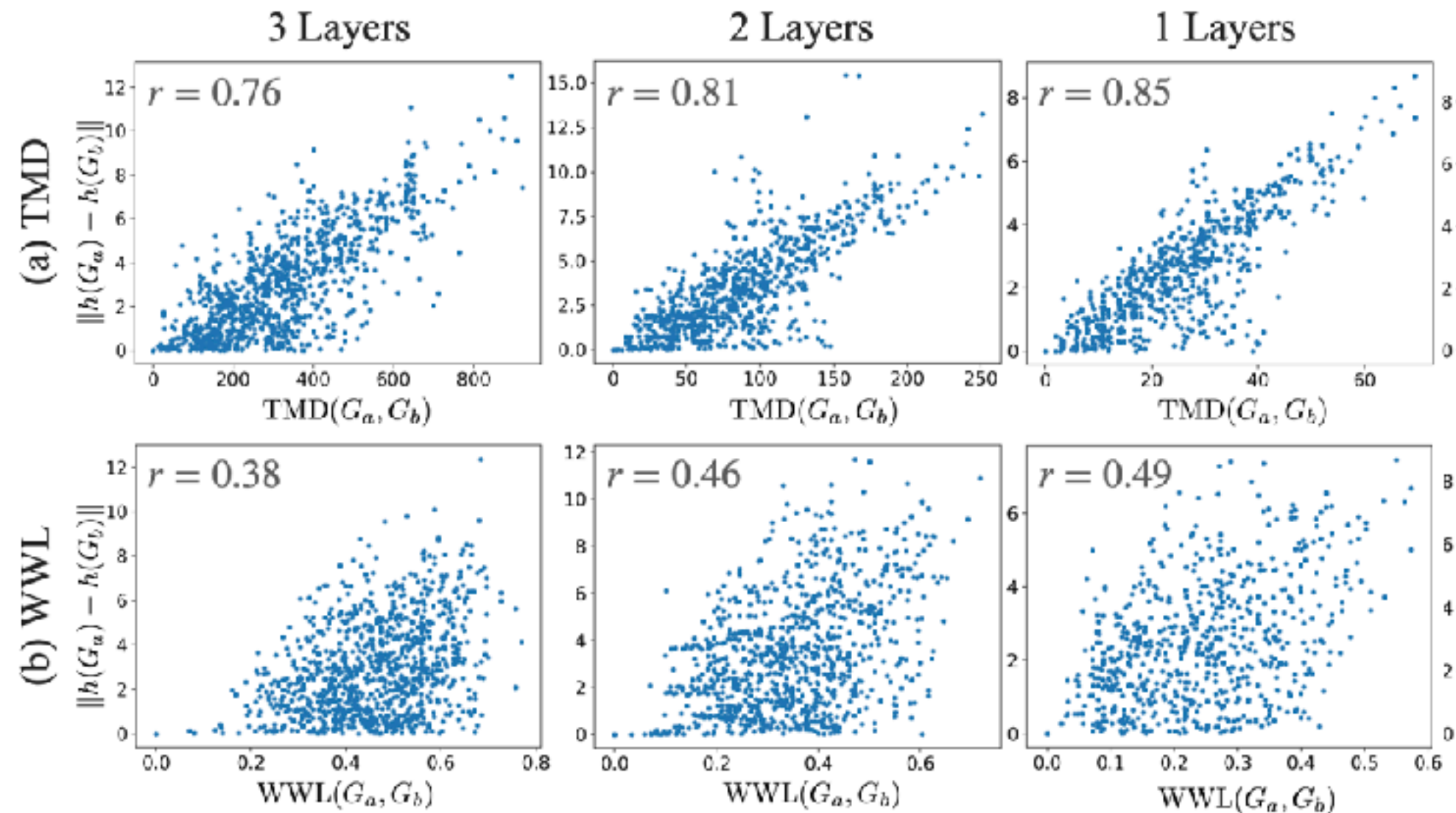


MUTAG data,
randomly sampled pairs

Empirically

- comparison with Wasserstein Weisfeiler-Leman metric (*Togninalli et al 2019*)

1. **stability:**



MUTAG data,
randomly sampled pairs

- ## 2. correlation with **accuracy drops under domain shifts** (PTC data):
- WWL: 0.489**
TMD: 0.712

Large perturbations: inductive biases

K. Xu, J. Li, M. Zhang, S. Du, K. Kawarabayashi, S. Jegelka. How Neural Networks Extrapolate: From Feedforward to Graph Neural Networks. ICLR, 2021.

Generalization to very different data

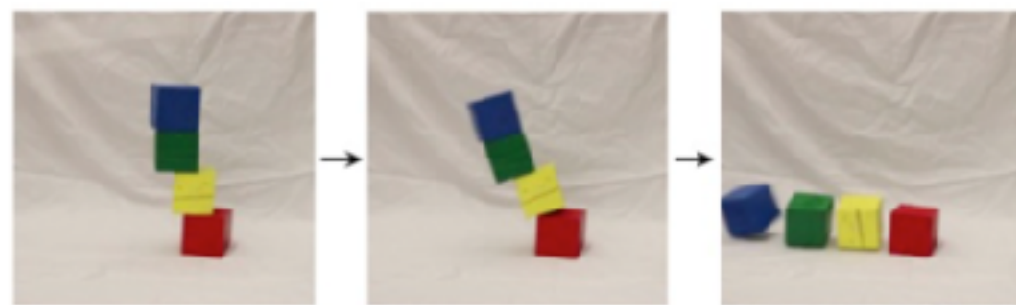


Train

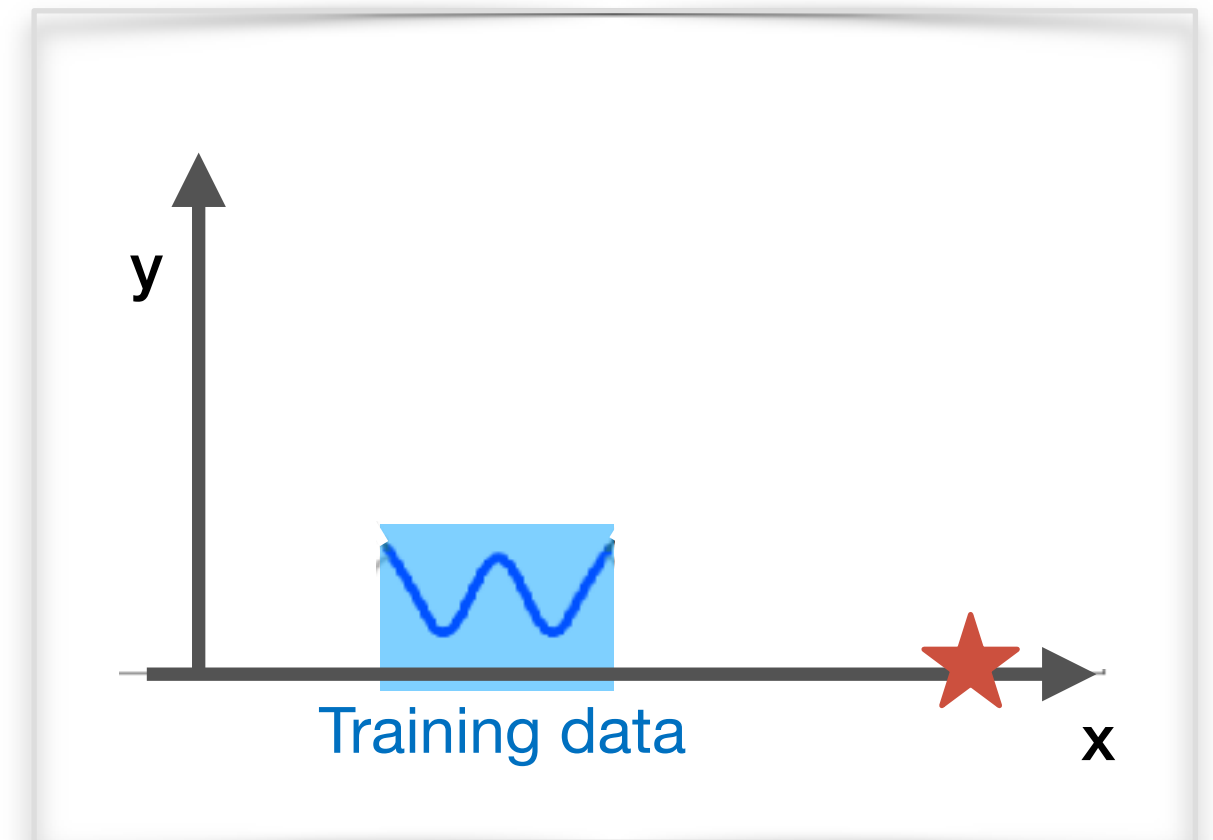


Test

different graph size,
graph structure, edge weights, ...
(Battaglia et al 2018, Dai et al 2018, Velickovic et al 2020)



Physical reasoning
different position, mass, number of objects

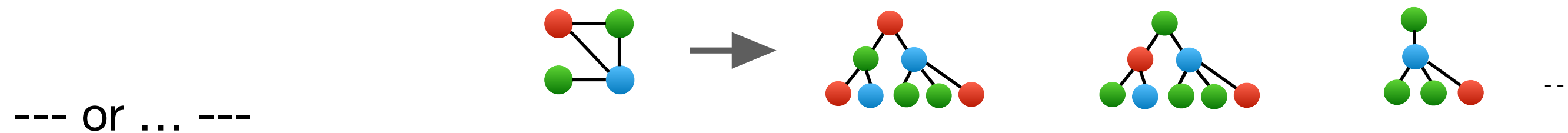


Big picture: when may extrapolation “work”?

1) Data distributions in training and test are sufficiently similar

same distribution of computation trees (message passing GNNs) (*Yehudai-Fetaya-Meirom-Chechik-Maron 21*)

shared underlying structure (spectral GNNs) (*Levie et al 2019, Ruiz et al 2020*)



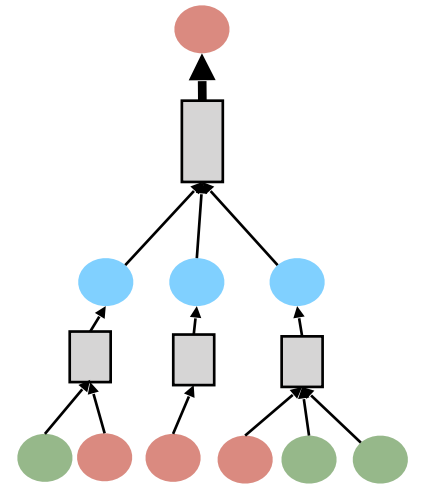
2) Understand what the model “learns”, and work around that: restrict the model via prior knowledge

(*Xu-Zhang-Li-Du-Kawarabayashi-Jegelka 21*)

Neural network structure, optimization algorithm, data geometry

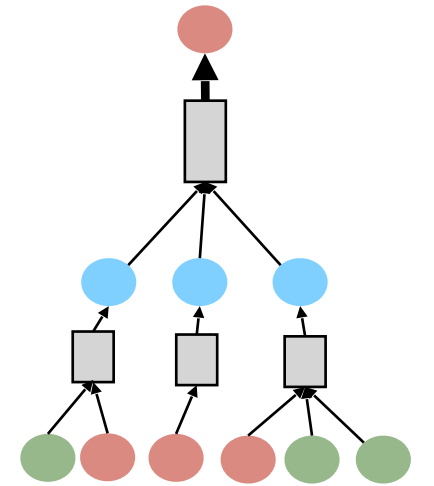
Neural network architecture and algorithms

- Algorithm = structured arrangement of subroutines
- (Graph) Neural network = structured arrangement of learnable “modules”



Neural network architecture and algorithms

- Algorithm = structured arrangement of subroutines
- (Graph) Neural network = structured arrangement of learnable “modules”



Bellman-Ford

for $k = 1 \dots |S| - 1$:

for u in S :

$$d[k][u] = \min_v d[k-1][v] + \text{cost}(v, u)$$

GNN

for $k = 1 \dots \text{GNN iter}$:

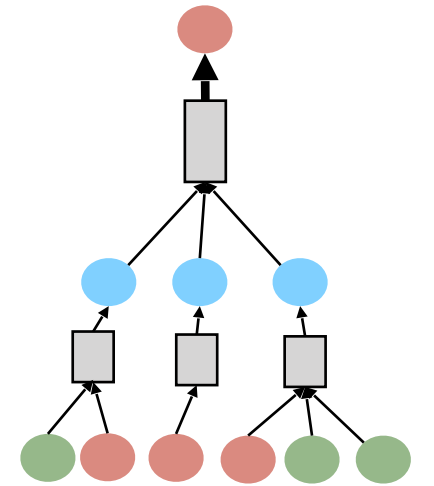
for u in S :

$$h_u^{(k)} = \Sigma_v \text{MLP}(h_v^{(k-1)}, h_u^{(k-1)})$$



Neural network architecture and algorithms

- Algorithm = structured arrangement of subroutines
- (Graph) Neural network = structured arrangement of learnable “modules”



Bellman-Ford

for $k = 1 \dots |S| - 1$:

for u in S :

$$d[k][u] = \min_v d[k-1][v] + \text{cost}(v, u)$$

GNN

for $k = 1 \dots \text{GNN iter}$:

for u in S :

$$h_u^{(k)} = \sum_v \text{MLP}(h_v^{(k-1)}, h_u^{(k-1)})$$

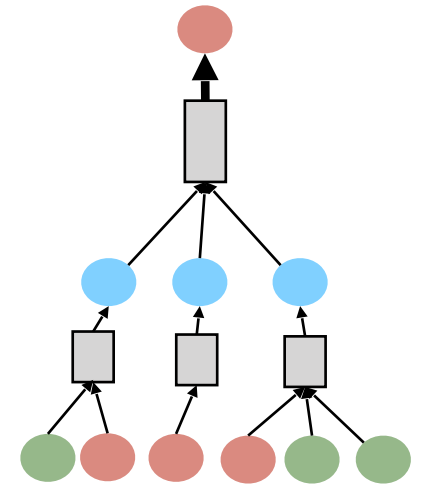


Algorithmic Alignment:

Neural Network can mimic algorithm via *few, easy-to-learn* “modules”

Neural network architecture and algorithms

- Algorithm = structured arrangement of subroutines
- (Graph) Neural network = structured arrangement of learnable “modules”



Bellman-Ford

for $k = 1 \dots |S| - 1$:

for u in S :

$$d[k][u] = \min_v d[k-1][v] + \text{cost}(v, u)$$

GNN

for $k = 1 \dots \text{GNN iter}$:

for u in S :

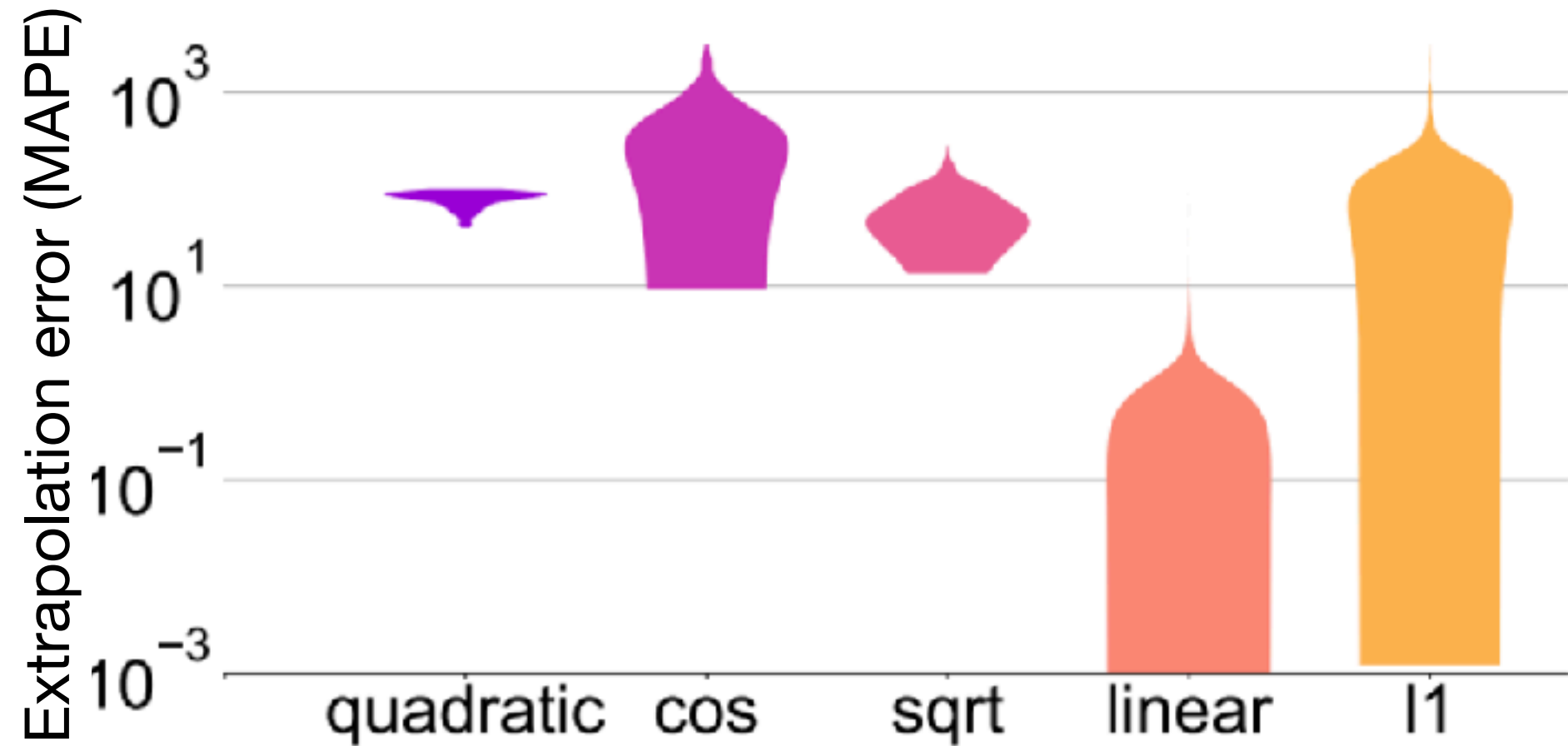
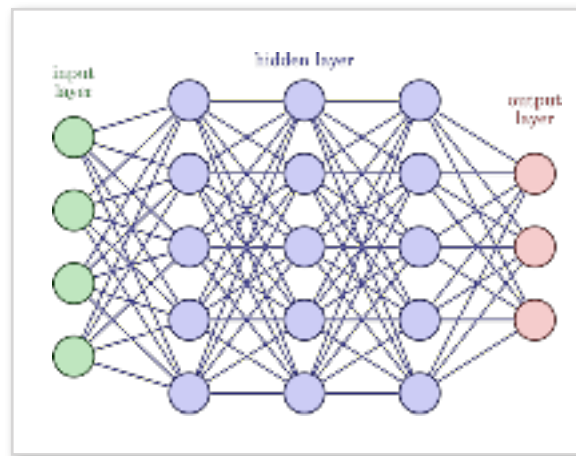
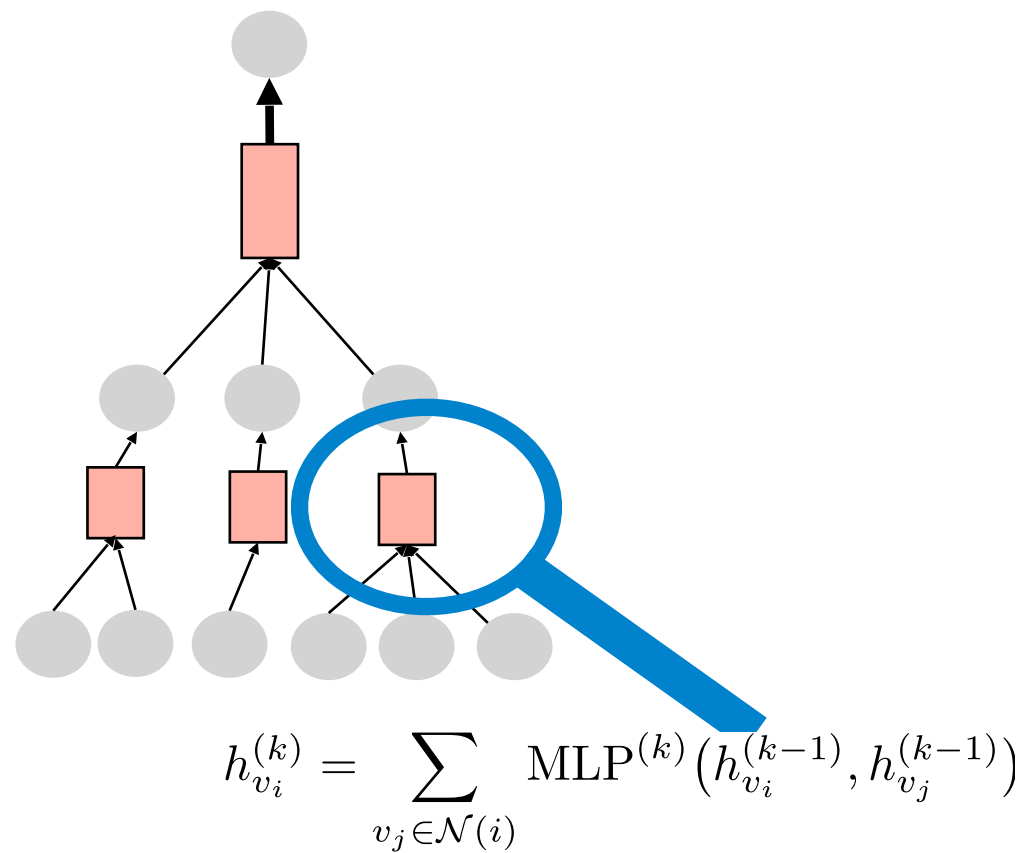
$$h_u^{(k)} = \sum_v \text{MLP}(v^{(k-1)}, h_u^{(k-1)})$$



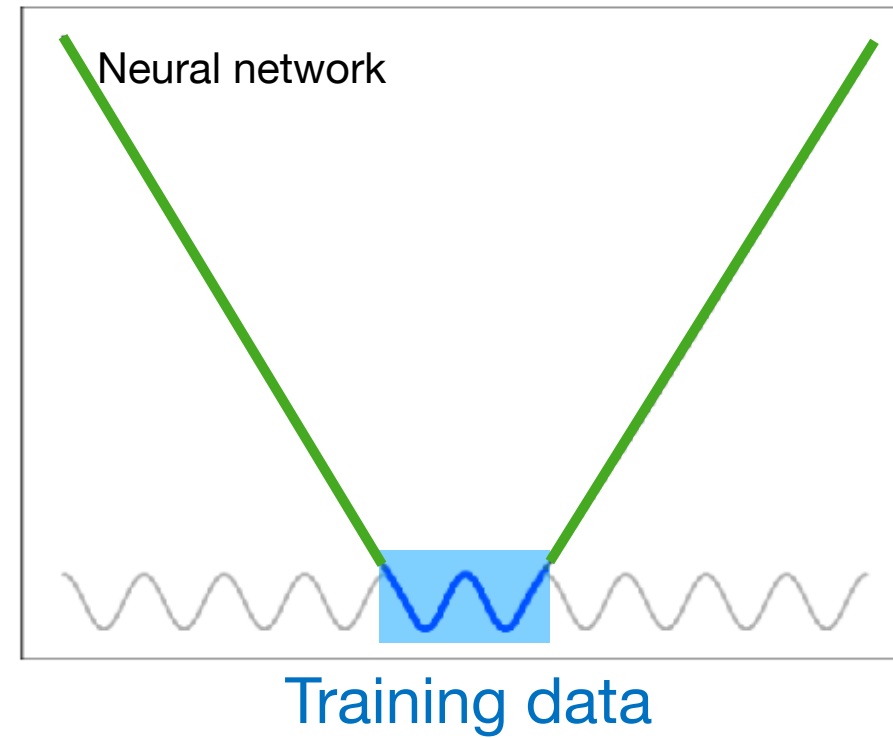
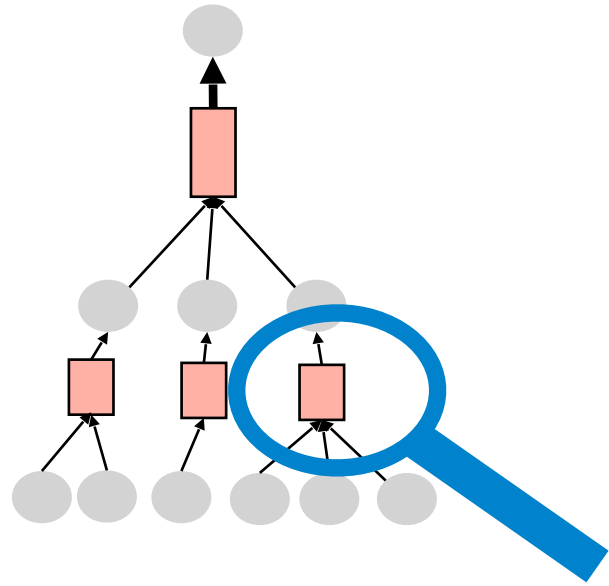
Algorithmic Alignment:

Neural Network can mimic algorithm via *few, easy-to-learn* “modules”

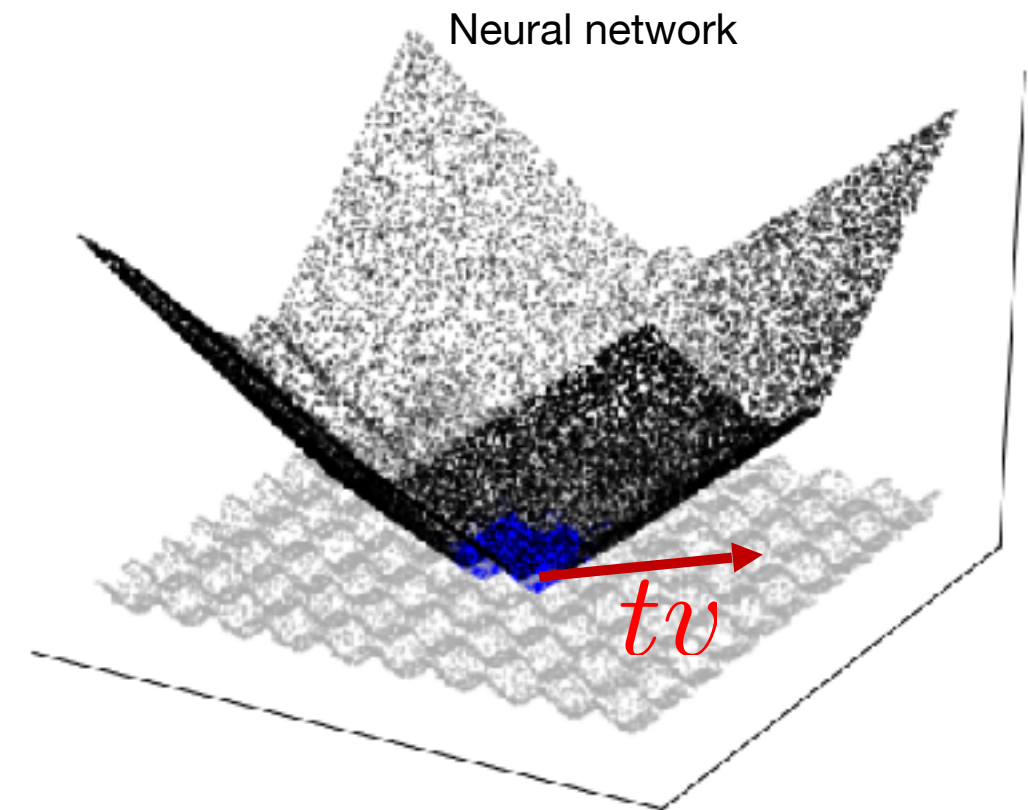
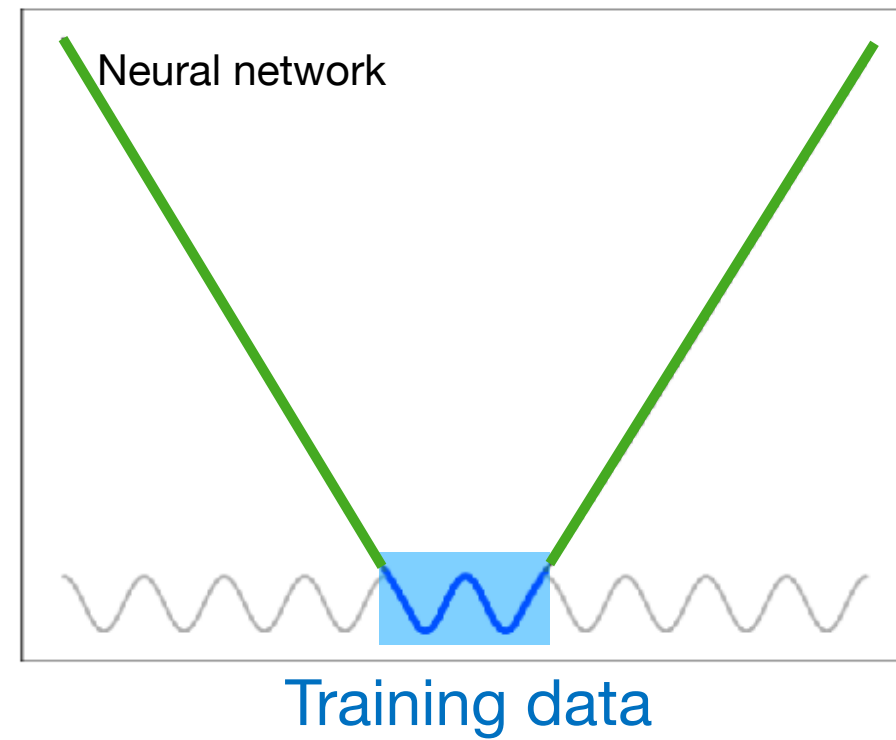
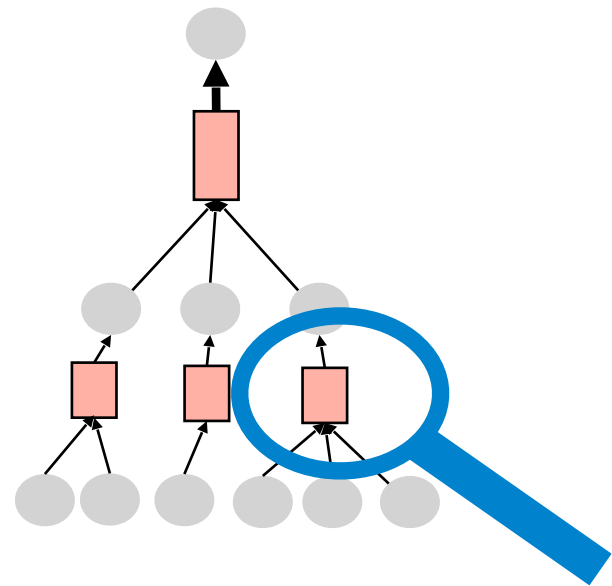
Empirically: MLP with ReLU activations



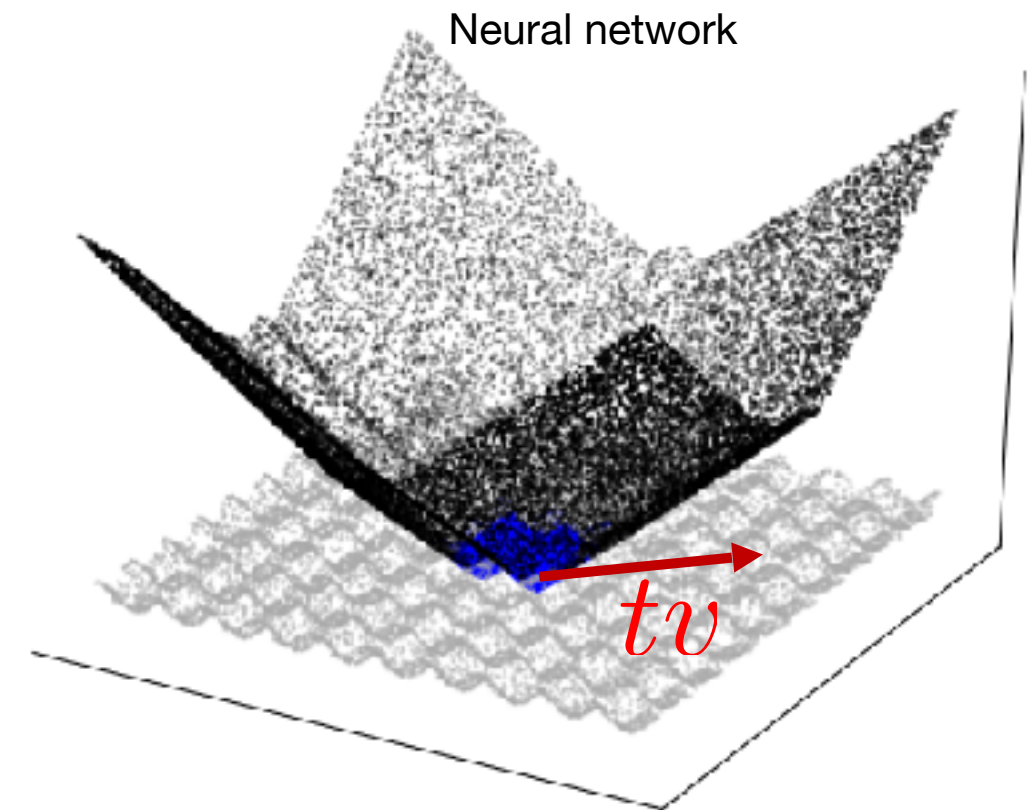
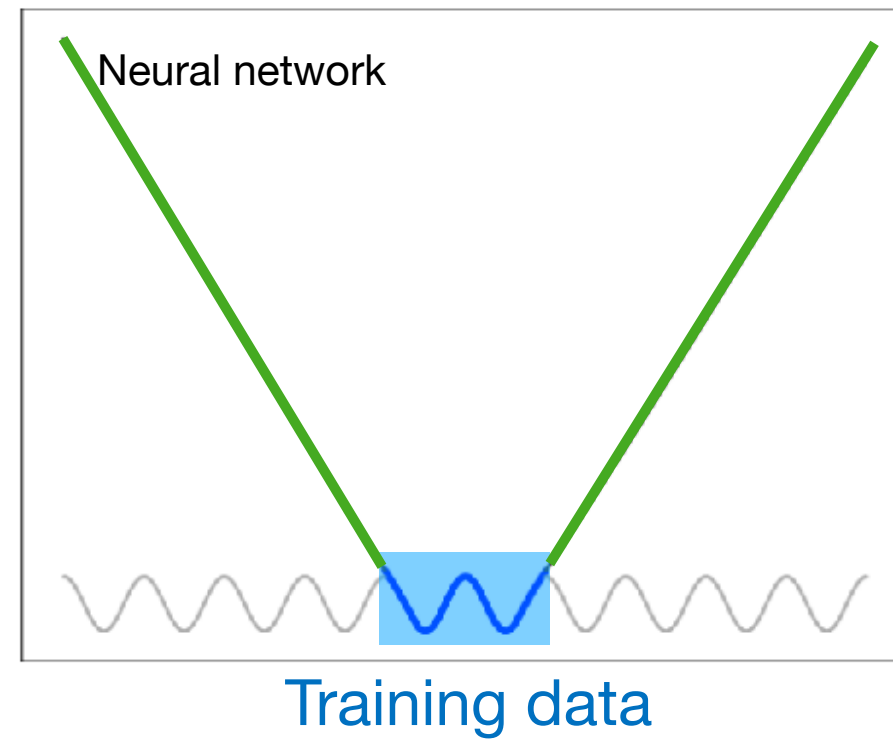
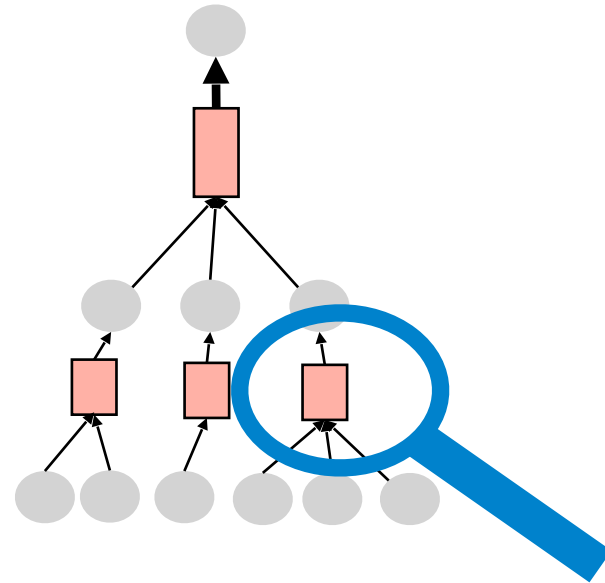
Extrapolation in fully connected ReLU networks



Extrapolation in fully connected ReLU networks



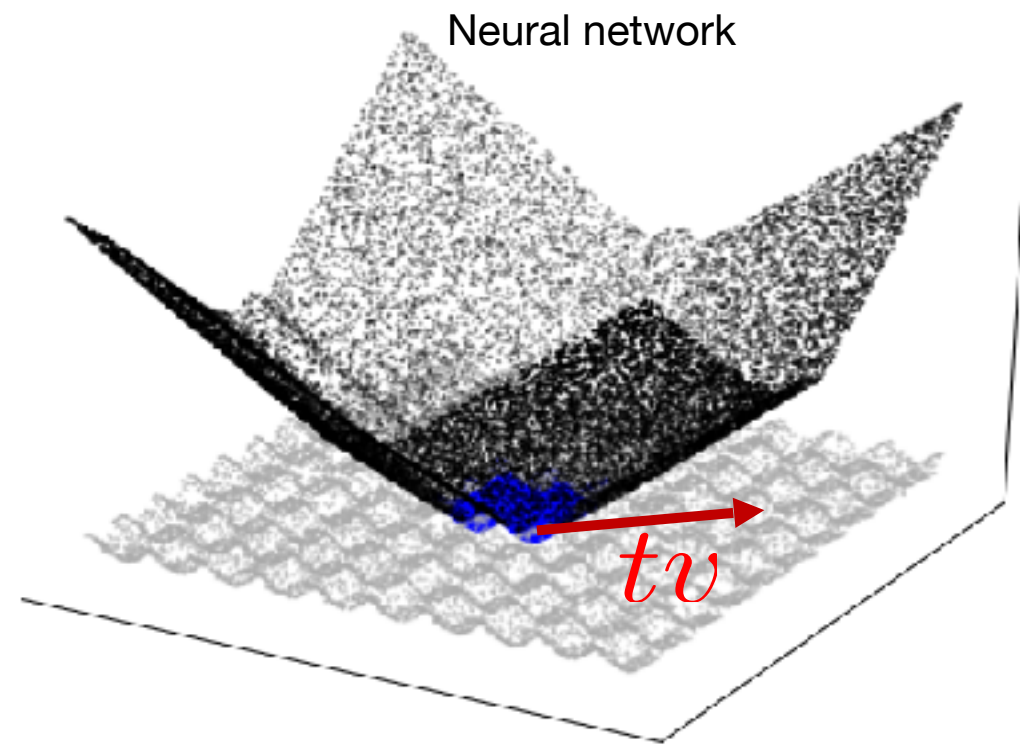
Extrapolation in fully connected ReLU networks



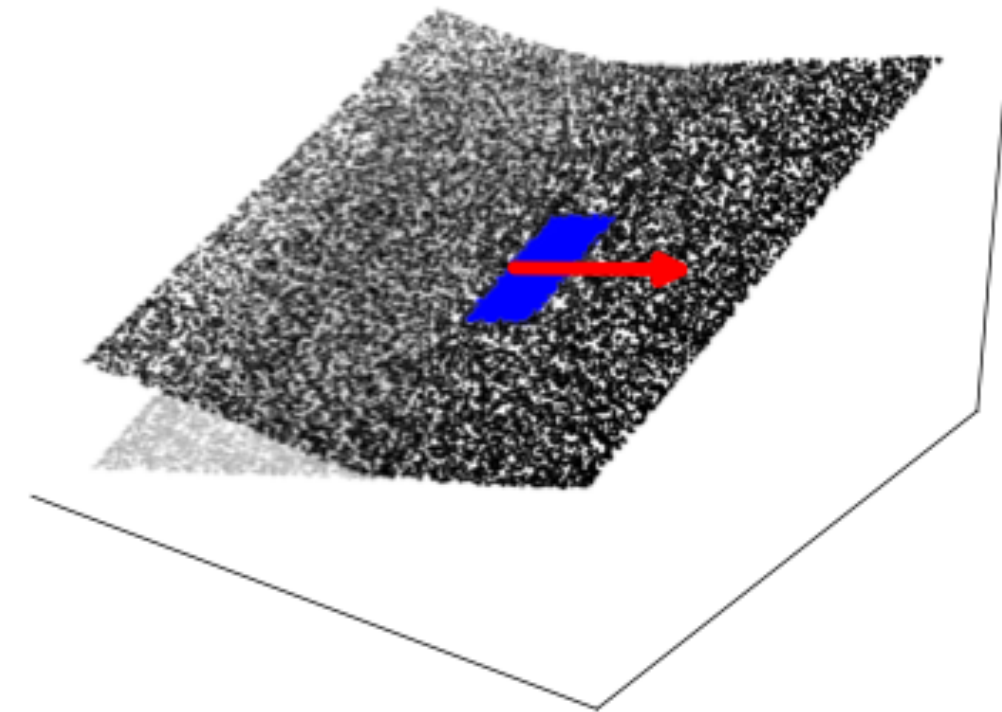
Theorem (Xu-Zhang-Li-Du-Kawarabayashi-J 21)

Let f be a 2-layer ReLU MLP trained with Gradient Descent. **Along any direction** $v \in \mathbb{R}^d$ **f approaches a linear function**: let $x = tv$. As $t \rightarrow \infty$: $f(x + hv) - f(x) \rightarrow \beta_v h$ with rate $O(1/t)$.

Implications

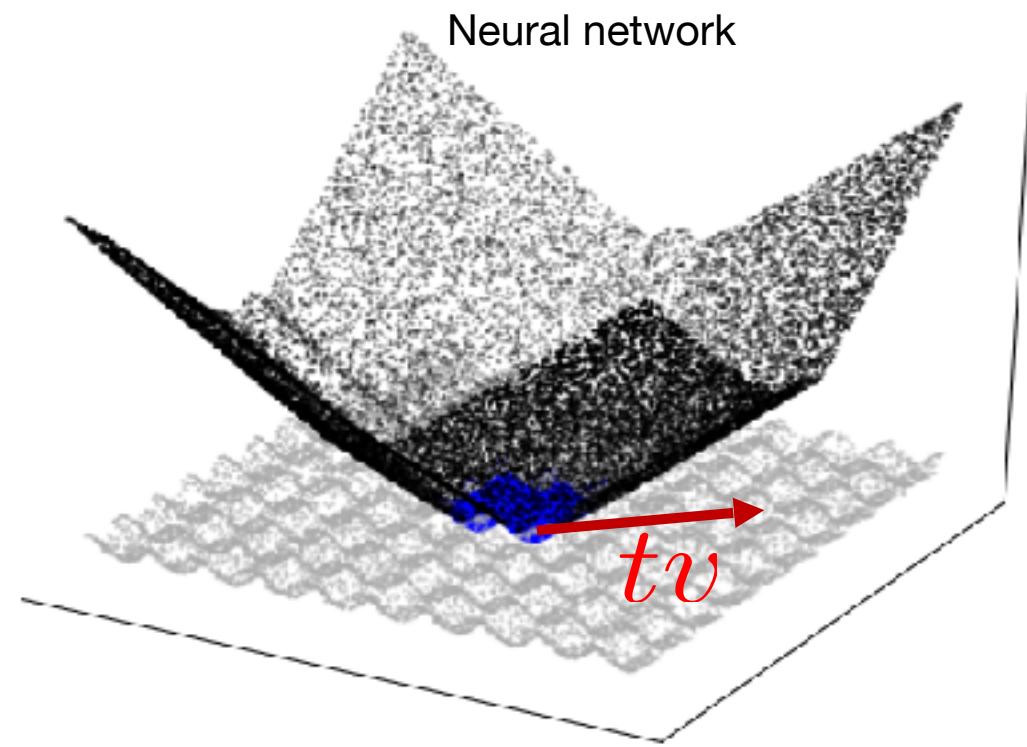


1. Can only extrapolate linear functions

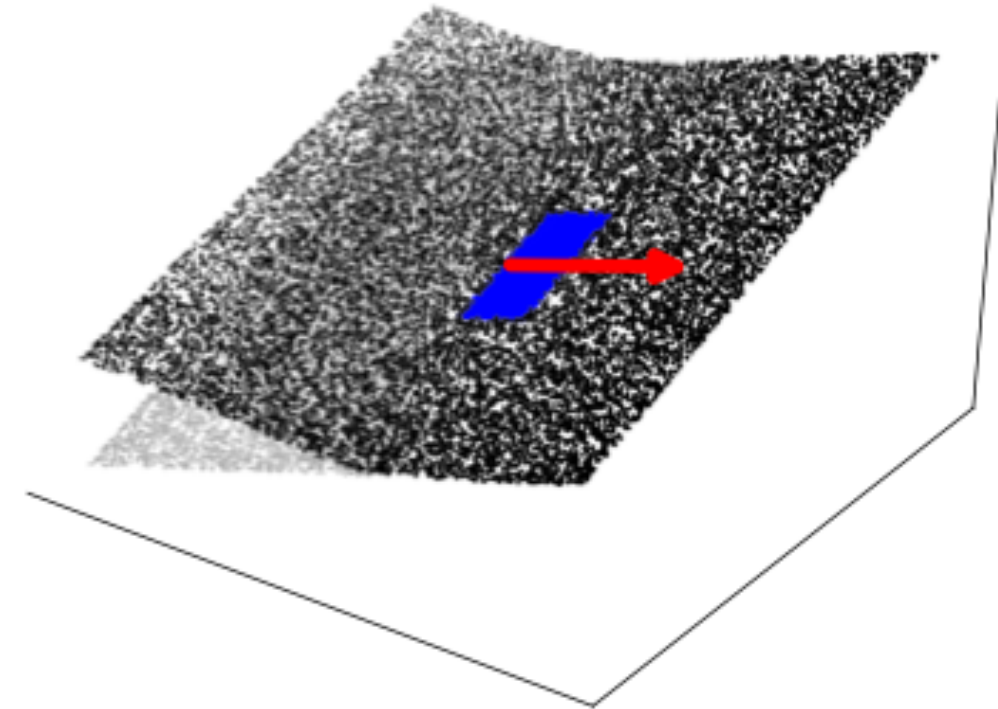


2. Training Data geometry

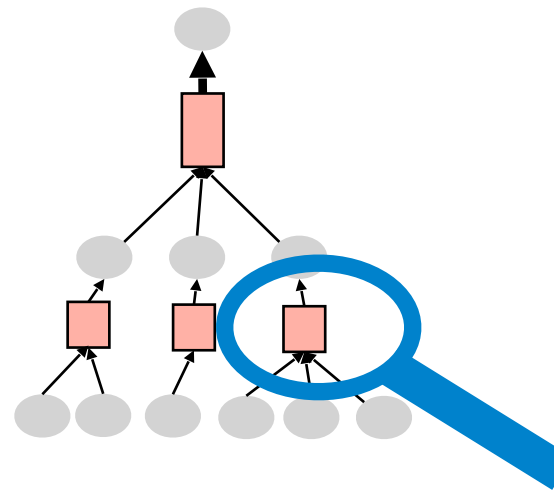
Implications



1. Can only extrapolate linear functions



2. Training Data geometry



Implications for the full GNN

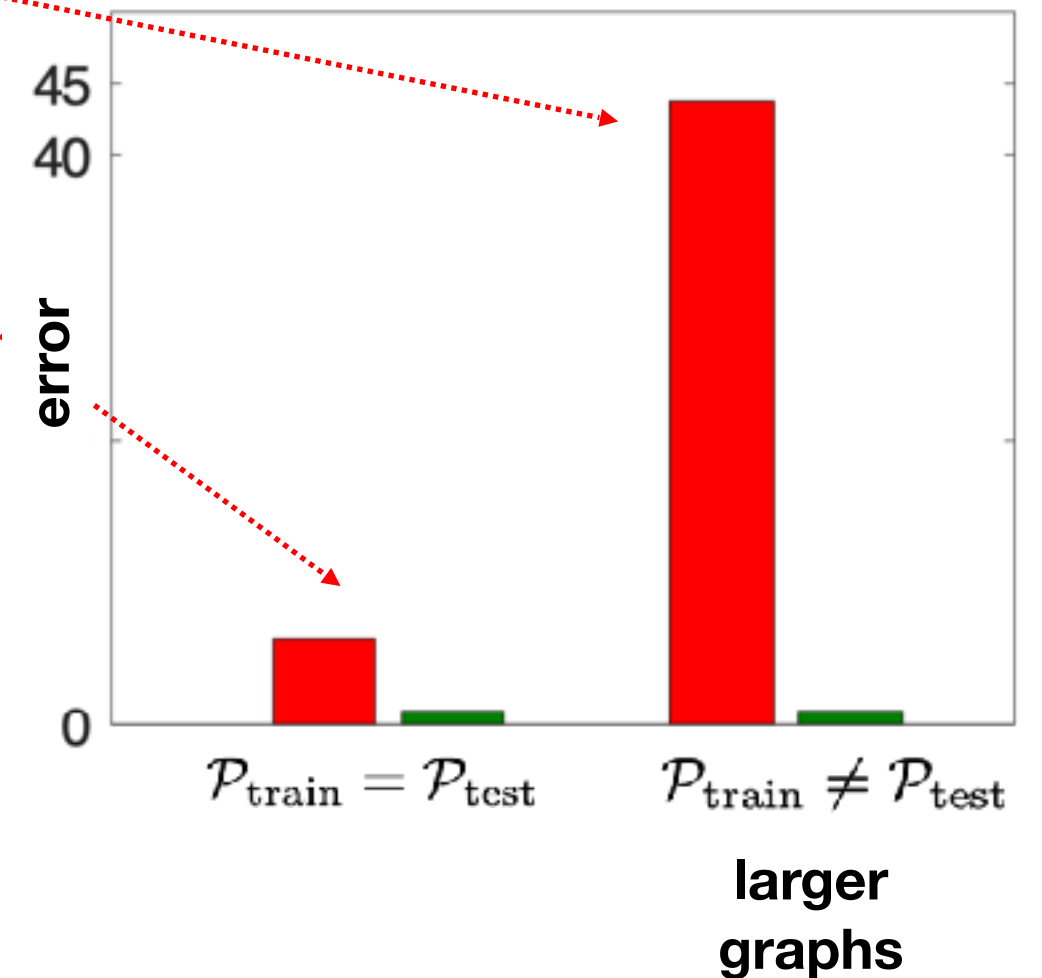
Shortest Path: $\text{dist}[k][v] = \min_{u \in \mathcal{N}(v)} \text{dist}[k-1][u] + w(u, v)$
(target)

GNN: $h_v^{(k)} = \sum_{u \in \mathcal{N}(v)} \text{MLP}(h_u^{(k-1)}, h_v^{(k-1)}, w(u, v))$ **Need MLP to be nonlinear!**

Implications for the full GNN

Shortest Path: $\text{dist}[k][v] = \min_{u \in \mathcal{N}(v)} \text{dist}[k-1][u] + w(u, v)$
(target)

GNN: $h_v^{(k)} = \sum_{u \in \mathcal{N}(v)} \text{MLP}(h_u^{(k-1)}, h_v^{(k-1)}, w(u, v))$ **Need MLP to be nonlinear!**

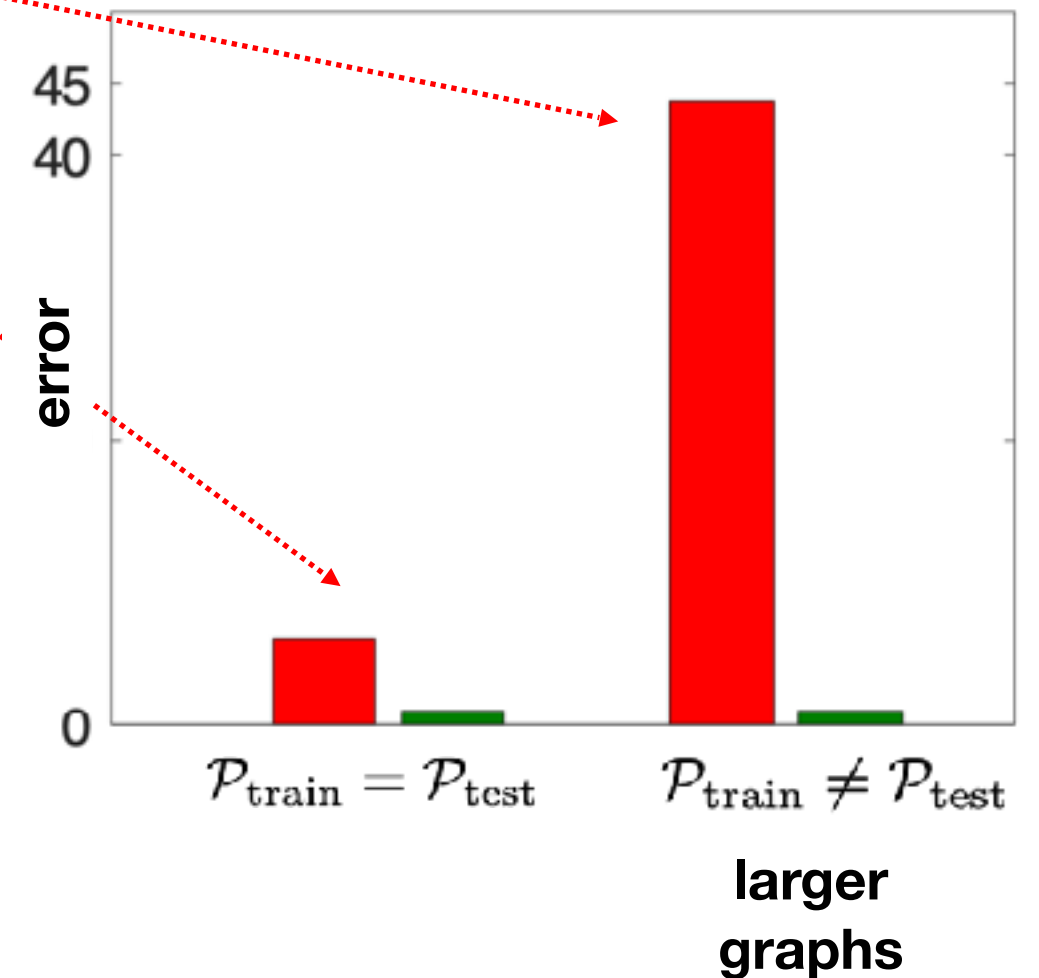


Implications for the full GNN

Shortest Path: $\text{dist}[k][v] = \min_{u \in \mathcal{N}(v)} \text{dist}[k-1][u] + w(u, v)$
(target)

GNN: $h_v^{(k)} = \sum_{u \in \mathcal{N}(v)} \text{MLP}(h_u^{(k-1)}, h_v^{(k-1)}, w(u, v))$ **Need MLP to be nonlinear!**

GNN II: $h_v^{(k)} = \max_{u \in \mathcal{N}(v)} \text{MLP}(h_u^{(k-1)}, h_v^{(k-1)}, w$
(Veličković et al 2020)



Implications for the full GNN

Shortest Path: $\text{dist}[k][v] = \min_{u \in \mathcal{N}(v)} \text{dist}[k-1][u] + w(u, v)$
(target)

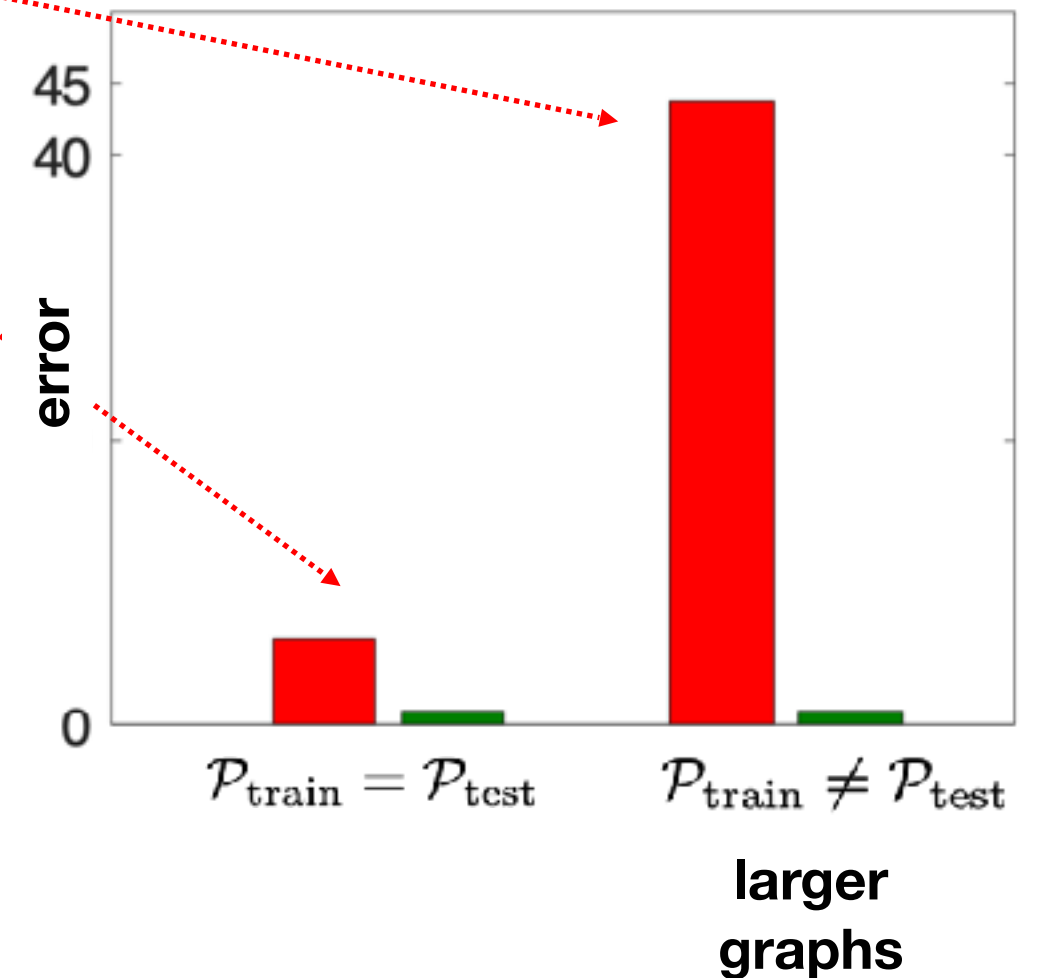
GNN: $h_v^{(k)} = \sum_{u \in \mathcal{N}(v)} \text{MLP}(h_u^{(k-1)}, h_v^{(k-1)}, w(u, v))$

Need MLP to be nonlinear!

GNN II: $h_v^{(k)} = \max_{u \in \mathcal{N}(v)} \text{MLP}(h_u^{(k-1)}, h_v^{(k-1)}, w)$
(Veličkovic et al 2020)

Task-specific nonlinearities help extrapolation.
Empirically reflected in many works

*(Trask et al 2018, Johnson et al 2017, Yi et al 2018,
Mao et al 2019, Cranmer et al 2019, 2020, Veličkovic et al 2020 ...)*



Encode nonlinearities in the ...

... architecture

$$\text{NALU: } \mathbf{y} = \mathbf{g} \odot \mathbf{a} + (1 - \mathbf{g}) \odot \mathbf{m}$$

$$\mathbf{m} = \exp \mathbf{W}(\log(|\mathbf{x}| + \epsilon)), \mathbf{g} = \sigma(\mathbf{G}\mathbf{x})$$

Exp log for learning multiplication

(Trask et al 2018)

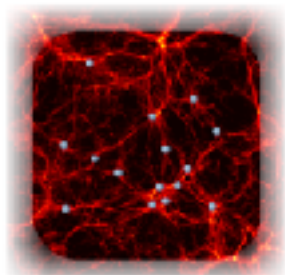


Q: What direction is the closest creature facing?
P: none, filter_creature, filter_closest, unique, query_direction

A: left

Library of programs

(Johnson et al 2017, Yi et al 2018, Mao et al 2019, ...)



Learning physics laws

(Cranmer et al 2019, 2020)

... input representation

Encode nonlinearities in the ...

... architecture

$$\text{NALU: } \mathbf{y} = \mathbf{g} \odot \mathbf{a} + (1 - \mathbf{g}) \odot \mathbf{m}$$

$$\mathbf{m} = \exp \mathbf{W}(\log(|\mathbf{x}| + \epsilon)), \mathbf{g} = \sigma(\mathbf{G}\mathbf{x})$$

Exp log for learning multiplication

(Trask et al 2018)

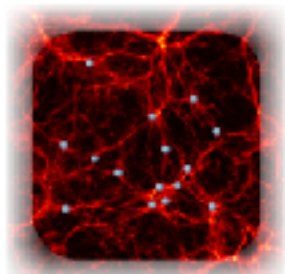


Library of programs

(Johnson et al 2017,
Yi et al 2018,
Mao et al 2019, ...)

Q: What direction is the closest creature facing?
P: none, filter_creature, filter_closest, unique, query_direction

A: left

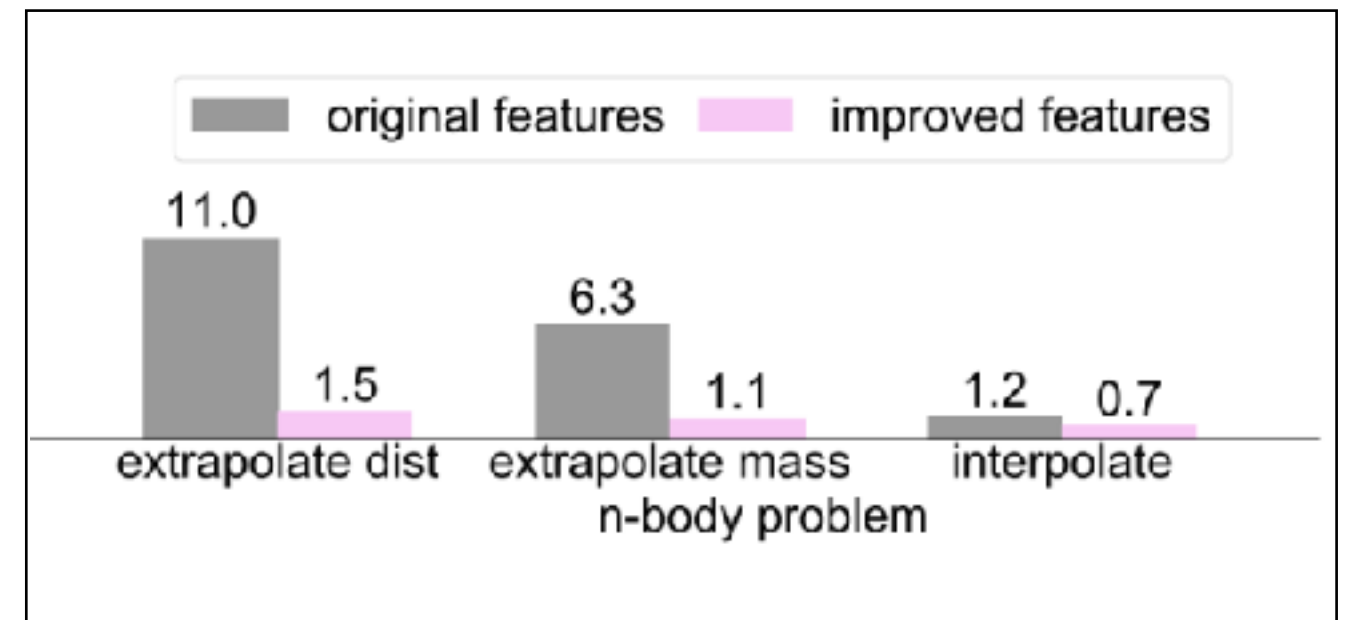


Learning physics laws

(Cranmer et al 2019,2020)



... input representation



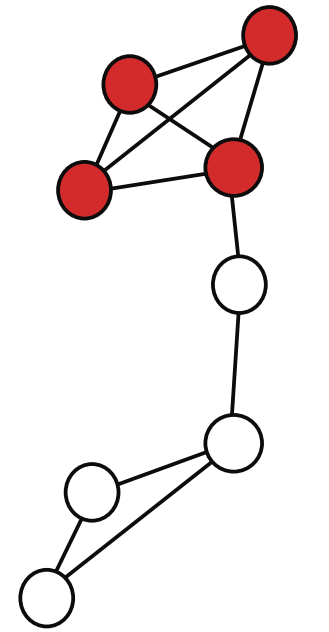
Prior knowledge or *representation learning*

Neural Network Losses from Set Function Extensions

N. Karalias, J. Robinson, A. Loukas, S. Jegelka. Neural set function extensions: learning with discrete functions in high dimensions. NeurIPS, 2022

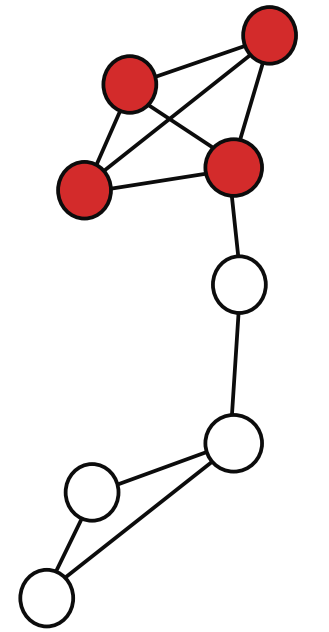
Setup

- use NN as “solver”, and objective function of the optimization problem as a loss



Setup

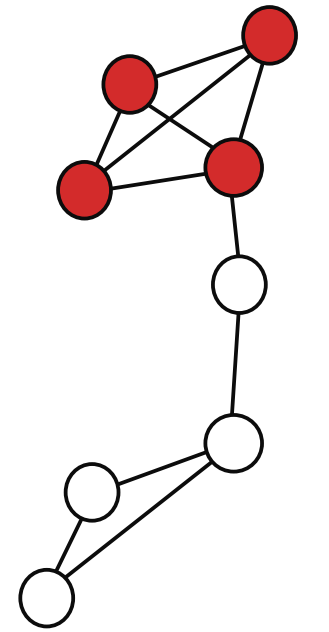
- use NN as “solver”, and objective function of the optimization problem as a loss
- What if the objective is a set function? $F(S), S \subseteq [n]$



Setup

- use NN as “solver”, and objective function of the optimization problem as a loss
- What if the objective is a set function? $F(S), S \subseteq [n]$
- Continuous extension:

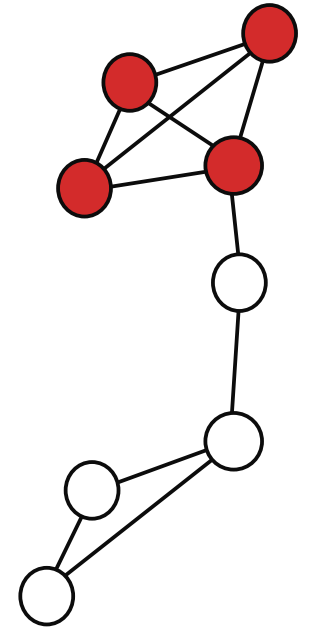
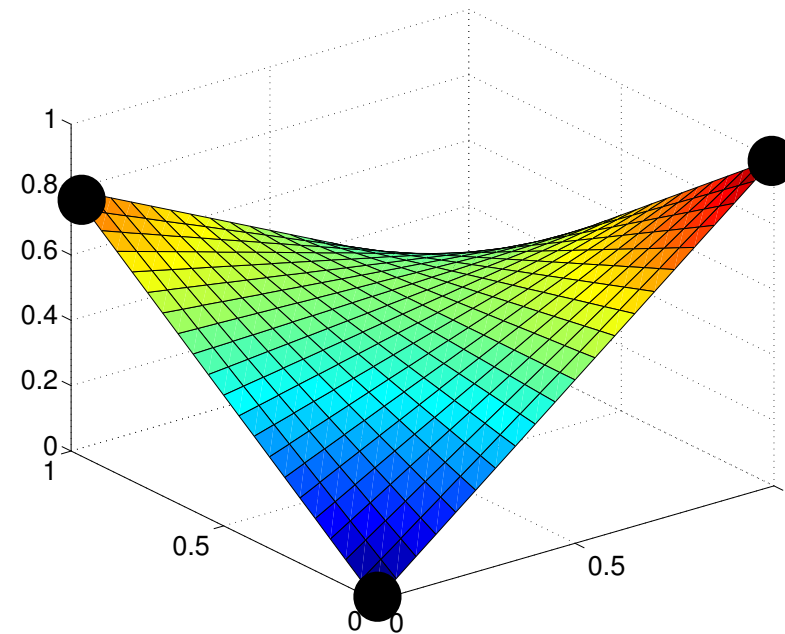
$$F : \{0, 1\}^n \rightarrow \mathbb{R} \quad \longrightarrow \quad f : [0, 1]^n \rightarrow \mathbb{R}$$



Setup

- use NN as “solver”, and objective function of the optimization problem as a loss
- What if the objective is a set function? $F(S), S \subseteq [n]$
- Continuous extension:

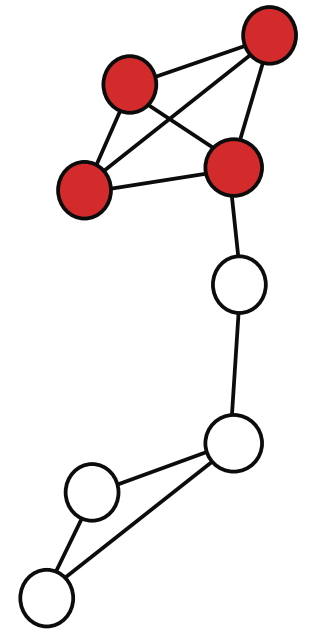
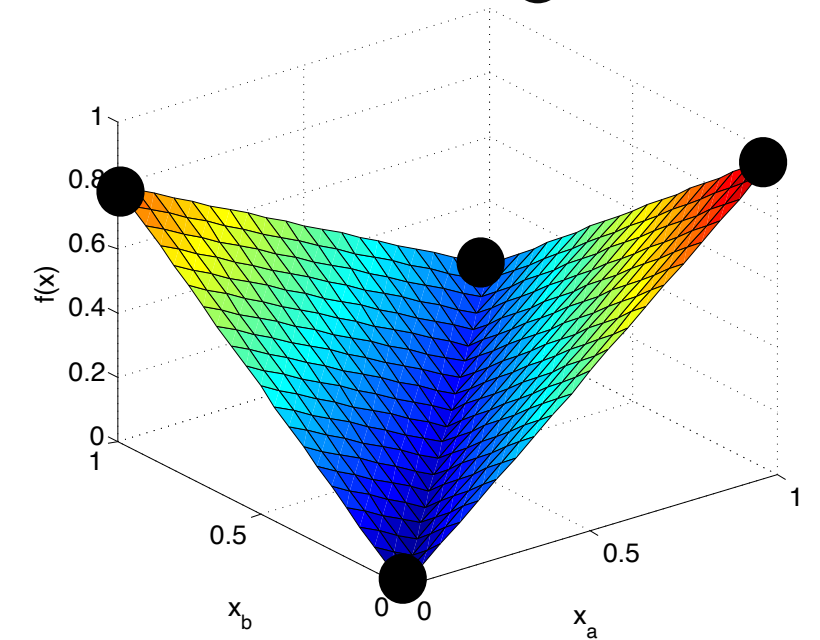
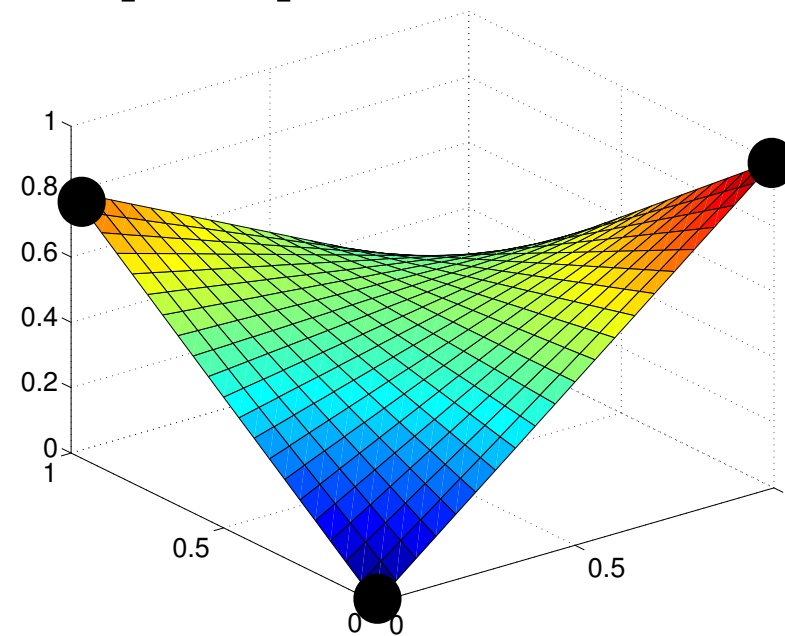
$$F : \{0, 1\}^n \rightarrow \mathbb{R} \quad \longrightarrow \quad f : [0, 1]^n \rightarrow \mathbb{R}$$



Setup

- use NN as “solver”, and objective function of the optimization problem as a loss
- What if the objective is a set function? $F(S), S \subseteq [n]$
- Continuous extension:

$$F : \{0, 1\}^n \rightarrow \mathbb{R} \quad \longrightarrow \quad f : [0, 1]^n \rightarrow \mathbb{R}$$



Setup

- use NN as “solver”, and objective function of the optimization problem as a loss

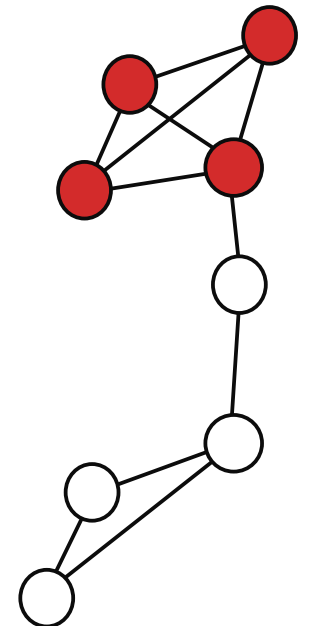
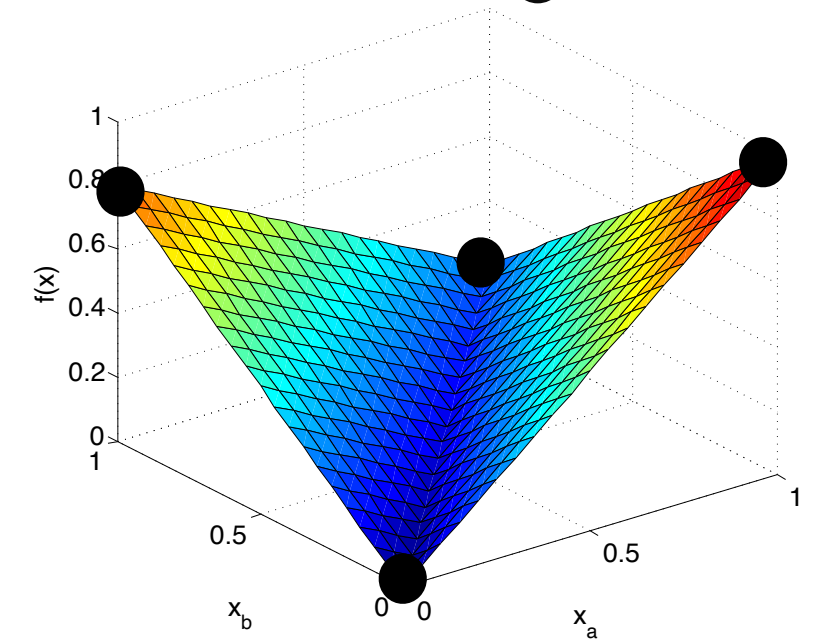
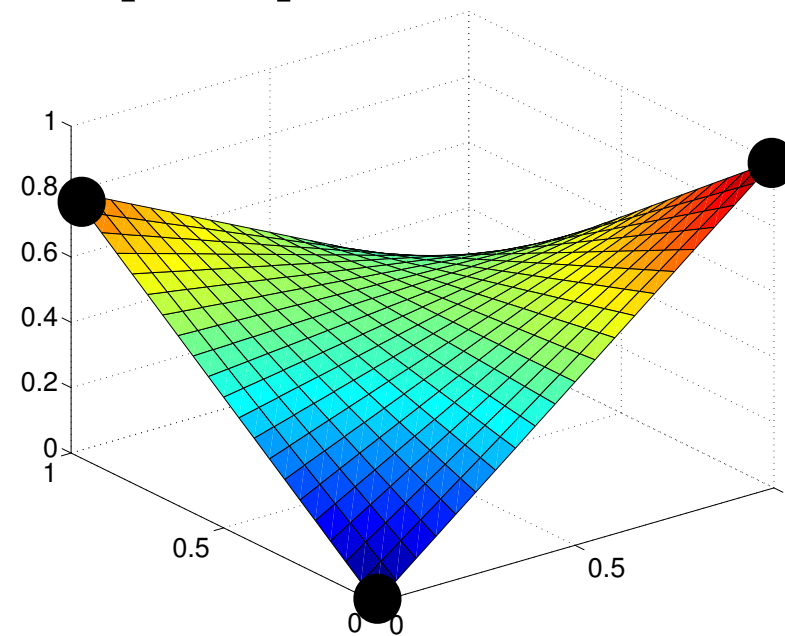
- What if the objective is a set function? $F(S), S \subseteq [n]$

- Continuous extension:

$$F : \{0, 1\}^n \rightarrow \mathbb{R} \quad \longrightarrow \quad f : [0, 1]^n \rightarrow \mathbb{R}$$

- Want that:

- f is continuous
- $f(\mathbf{1}_S) = F(S)$



Setup

- use NN as “solver”, and objective function of the optimization problem as a loss

- What if the objective is a set function? $F(S), S \subseteq [n]$

- Continuous extension:

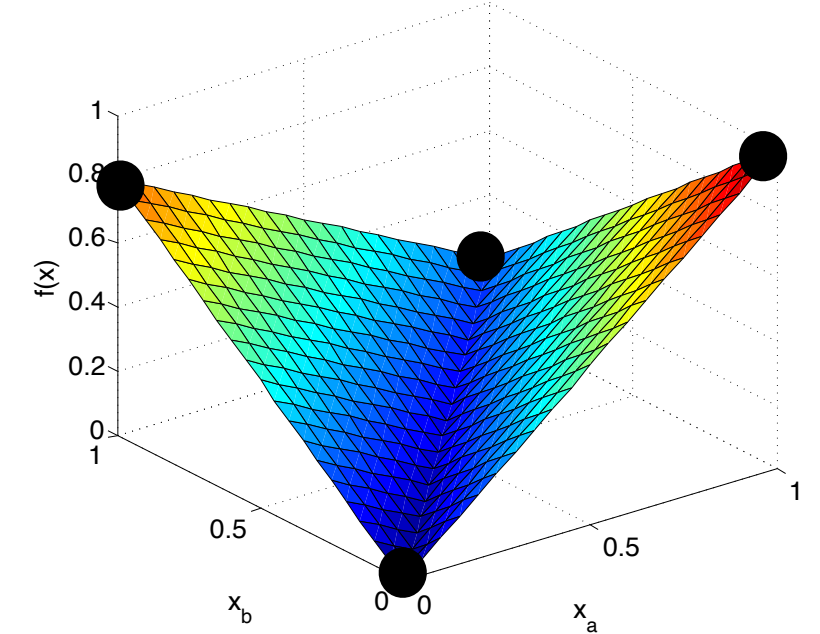
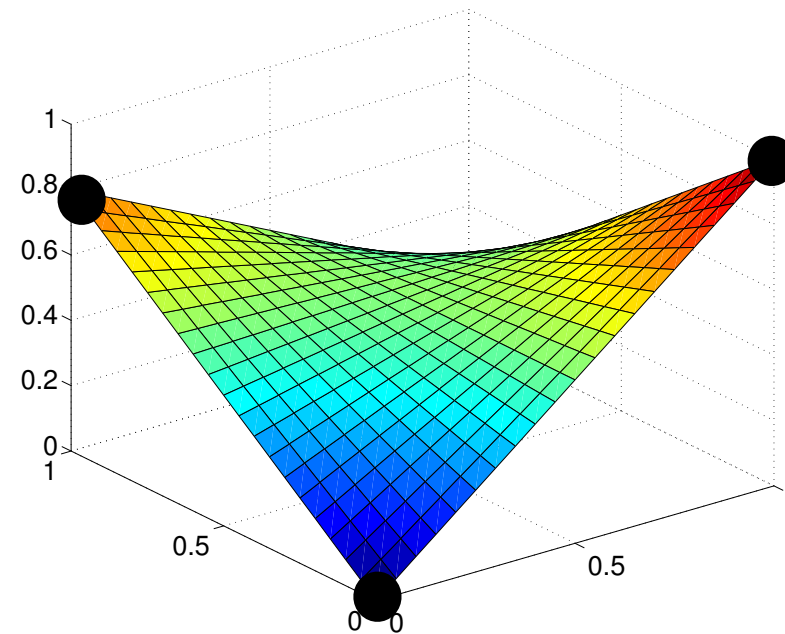
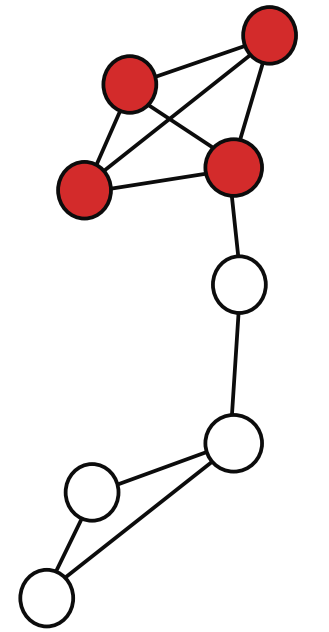
$$F : \{0, 1\}^n \rightarrow \mathbb{R} \quad \longrightarrow \quad f : [0, 1]^n \rightarrow \mathbb{R}$$

- Want that:

- f is continuous
- $f(\mathbf{1}_S) = F(S)$

- Strategy:

$$f(x) = \sum_{S \subseteq [n]} p_x(S) F(S)$$



Setup

- use NN as “solver”, and objective function of the optimization problem as a loss

- What if the objective is a set function? $F(S), S \subseteq [n]$

- Continuous extension:

$$F : \{0, 1\}^n \rightarrow \mathbb{R} \quad \longrightarrow \quad f : [0, 1]^n \rightarrow \mathbb{R}$$

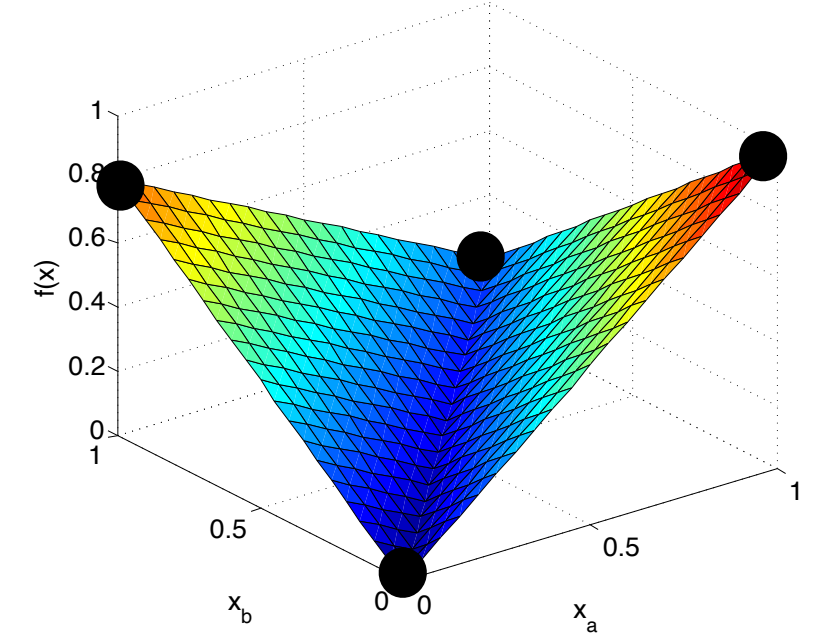
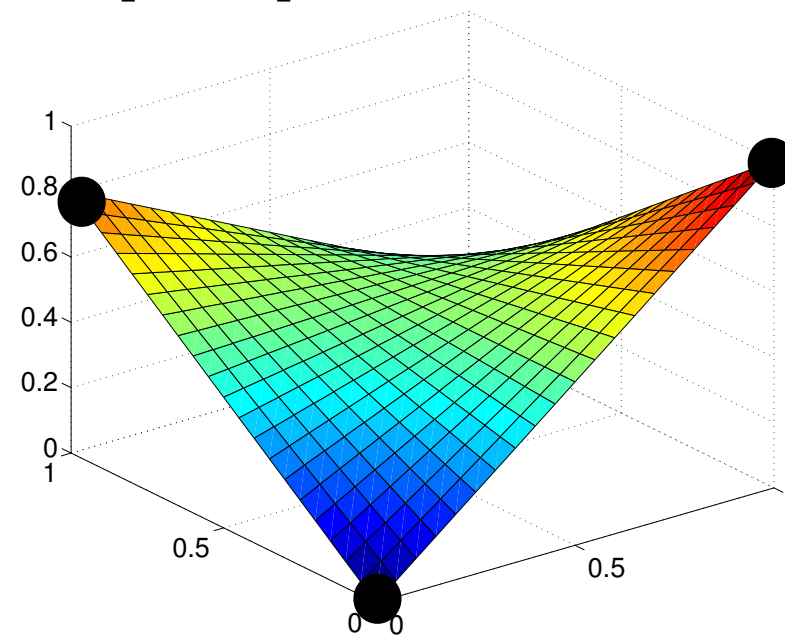
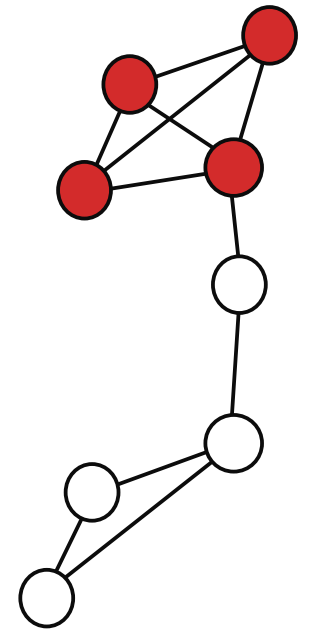
- Want that:

- f is continuous
- $f(\mathbf{1}_S) = F(S)$

- Strategy:

$$f(x) = \sum_{S \subseteq [n]} p_x(S) F(S)$$

marginals:
 $p_x(i \in S) = x_i$



Setup

- use NN as “solver”, and objective function of the optimization problem as a loss

- What if the objective is a set function? $F(S), S \subseteq [n]$

- Continuous extension:

$$F : \{0, 1\}^n \rightarrow \mathbb{R} \quad \longrightarrow \quad f : [0, 1]^n \rightarrow \mathbb{R}$$

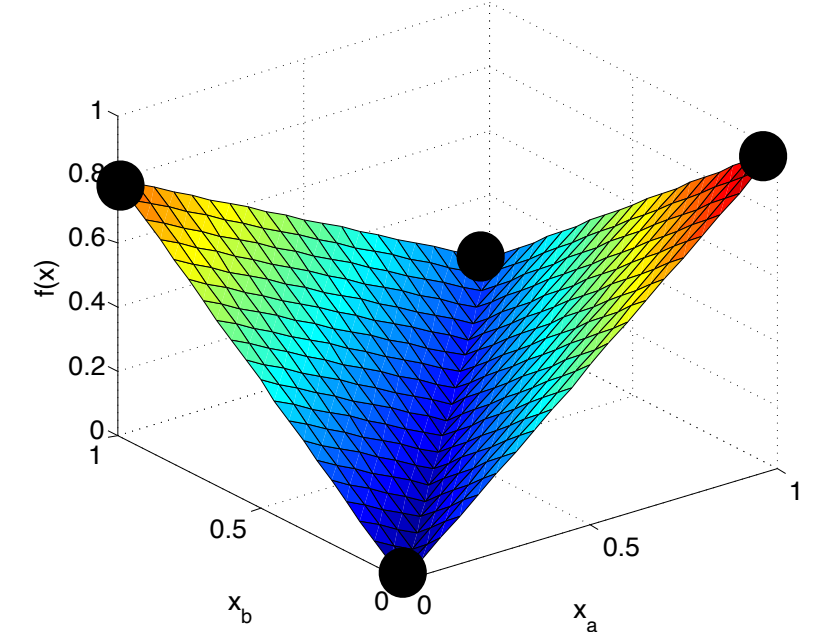
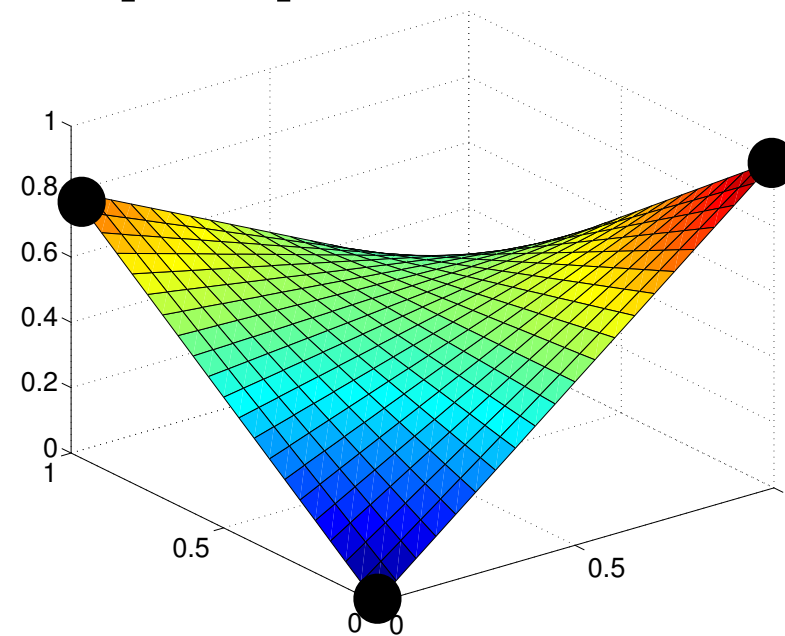
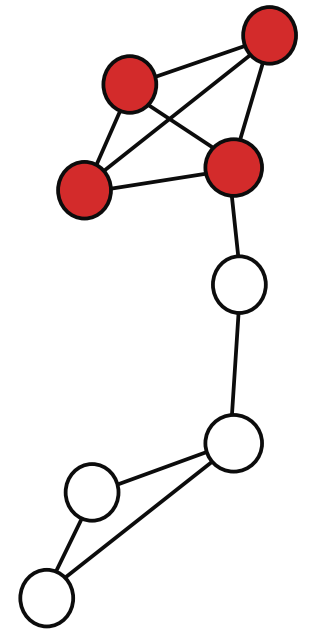
- Want that:

- f is continuous
- $f(\mathbf{1}_S) = F(S)$

- Strategy:

$$f(x) = \sum_{S \subseteq [n]} p_x(S) F(S)$$

marginals:
 $p_x(i \in S) = x_i$

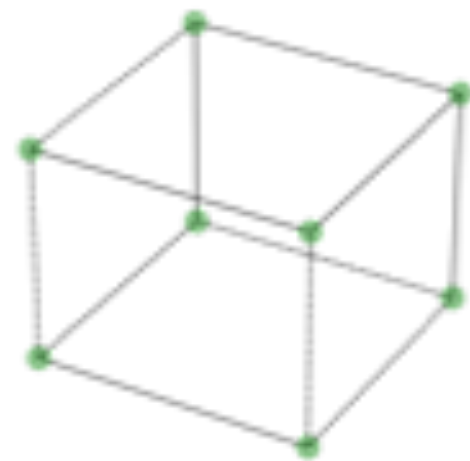


**Example: Lovász extension
of submodular set function**

Extensions: higher-dimensional

NN outputs...

$$S \in \{0, 1\}^n$$



$$F(S)$$

Extensions: higher-dimensional

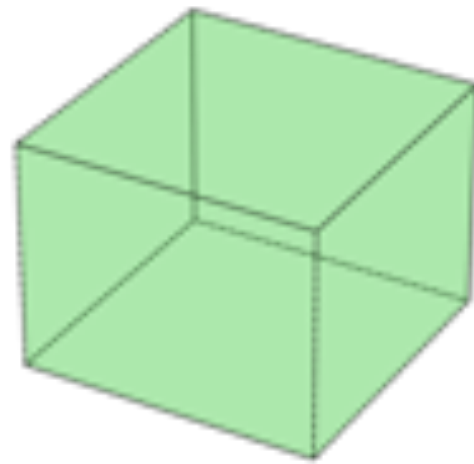
NN outputs...

$$S \in \{0, 1\}^n$$



$$F(S)$$

$$x \in [0, 1]^n$$



$$f(x) = \sum_{S \subseteq [n]} p_x(S) F(S)$$

We want:

$$f(\mathbf{1}_S) = F(S)$$

Extensions: higher-dimensional

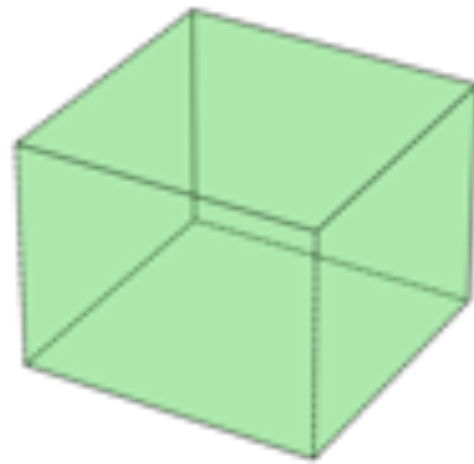
NN outputs...

$$S \in \{0, 1\}^n$$



$$F(S)$$

$$x \in [0, 1]^n$$

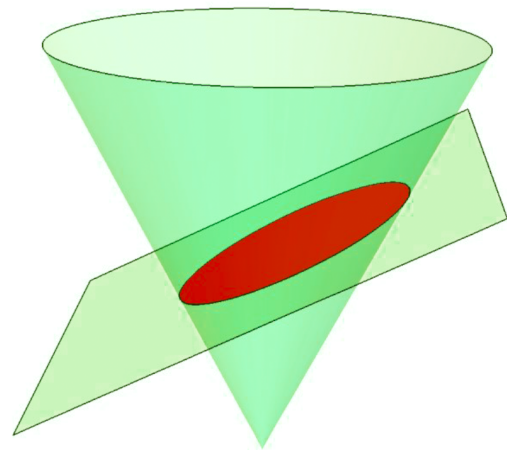


$$f(x) = \sum_{S \subseteq [n]} p_x(S) F(S)$$

We want:

$$f(\mathbf{1}_S) = F(S)$$

$$X \in \mathbb{S}_+$$



$$f(X) = \sum_{S, T \subseteq [n]} p_X(S, T) F(S \cap T)$$

$$f(\mathbf{1}_S \mathbf{1}_S^\top) = F(S)$$

Derivation in a nutshell

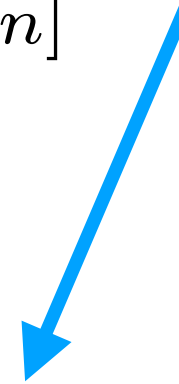
- vector extension for $x \in [0, 1]^n$: $p_x(S)$ solution to dual of “ f is a convex envelope of F ”:

$$\min_{\{y_S \geq 0\}_{S \subseteq [n]}} \sum_{S \subseteq [n]} y_S F(S) \quad \text{s.t.} \quad \sum_{S \subseteq [n]} y_S \mathbf{1}_S = x, \quad \sum_{S \subseteq [n]} y_S = 1$$

Derivation in a nutshell

- vector extension for $x \in [0, 1]^n$: $p_x(S)$ solution to dual of “ f is a convex envelope of F ”:

$$\min_{\{y_S \geq 0\}_{S \subseteq [n]}} \sum_{S \subseteq [n]} y_S F(S) \quad \text{s.t.} \quad \sum_{S \subseteq [n]} y_S \mathbf{1}_S = x, \quad \sum_{S \subseteq [n]} y_S = 1$$



=> marginals

$$p_x(i \in S) = x_i$$

Derivation in a nutshell

- vector extension for $x \in [0, 1]^n$: $p_x(S)$ solution to dual of “ f is a convex envelope of F ”:

$$\min_{\{y_S \geq 0\}_{S \subseteq [n]}} \sum_{S \subseteq [n]} y_S F(S) \quad \text{s.t.} \quad \sum_{S \subseteq [n]} y_S \mathbf{1}_S = x, \quad \sum_{S \subseteq [n]} y_S = 1$$

- use SDP version of this

Derivation in a nutshell

- vector extension for $x \in [0, 1]^n$: $p_x(S)$ solution to dual of “ f is a convex envelope of F ”:

$$\min_{\{y_S \geq 0\}_{S \subseteq [n]}} \sum_{S \subseteq [n]} y_S F(S) \quad \text{s.t.} \quad \sum_{S \subseteq [n]} y_S \mathbf{1}_S = x, \quad \sum_{S \subseteq [n]} y_S = 1$$

- use SDP version of this

- any valid vector extension leads to a valid matrix extension: Let $X = \sum_i \lambda_i v_i v_i^\top$

Derivation in a nutshell

- vector extension for $x \in [0, 1]^n$: $p_x(S)$ solution to dual of “ f is a convex envelope of F ”:

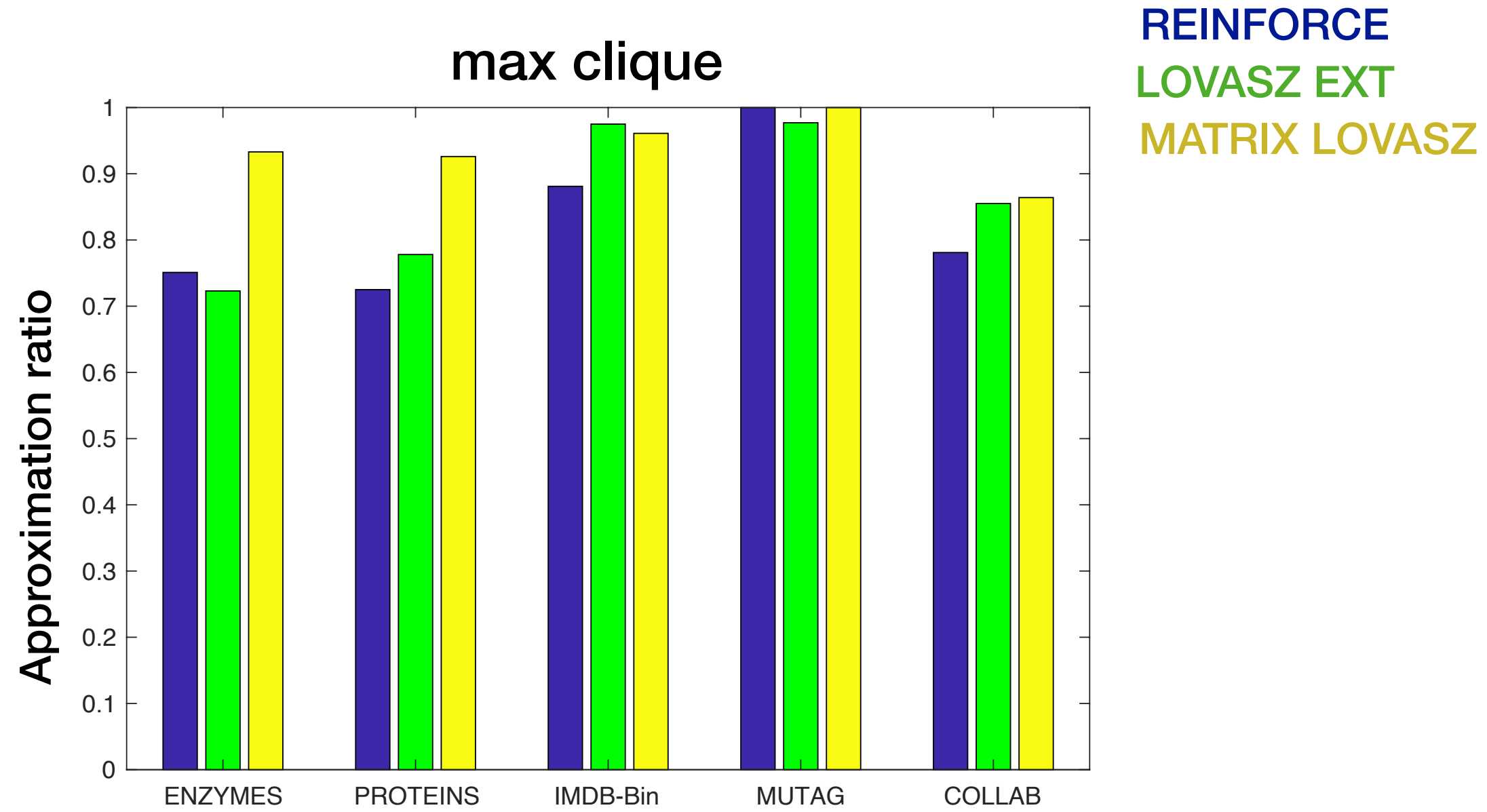
$$\min_{\{y_S \geq 0\}_{S \subseteq [n]}} \sum_{S \subseteq [n]} y_S F(S) \quad \text{s.t.} \quad \sum_{S \subseteq [n]} y_S \mathbf{1}_S = x, \quad \sum_{S \subseteq [n]} y_S = 1$$

- use SDP version of this

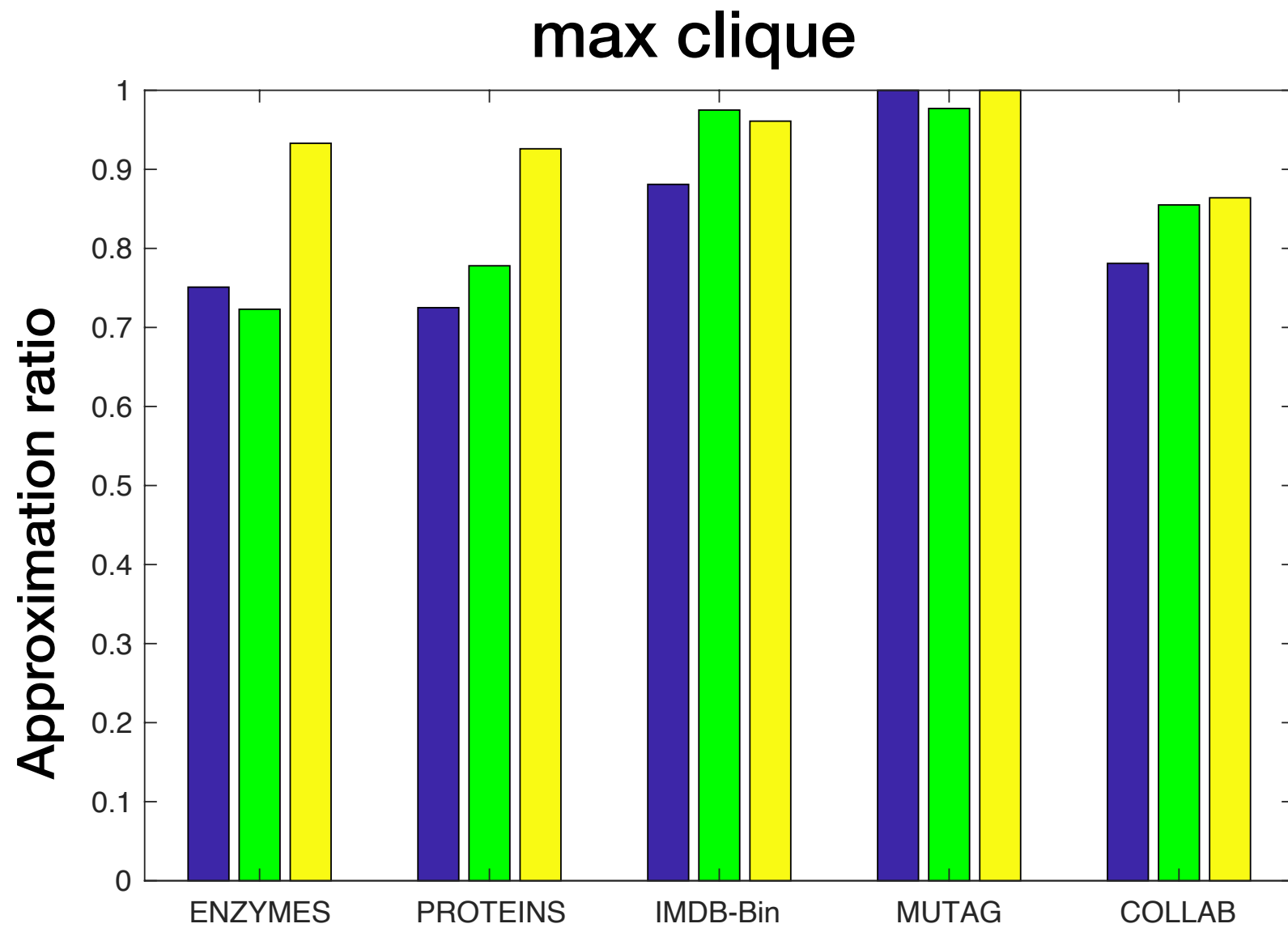
- any valid vector extension leads to a valid matrix extension: Let $X = \sum_i \lambda_i v_i v_i^\top$

$$\sum_{S, T \subseteq [n]} p_X(S, T) F(S \cap T) \quad \text{with} \quad p_X(S, T) = \sum_i \lambda_i p_{v_i}(S) p_{v_i}(T)$$

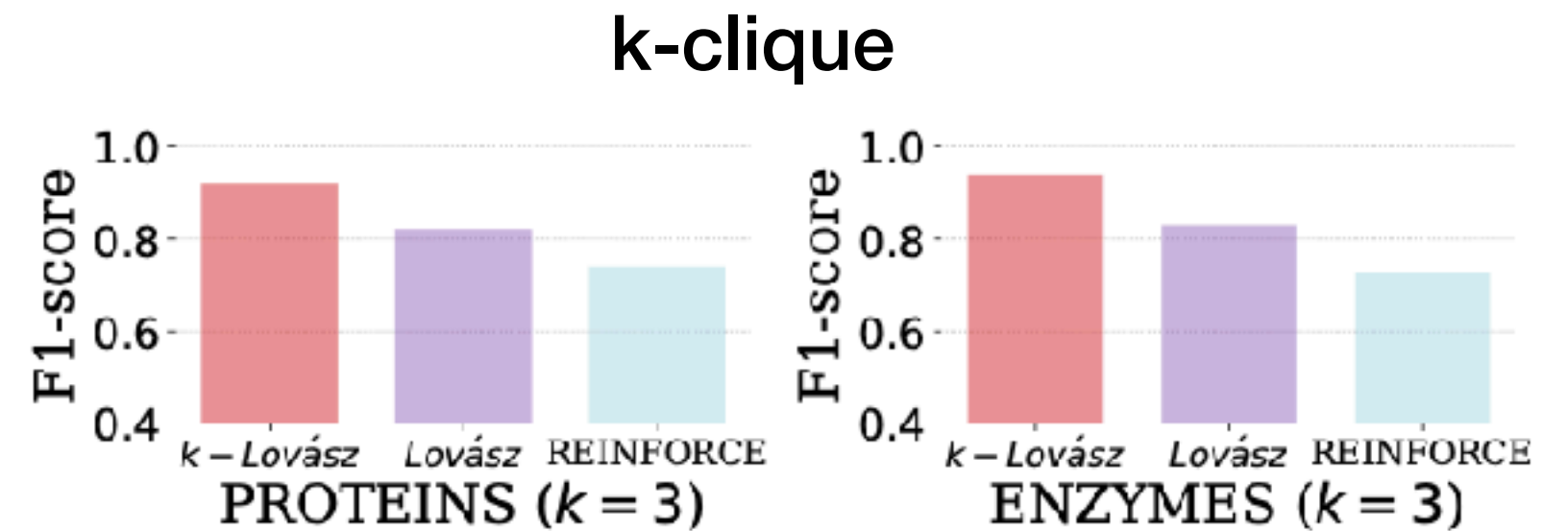
Empirical results



Empirical results



REINFORCE
LOVASZ EXT
MATRIX LOVASZ



GNNs for Learning for Combinatorial Optimization

- Active, recent area, but need to understand learned functions
- **To what examples will my models generalize?** - understanding the data space metric from the GNN perspective
- **What are important architectural choices?** - understanding the model nonlinearities and alignment with algorithms
- **How to train the model?** - choice of loss function
higher-dimensional extensions of set functions tend to work better

Additional slides

$$\tilde{\mathfrak{F}}(\mathbf{x}) = \max_{\mathbf{z}, b \in \mathbb{R}^n \times \mathbb{R}} \{\mathbf{x}^\top \mathbf{z} + b\} \text{ subject to } \mathbf{1}_S^\top \mathbf{z} + b \leq f(S) \text{ for all } S \subseteq [n]. \quad (\text{primal LP})$$

$$\tilde{\mathfrak{F}}(\mathbf{x}) = \min_{\{y_S \geq 0\}_{S \subseteq [n]}} \sum_{S \subseteq [n]} y_S f(S) \text{ subject to } \sum_{S \subseteq [n]} y_S \mathbf{1}_S = \mathbf{x}, \quad \sum_{S \subseteq [n]} y_S = 1, \text{ for all } S \subseteq [n],$$

(dual LP)

$$\max_{\mathbf{Z} \succeq 0, b \in \mathbb{R}} \{\text{Tr}(\mathbf{X}^\top \mathbf{Z}) + b\} \text{ subject to } \frac{1}{2} \text{Tr}((\mathbf{1}_S \mathbf{1}_T^\top + \mathbf{1}_T \mathbf{1}_S^\top) \mathbf{Z}) + b \leq f(S \cap T) \text{ for } S, T \subseteq [n],$$

(primal SDP)

$$\min_{\{y_{S,T} \geq 0\}} \sum_{S, T \subseteq [n]} y_{S,T} f(S \cap T) \text{ subject to } \mathbf{X} \preceq \sum_{S, T \subseteq [n]} \frac{1}{2} y_{S,T} (\mathbf{1}_S \mathbf{1}_T^\top + \mathbf{1}_T \mathbf{1}_S^\top) \text{ and } \sum_{S, T \subseteq [n]} y_{S,T} = 1$$

(dual SDP)