# How Neural Networks Extrapolate: From Feedforward to Graph Neural Networks

Keyulu Xu

MIT

# Approaches of ML for algorithms

## **“End-to-end” learning of algorithm**

- train on a set of (input, output)
- algorithm implemented as neural network

## Learning a “part” of the algorithm

- algo depends on certain *“prediction”* of input *(Kraska et 2018, Balcan et al 2018, Hsu et al 2019, Dong et al 2020..)*
- learned algorithm configuration *(Leyton-Brown et al 2002, Hutter et al 2011, Gupta et al 2015, Balcan et al 2017..)*

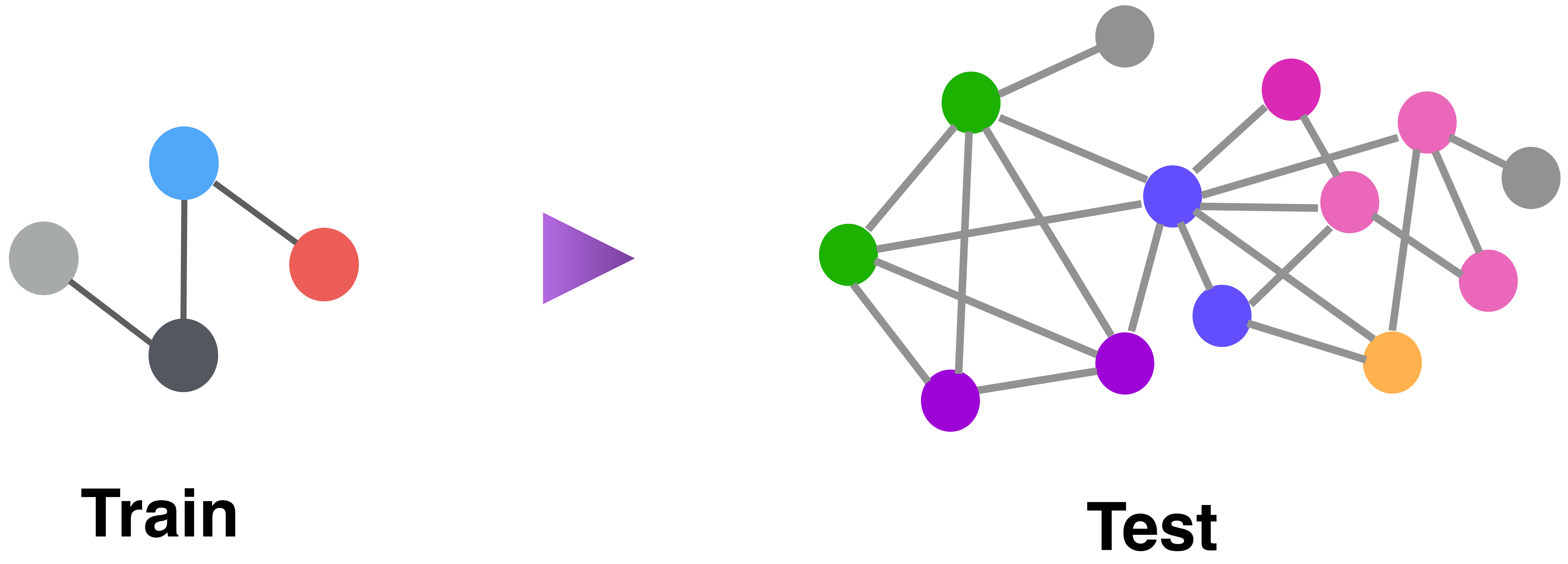
## Deep RL & unsupervised

- optimize certain reward or unsupervised objective *(Dai et al 2018, Mao et al 2018, Abe et al 2020, Karalias et al 2020..)*

# Extrapolation

Train NN $f$ to learn underlying function $g : \mathcal{X} \to \mathbb{R}$ with training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n} \subset \mathcal{D}$

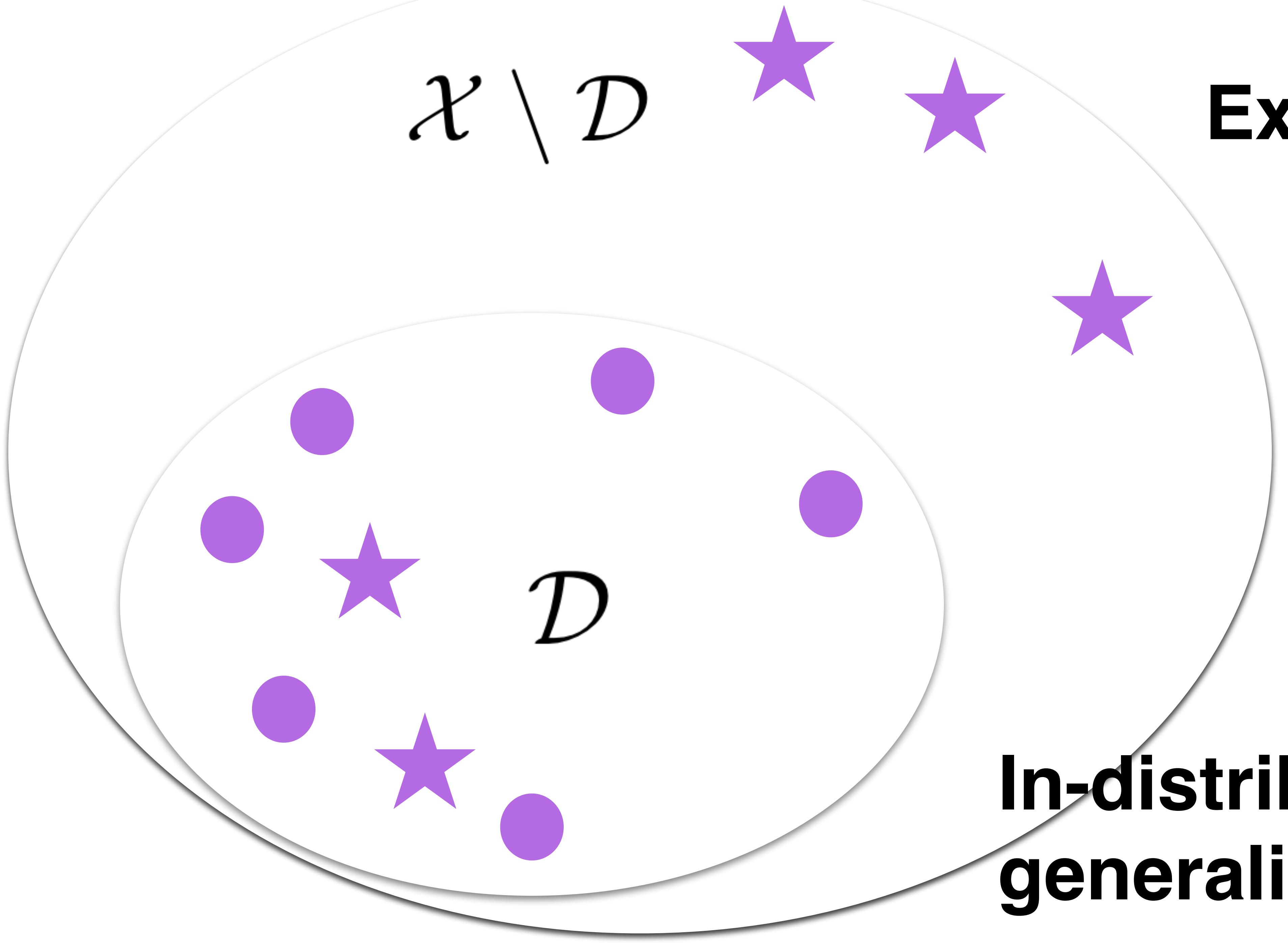
$$\mathbb{E}_{\boldsymbol{x} \sim \mathcal{P}_{\text{test}}}[\ell(f(\boldsymbol{x}), g(\boldsymbol{x}))]$$



# Extrapolation vs. interpolation

Train NN $f$ to learn underlying function $g : \mathcal{X} \rightarrow \mathbb{R}$ with training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n} \subset \mathcal{D}$

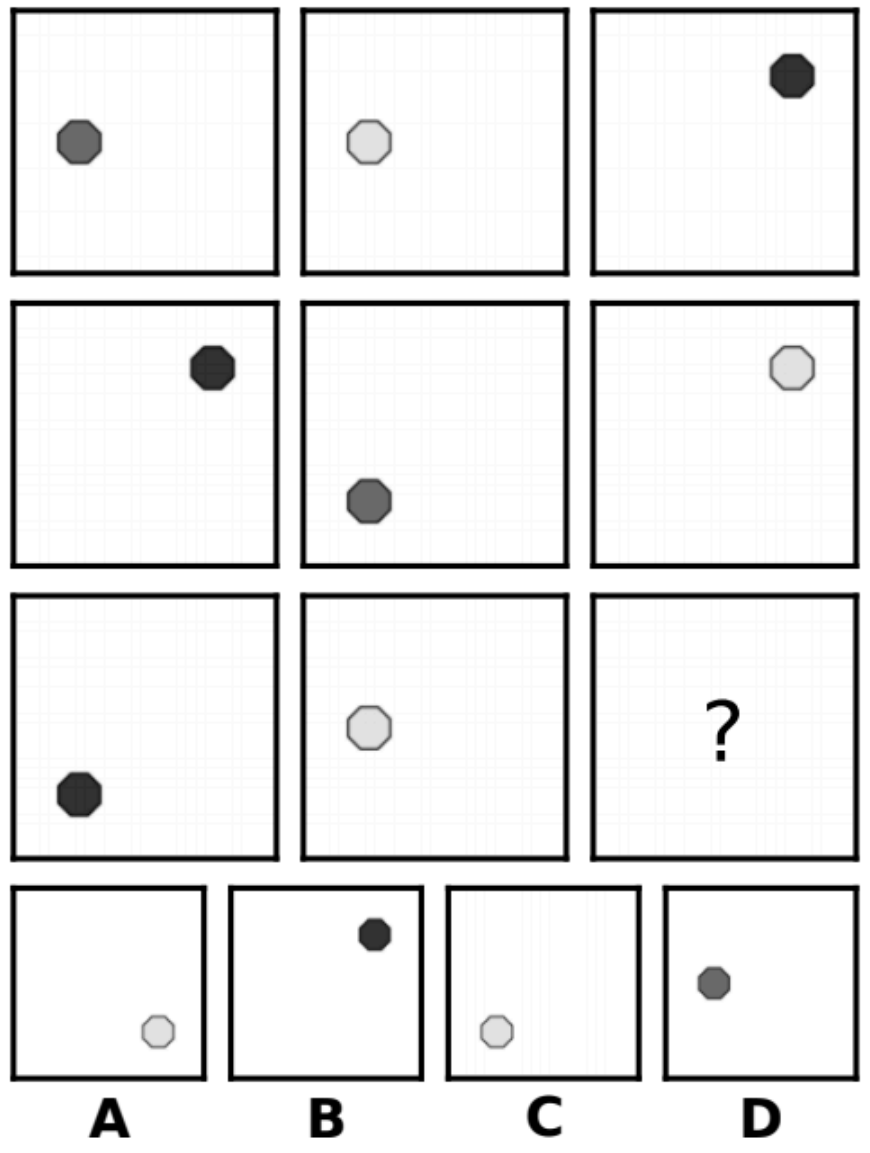
$$\mathbb{E}_{\boldsymbol{x} \sim \mathcal{P}_{\text{test}}} [\ell(f(\boldsymbol{x}), g(\boldsymbol{x}))]$$

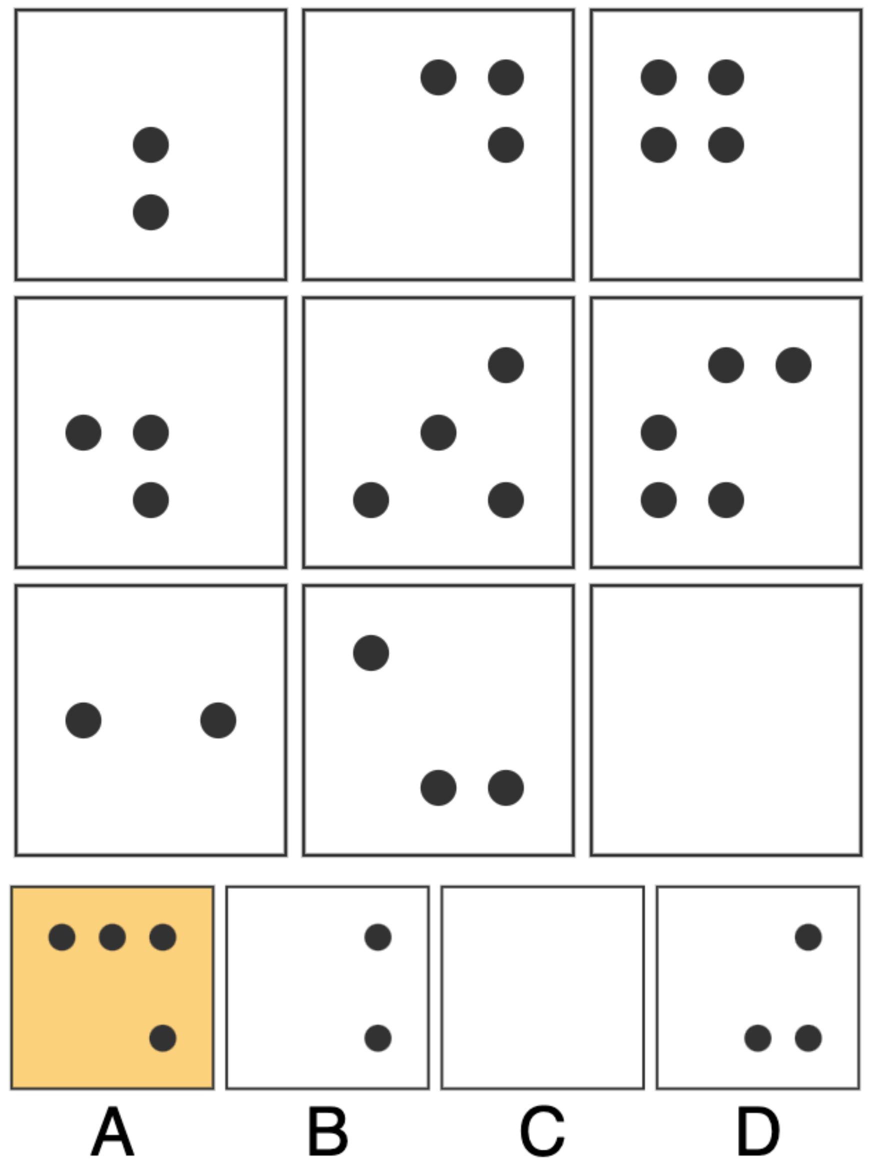



**Extrapolation**

**In-distribution generalization** $\mathcal{P}_{\text{train}} = \mathcal{P}_{\text{test}}$

# Evaluation of NN extrapolation & interpolation




IQ tests
shape, color, number of objects

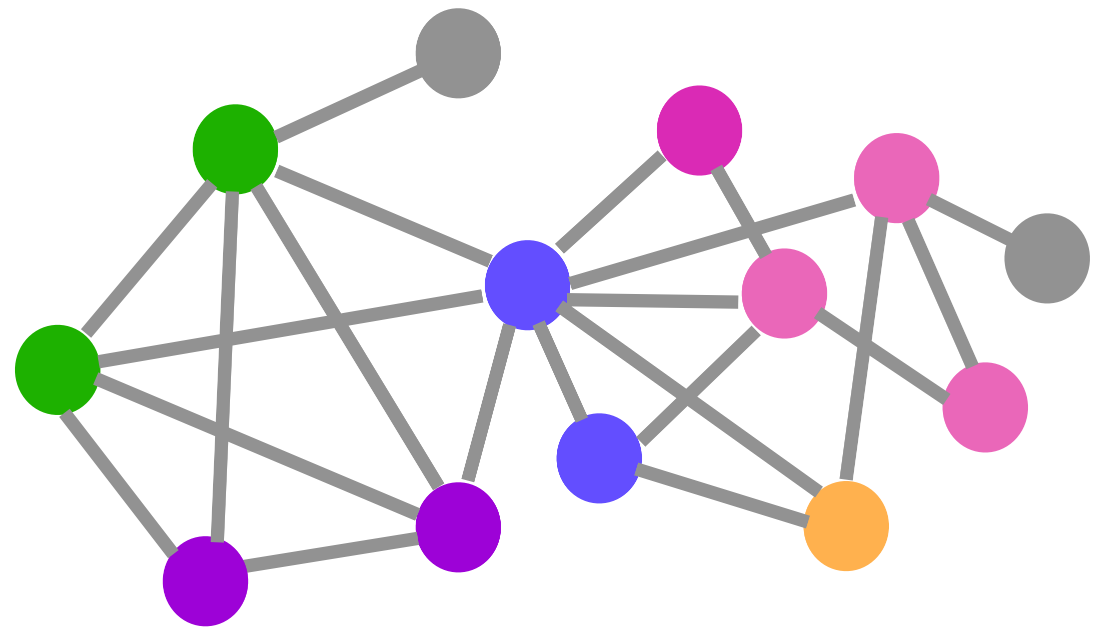
*(Santoro et al. 2018, Zhang et al 2019)*

Physical reasoning
position, mass, number of objects

*(Wu et al. 2017, Battagalia et al 2016, Janner et al 2019)*

# Evaluation of NN extrapolation & interpolation

**Question:** `Calculate -841880142.544 + 411127.`
**Answer:** `-841469015.544`
**Question:** `Let x(g) = 9*g + 1. Let q(c) = 2*c + 1. Let f(i) = 3*i - 39. Let w(j) = q(x(j)). Calculate f(w(a)).`
**Answer:** `54*a - 30`
**Question:** `Let e(l) = l - 6. Is 2 a factor of both e(9) and 2?`
**Answer:** `False`



Mathematical reasoning

length, number range, complexity

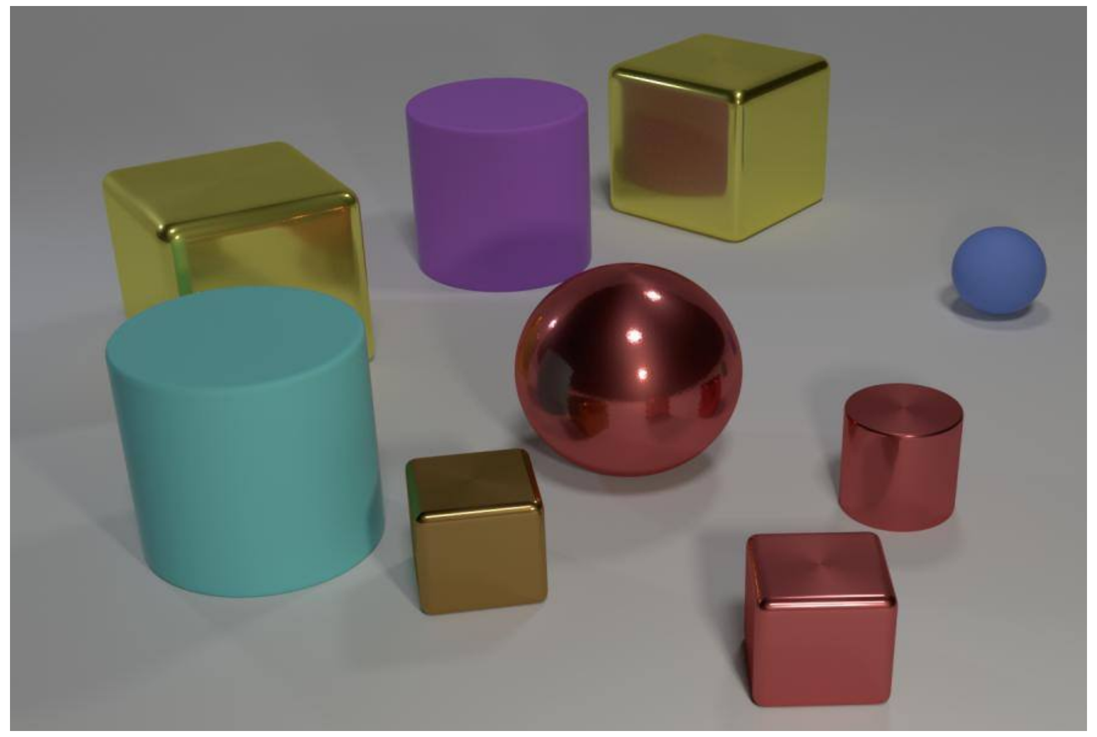
*(Saxton et al. 2019, Lample et al 2020)*

Graph algorithms

graph size, graph structure, edge weights

*(Battagalia et al 2018, Dai et al 2018, Velickovic et al 2020)*

# Architectures (Part I)

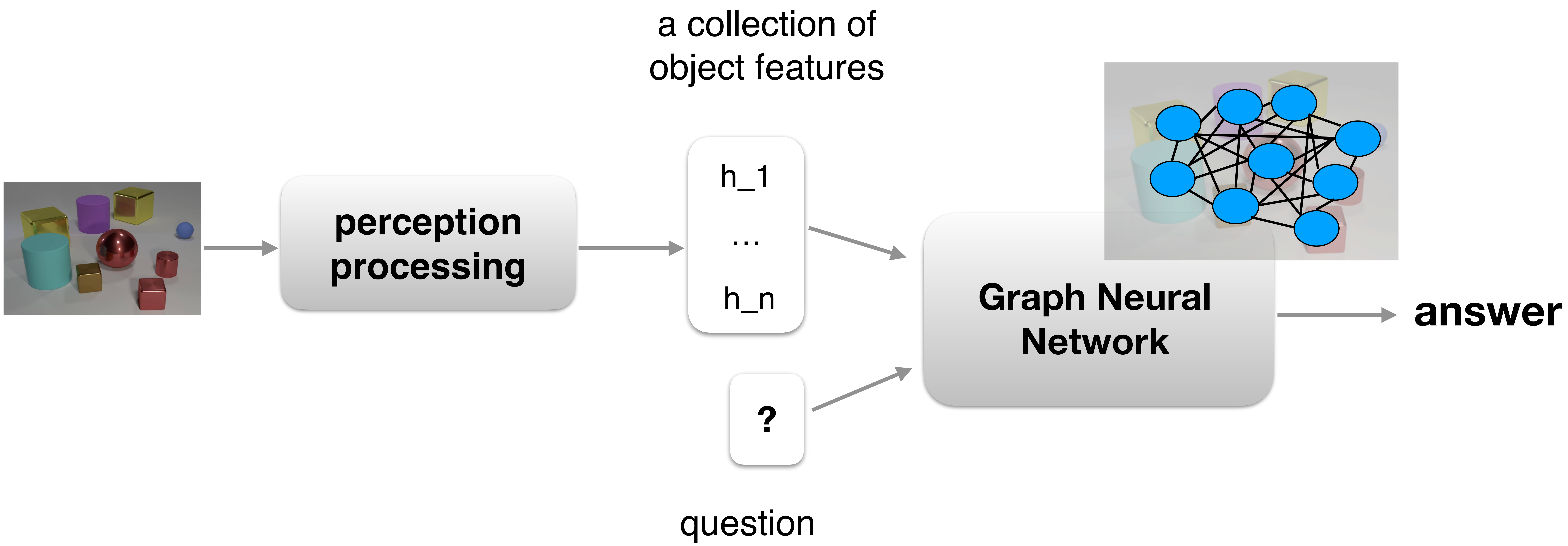


→ **Neural Network** *MLP* *CNN* *RNN* → **answer**

**?** *e.g., colors of the furthest pair of objects?*

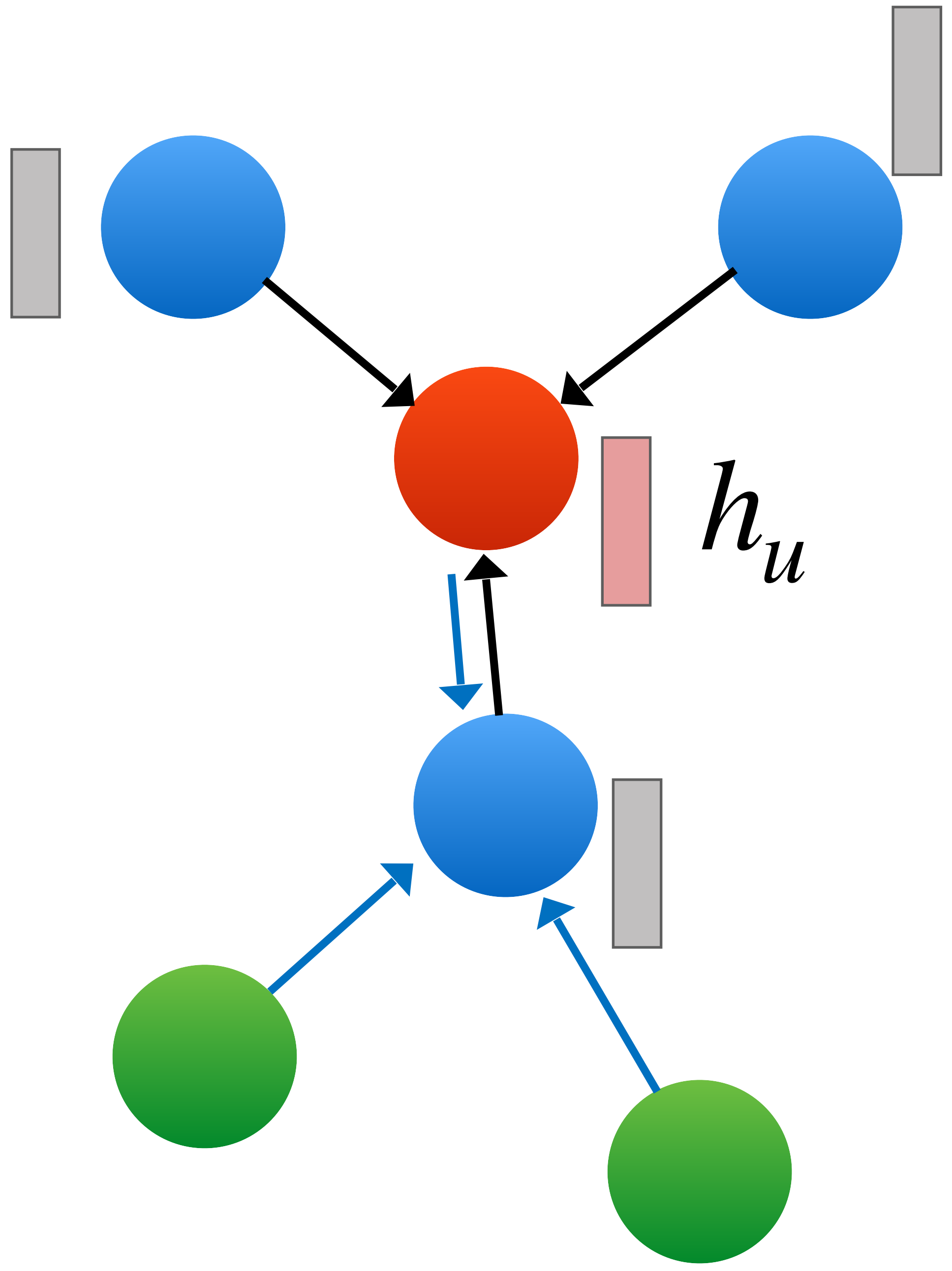
*(Weston et al., 2015; Johnson et al., 2017a; Wu et al. 2017, Fleuret et al., 2011; Antol et al., 2015; Battaglia et al., 2016, 2018; Watters et al., 2017; Fragkiadaki et al., 2016; Chang et al., 2017, 2019; Saxton et al., 2019; Santoro et al., 2018…)*

# Architectures (Part II)




(Weston et al., 2015; Johnson et al., 2017a; Wu et al. 2017, Fleuret et al., 2011; Antol et al., 2015; Battaglia et al., 2016, 2018; Watters et al., 2017; Fragkiadaki et al., 2016; Chang et al., 2017, 2019; Saxton et al., 2019; Santoro et al., 2018…)

# Graph Neural Networks (GNNs)




In each round:

For $u \in V$ concurrently:

**Aggregate** over neighbors

Representation of neighbor node $v$ in round $k-1$

$$h_u^{(k)} = \text{AGGREGATE}^{(k)}\left( \left\{ \left( h_v^{(k-1)}, h_u^{(k-1)} \right) \right\} \middle| v \in \mathcal{N}(u) \right)$$

............

Graph-level **readout**

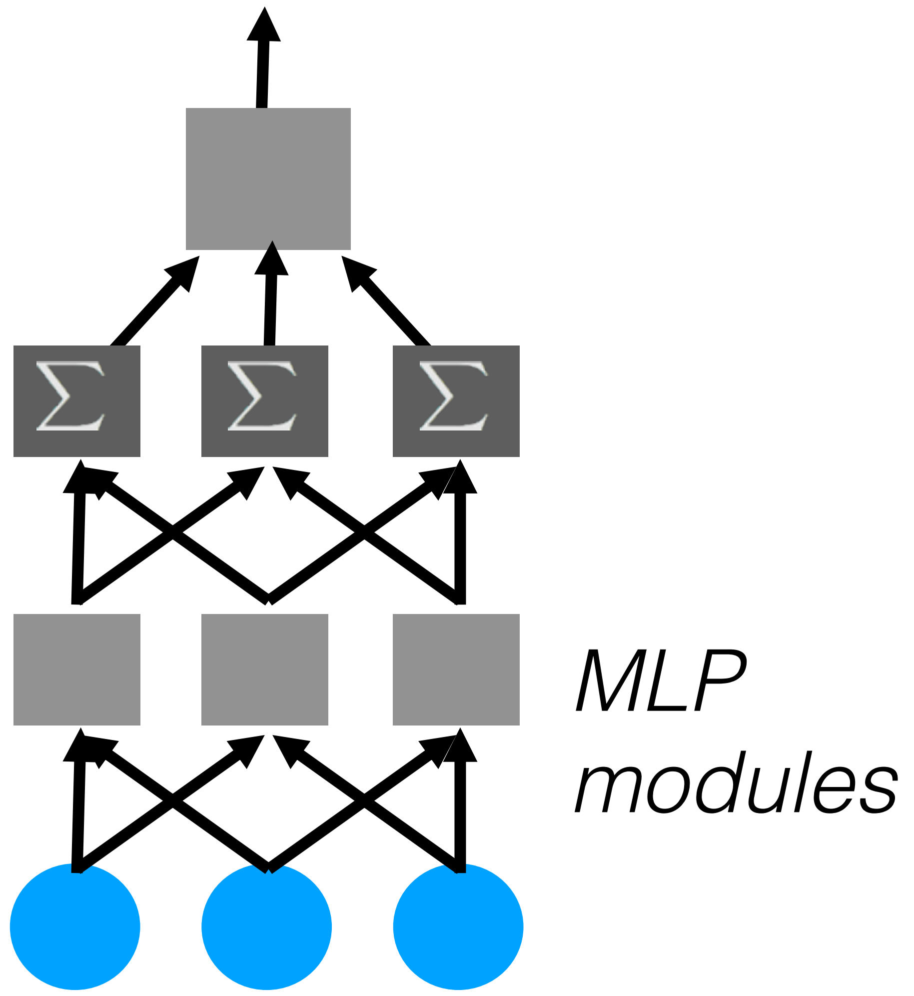
$$h_G = \text{READOUT}\left( \left\{ h_u^{(K)} \right\} \middle| u \in V \right)$$

(Gori et al. 2005, Merkwirth & Lengauer 2005, Scarselli et al 2009, Duvenaud et al., 2015, Battaglia et al., 2016, Dai et al., 2016, Defferrard et al., 2016, Kearnes et al., 2016, Li et al., 2016, Gilmer et al., 2017, Hamilton et al., 2017, Kipf & Welling, 2017, Velickovic et al., 2018, Xu et al., 2018)

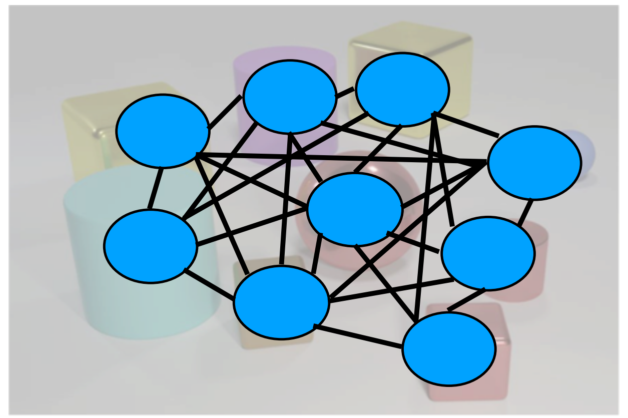
# Training GNNs

1. Parameterize AGGREGATE$^{(k)}$ and READOUT

$$\boldsymbol{h}_u^{(k)} = \sum_{v \in \mathcal{N}(u)} \mathrm{MLP}^{(k)}\left(\boldsymbol{h}_u^{(k-1)}, \boldsymbol{h}_v^{(k-1)}, \boldsymbol{w}_{(v,u)}\right), \quad \boldsymbol{h}_G = \mathrm{MLP}^{(K+1)}\left(\sum_{u \in G} \boldsymbol{h}_u^{(K)}\right)$$




Other aggregation also possible, e.g., attention



2. Specify a loss on node/graph/edge representations

3. Train on data points with SGD

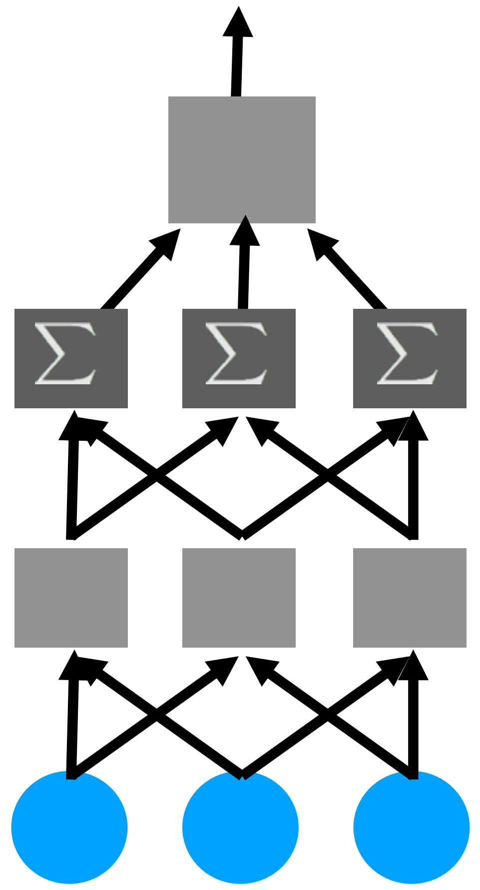
# Puzzle

Feedforward NN *Similarly for CNN, RNN etc*

*(Barnard & Wessels 1992, Haley & Soloway 1992, Santoro et al 2018, Saxton et al 2019)*

Graph neural network (GNN)

*sometimes succeed even in extrapolation*



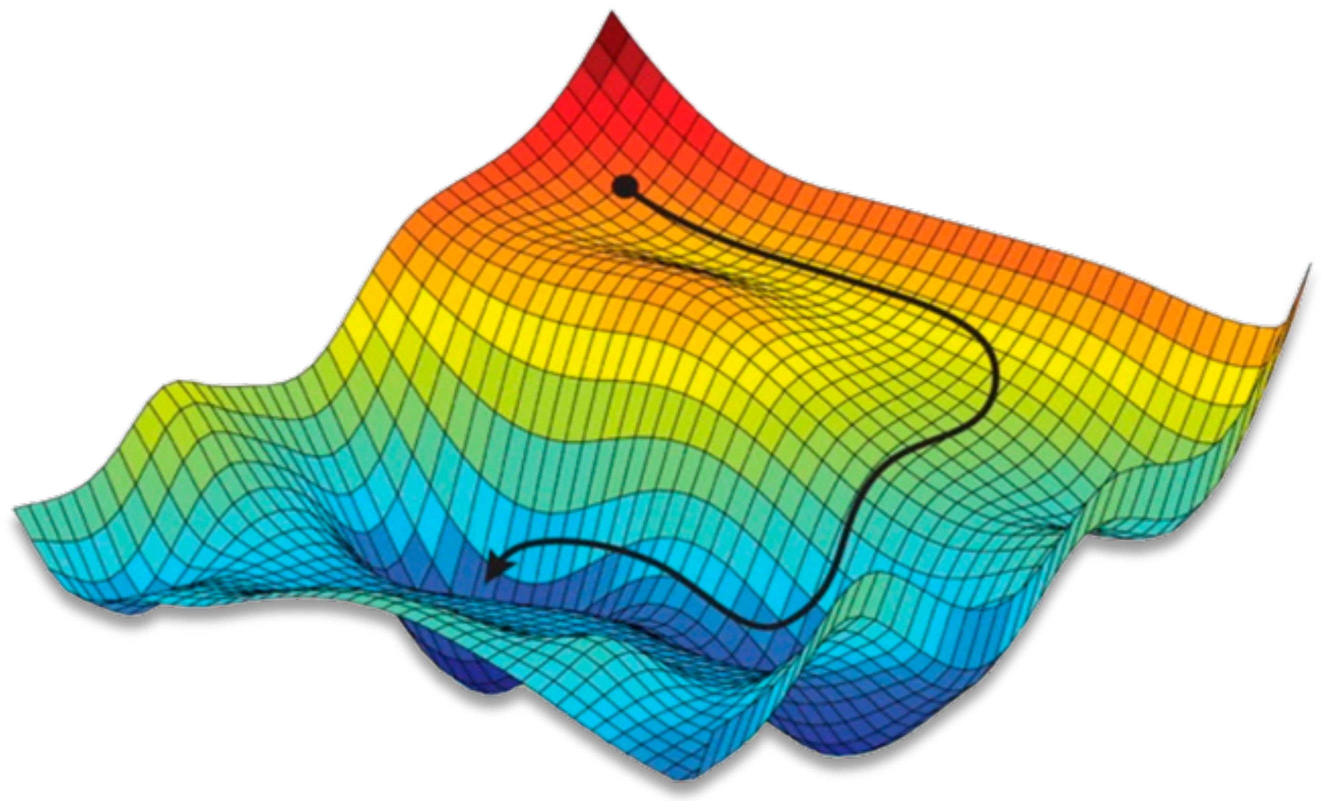
*MLP modules*

**Under what conditions does a NN extrapolate (interpolate) well in a task?**

# What algorithm/function does a NN learn

Despite universal approximation… *(Cybenko 1989, Funahashi 1989, Hornik et al1989, Kurkova 1992, Zhang et al 2017)*

What NN learns depends on **training algorithm**, **architecture**, **data**



Parameter trajectory

$\theta(t)$

*GD*
*(SGD, Adam)*

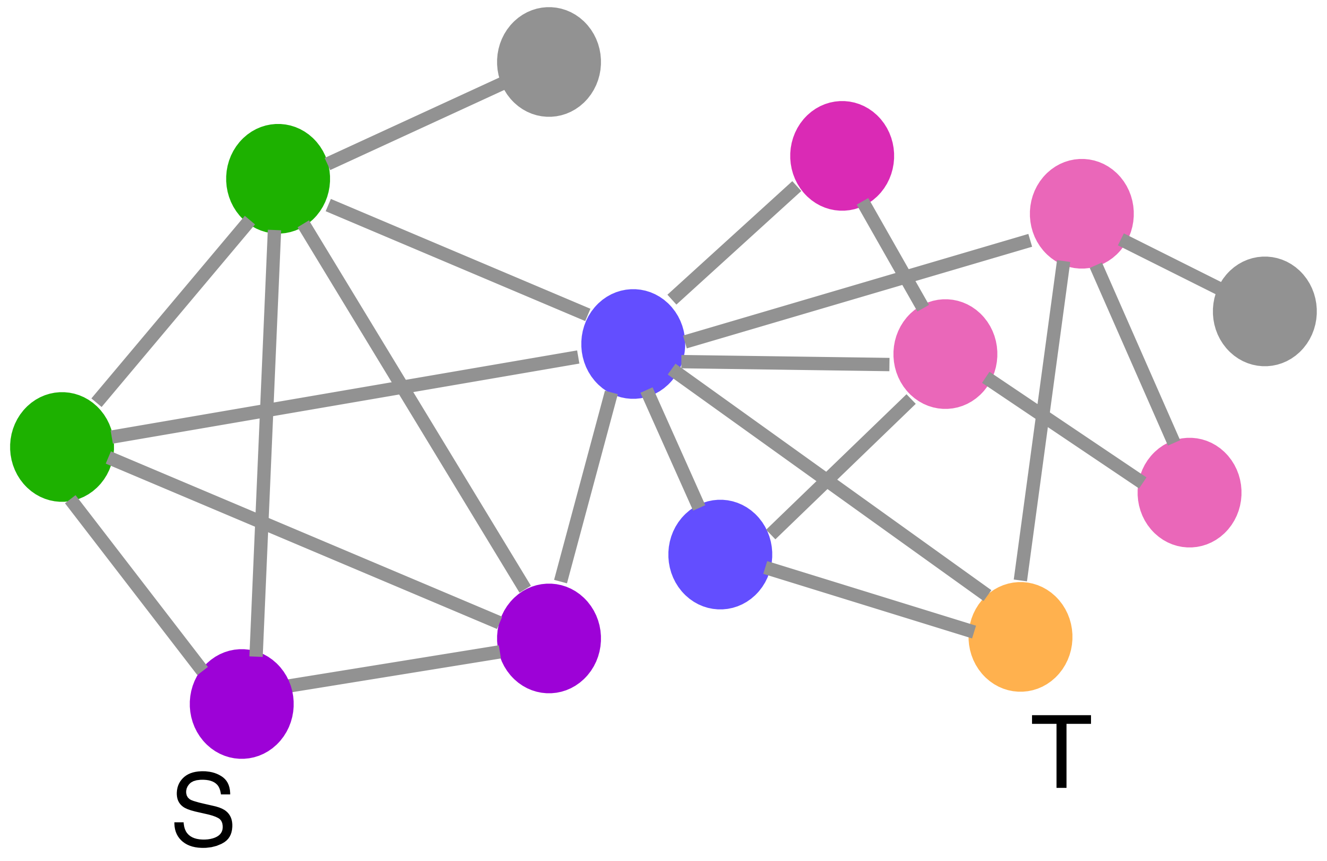
***GNN (structured networks)***

*MLP*

***task structure***

***input distribution***

# Interpolation: task and network structure




e.g., shortest distance between S and T

**Graph Neural Network**

for k = 1 … GNN iter:

for u in S:

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

*No need to learn for-loops*

**Bellman-Ford algorithm**

for k = 1 … |S| - 1:

for u in S:

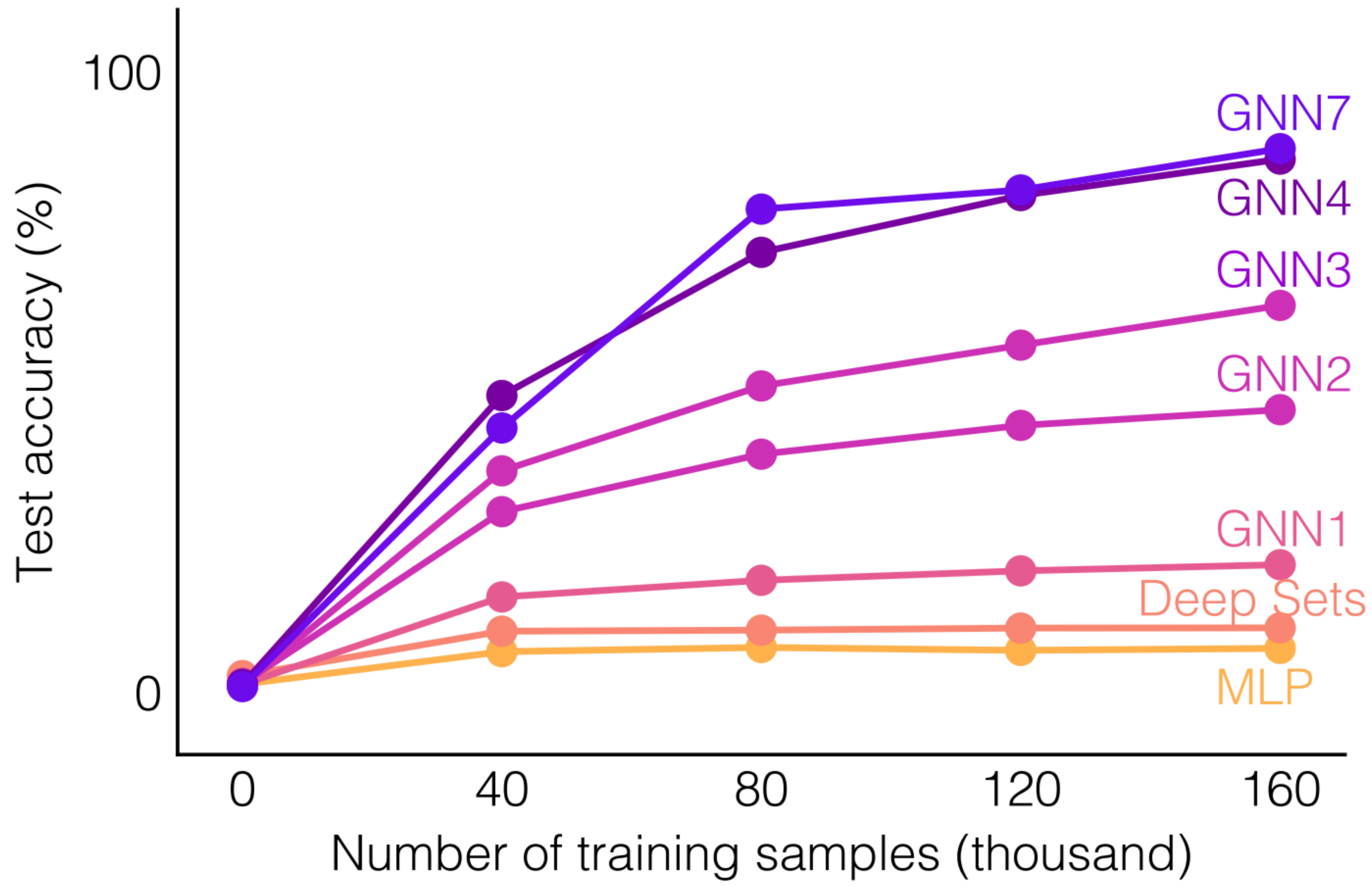
$d[k][u] = \min_v d[k-1][v] + \text{cost}(v, u)$

*Learns a simple reasoning step*

# Algorithmic alignment: formalizing inductive biases

**Algorithmic alignment (XLZDKJ'20)**
Network can simulate algorithm via *few, easy-to-learn* "modules".

**Claim:** Better algo alignment implies better generalization.



# Better alignment implies better generalization

**Algorithmic alignment (XLZDKJ'20)**
A neural network $(M, \epsilon, \delta)$-aligns with an algorithm if it can simulate the algorithm via $n$ weight-shared modules, each of which is $(\epsilon, \delta)$ PAC-learnable with $M/n$ samples.

* *Sample complexity of modules by e.g., NTK*

**Theorem (XLZDKJ'20)**
If a neural network and a task algorithm $(M, \epsilon, \delta)$-align, then, under assumptions*, the task is $(O(\epsilon), O(\delta))$ PAC-learnable by the network with $M$ examples.

* *Lipschitznes and SGD sequential training*
* *Related work experimenting assumptions: Veličković et al 2020*

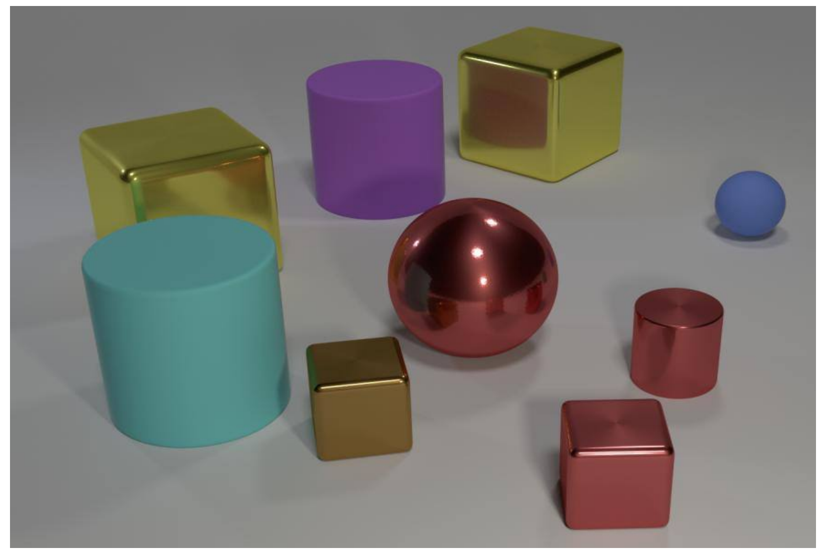
# GNNs can sample-efficiently learn DP

$$\text{Answer}[k][i] = \text{DP-Update}(\{\text{Answer}[k-1][j],\ j = 1 \ldots n\})$$

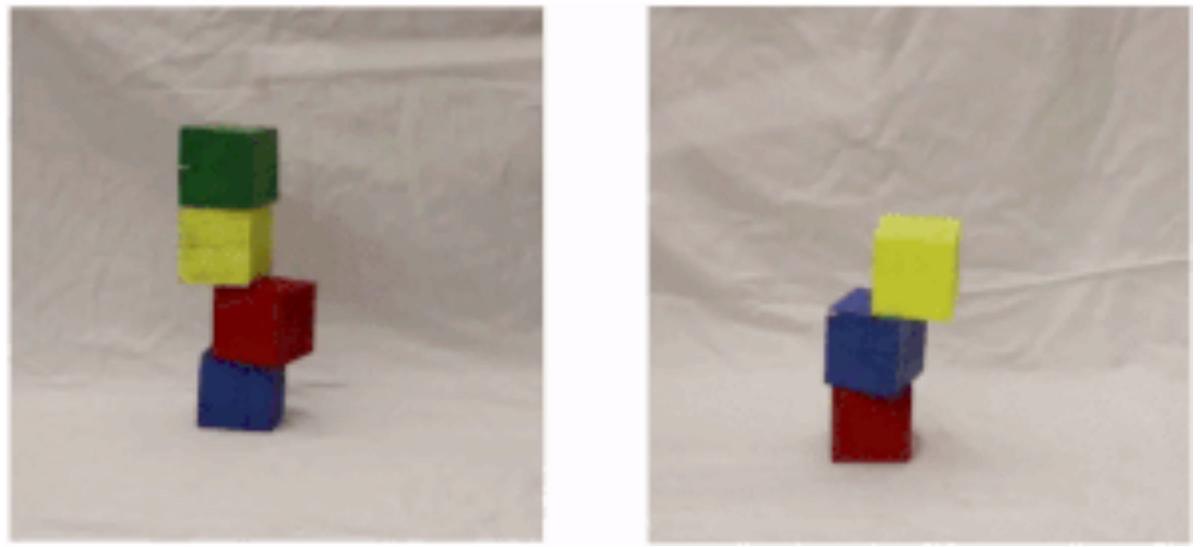
$$h_s^{(k)} = \sum\nolimits_{t \in S} \text{MLP}_1^{(k)}\left(h_s^{(k-1)}, h_t^{(k-1)}\right)$$

Reasoning tasks as dynamic programming (DP):

graph algorithms

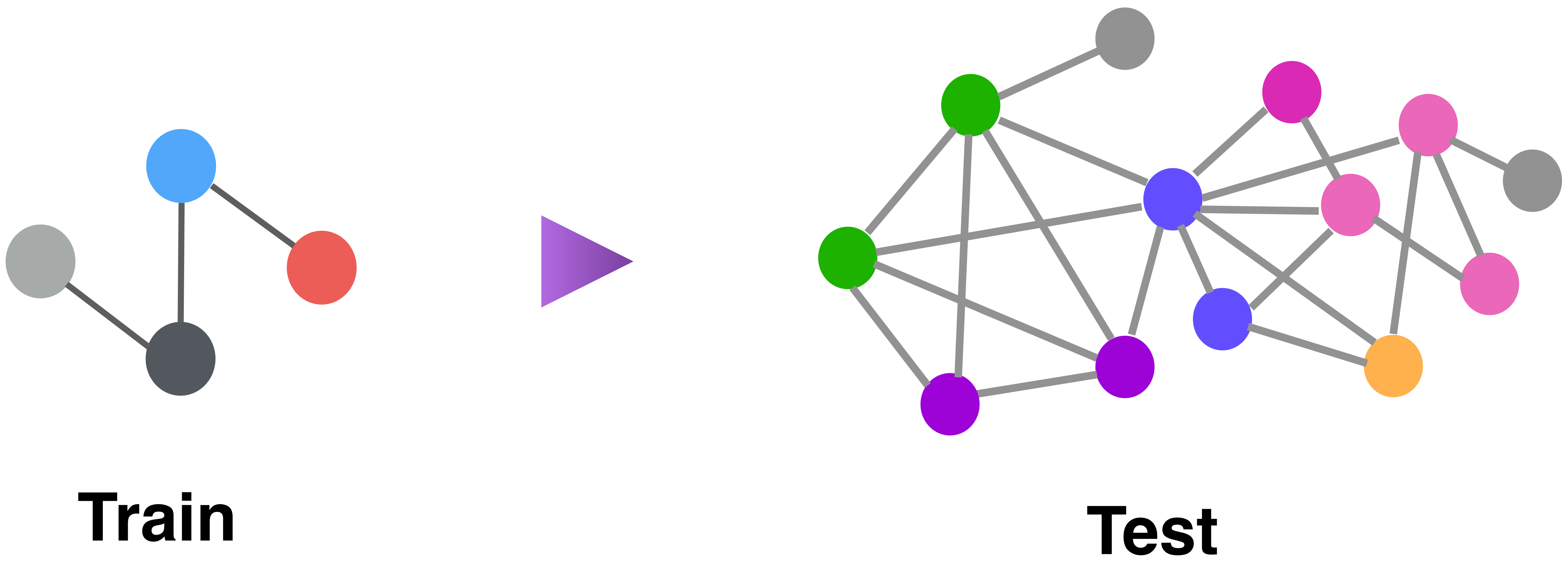


visual question answering



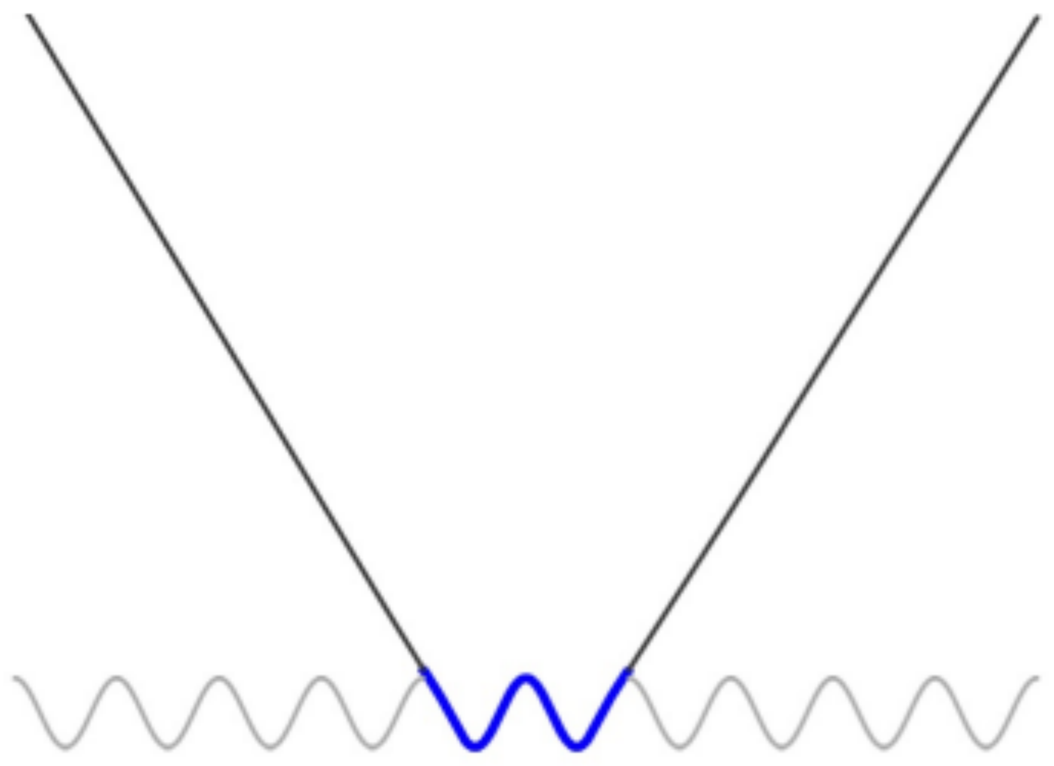
Intuitive physics

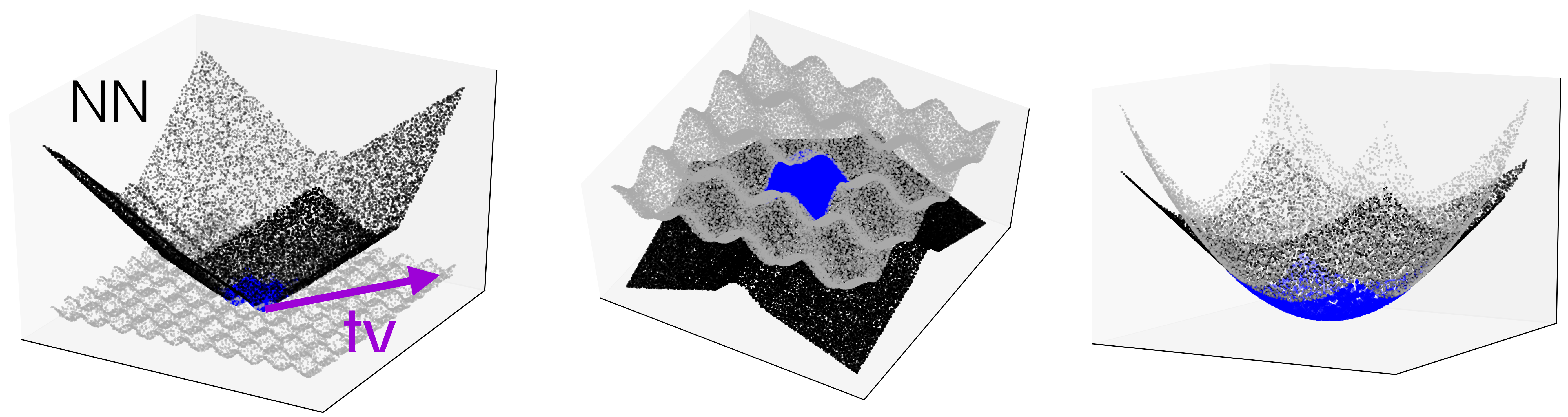
# Extrapolation




Extrapolate graph size, edge weights, node features, graph structure…

# Linear extrapolation behavior of ReLU MLPs

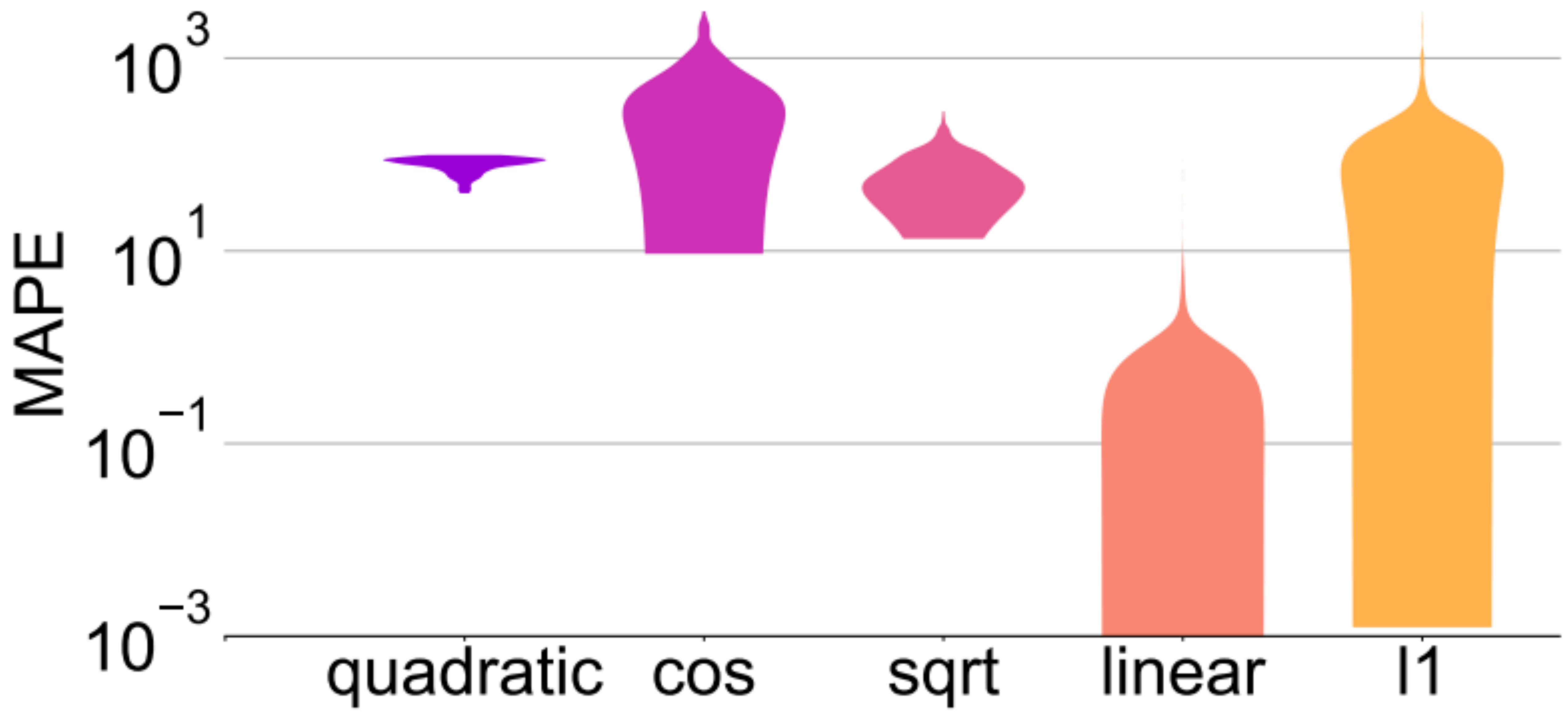





**Theorem (XZLDKJ'21)**
Let $f$ be a two-layer ReLU MLP trained by GD*. For any direction $v \in \mathbb{R}^d$, let $x = tv$. For any $h > 0$, as $t \to \infty$, $f(x + hv) - f(x) \to \beta_v h$ with rate $O(1/t)$

* *Assumption: NTK regime*

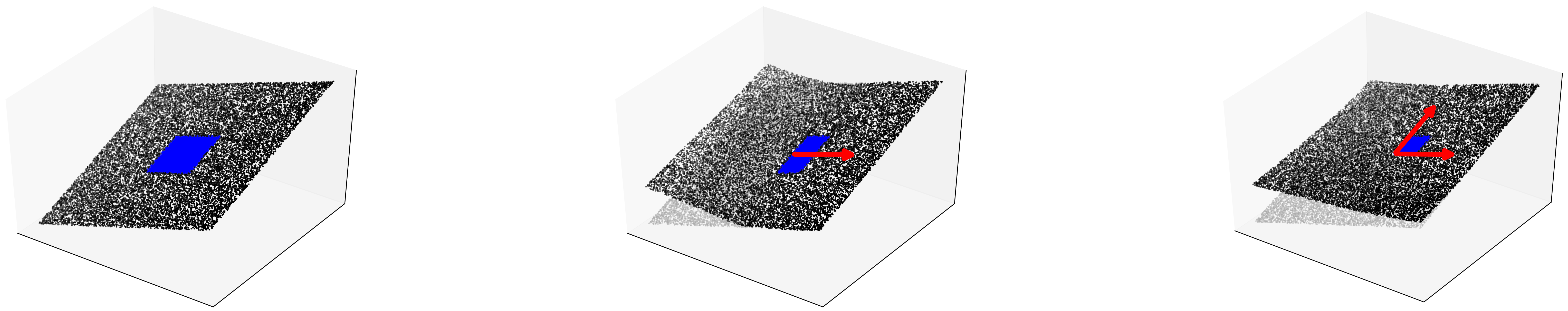
# Implication of linear extrapolation




MAPE extrapolation error: lower the better

* *Note: this does not follow from ReLU networks have finitely many linear egions, which only implies asymptotic behavior*

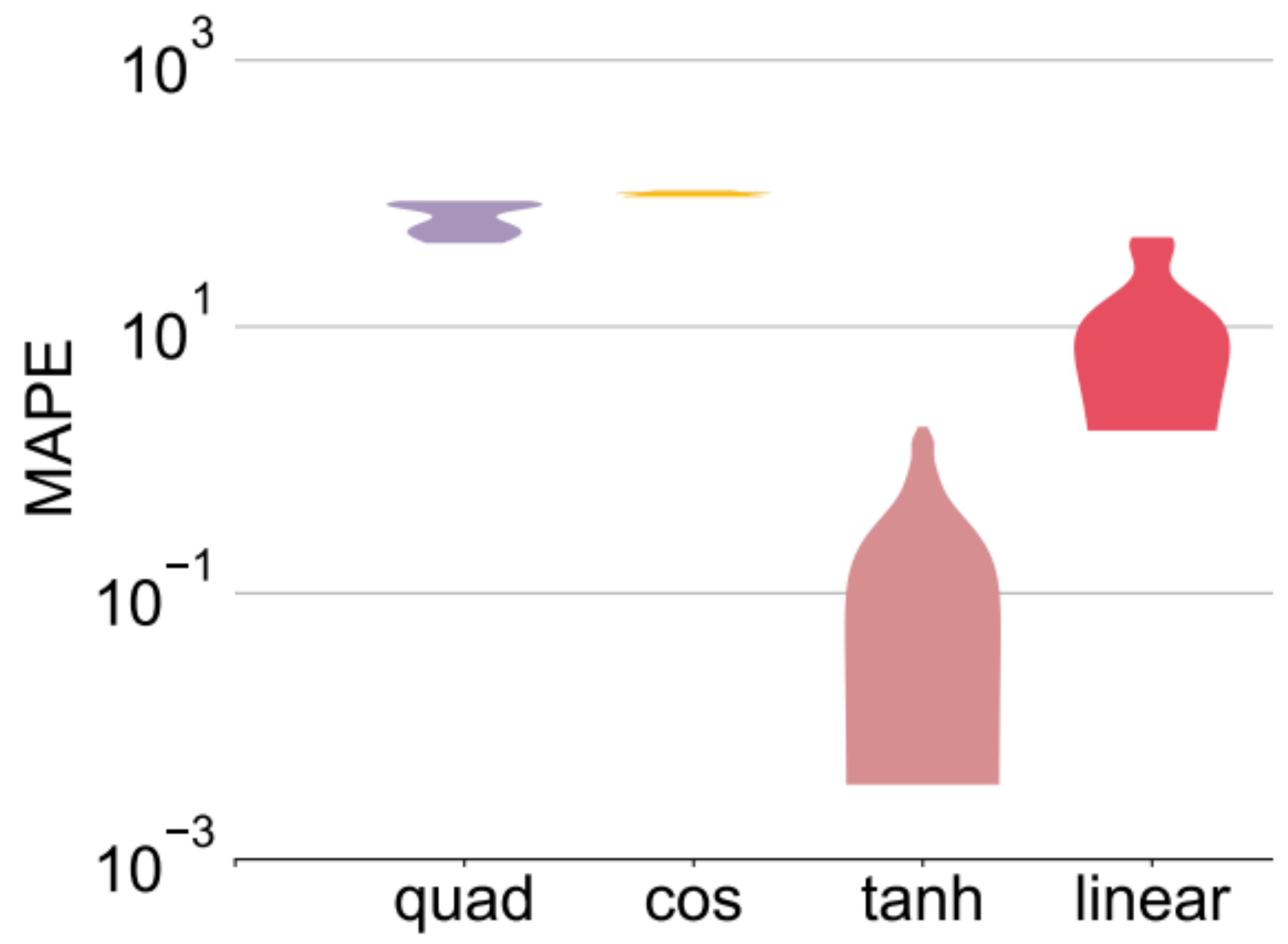
# Data geometry for learning linear functions



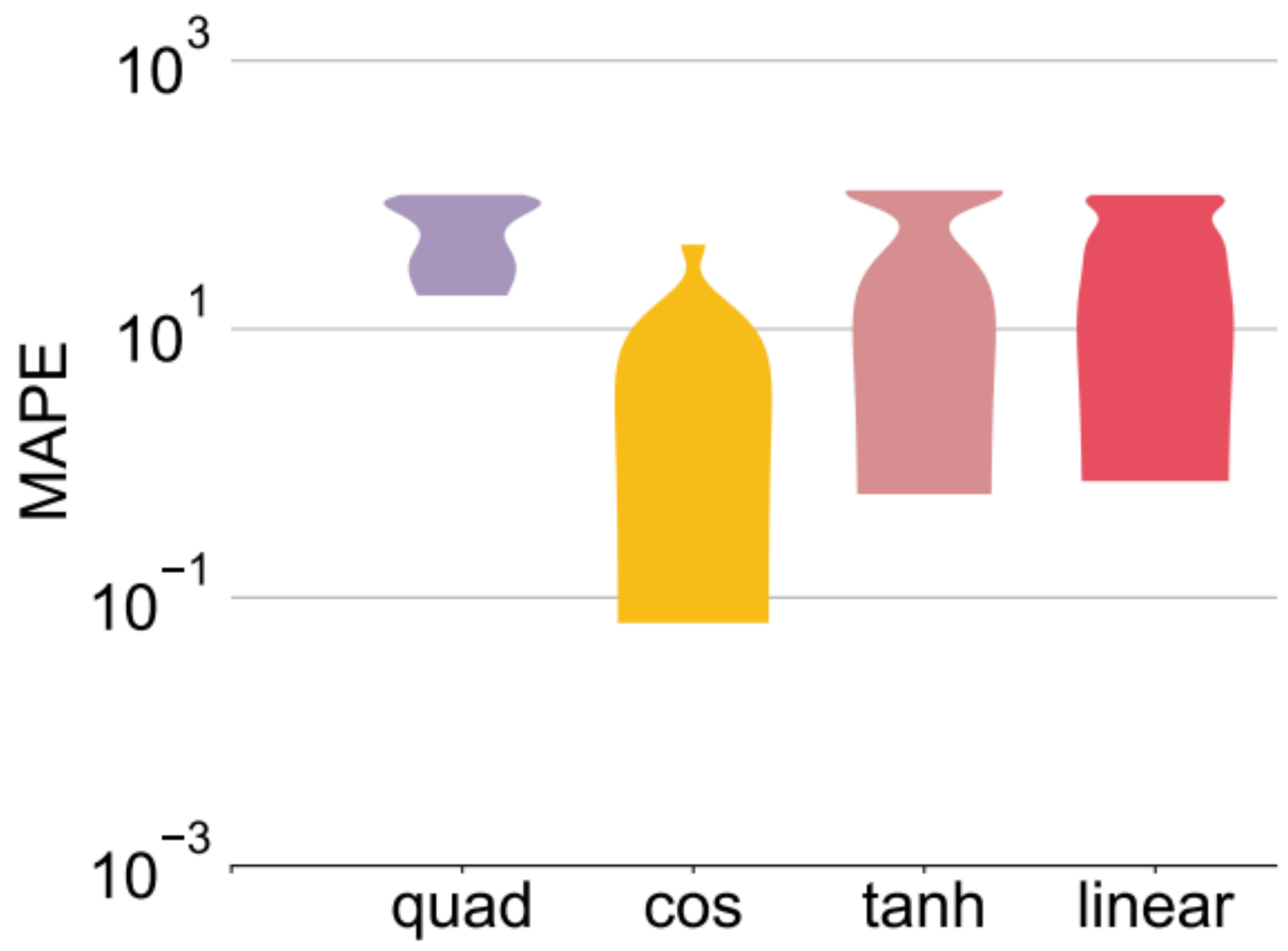
**Theorem (XZLDKJ'21)**
Let $f$ be a two-layer ReLU MLP trained by GD*. Suppose target function is $\beta^\intercal x$ and support of training distribution covers all directions. As the number of training examples $n \to \infty, f(x) \to \beta^\intercal x$.

* *Assumption: NTK regime*

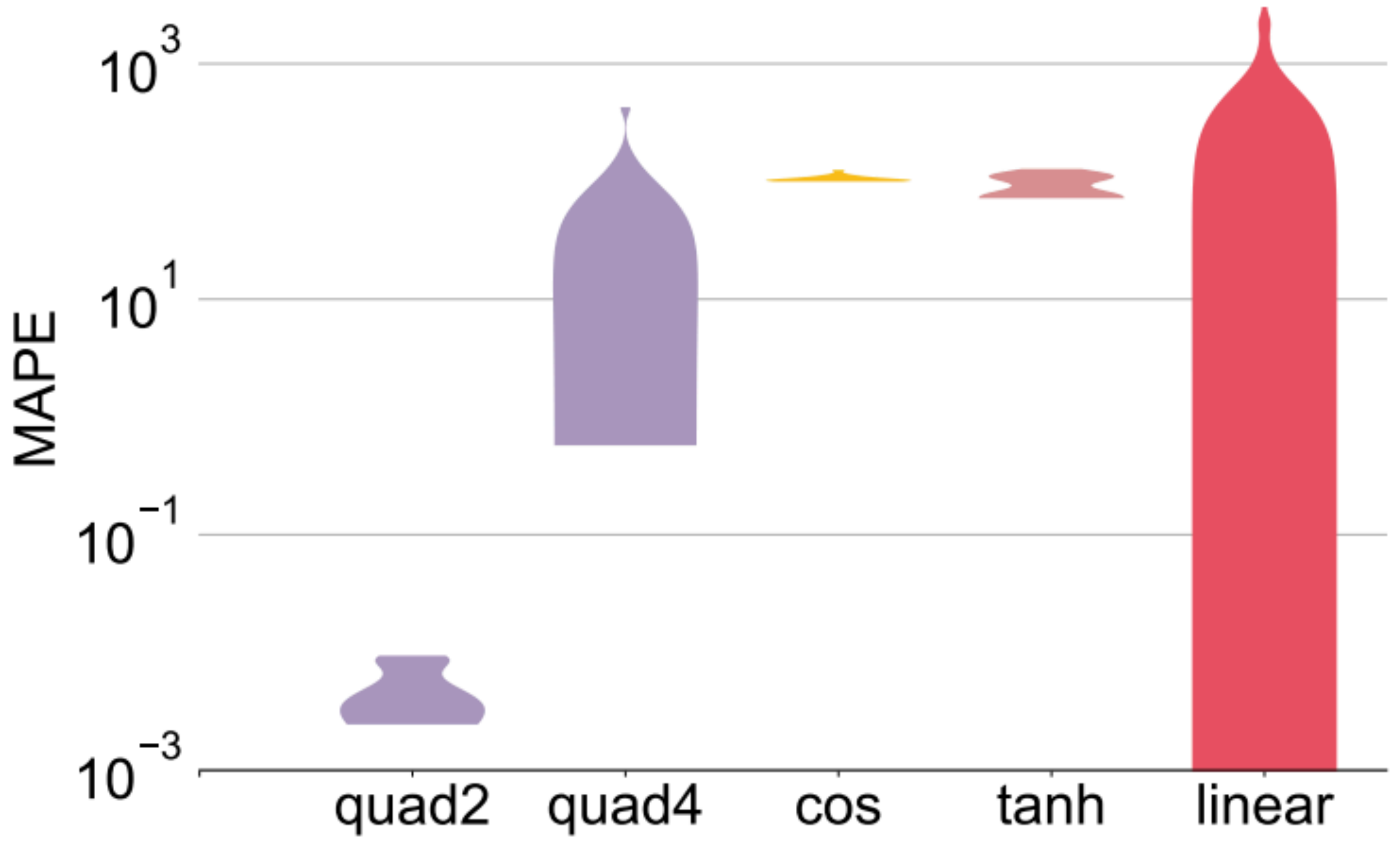
# Feedforward networks with other activation



(a) tanh activation



(b) cosine activation



(c) quadratic activation

Extrapolates well if activation is “similar” to target function

# Proof idea

Function implemented by NN after GD traininig:

*(Jacot et al 2018, Li and Liang 2018, Allen-Zhu et al 2019, Arora et al 2019ab, Du et al 2019)*

$$f_{\mathrm{NTK}}(\boldsymbol{x}) = (\mathrm{NTK}(\boldsymbol{x}, \boldsymbol{x}_1), \ldots, \mathrm{NTK}(\boldsymbol{x}, \boldsymbol{x}_n)) \cdot \mathrm{NTK}_{\mathrm{train}}^{-1} \boldsymbol{Y}$$

$$\mathrm{NTK}(\boldsymbol{x}, \boldsymbol{x}') = \mathop{\mathbb{E}}_{\boldsymbol{\theta} \sim \mathcal{W}} \left\langle \frac{\partial f(\boldsymbol{\theta}(t), \boldsymbol{x})}{\partial \boldsymbol{\theta}}, \frac{\partial f(\boldsymbol{\theta}(t), \boldsymbol{x}')}{\partial \boldsymbol{\theta}} \right\rangle$$

In functional form: equivalent to

$$f_{NTK}(\boldsymbol{x}) = \phi(\boldsymbol{x})^\top \boldsymbol{\beta}_{NTK}$$

~~training algorithm~~, architecture, data

$$\min_{\boldsymbol{\beta}} \|\boldsymbol{\beta}\|_2$$

$$s.t. \quad \phi(\boldsymbol{x}_i)^\top \boldsymbol{\beta} = y_i, \quad for\ i = 1, \ldots, n$$

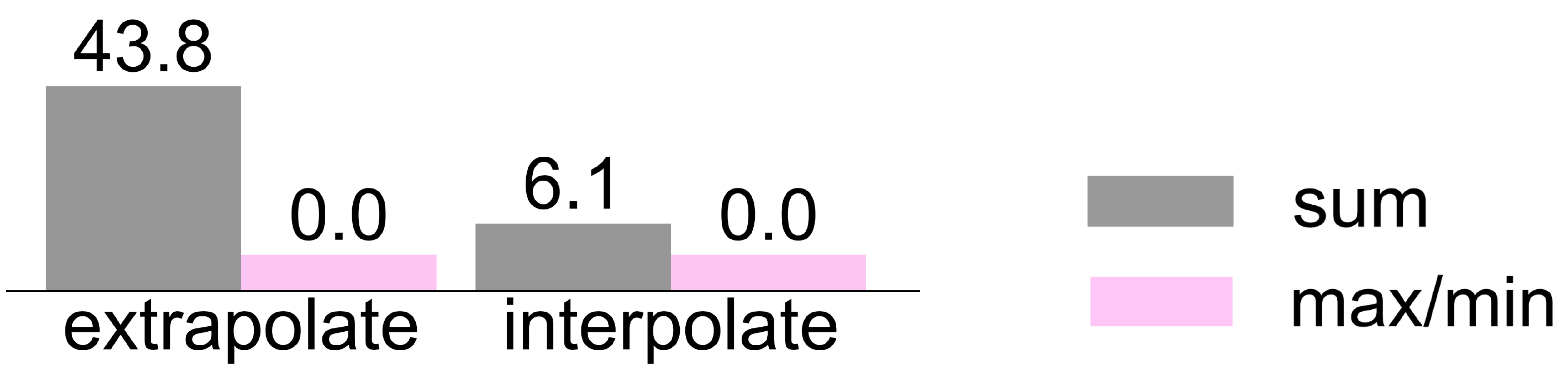
# Implications for GNNs

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

GNN (sum): $h_u^{(k)} = \Sigma_v \ \mathbf{MLP}^{(k)}\left(h_v^{(k-1)}, h_u^{(k-1)}, w(v,u)\right)$

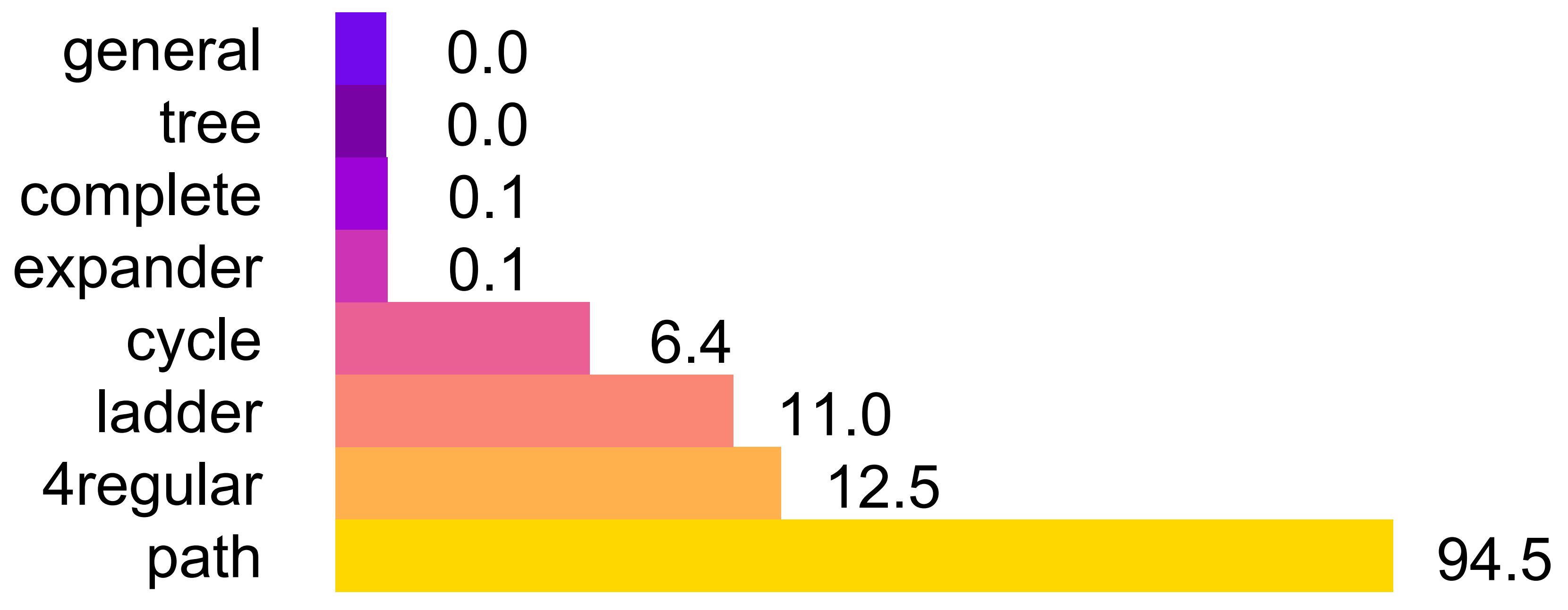
GNN that encodes the nonlinearity min

$$h_u^{(k)} = \mathbf{min}_v \ \mathbf{MLP}^{(k)}\left(h_v^{(k-1)}, h_u^{(k-1)}, w(v,u)\right)$$

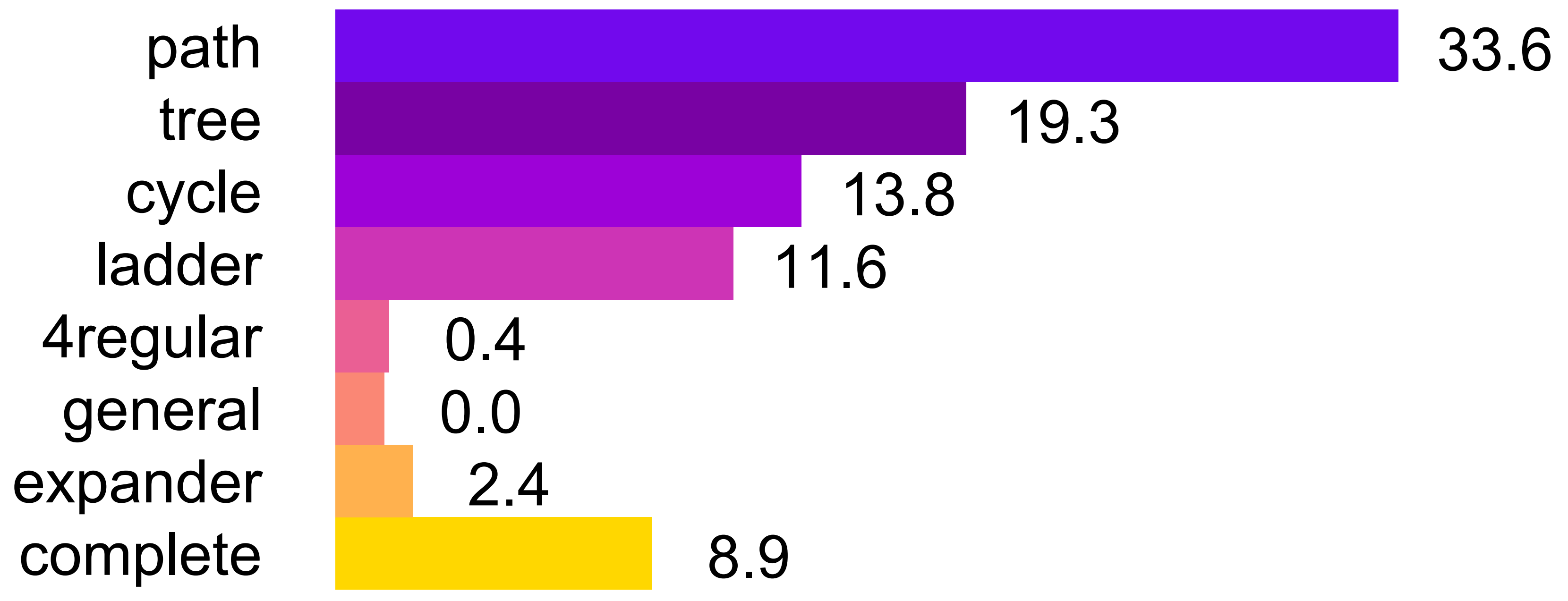


# Data distribution and architecture

**Data diversity:** feature direction (MLP), graph structure (GNN)




Max Degree




Shortest Path

**Theorem (XZLDKJ'21)**
A GNN encoding max in aggregation trained by GD* learns max degree if training data $\{\deg_{\max}(G_i), \deg_{\min}(G_i), N_i^{\max} \deg_{\max}(G_i), N_i^{\min} \deg_{\min}(G_i)\}_{i=1}^{n}$ spans $\mathbb{R}^4$.

* *Assumption: NTK regime*

# Linear algorithmic alignment

**Linear algorithmic alignment (XZLDKJ'21)**
Network can simulate underlying function via ~~easy-to-learn~~ *linear* "modules"

**Hypothesis:** Linear algo alignment helps *extrapolation*

**Application:** Encode nonlinearity in **architecture** or **input representation**.

# Encoding nonlinearities in architecture

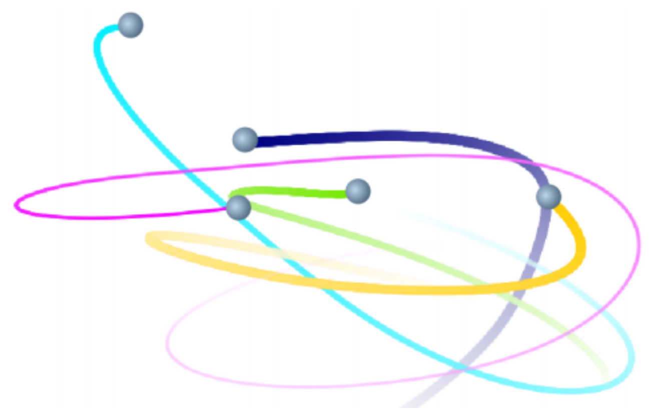
*Activation, pooling, symbolic operations etc…*

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

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

Encode **exp log** for learning multiplication

*(Trask et al. 2018, Madsen & Johansen 2020)*



$$\vec{a}_i = \frac{C}{M_i} \sum_{j \neq i} (1 - r_{ij}) \hat{r}_{ij}$$

Symbolic output

*(Cranmer et al 2020)*

**Q:** What direction is the closest creature facing?

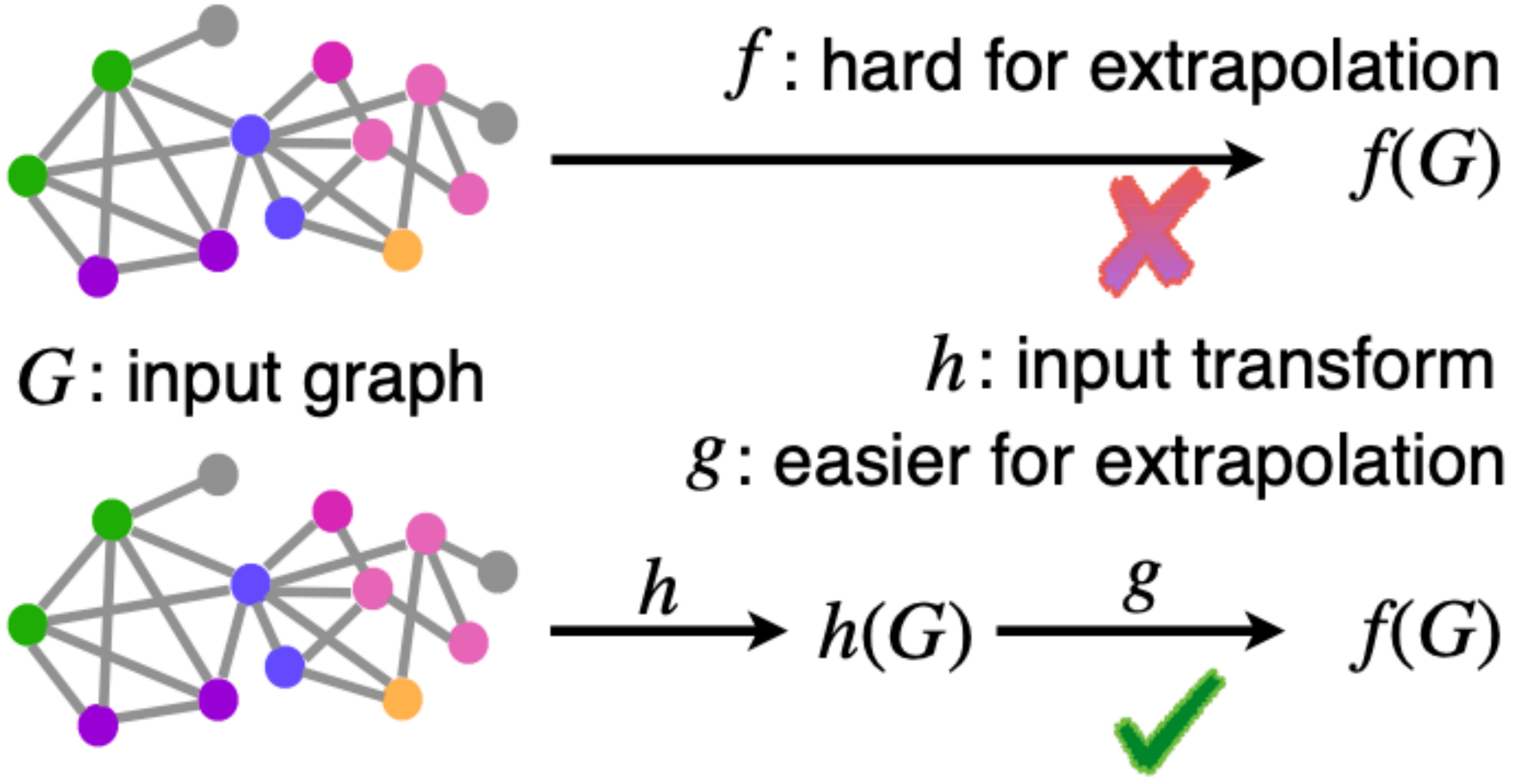
**P:** `scene, filter_creature, filter_closest, unique, query_direction`

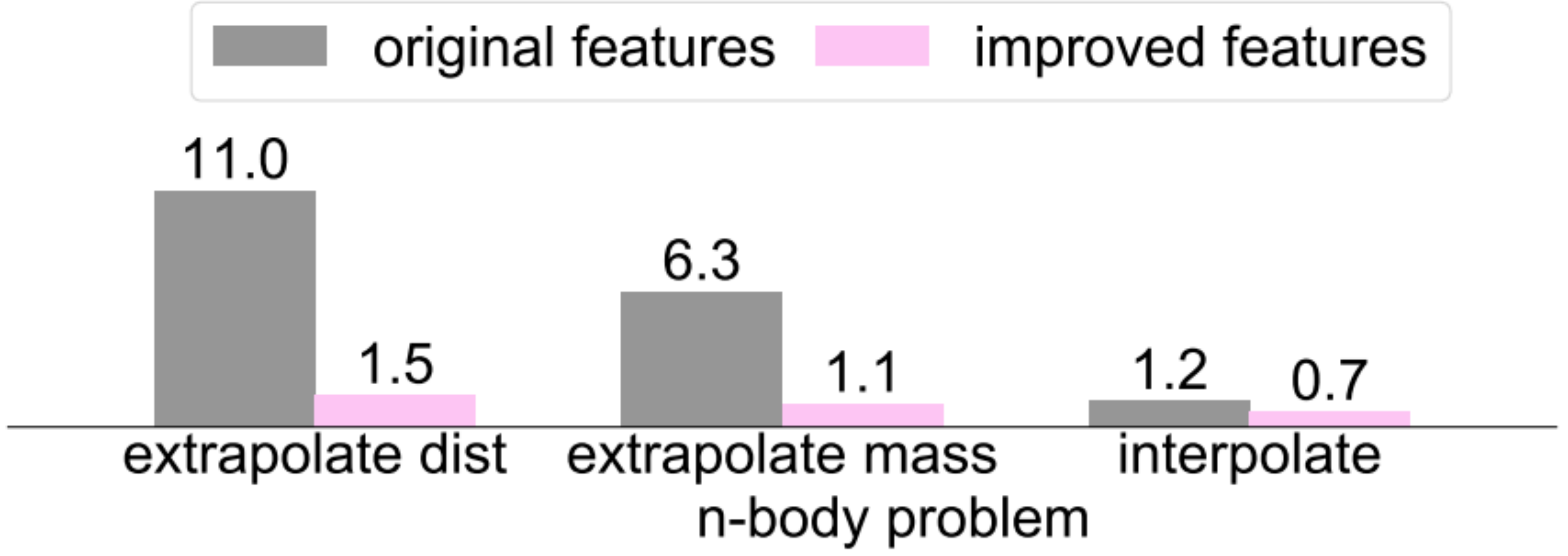
**A:** left

Encode **a library of programs** (~2K)

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

# Encoding nonlinearities in input representation




Specialized features, feature transformation

Representation learning with out-of-distribution data (e.g., BERT)

# Summary

1. Interpolation: alignment of task and network structure
2. How feedforward networks extrapolate
3. Linear algorithmic alignment for structured networks, e.g., GNNs

*What Can Neural Networks Reason About? K. Xu, J. Li, M. Zhang, S. S. Du, K. Kawarabayashi, S. Jegelka. ICLR 2020*

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