How Neural Networks Extrapolate: From Feedforward to Graph Neural Networks

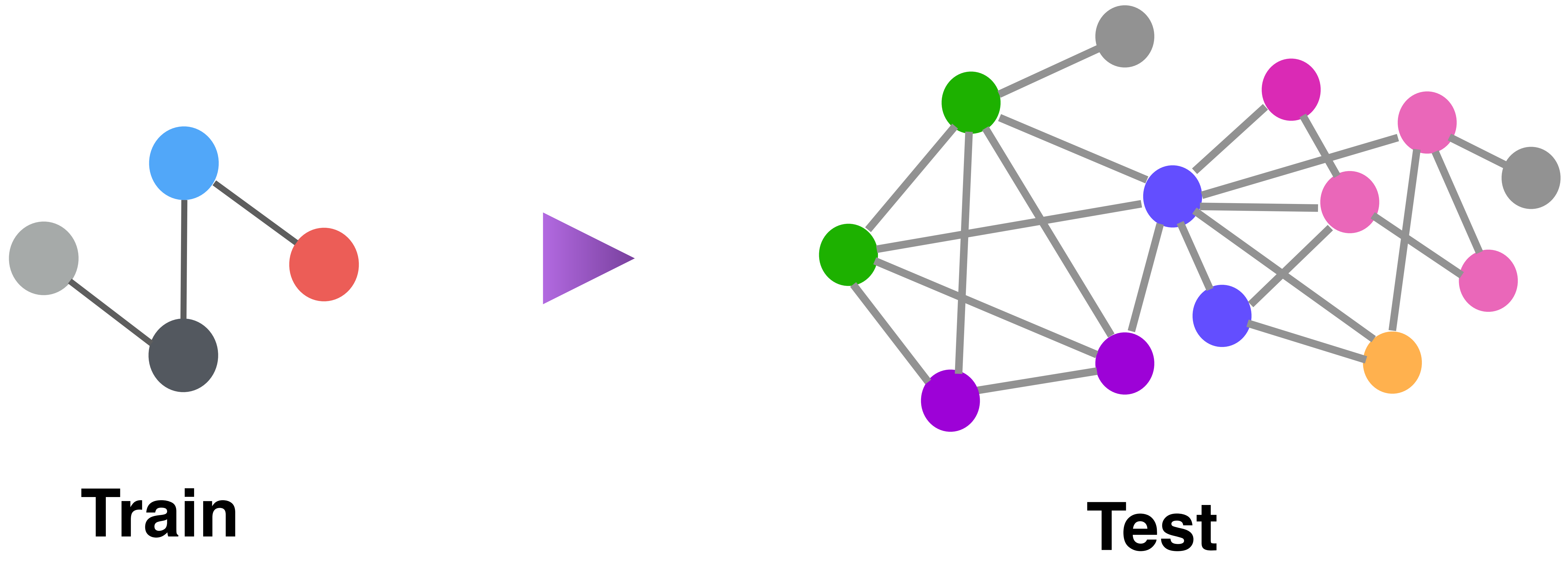
Keyulu Xu, Mozhi Zhang, Jingling Li, Simon S. Du

Ken-ichi Kawarabayashi, Stefanie Jegelka

Extrapolation

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

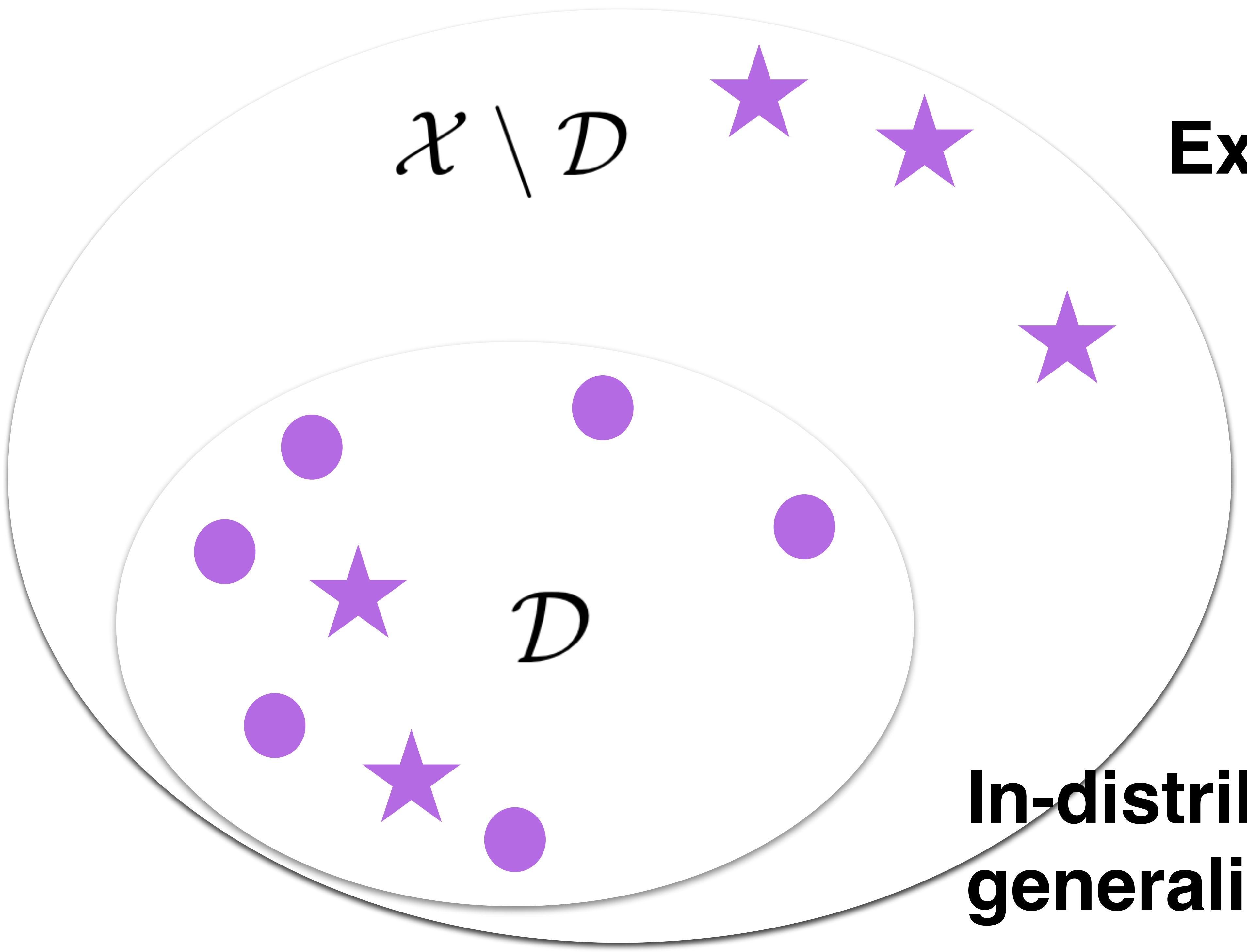
$$\mathbb{E}_{\boldsymbol{x} \sim \mathcal{P}_{\text{test}}}[\ell(f(\boldsymbol{x}), g(\boldsymbol{x}))]$$



Extrapolation

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

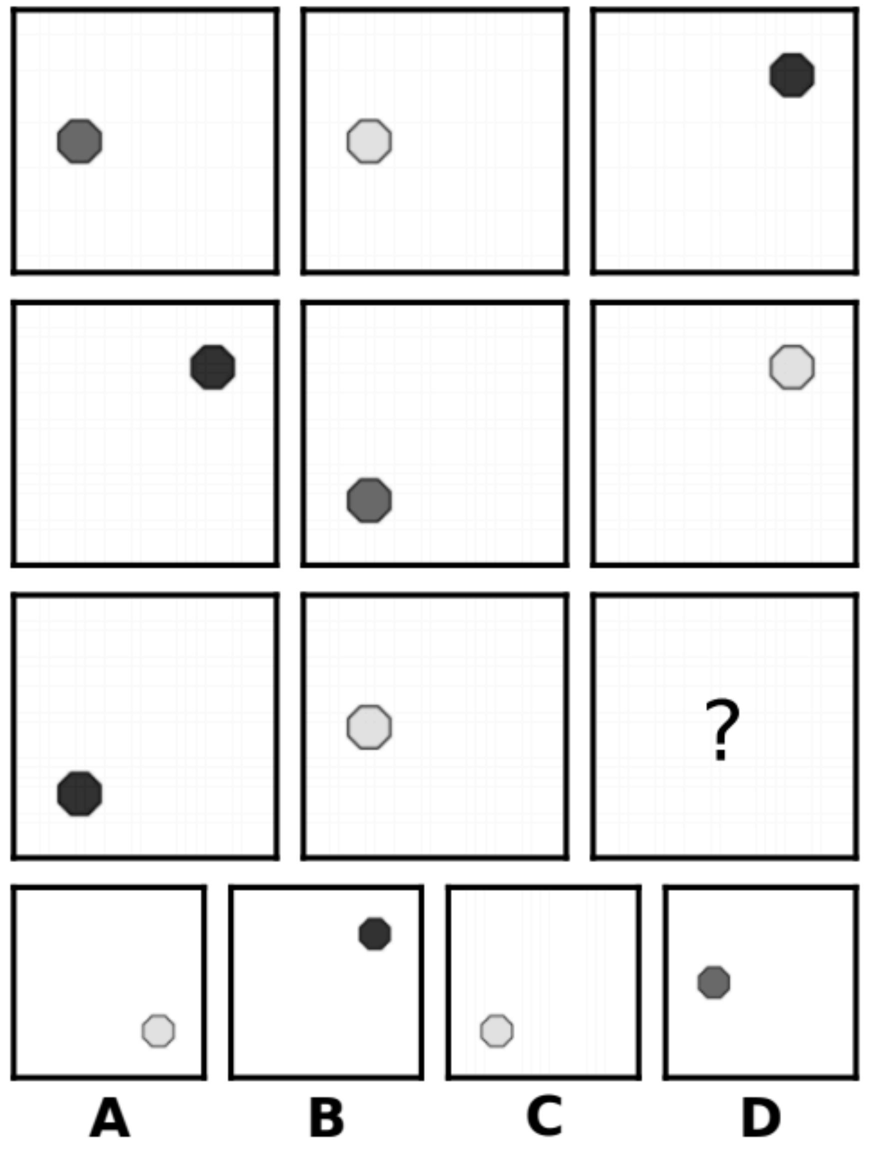
$$\mathbb{E}_{\boldsymbol{x} \sim \mathcal{P}_{\text{test}}}[\ell(f(\boldsymbol{x}), g(\boldsymbol{x}))]$$

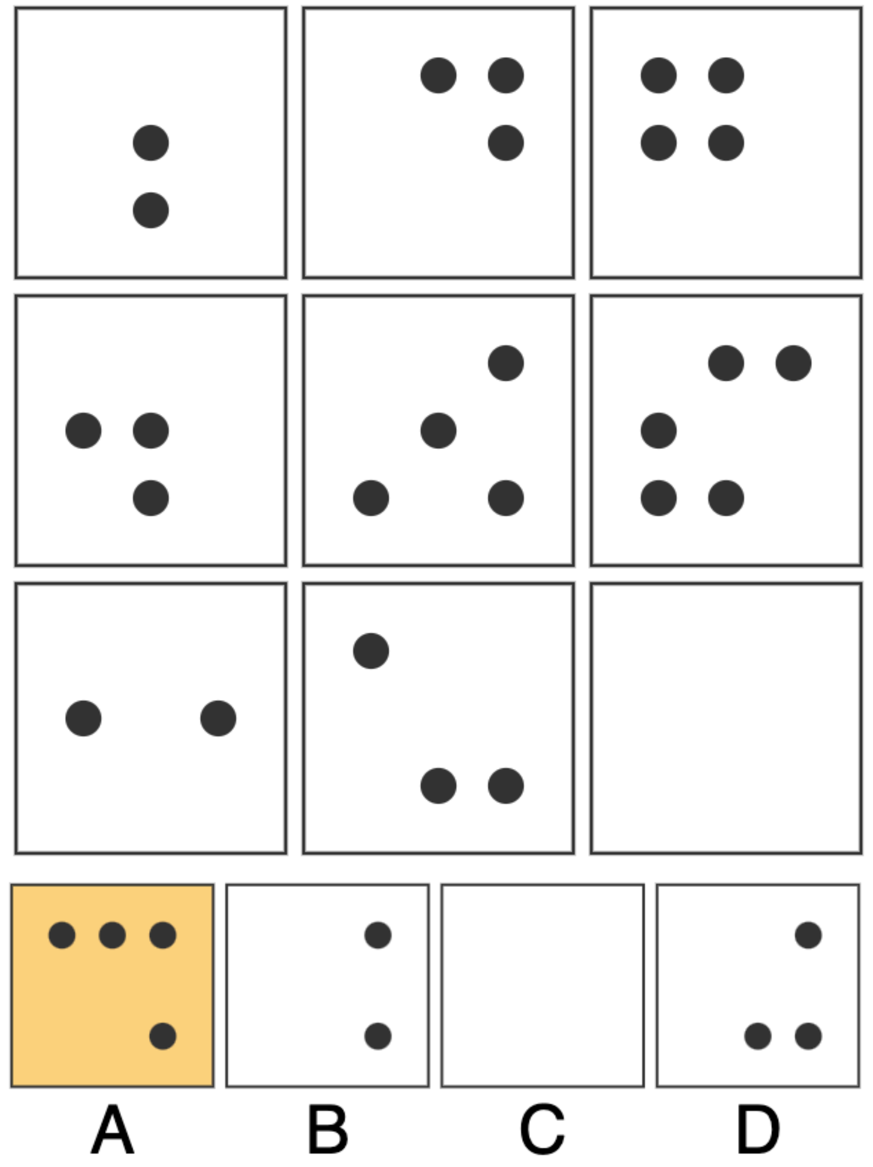


Extrapolation

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

Evaluation of NN extrapolation





IQ tests
shape, color, number of objects

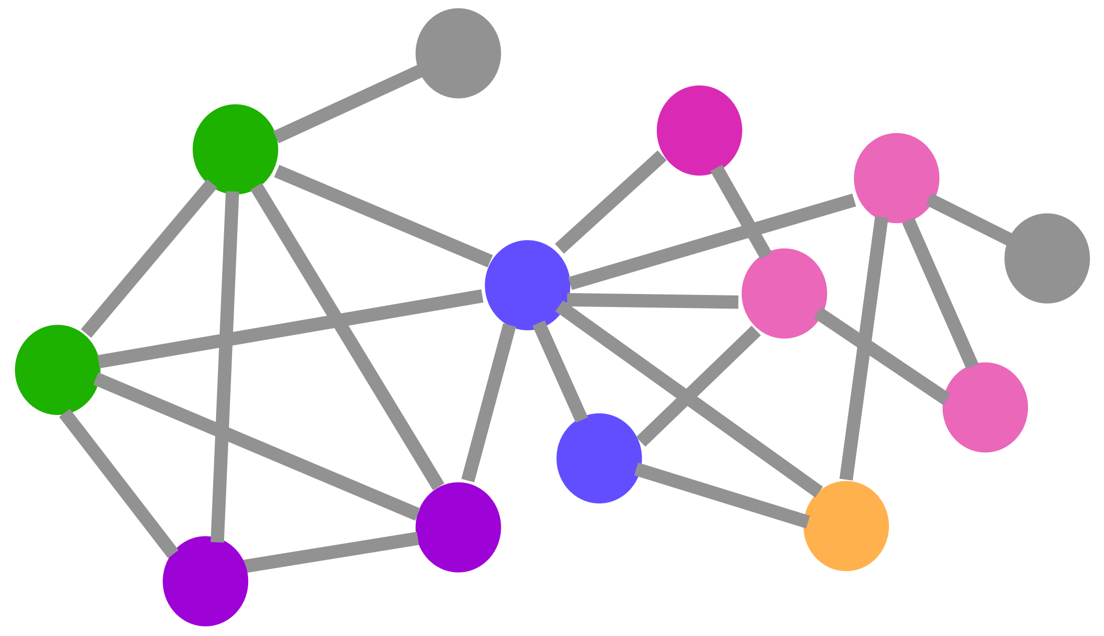
(Santoro et al. 2018, Zhang et al 2019)

Physical reasoning
position, mass, number of objects

(Wu et al. 2017, Battagalia et al 2016, Janner et al 2019, Kramer et 2020)

Evaluation of NN extrapolation

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)

Puzzle

Feedforward NN (multilayer perceptron)

Similarly for CNN, RNN etc

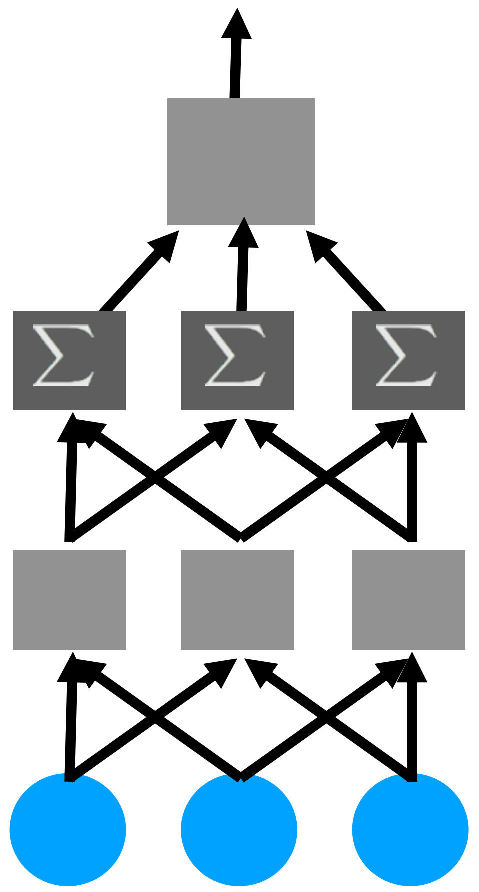
$$\mathrm{MLP}(\boldsymbol{x}) = \boldsymbol{W}^{(d)} \cdot \sigma\left(\boldsymbol{W}^{(d-1)}\sigma\left(...\sigma\left(\boldsymbol{W}^{(1)}\boldsymbol{x}\right)\right)\right)$$

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

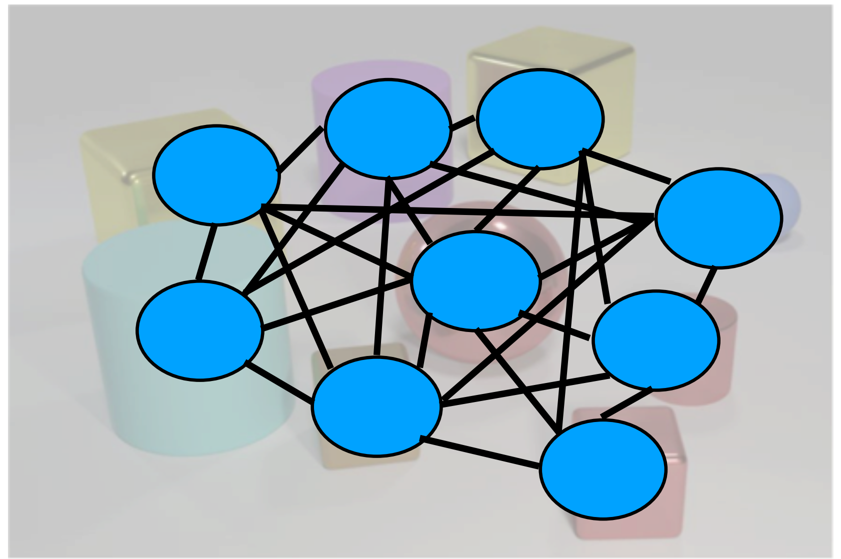
Graph neural network (GNN)

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

variants of GNN architectures may be used



MLP modules



$\boldsymbol{h}_u^{(k)}$

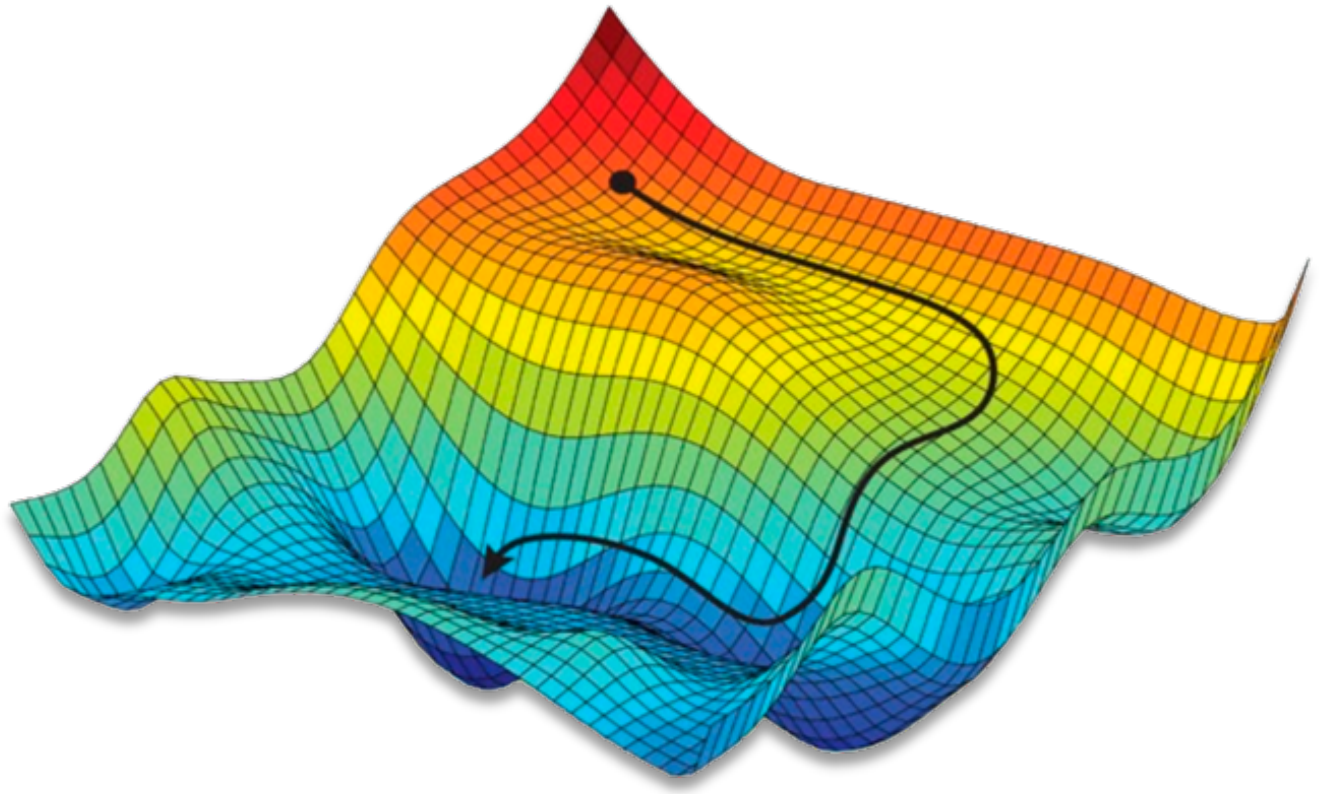
some success

How neural networks extrapolate

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**

training algorithm	architecture	data
GD	*GNN*	*feature geometry*
	MLP	*graph structure*



Parameter trajectory

$\theta(t)$

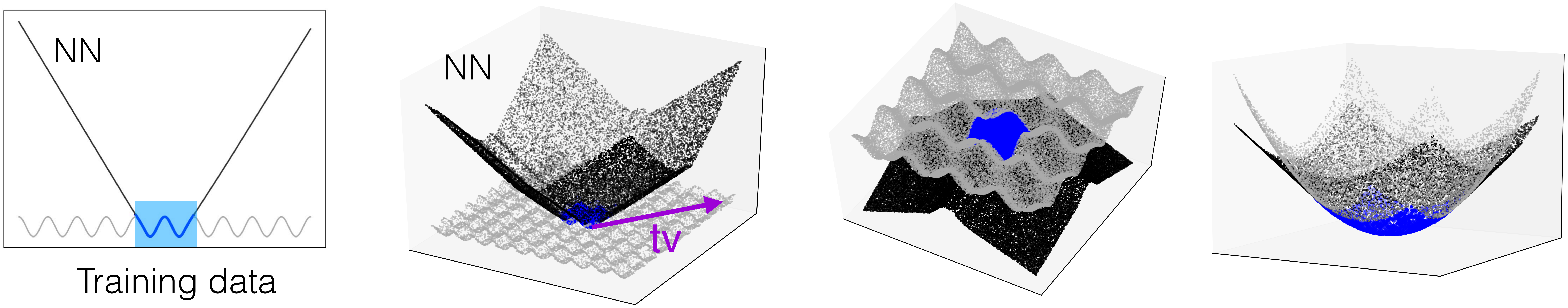
Theory: Gradient descent training in NTK regime

Wide NN trained by GD = kernel GD with NTK

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

Experiments: same conclusion **in regular regimes**

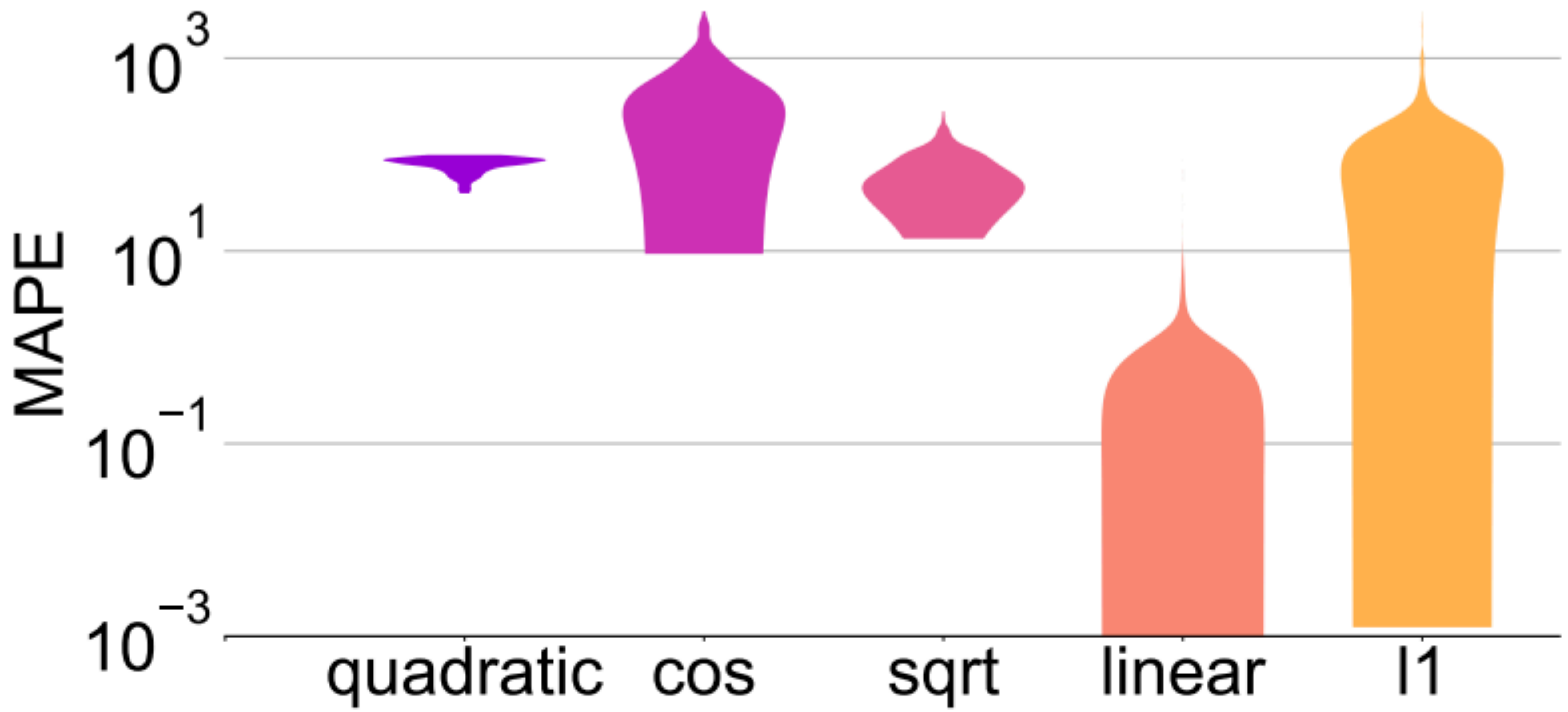
Linear extrapolation behavior of ReLU MLPs



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*

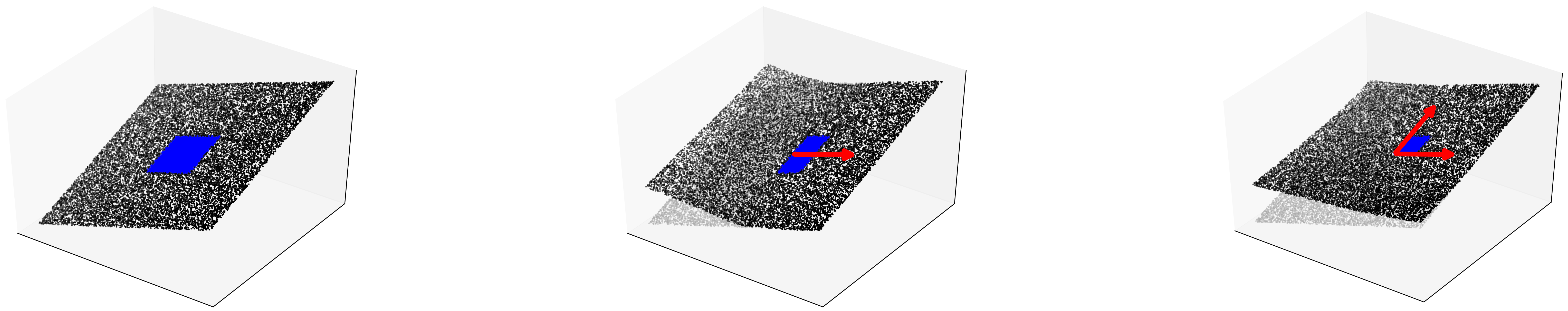
Implication of linear extrapolation



MAPE extrapolation error: lower the better

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

Provable learning of linear functions with diverse training data



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*

Provable learning of linear functions II

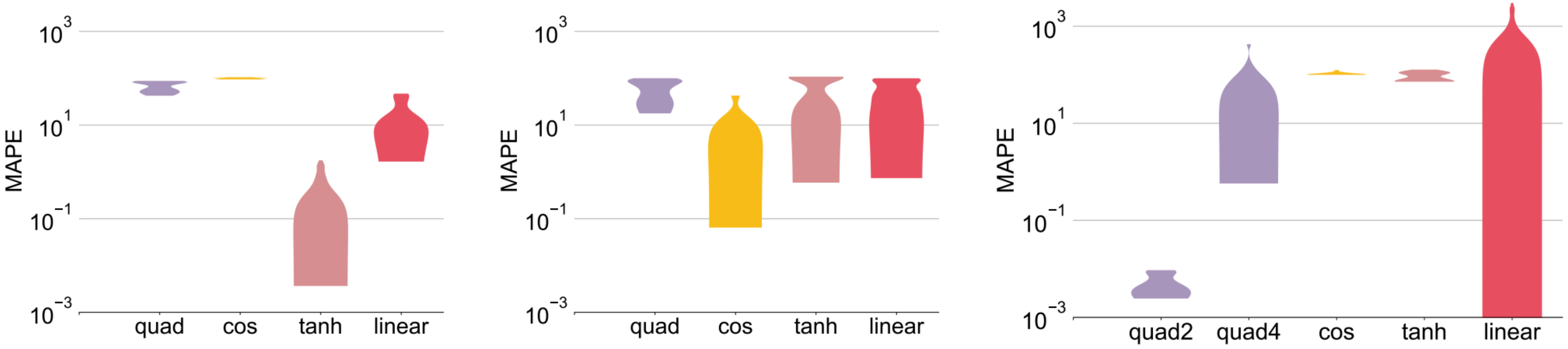
Provable extrapolation with **2d diverse training data**

* *mainly of theoretical interest*

Lemma (XZLDKJ'21)
Let f be a two-layer ReLU MLP trained by GD*. Suppose target function is $\beta^{\intercal} x$ and training set contains an orthogonal basis and their opposite vectors, then $f(x) = \beta^{\intercal} x$.

* *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

Implications for GNNs

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

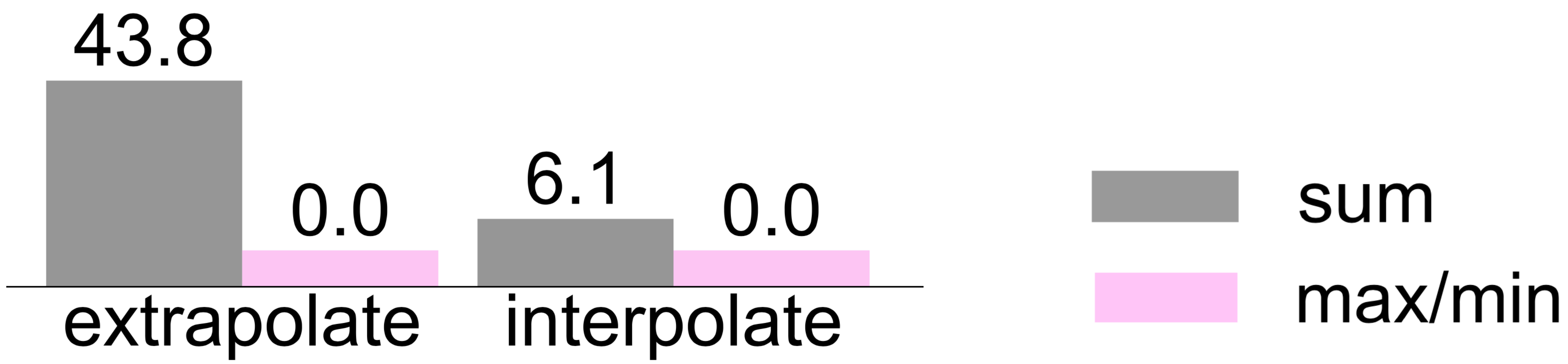
GNN (sum): $h_u^{(k)} = \boxed{\Sigma_v}\ \mathbf{MLP}^{(k)}\left(h_v^{(k-1)}, h_u^{(k-1)}, w(v,u)\right)$

✗ *MLP has to learn non-linear steps*

GNN that encodes the nonlinearity min

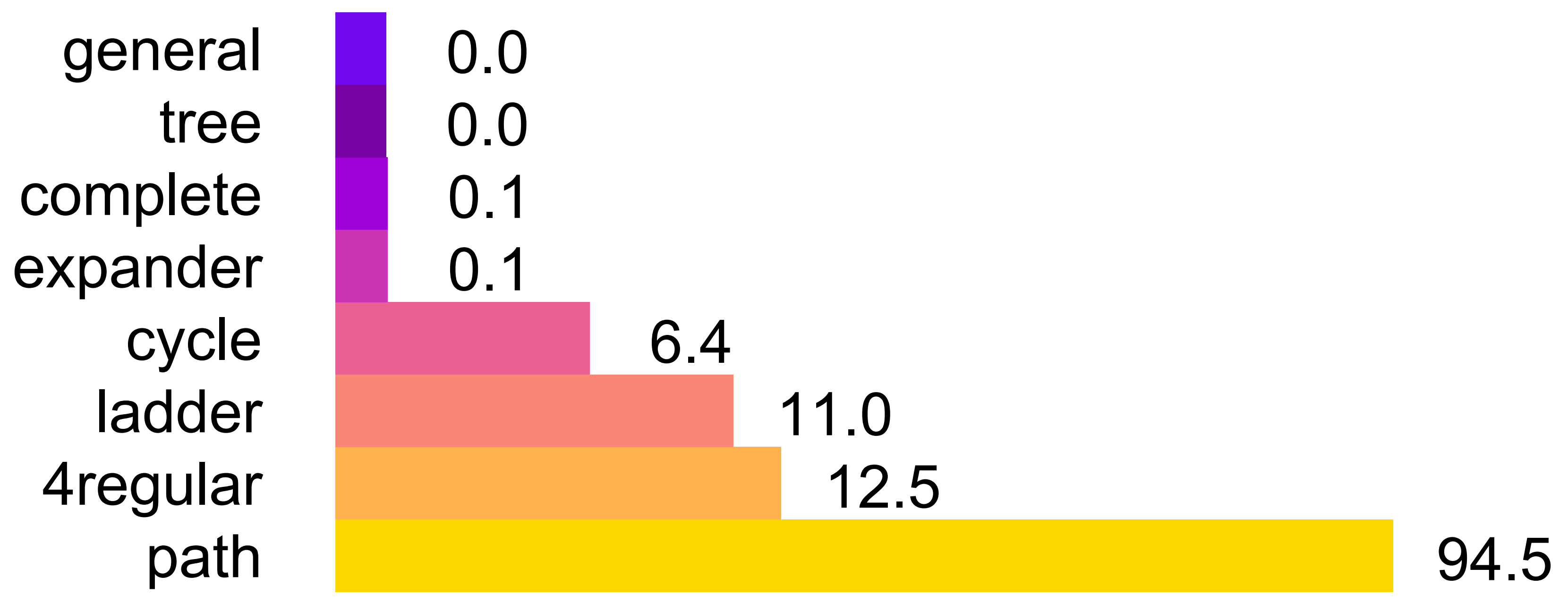
$$h_u^{(k)} = \boxed{\mathbf{min_v}}\ \mathbf{MLP}^{(k)}\left(h_v^{(k-1)}, h_u^{(k-1)}, w(v,u)\right)$$

✓ *MLP learns linear steps*

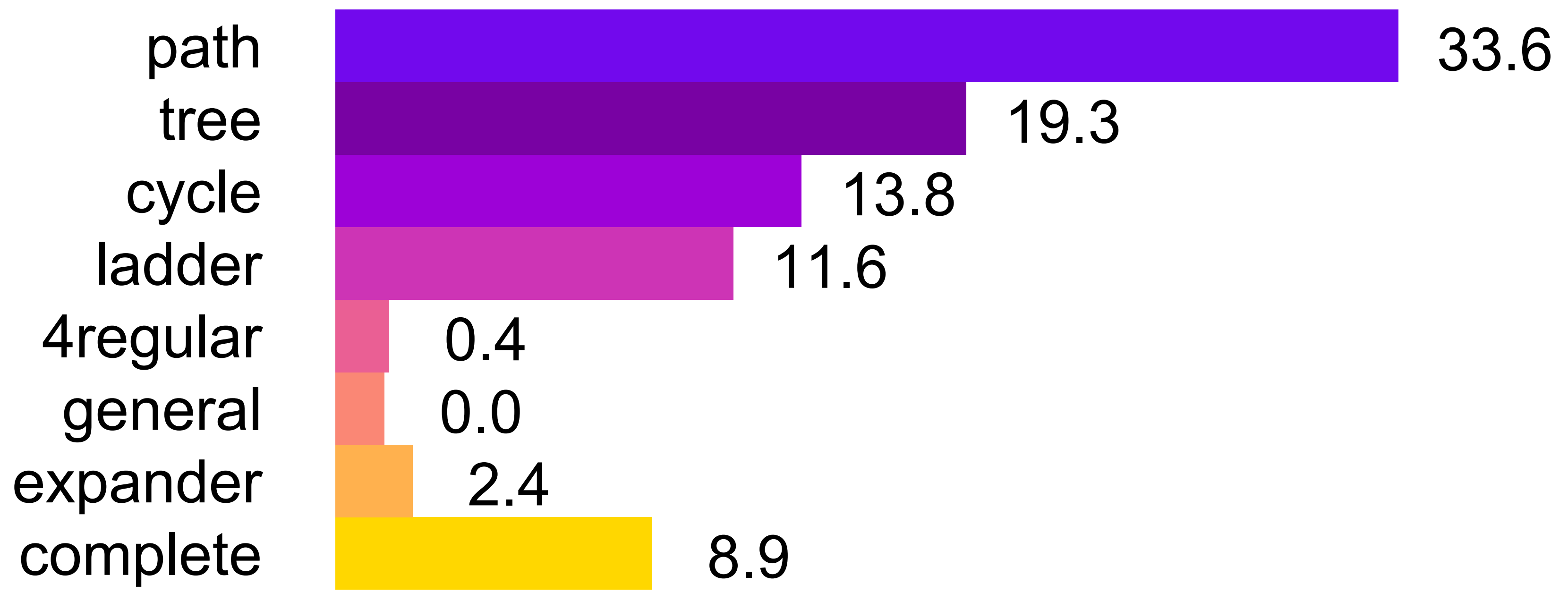


Provable extrapolation: nonlinearity and data distribution

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*

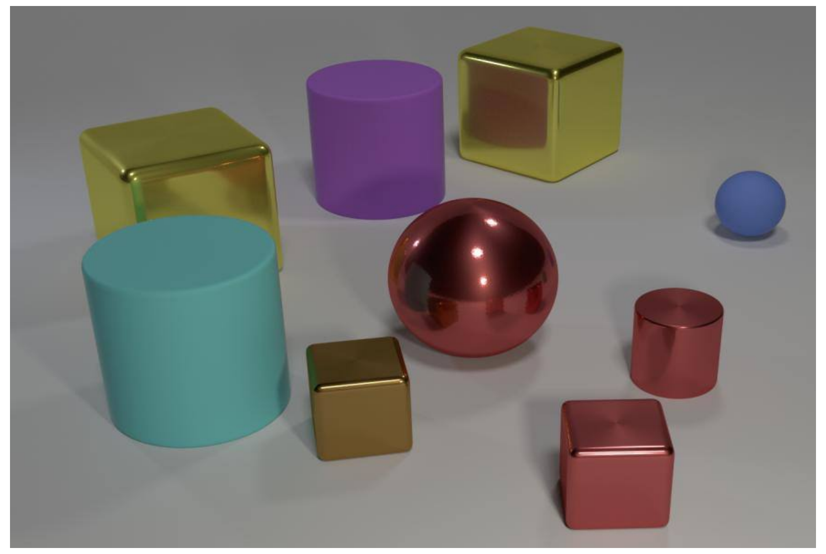
GNNs can extrapolate DP (under conditions)

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

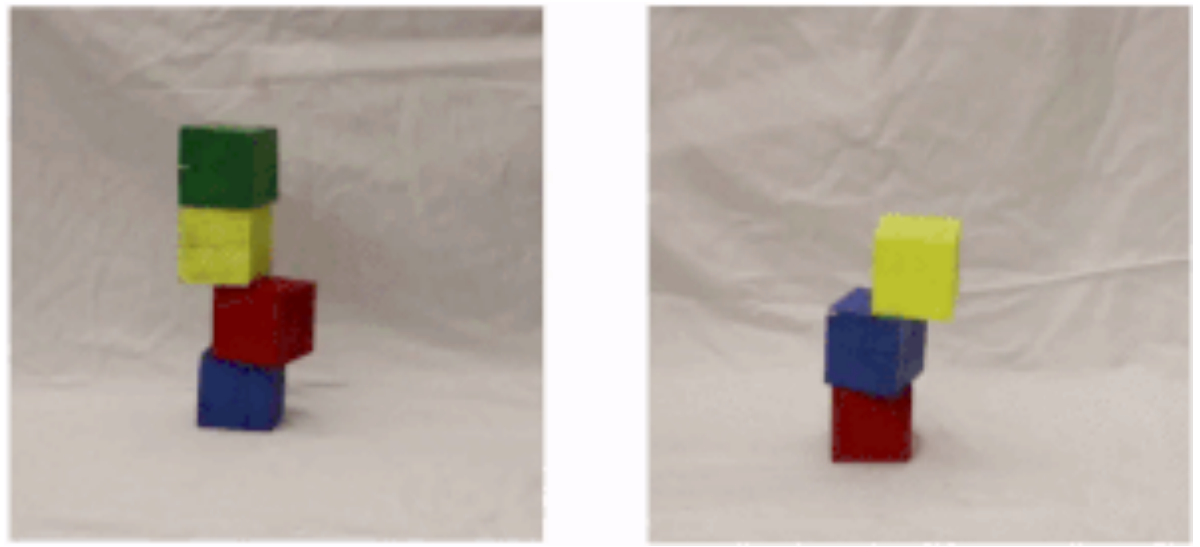
$$h_s^{(k)} = \sum\nolimits_{t \in S} \text{MLP}_1^{(k)} \left(h_s^{(k-1)}, h_t^{(k-1)} \right)$$

Reasoning tasks as dynamic programming (DP):

graph algorithms



visual question answering



Intuitive physics

Linear algorithmic alignment

Linear algorithmic alignment (XZLDKJ'21)
Network can simulate underlying function via *linear* "modules".

Hypothesis: Linear algo alignment helps *extrapolation.*

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

* *Interpolation version (Xu et al 2020): align with easy-to-learn (possibly nonlinear) modules*

Encoding nonlinearities in architecture

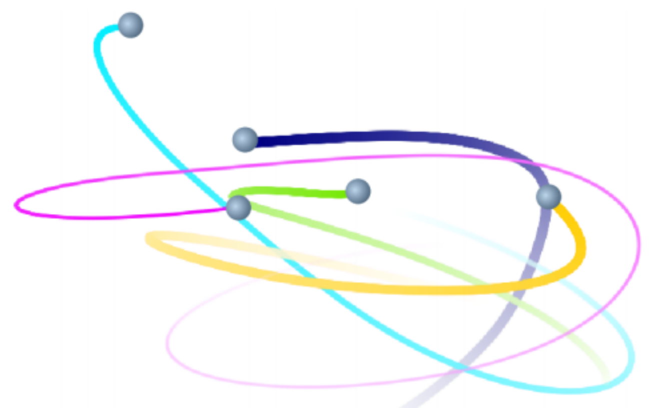
Activation, pooling, symbolic operations etc…

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

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

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?

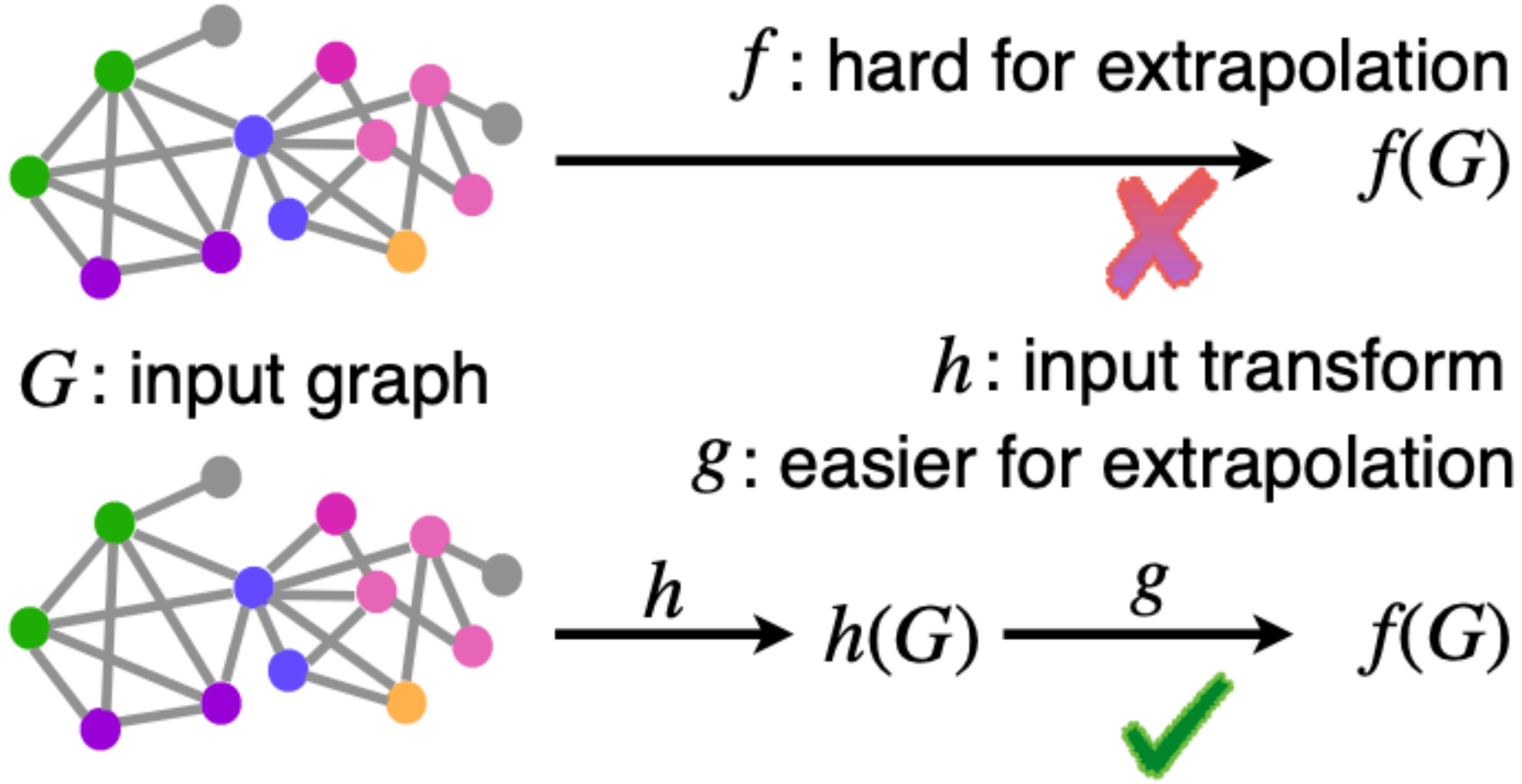
P: `scene, filter_creature, filter_closest, unique, query_direction`

A: left

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

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

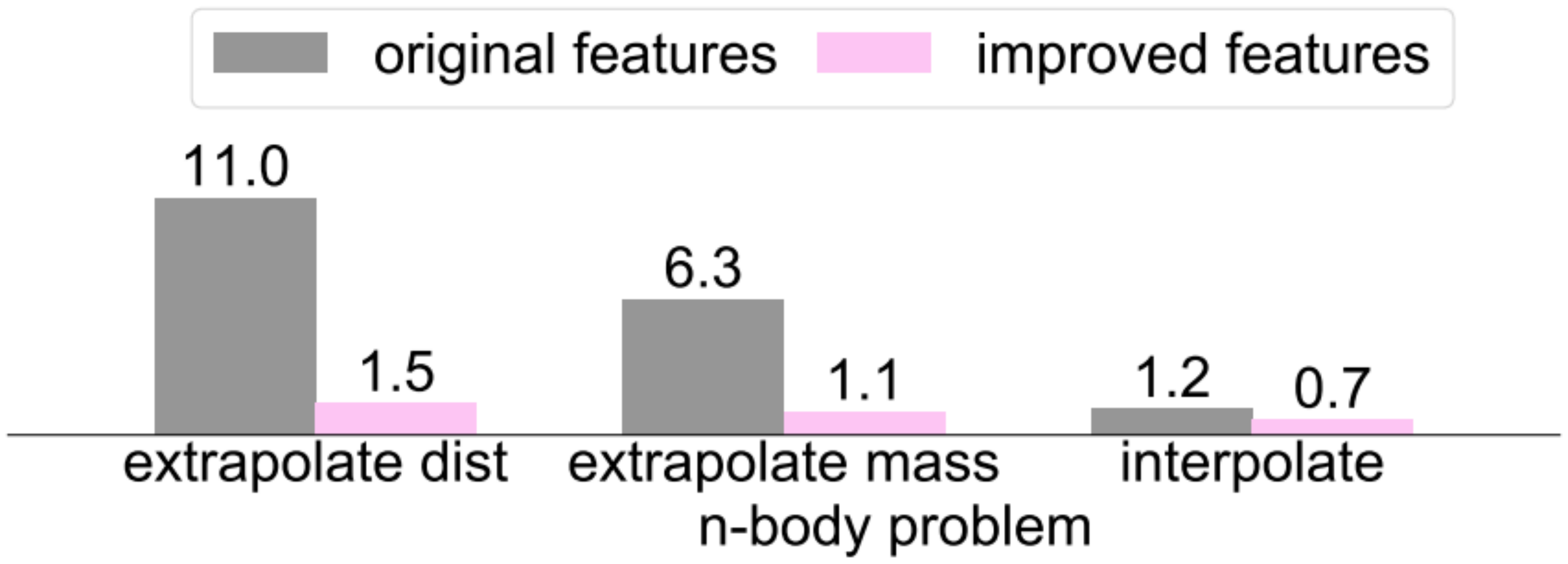
Encoding nonlinearities in input representation



Specialized features, feature transformation

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

Encoding nonlinearities in input representation



Generalization across **languages** *(Mikolov et al 2013, Zhang et al. 2019, Devlin et al 2019, Wu et al 2019, Yuan et al 2020)*

Symbolic **mathematics** *(Lample et al 2020)*

Quantitative **finance** *(Fama & French 1993, Banz 1981, Ross 1976)*

Summary

1. **Linear extrapolation** of ReLU MLPs (non-asymptotic analysis)

2. Provable learning of linear functions with **diverse training data**

3. **Linear algorithmic alignment** for structured networks, e.g., GNNs

Code & slides:

https://people.csail.mit.edu/keyulux/