

Lecture 6:

Message Passing, Symmetries and Reasoning

Kimon Fountoulakis



UNIVERSITY OF
WATERLOO

DAVID R. CHERITON SCHOOL
OF COMPUTER SCIENCE

Data model

Our data $(x, y) \sim P$,

where P is defined over $\mathcal{X} \times \mathcal{Y}$, where \mathcal{X} and \mathcal{Y} are data and label domains.

Typical example: $\mathcal{X} = \mathbb{R}^d$ for some large d .

Training data

We are given a dataset \mathcal{D} of N observations

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N, \text{ which are drawn i.i.d}$$

Problem setting

We assume there exists a function f , such that

$$f(x_i) = y_i$$

and we are given a parameterized class of models

$$\mathcal{F} = \{f_{\theta \in \Theta}\}$$

Usually, θ corresponds to parameter weights in a neural network.

Problem setting

Pick an $\tilde{f} \in \mathcal{F}$, such that

$$\tilde{f}(x_i) = y_i$$

for all $i \in [N]$

and hopefully, works well on unseen data too.

Performance measure

For some loss function L , we define the risk:

$$\mathcal{R}(\tilde{f}) = \mathbb{E}_{x \sim P} L(\tilde{f}(x), f(x))$$

Typical example for the loss: $L(y, y') = \frac{1}{2} |y - y'|^2$

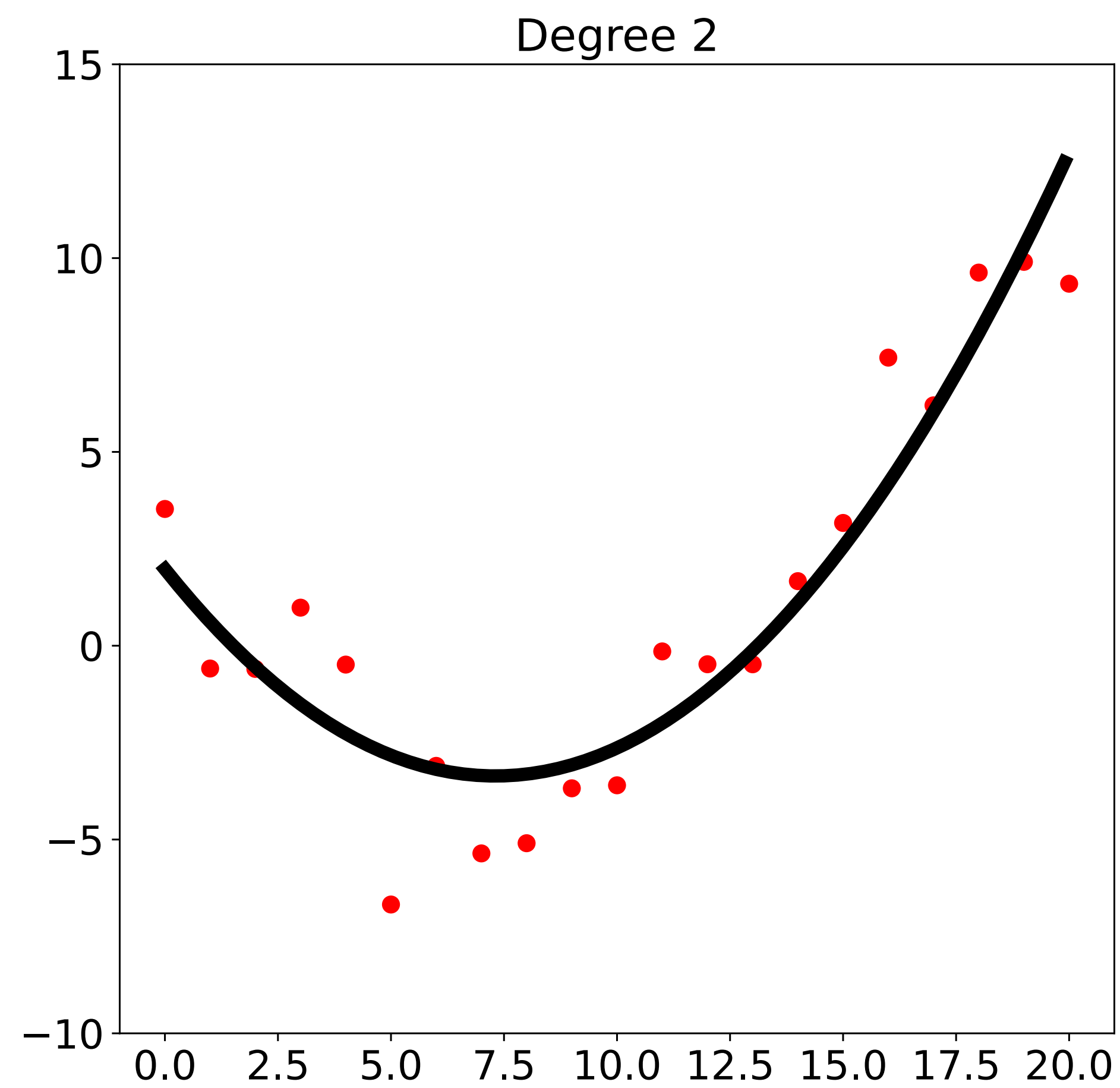
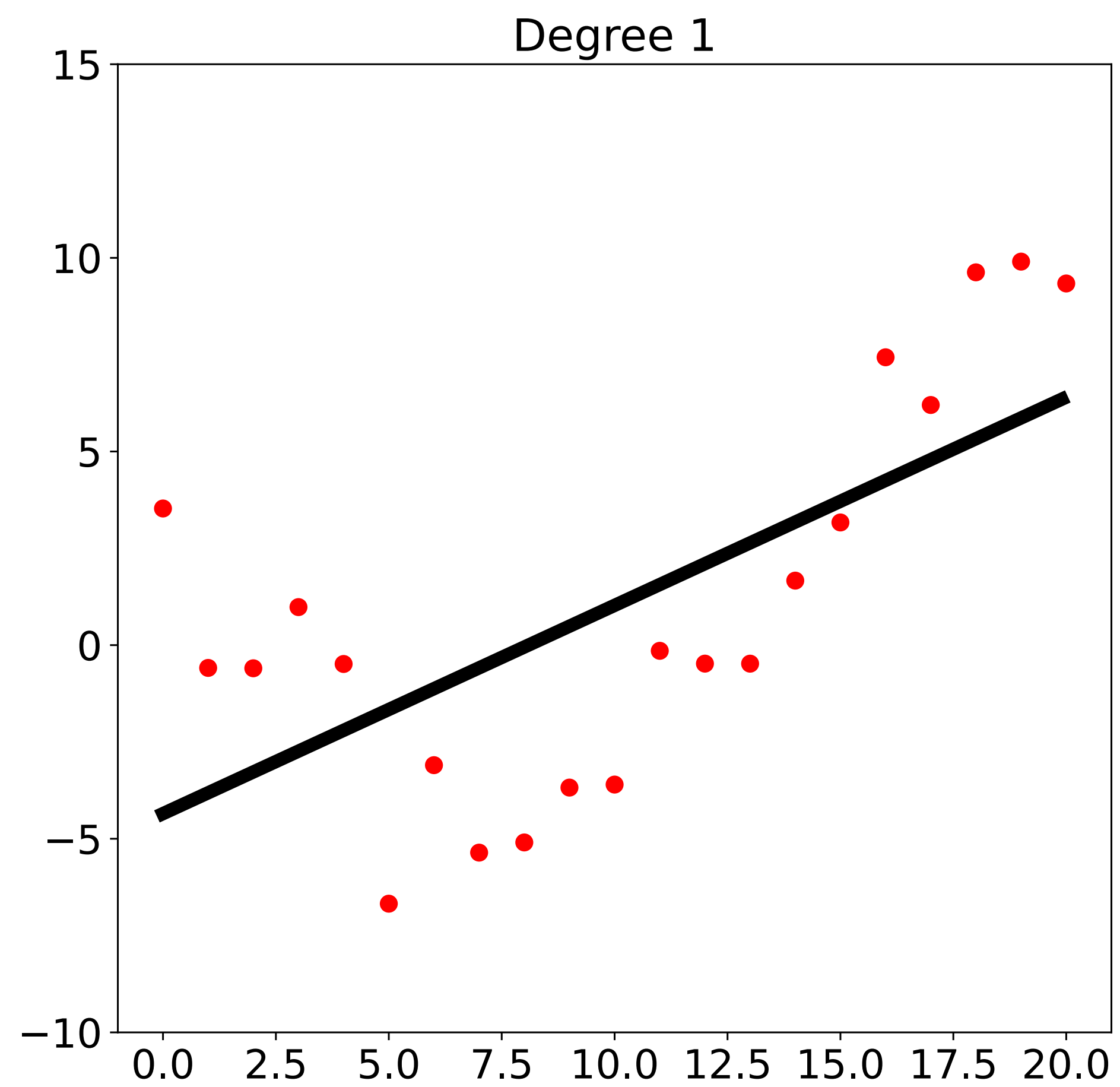
In practice, we use the empirical risk.

The problem

The 1 million dollar question is of course: what is the right choice of

$$\mathcal{F} = \{f_{\theta \in \Theta}\} \text{ ?}$$

What class of models should we consider?



Expressive power

Ideally, we want to make sure that \mathcal{F} is complex enough such that some function in it, $\tilde{f} \in \mathcal{F}$, can interpolate any dataset.

***Universal approximation Theorems:** important topic in the 90s.*

A 2-layer MLP can approximate any continuous function

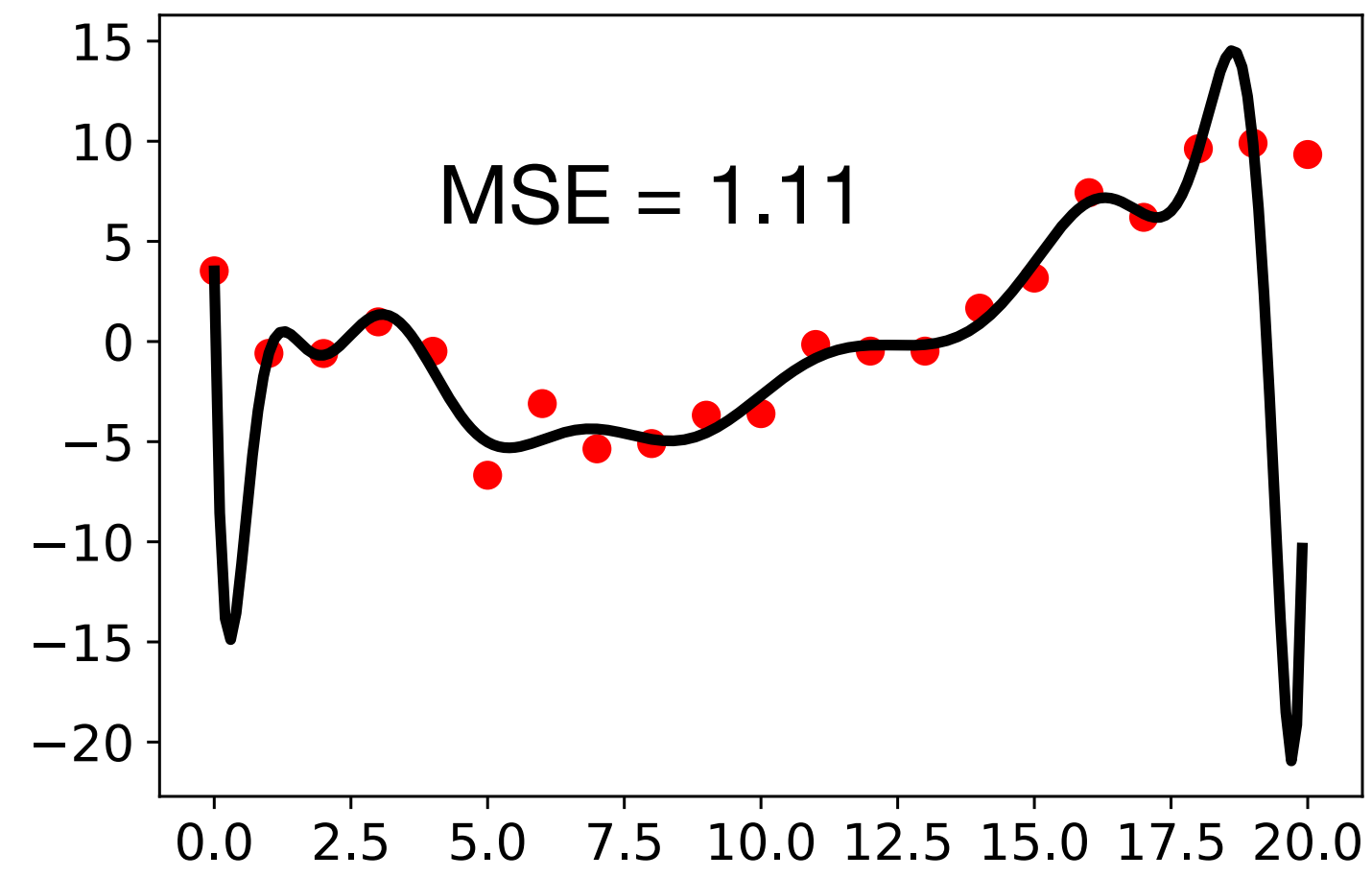
Expressive power

But expressive power by itself is not really important. *It does imply good generalization to unseen data.*

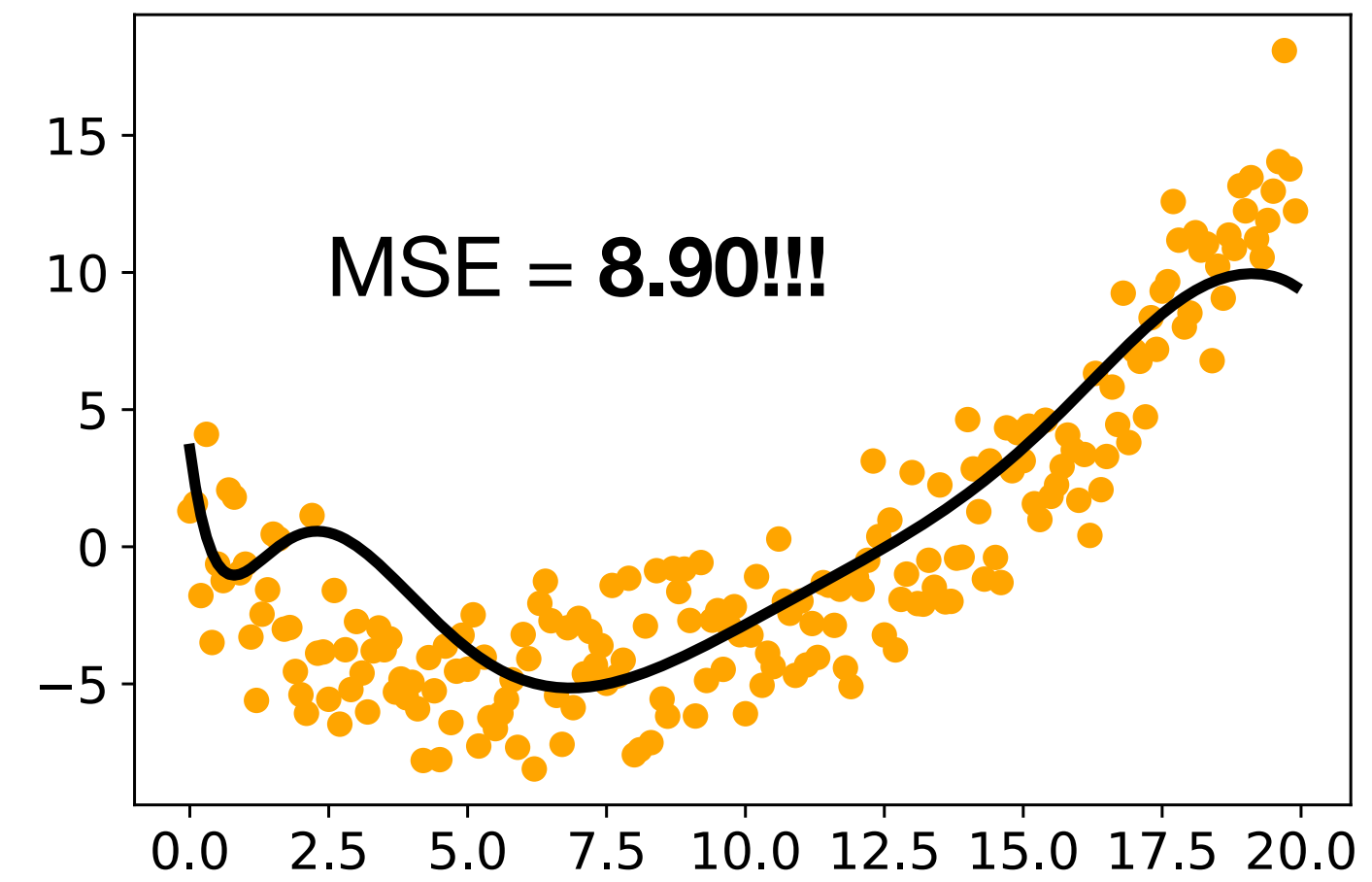
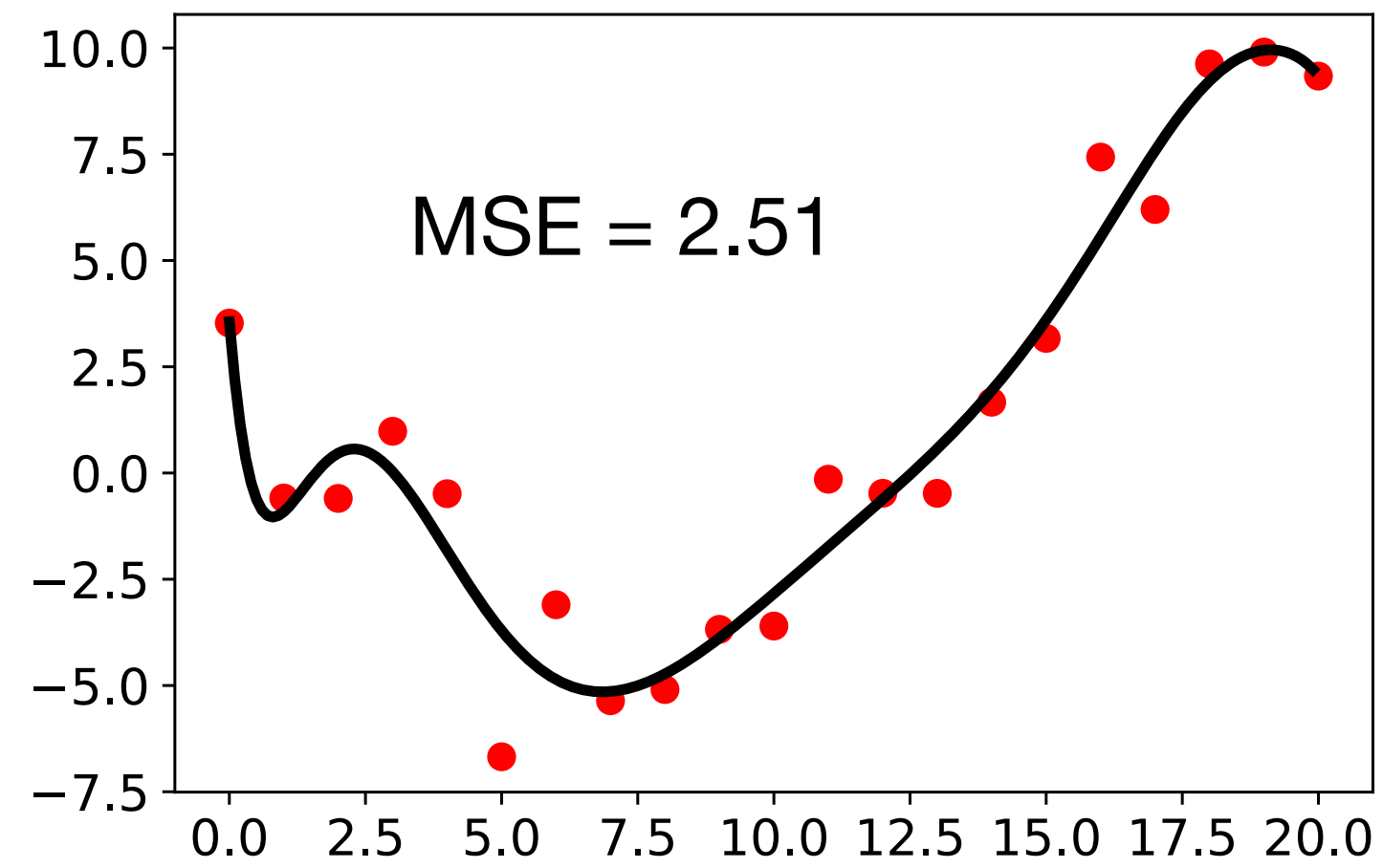
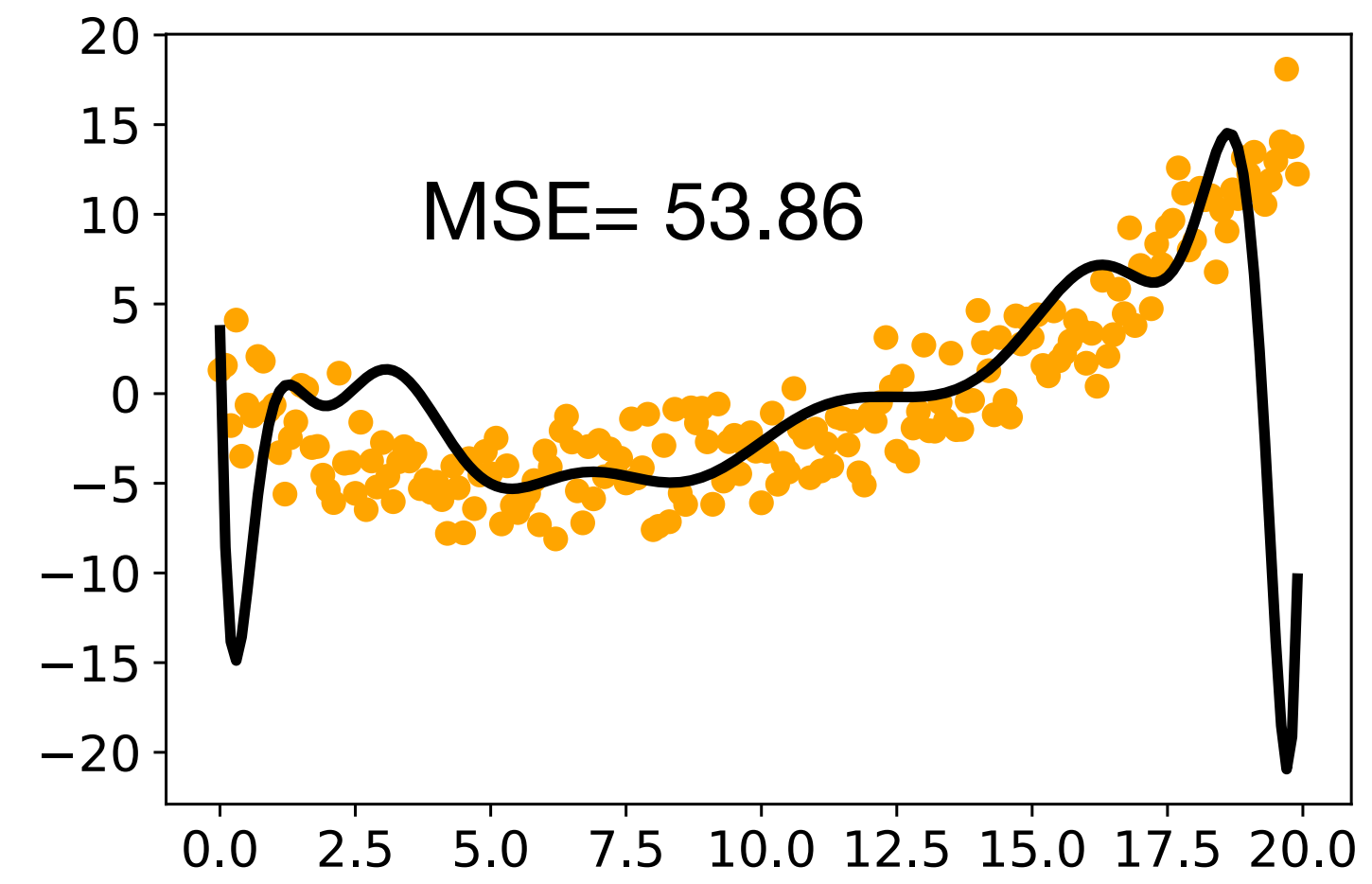
Otherwise, we would only use 2-layer MLPs for ALL problems.

A too complex class $\tilde{f} \in \mathcal{F}$ could actually hurt generalization.

Train data



Test data



Function complexity

A very popular approach is to find a function \tilde{f} which interpolates the data

$$\tilde{f}(x_i) = y_i$$

for all $i \in [N]$.

And has as low “**complexity**” as possible, denoted by $c(\tilde{f})$.

Function complexity

$$\tilde{f} \in \operatorname{argmin}_{g \in \mathcal{F}} c(g) \quad \text{subject to: } g(x_i) = y_i \quad \forall i$$

Function complexity

$$c(g) = \|g\|$$

Function complexity

$$c(g) = \int_{-\infty}^{+\infty} |g''(x)|^2$$

Controls the curvature of g .

Function complexity

For neural networks, complexity is often defined as the norm of the weights

$$c(g_{\theta}) = \|\theta\|_2^2$$

Also known as “weight decay”.

Beyond Simple Weight Decay: Graph Symmetries and Alignment

Symmetries

Another popular approach is to exploit ***symmetries*** in the data.

Then limit the class \mathcal{F} accordingly.

What's is a graph symmetry?

Properties in the data that don't change under some transformation of the data.

Regularization

Another approach is to look for “structure” in the data

Then limit the class \mathcal{F} by appropriate regularization.

Graph symmetry: node permutation

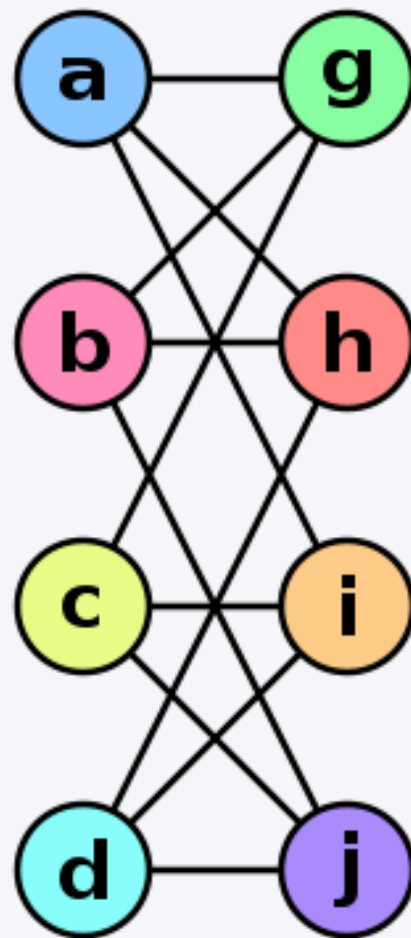
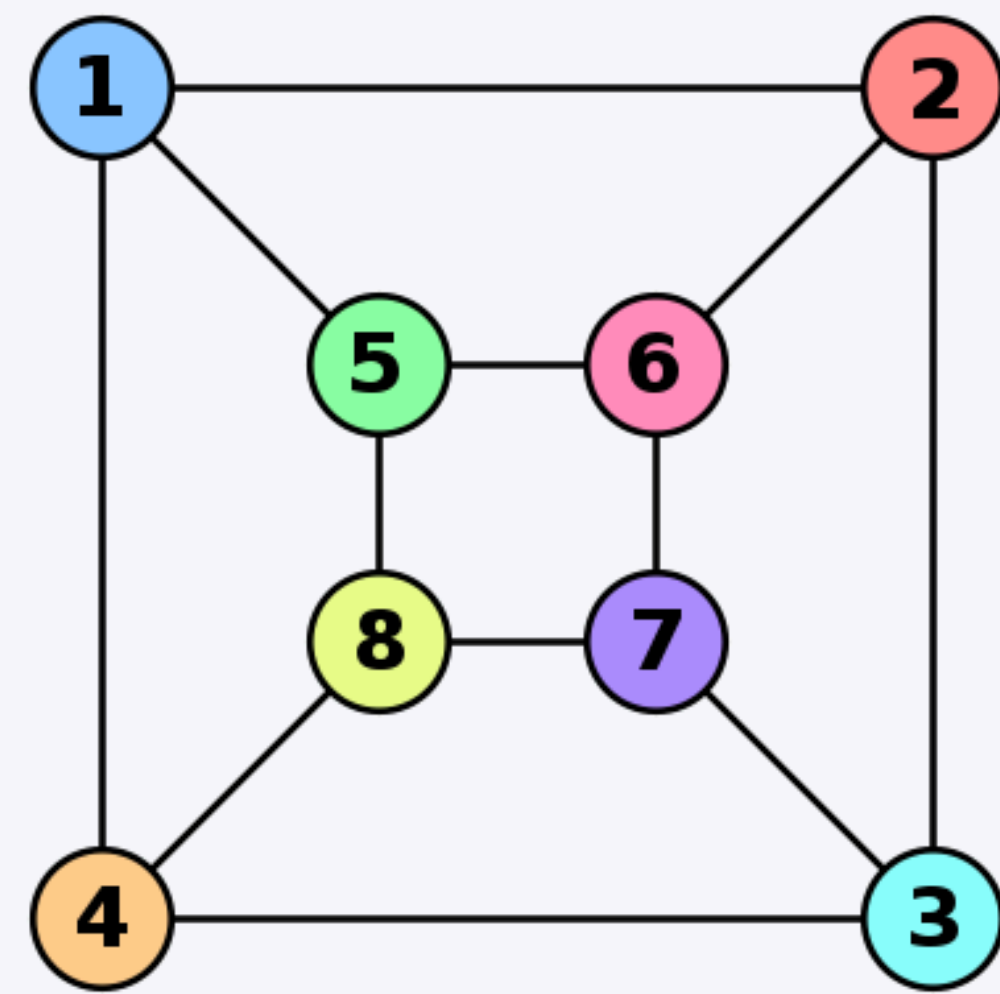
Let \mathcal{V} be the set of nodes in the graphs.

Often, we order the nodes from 1 to n . Or we give them “labels”.

Any decision that we make about the graph/nodes should not depend on any assumed ordering.

Our decision should be invariant to node permutations.

Isomorphic graphs

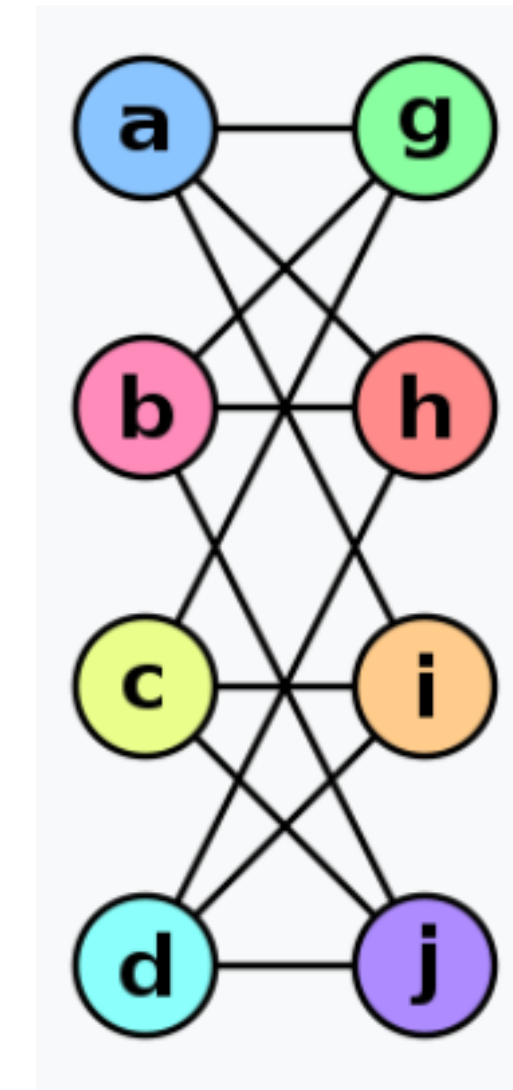
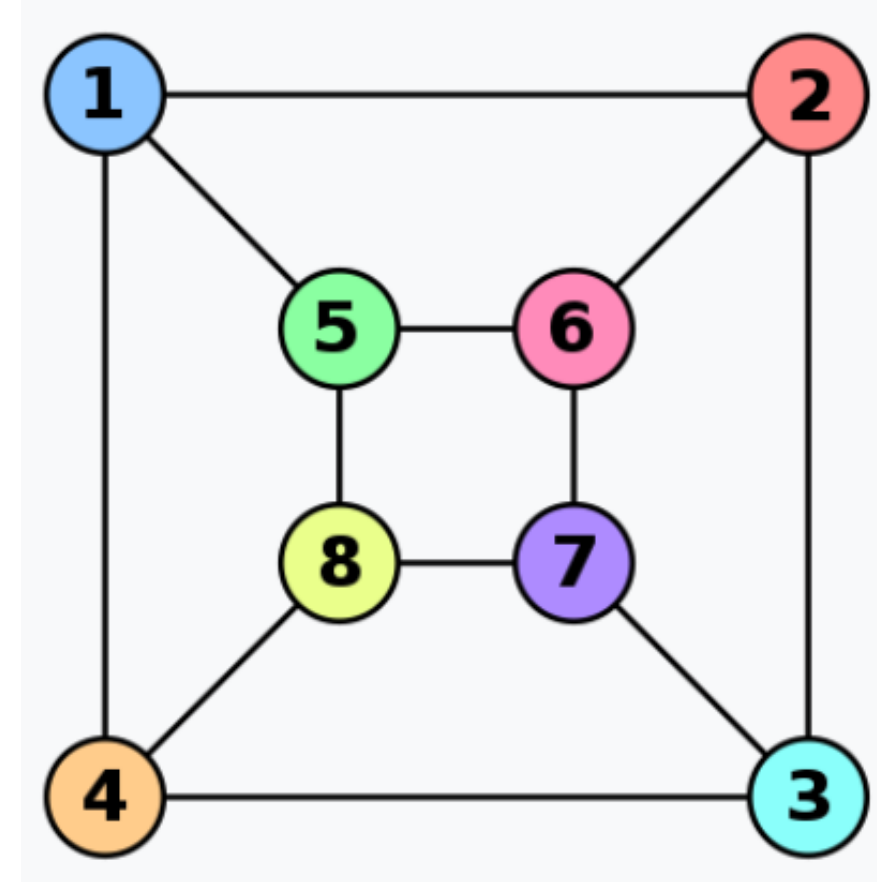
Graph G	Graph H	An isomorphism between G and H
		$f(a) = 1$ $f(b) = 6$ $f(c) = 8$ $f(d) = 3$ $f(g) = 5$ $f(h) = 2$ $f(i) = 4$ $f(j) = 7$

Graph G and H are equivalent up renaming the nodes.

f preserves the edges in the graph. It only changes the “names” of the nodes.

Permutation invariance

If two graphs are isomorphic



Then the output of a graph neural network should be the same:

$$GNN \left(\begin{array}{c} \text{Graph 1} \end{array} \right) = GNN \left(\begin{array}{c} \text{Graph 2} \end{array} \right)$$

Example on Sets (special graph with no edges)

By stacking the node features as rows in the $n \times d$ matrix X , we assume an ordering of the nodes.

So ordering the nodes seems unavoidable.

Permutation matrix

A permutation matrix $P \in \mathbb{R}^{n \times n}$ is obtained by permuting the rows and columns of the identity matrix. For example:

$$P = \begin{bmatrix} 1, & 0, & 0, & 0, & 0, & 0 \\ 0, & 0, & \textcolor{red}{1}, & 0, & 0, & 0 \\ 0, & 0, & 0, & \textcolor{red}{1}, & 0, & 0 \\ 0, & \textcolor{red}{1}, & 0, & 0, & 0, & 0 \\ 0, & 0, & 0, & 0, & 1, & 0 \\ 0, & 0, & 0, & 0, & 0, & 1 \end{bmatrix} \qquad P \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} = \begin{bmatrix} a \\ \textcolor{red}{c} \\ \textcolor{red}{f} \\ \textcolor{red}{b} \\ e \\ f \end{bmatrix}$$

Permutation matrices are orthogonal, i.e., $P^T P = I$ and $P^{-1} = P^T$.

Permutation invariance in sets

Given a model f , then for any permutation matrix P we need

$$f(PX) = f(X)$$

Example:

$$f(X) = \phi \left(\sum_{v \in \mathcal{V}} \psi(x_v) \right)$$

The sum function is independent of the order of which its input is provided!

Permutation invariance in graphs

Given a model f , then for any permutation matrix P we need

$$f(PX, PAP^T) = f(X, A)$$

Permutation invariant architectures: *general message passing*

$$h_u = \phi(x_u, \bigoplus_{v \in \mathcal{N}(u)} \psi(x_u, x_v))$$

$\phi(x, z) = \sigma(Wx + Uz + b)$

Some aggregation function,
e.g., $\sum_{v \in \mathcal{N}(u)}$ or $\max_{v \in \mathcal{N}(u)}$

MLP, GCN or GAT
see, next slide.

Permutation invariant architectures: *graph convolution*

$$h_u = \phi(x_u, \bigoplus_{v \in \mathcal{N}(u)} \boxed{c_{uv} \psi(x_v)})$$


$$\psi(x) = (Wx + b)$$

c_{uv} are constant weights

Permutation invariant architectures: *graph attention*

$$h_u = \phi(x_u, \bigoplus_{v \in \mathcal{N}(u)} \alpha(x_u, x_v) \psi(x_v))$$


$$\psi(x) = (Wx + b)$$

$\alpha(x_u, x_v)$ are learnable

Alignment and an Application on Reasoning!

Problem Setting

The set S is the set of objects to reason about.

Each $s \in S$ is represented by a vector x .

x can be some description or an image or information about the specific question that we want to answer.

Problem Setting

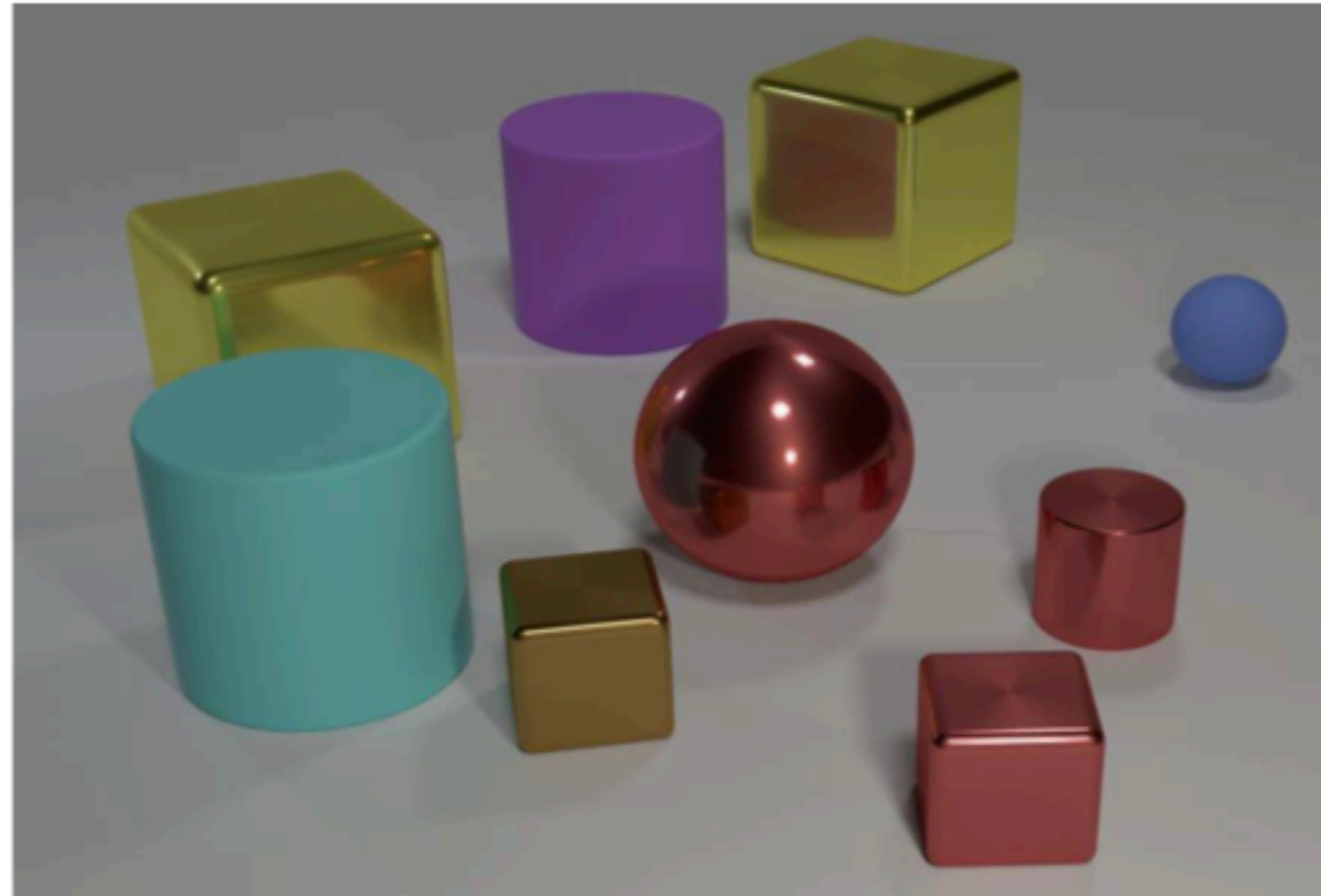
Given a set of universes $\{S_1, \dots, S_M\}$ and answer labels $\{y_1, \dots, y_M\}$,

we want to learn a function g such that

$$g(S_i) = y_i$$

and we also want g to perform well on unseen sets S .

Example: relational argmax



Relational argmax

What are the colors of the
furthest pair of objects?

Example: relational argmax

Each object $s \in S$ is represented by a vector

$$x_s = [h_1, h_2]$$

h_1 : is the location of the object

h_2 : is the colour

Example: relational argmax

The colour of the furthest pair of objects is given in two steps:

1) Compute the objects that are the furthest apart:

$$\{X_{s_1}, X_{s_2}\} := \operatorname{argmax}_{s_1, s_2 \in S} \|h_1(X_{s_1}) - h_1(X_{s_2})\|_1$$

2) Return their colour:

$$y(S) = \left(h_2(X_{s_1}), h_2(X_{s_2}) \right)$$

Candidate architectures: Deep Sets Architecture (no edges, only nodes)

$$y = \text{MLP}_1 \left(\sum_{s \in S} \text{MLP}_2(x_s) \right)$$

Candidate architectures: Message Passing Architecture

There is no graph in our reasoning task, but... we can make one.

Consider every $s \in S$ as a node.

Assume that all nodes are connected to each other (complete graph).

Candidate architectures: Message Passing Architecture

For every node s and every layer k compute:

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

and the global pooling layer at the last layer:

$$y = \text{MLP}_2\left(\sum_{t \in S} h_t^{(K)}\right)$$

Candidate architectures

Both architectures are reasonable candidates since they are both permutation-invariant and there is no ordering in the input set!

Universal Approximation

Proposition 3.1. *Let $f : \mathbb{R}^{d \times N} \rightarrow \mathbb{R}$ be any continuous function over sets S of bounded cardinality $|S| \leq N$. If f is permutation-invariant to the elements in S , and the elements are in a compact set in \mathbb{R}^d , then f can be approximated arbitrarily closely by a GNN (of any depth).*

Proposition 3.2. *For any GNN \mathcal{N} , there is an MLP that can represent all functions \mathcal{N} can represent.*

This means that MLPs are as powerful as GNNs.

But in practice, there is a huge difference in generalization performance.

Observation

A lot of reasoning tasks can be written down as algorithmic steps.

Example: compute the shortest path between two places.

It is formulated as a shortest path problem on a map and solved using Bellman-Ford's algorithm.

Algorithmic Alignment

The concept of alignment helps us understand the gap between theory and practice.

How?

Study how well network structure and the task interact,

and how this structure affects generalization performance of the network.

Algorithmic Alignment

Graph Neural Network

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

for u in S : *No need to learn for-loops*

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

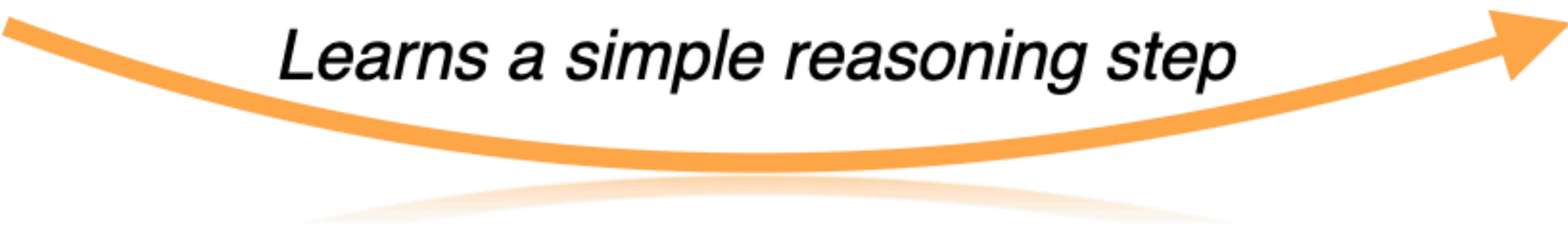
Bellman-Ford algorithm

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

for u in S :

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

Learns a simple reasoning step



Algorithmic Alignment

Intuitively, the better the alignment the “easier” would be to learn to solve a reasoning problem.

What does “easier” mean?

It means, less training data!!

More formally, *smaller* sample complexity.

Probably Approximately Correct Learning

Fix an error parameter $\epsilon > 0$, and a failure probability $\delta \in (0,1)$.

Suppose $\{x_i, y_i\}_{i=1}^C$ are i.i.d. samples from some distribution P .

Suppose that $g(x_i) = y_i$ for some underlying function g .

Let f be a function generated by a learning procedure which uses $\{x_i, y_i\}_{i=1}^C$.

We say that g is (C, ϵ, δ) -learnable if

$$\mathbb{P}_{x \sim P}[\|g(x) - f(x)\| \leq \epsilon] \geq 1 - \delta$$

Sample Complexity

We say that g is (C, ϵ, δ) -learnable if

$$\mathbb{P}_{x \sim P}[\|g(x) - f(x)\| \leq \epsilon] \geq 1 - \delta$$

The ***sample complexity of learning*** g with tolerance ϵ and failure probability δ is the ***minimum number of samples*** C such that g is (C, ϵ, δ) -learnable.

Algorithmic Alignment: formal definition

Let g represent an algorithm/function which we want to approximate using neural networks. For example, Bellman-Ford's algorithm.

g consists of m modules/sub-routines f_1, \dots, f_m , which placed in the right order gives the algorithm g .

Assume that you have a neural network \mathcal{N} which also consists of m modules $\mathcal{N}_1, \dots, \mathcal{N}_m$.

Algorithmic Alignment: formal definition

Then the neural network \mathcal{N} *simulates* algorithm g if we can replace f_i with \mathcal{N}_i and we get the correct output for every input.

\mathcal{N} *C-aligns* with g if there is a way to train the modules \mathcal{N}_i with sample complexity C_i that satisfies:

$$\max_{i \in [m]} C_i \leq \frac{C}{m}$$

Algorithmic Alignment Reduces Sample Complexity

If \mathcal{N} C-aligns with g , then g is (C, ϵ, δ) -learnable by \mathcal{N} .

In other words, the sample complexity of learning g using \mathcal{N} is determined by the alignment sample complexity C , which depends on training individual modules \mathcal{N}_i .

Algorithmic Alignment: shortest path

Graph Neural Network

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

for u in S : *No need to learn for-loops*

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

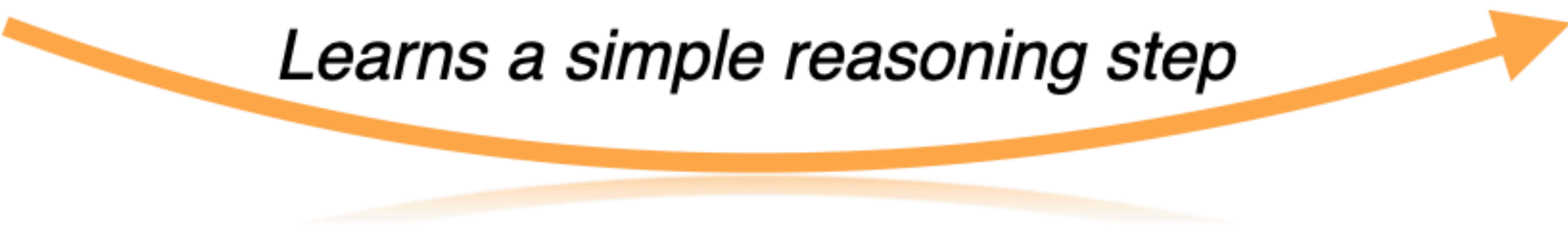
Bellman-Ford algorithm

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

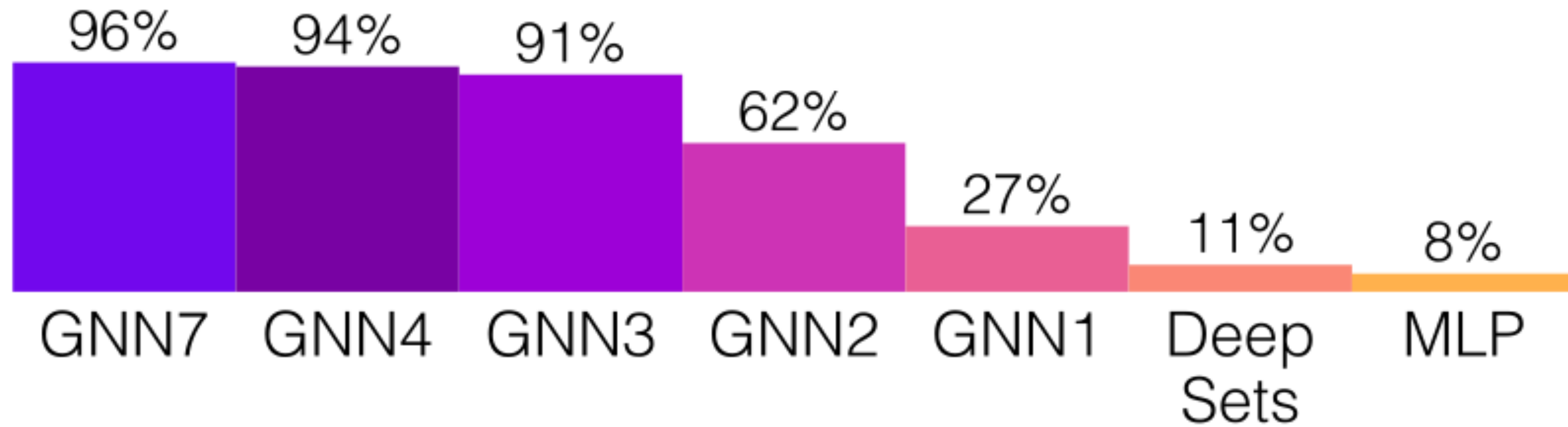
for u in S :

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

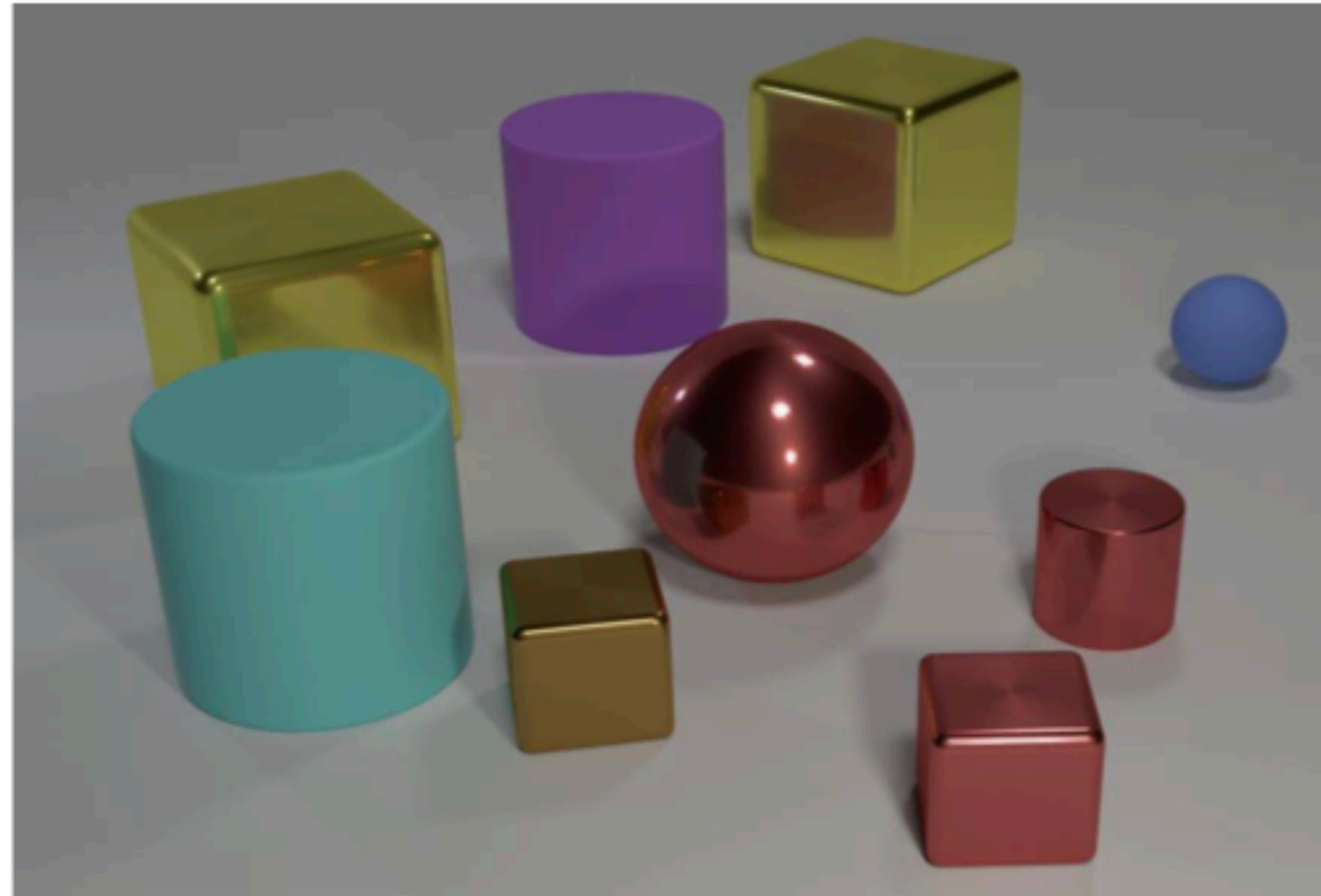
Learns a simple reasoning step



Performance on shortest path



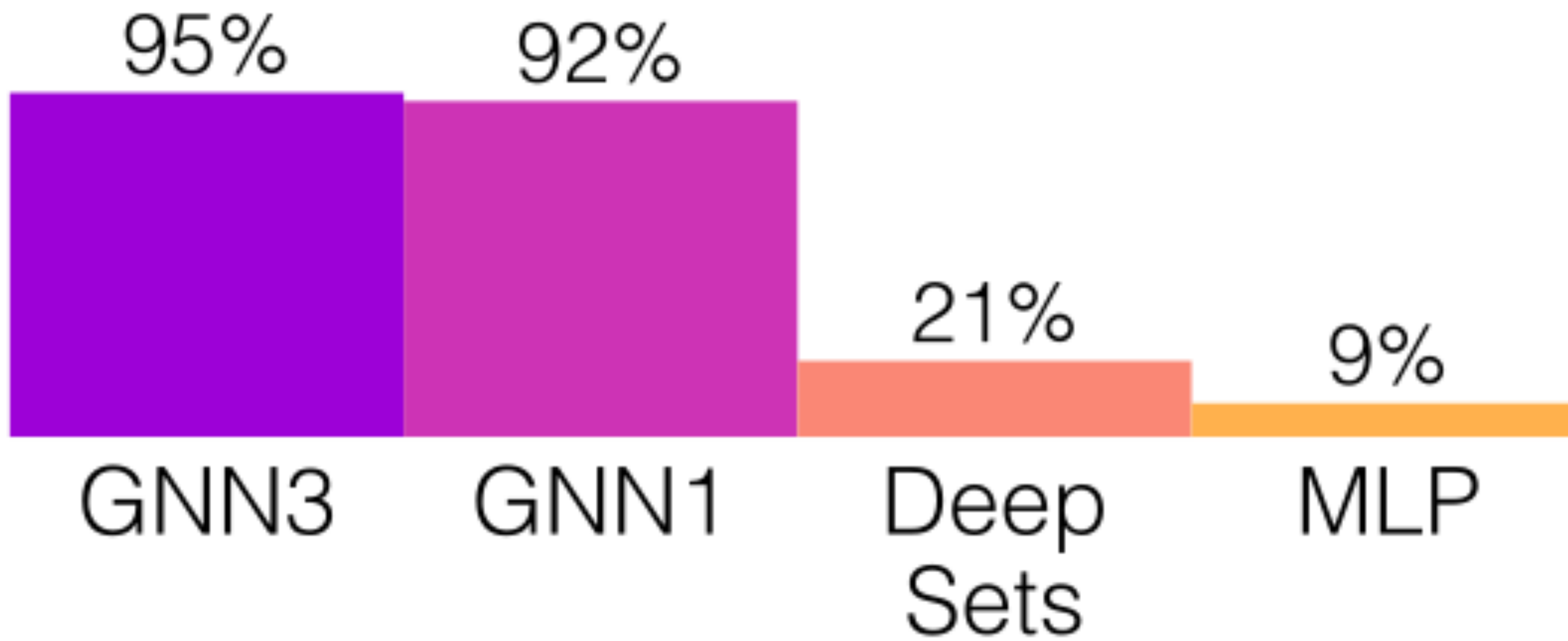
Example: relational argmax



Relational argmax

What are the colors of the
furthest pair of objects?

Performance on relational argmax



Thank you!