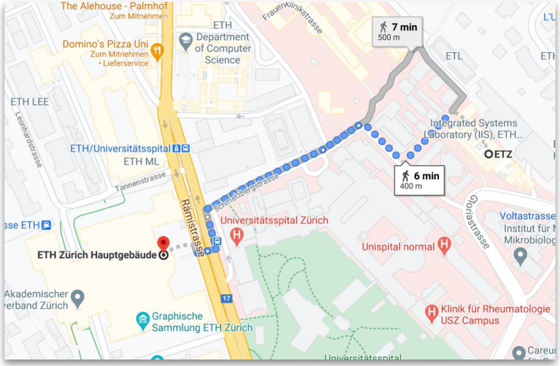
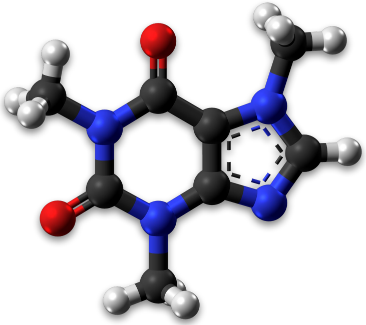
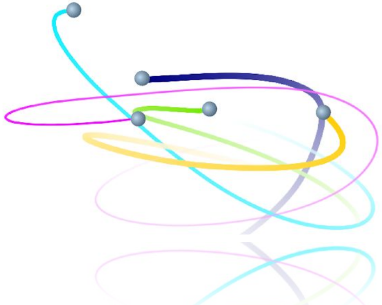
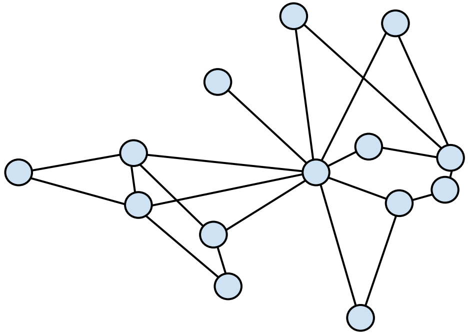




Graph Neural Networks

Algorithmic Alignment & Necessity






Cell

Article

A Deep Learning Approach to Antibiotic Discovery

Jonathan M. Stokes,[1,2,3] Kevin Yang,[3,4,10] Kyle Swanson,[3,4,10] Wengong Jin,[3,4] Andres Cubillos-Ruiz,[1,2,5] Nina M. Donghia,[1,5] Craig R. MacNair,[6] Shawn French,[6] Lindsey A. Carfrae,[6] Zohar Bloom-Ackermann,[2,7] Victoria M. Tran,[2] Anush Chiappino-Pepe,[5,7] Ahmed H. Badran,[2] Ian W. Andrews,[1,2,5] Emma J. Chory,[1,2] George M. Church,[5,7,8] Eric D. Brown,[6] Tommi S. Jaakkola,[3,4] Regina Barzilay,[3,4,9,*] and James J. Collins[1,2,5,8,9,11,*]

[1]Department of Biological Engineering, Synthetic Biology Center, Institute for Medical Engineering and Science, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

[2]Broad Institute of MIT and Harvard, Cambridge, MA 02142, USA

[3]Machine Learning for Pharmaceutical Discovery and Synthesis Consortium, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

[4]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

[5]Wyss Institute for Biologically Inspired Engineering, Harvard University, Boston, MA 02115, USA

[6]Department of Biochemistry and Biomedical Sciences, Michael G. DeGroote Institute for Infectious Disease Research, McMaster University, Hamilton, ON L8N 3Z5, Canada

[7]Department of Genetics, Harvard Medical School, Boston, MA 02115, USA

[8]Harvard-MIT Program in Health Sciences and Technology, Cambridge, MA 02139, USA

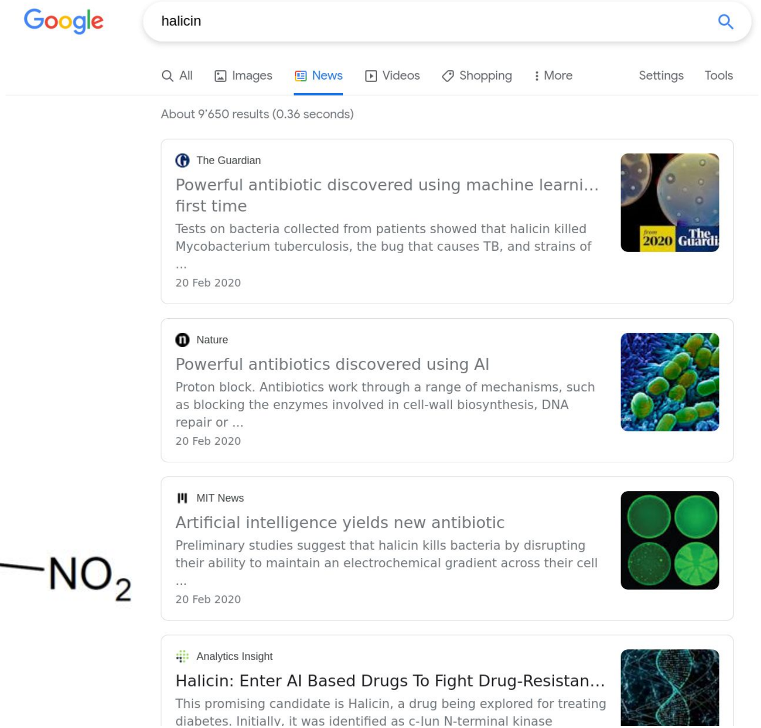
[9]Abdul Latif Jameel Clinic for Machine Learning in Health, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

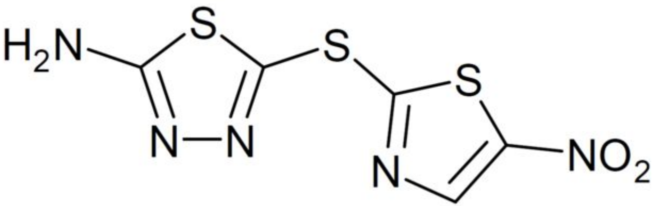
[10]These authors contributed equally

[11]Lead Contact

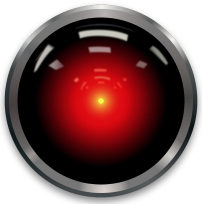
*Correspondence: regina@csail.mit.edu (R.B.), jimjc@mit.edu (J.J.C.)

https://doi.org/10.1016/j.cell.2020.01.021





Halicin



MLP is all you need?

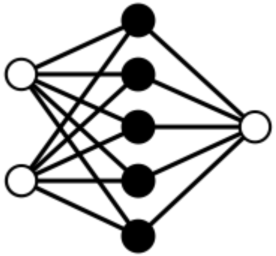
Theorem 4.1.1 (universal approximation theorem):
An arbitrary continuous function, defined on [0,1] *can be arbitrary well uniformly approximated by a multilayer feed-forward neural network with one hidden layer (that contains only finite number of neurons) using neurons with arbitrary activation functions in the hidden layer and a linear neuron in the output layer. Formally:*

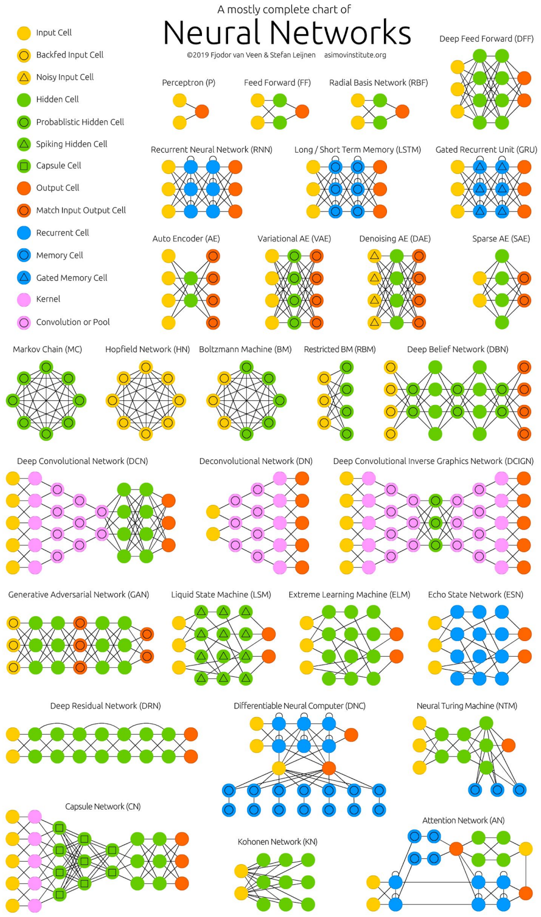
Let $\varphi(.)$ *be the arbitrary activation*[5] *function. Then* $\forall f \in C([0,1])$, $\forall \varepsilon > 0$: $\exists\, n \in \mathrm{N}$, w_i, a_i, $b_i \in \mathrm{R}$, $i \in \{0 \ldots n\}$:

$$(A_n f)(x) = \sum_{i=1}^{n} w_i \varphi(a_i x + b_i)$$

as an approximation of the function f(.); that is

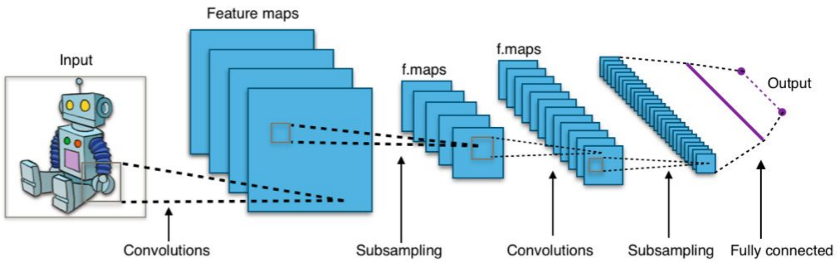
$$\sup_{x \in [0,1]} \left| (A_n f)(x) - \mathrm{f}(x) \right| < \varepsilon$$

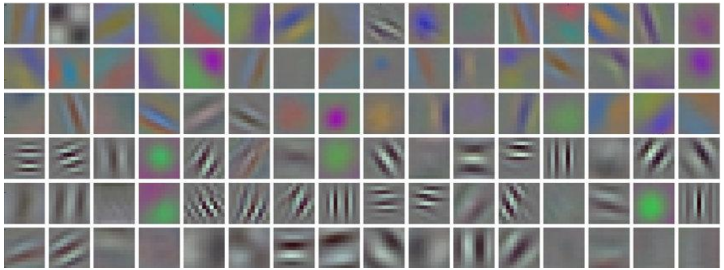




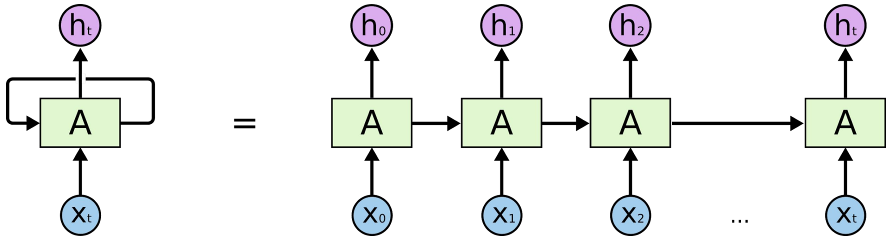
MLP is all you need? No!

- Inductive bias for Images: **Convolutions**





- Inductive bias for Time Series: **Hidden states**



Battaglia et al., 2018

MLP is all you need? No!

- Prior distribution

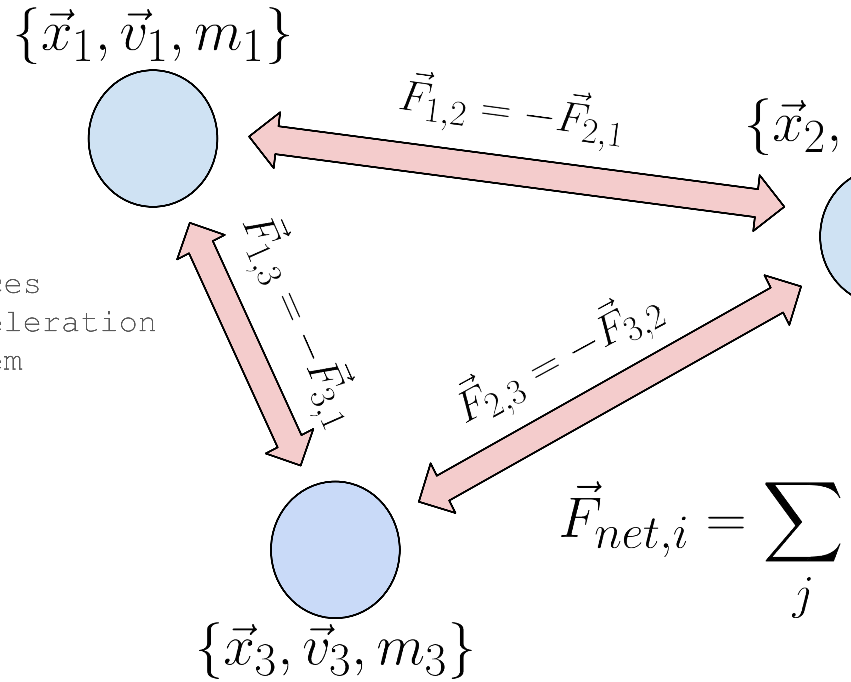
$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{p(X)}$$

- Ridge / Lasso regularization

$$\mathcal{L} = \mathcal{L}\left(Y, f_\theta(X)\right) + \lambda||\theta||_p$$

Battaglia et al., 2018

Example: Particle Physics - Predict particle movement



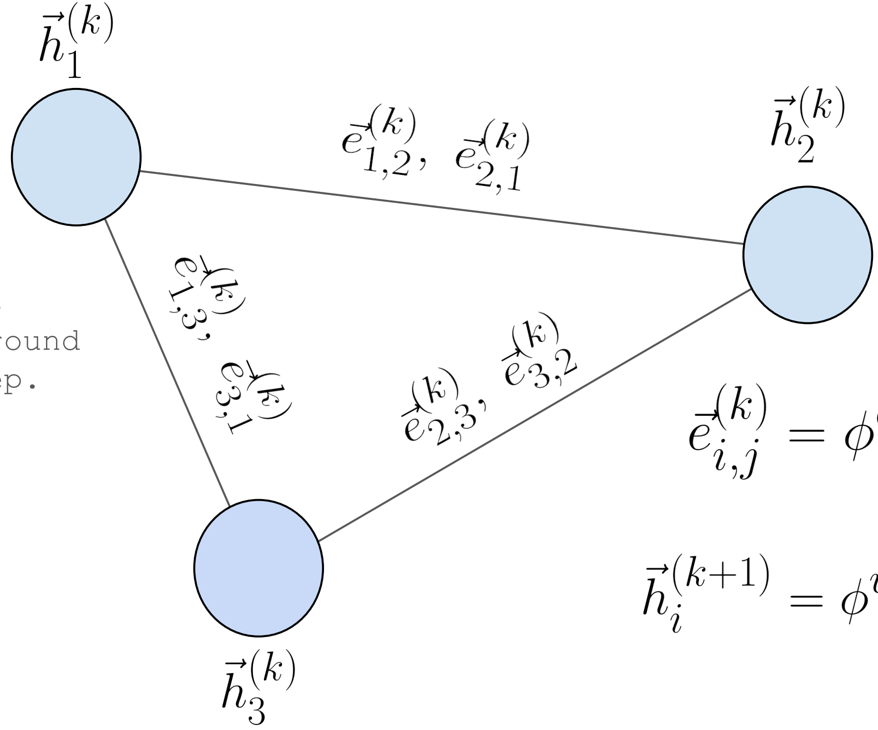
Step 1: Compute forces
Step 2: Compute acceleration
Step 3: Evolve system

$$\vec{a}_i = \frac{1}{m_i}\vec{F}_{net,i}$$

$$\vec{F}_{net,i} = \sum_j \vec{F}_{i,j} = C \sum_j (1 - r_{i,j})\hat{r}_{i,j}$$

Cranmer et al., 2020

Example: Particle Physics - Predict particle movement



Step 1: Compute messages
Step 2: Pass messages around
Step 3: Update hidden rep.

$$\vec{e}_{i,j}^{(k)} = \phi^e \left(\vec{h}_i^{(k)}, \vec{h}_j^{(k)} \right)$$

$$\vec{h}_i^{(k+1)} = \phi^v \left(\vec{h}_i^{(k)}, \sum_{j \in \mathcal{N}_i} \vec{e}_{i,j}^{(k)} \right)$$

Cranmer et al., 2020

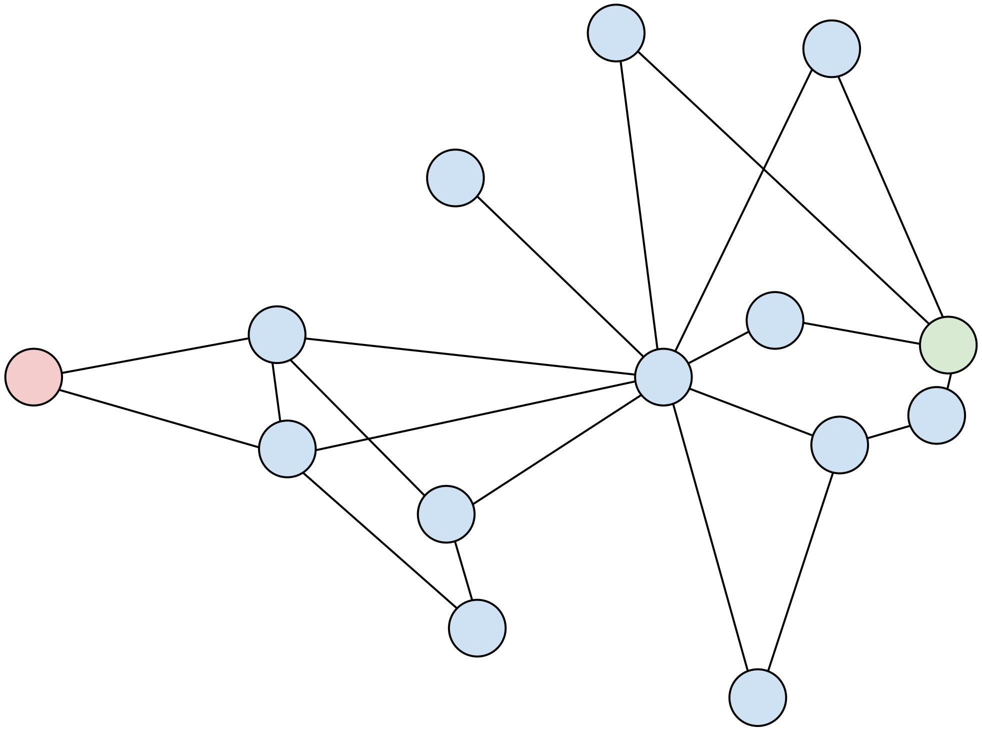
Example: Particle Physics - Predict particle movement

Physics		**GNNs**	
Particle	$\{\vec{x}_1, \vec{v}_1, m_1\}$	Node	$\vec{h}_1^{(k)}$
Force	$\vec{F}_{i,j}$	Edge / message	$\vec{e}_{i,j}^{(k)}$
Gravitation	$(1-r_{i,j})\hat{r}_{i,j}$	Edge model	ϕ^e
Net force	$\sum \vec{F}_{i,j}$	Aggregation	$\sum \vec{e}_{i,j}^{(k)}$
Acceleration	$\vec{a}_i$	Node model	ϕ^v

Message passing framework: Algorithmic alignment with physical task

Cranmer et al., 2020

Example: Graph Algorithms



Bellman-Ford algorithm

for k = 1 … |S| - 1:

for u in S:

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

GNN

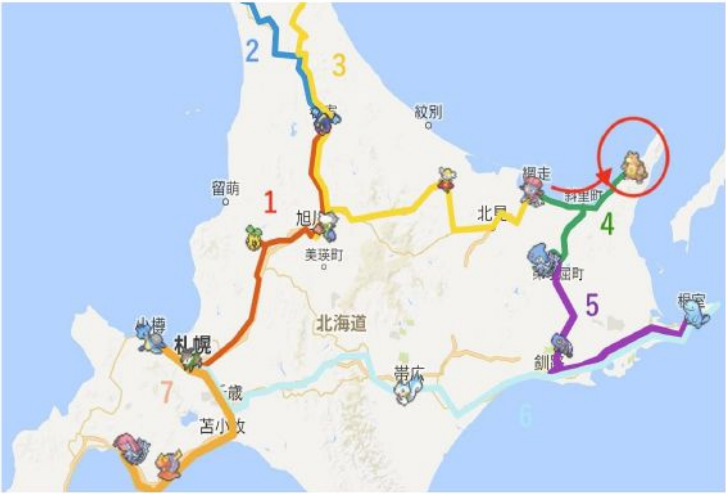
for k = 1 … GNN iter:

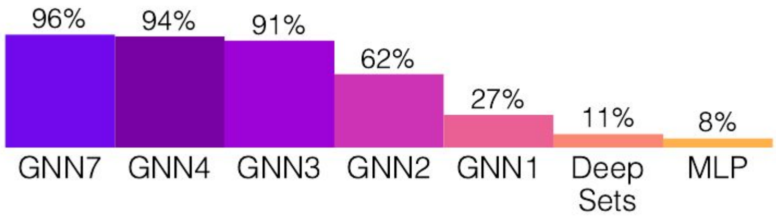
for u in S:

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

Xu et al., 2019

Example: Graph Algorithms





GNN

for k = 1 … GNN iter:

for u in S:

No need to learn for-loops

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

Xu et al., 2019

Can we mathematically define *Algorithmic Alignment*?

Definition 1.1 (PAC learning and sample complexity). Fix an error parameter $\boxed{\epsilon > 0}$ and failure probability $\boxed{\delta \in (0, 1)}$. Suppose $\boxed{\{x_i, y_i\}_{i=1}^{M}}$ are i.i.d. samples from some distribution $\mathcal{D}$, and the data satisfies $\boxed{y_i = g(x_i)}$ for some underlying function g. Let $\boxed{f = \mathcal{A}\left(\{x_i, y_i\}_{i=1}^{M}\right)}$ be the function generated by a learning algorithm $\mathcal{A}$. Then g is (M, ϵ, δ)-*learnable* with $\mathcal{A}$ if

$$\boxed{\mathbb{P}_{x \sim \mathcal{D}}\left[\|f(x) - g(x)\| \leq \epsilon\right] \geq 1 - \delta.}$$

The $\boxed{\textit{sample complexity } \mathcal{C}_{\mathcal{A}}(g, \epsilon, \delta)}$ is the $\boxed{\text{minimum M}}$ so that g is (M, ϵ, δ)-learnable with $\mathcal{A}$.

Xu et al., 2019

Can we mathematically define *Algorithmic Alignment*?

Definition 1.2 (Algorithmic alignment). Let g be a reasoning function and $\mathcal{N}$ a neural network with n modules $\mathcal{N}_i$. The module functions $f_1, ..., f_n$ generate g for $\mathcal{N}$ if, by replacing $\mathcal{N}_i$ with f_i, the network $\mathcal{N}$ simulates g. Suppose $\{x_i, y_i\}_{i=1}^M$ are i.i.d. samples from some distribution $\mathcal{D}$, and the data satisfies $y_i = g(x_i)$. Then $\mathcal{N}$ (M, ϵ, δ)-algorithmically aligns with g if (1) $f_1, ..., f_n$ generate g and (2) there are learning algorithms $\mathcal{A}_i$ for the $\mathcal{N}_i$'s such that

$$n \cdot \max_i \mathcal{C}_{\mathcal{A}_i}(f_i, \epsilon, \delta) \le M.$$

Bellman-Ford algorithm

for k = 1 ... |S| - 1:

for u in S:

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

GNN

for k = 1 ... GNN iter:

for u in S:

No need to learn for-loops

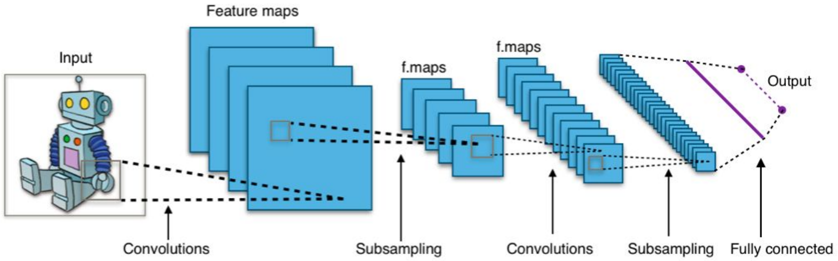
$h_u^{(k)} = \Sigma_v \; \mathrm{MLP}(h_v^{(k-1)}, h_u^{(k-1)})$

Xu et al., 2019

Can we mathematically define *Algorithmic Alignment*?

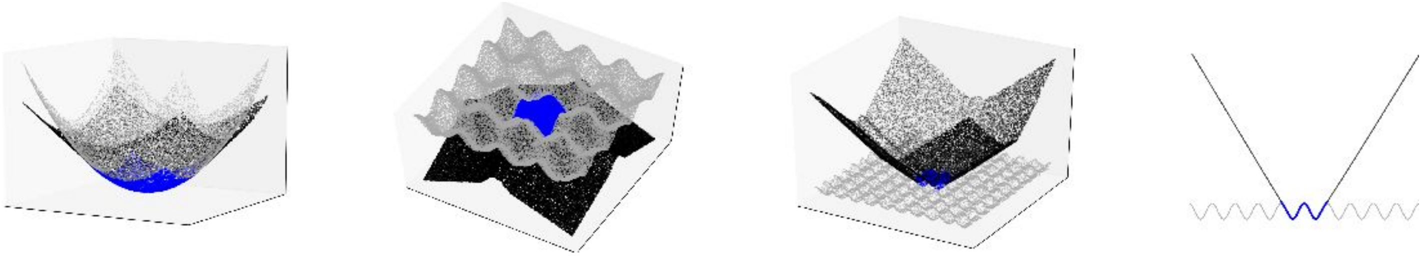
Definition 1.2 (Algorithmic alignment). Let g be a reasoning function and $\mathcal{N}$ a neural network with n modules $\mathcal{N}_i$. The module functions $f_1, ..., f_n$ generate g for $\mathcal{N}$ if, by replacing $\mathcal{N}_i$ with f_i, the network $\mathcal{N}$ simulates g. Suppose $\{x_i, y_i\}_{i=1}^M$ are i.i.d. samples from some distribution $\mathcal{D}$, and the data satisfies $y_i = g(x_i)$. Then $\mathcal{N}$ (M, ϵ, δ)-algorithmically aligns with g if (1) $f_1, ..., f_n$ generate g and (2) there are learning algorithms $\mathcal{A}_i$ for the $\mathcal{N}_i$'s such that

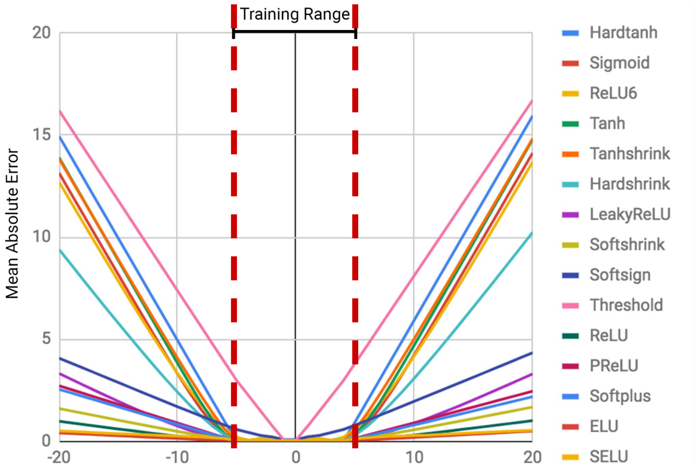
$$n \cdot \max_i \mathcal{C}_{\mathcal{A}_i}(f_i, \epsilon, \delta) \leq M.$$



Xu et al., 2019

When can GNNs extrapolate?





Xu et al., 2020 & Trask et al., 2018

When can GNNs extrapolate?

Definition 1.2 (**Algorithmic alignment**). Let g be a reasoning function and $\mathcal{N}$ a neural network with n modules $\mathcal{N}_i$. The module functions $f_1, ..., f_n$ generate g for $\mathcal{N}$ if, by replacing $\mathcal{N}_i$ with f_i, the network $\mathcal{N}$ simulates g. Suppose $\{x_i, y_i\}_{i=1}^M$ are i.i.d. samples from some distribution $\mathcal{D}$, and the data satisfies $y_i = g(x_i)$. Then $\mathcal{N}$ (M, ϵ, δ)-algorithmically aligns with g if (1) $f_1, ..., f_n$ generate g and (2) there are learning algorithms $\mathcal{A}_i$ for the $\mathcal{N}_i$'s such that

$$n \cdot \max_i \mathcal{C}_{\mathcal{A}_i}(f_i, \epsilon, \delta) \leq M.$$

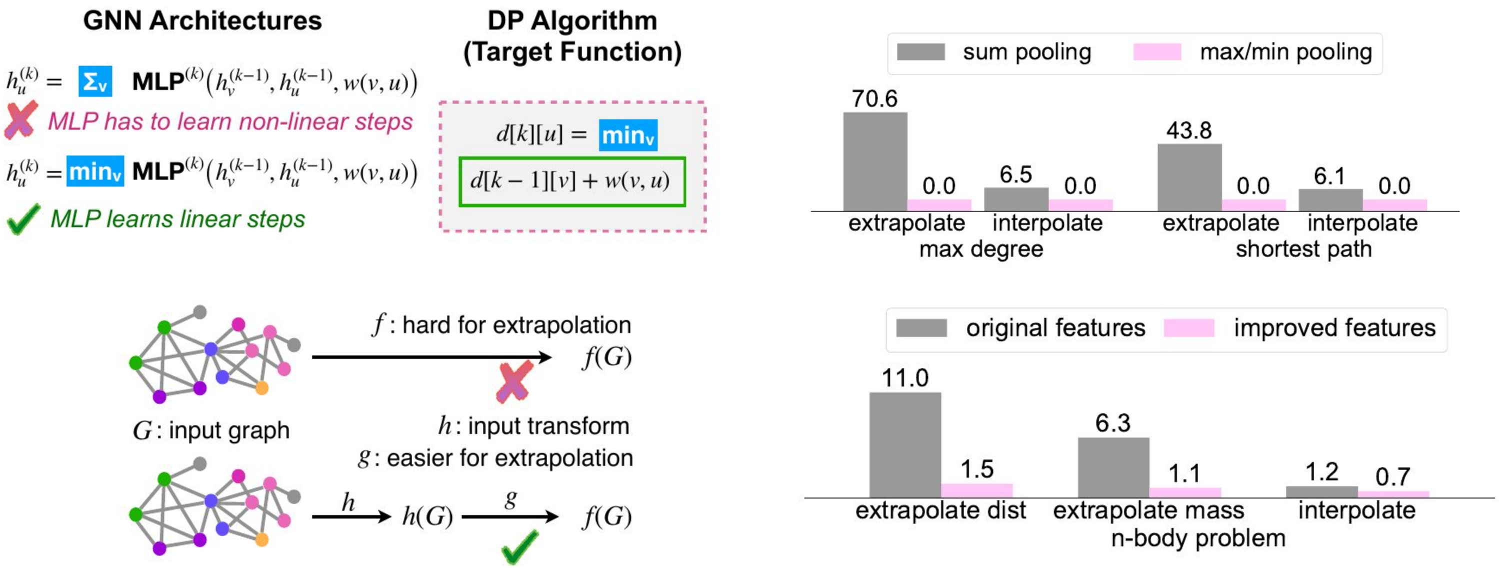
"easy to learn" = sample complexity grows ***polynomial*** = good ***interpolation***

good ***extrapolation*** = algorithm steps can be represented by ***linear*** functions via MLP

Xu et al., 2020

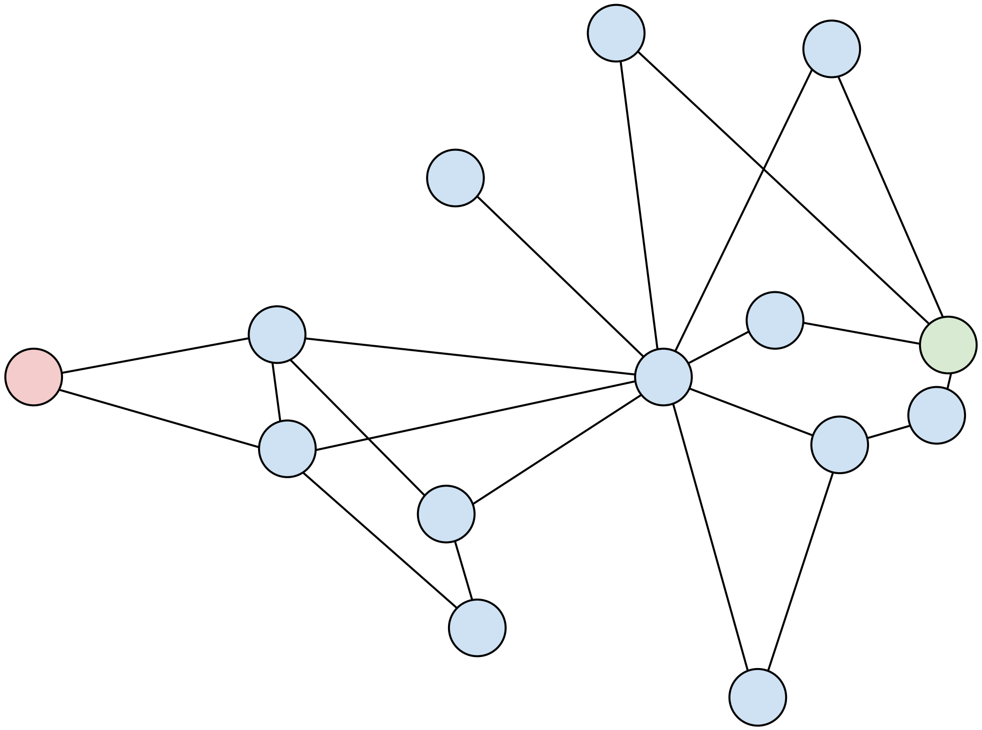
When can GNNs extrapolate?

good ***extrapolation*** = algorithm steps can be represented by ***linear*** functions via MLP



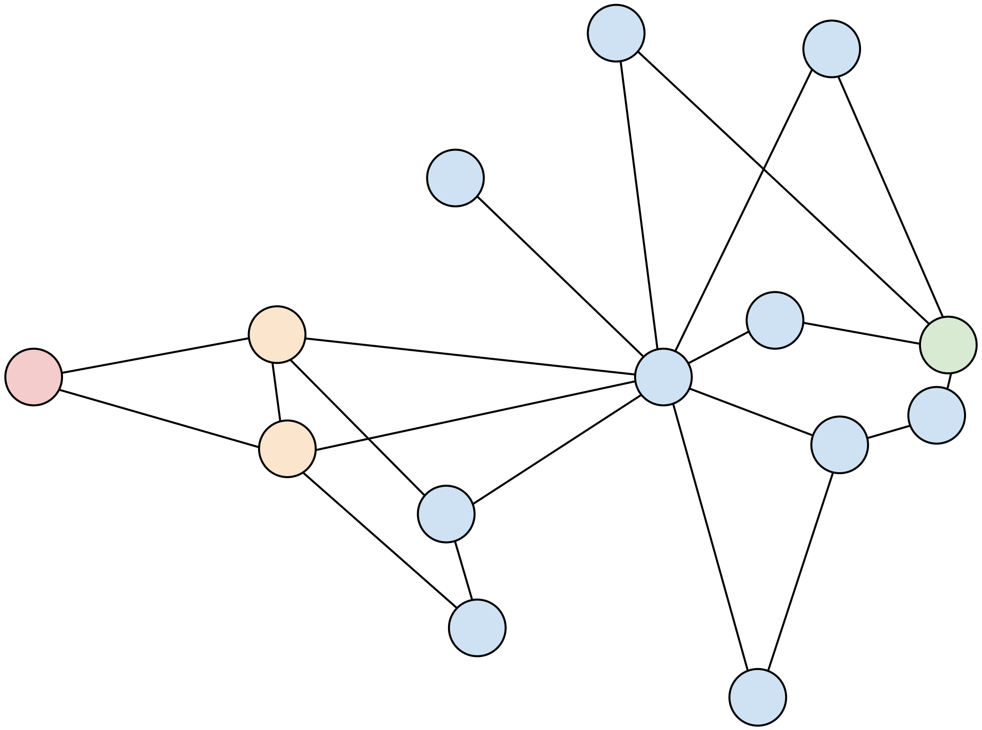
Xu et al., 2020

What about termination?



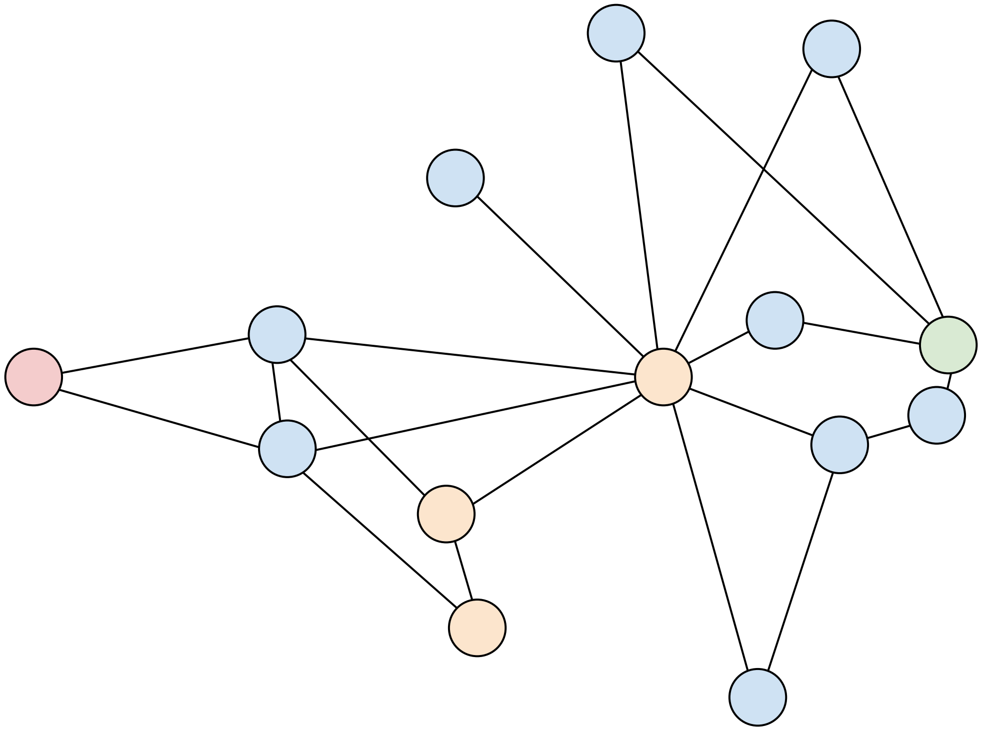
What about termination?

1 hop



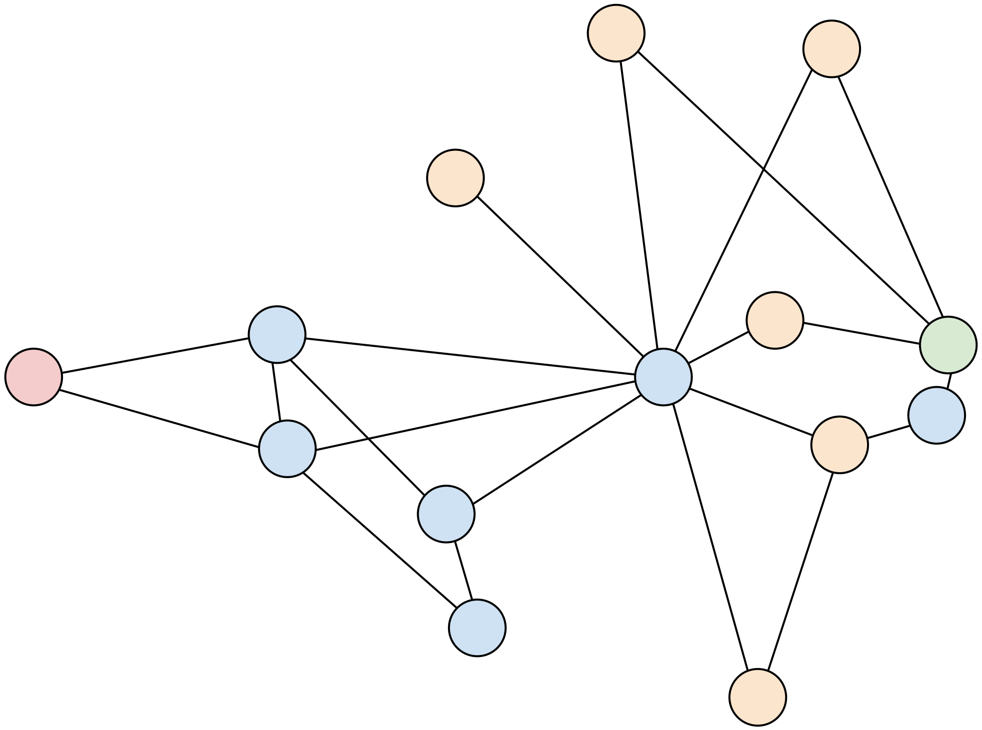
What about termination?

2 hops



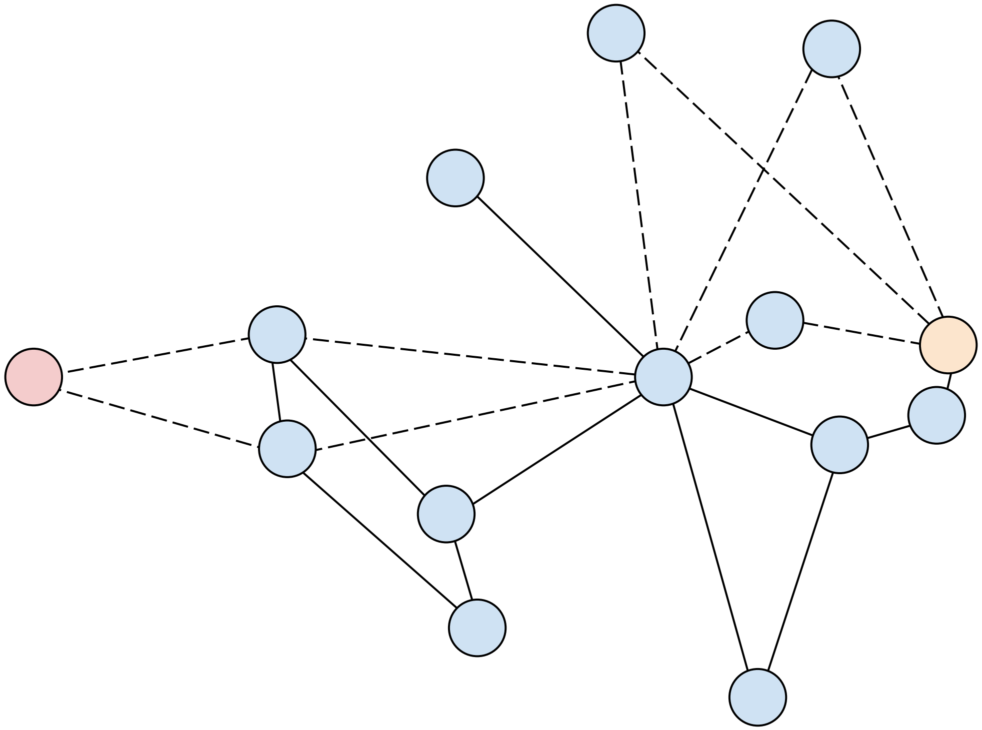
What about termination?

3 hops



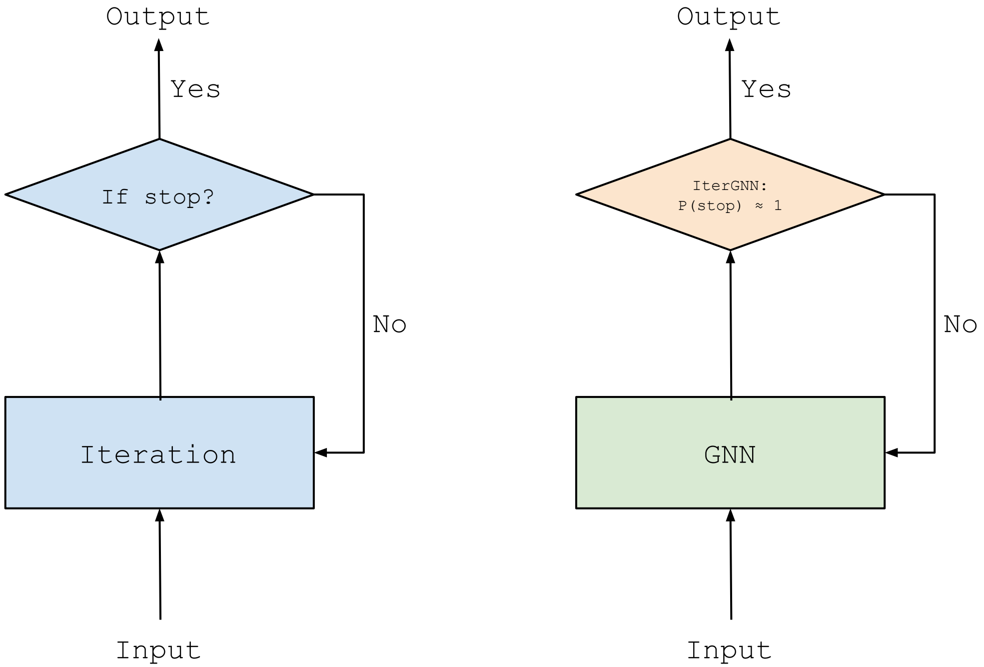
What about termination?

4 hops



What about termination?

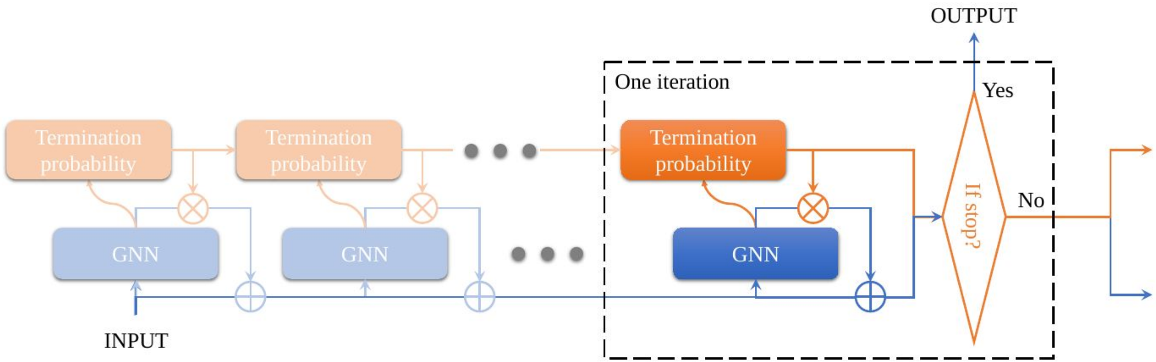
Idea: *Learn* termination with *IterGNN*



Tang et al., 2020

What about termination?

Idea: *Learn* termination with *IterGNN*

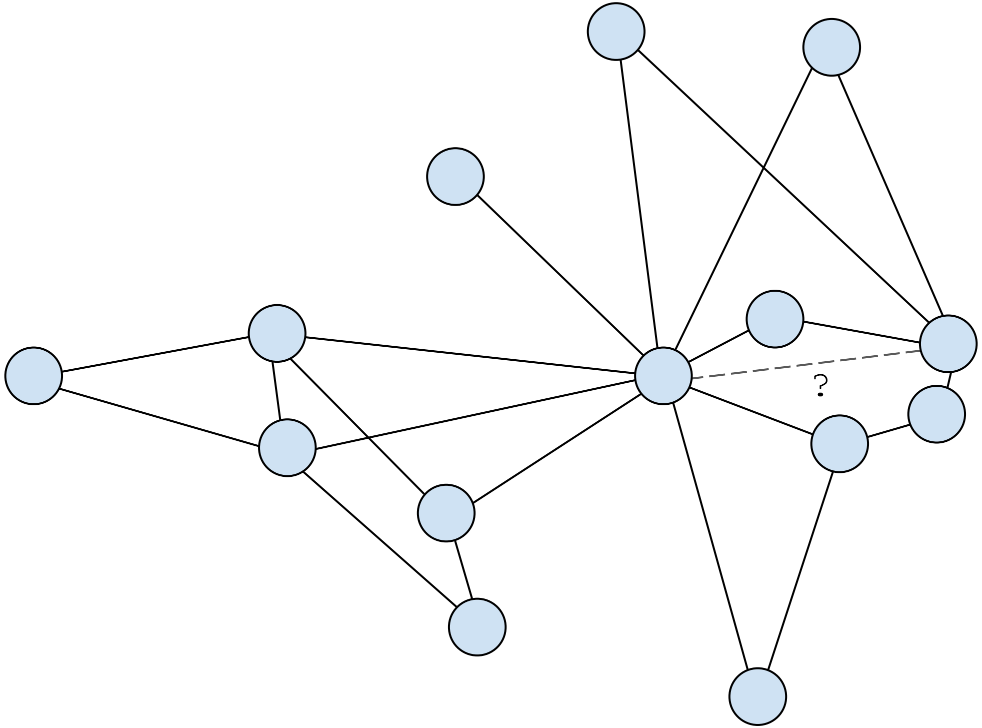


Algorithm 1: Iterative module. g is the stopping criterion and f is the iteration body

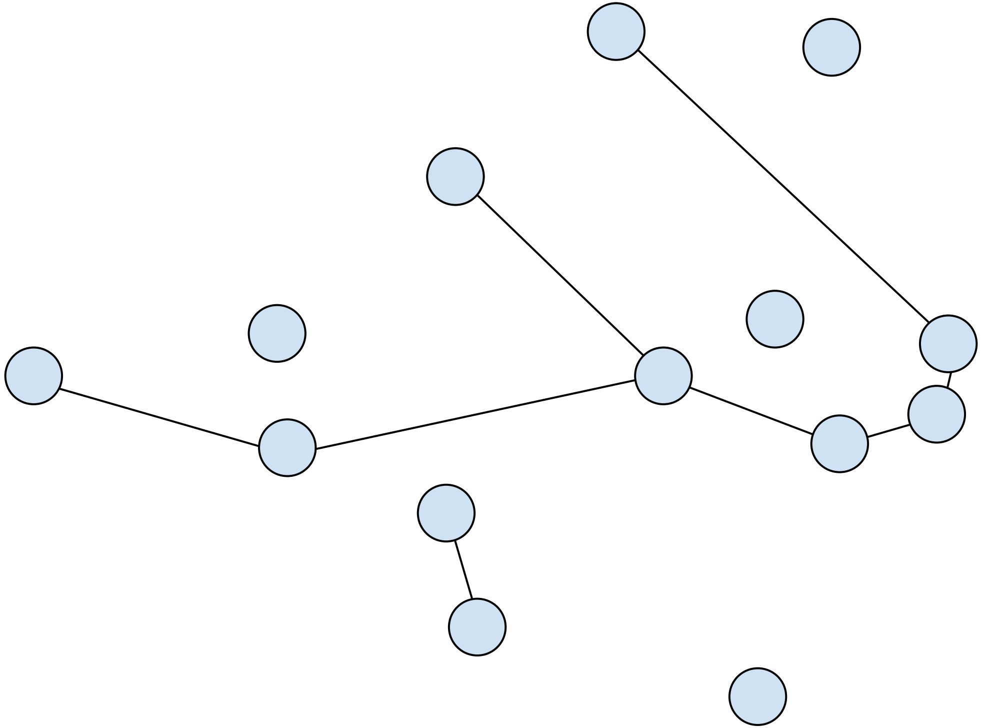
input: initial feature x; stopping threshold ϵ
$k \leftarrow 1$
$h^0 \leftarrow x$
while $\prod_{i=1}^{k-1}(1 - c^i) > \epsilon$ **do**
  $h^k \leftarrow f(h^{k-1})$
  $c^k \leftarrow g(h^k)$
  $k \leftarrow k + 1$
end while
return $h = \sum_{j=1}^{k-1}\left(\prod_{i=1}^{j-1}(1 - c^i)\right) c^j h^j$

Tang et al., 2020

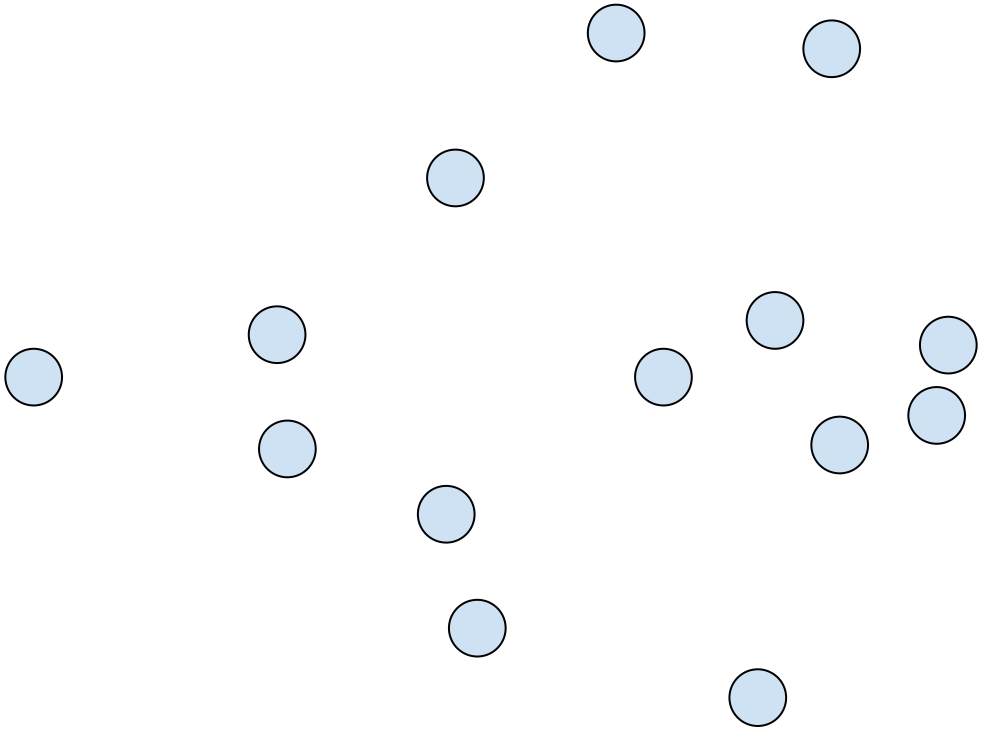
Static graph structure



Static graph structure



Static graph structure



How to overcome static graph structure?

Idea: *Augment* the graph with *dynamic* edges

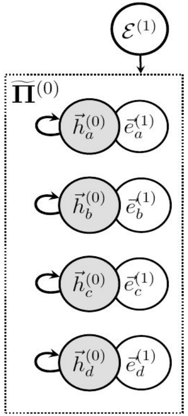
(1) encode entity representations

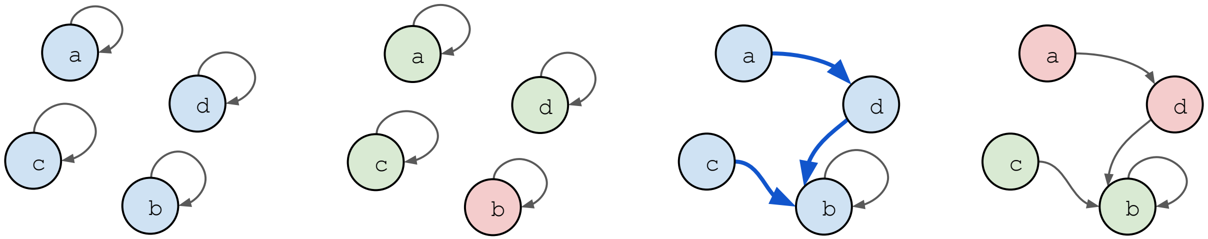
(2) compute new hidden representations

(3) decode answer

(4) calc pointer mask

(5) re-estimate pointer via self-attention

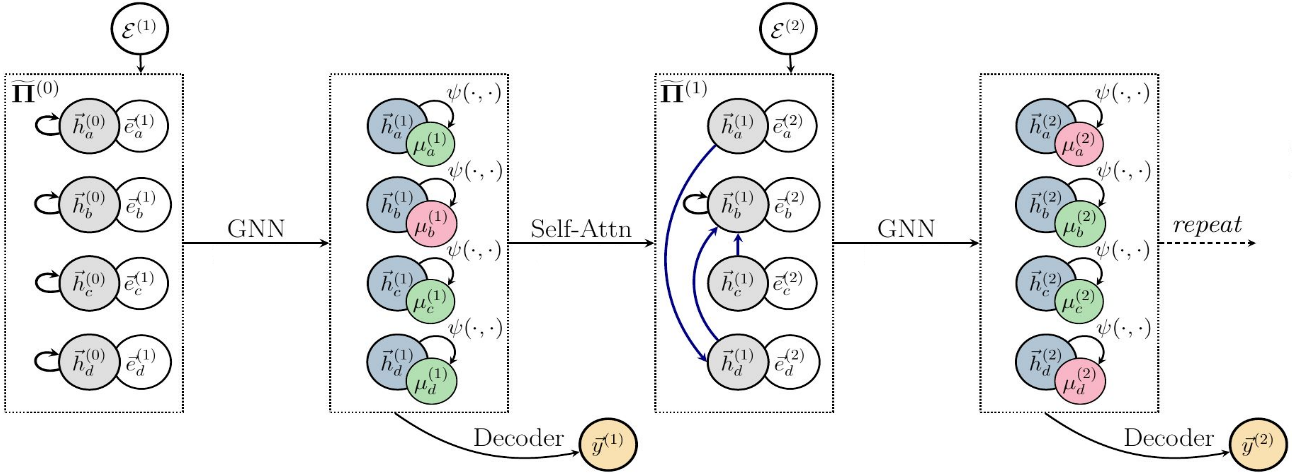




Veličković et al., 2020

How to overcome static graph structure?

Idea: *Augment* the graph with *inferred* edges



$$\vec{z}_i^{(t)} = f\left(\vec{e}_i^{(t)}, \vec{h}_i^{(t-1)}\right)$$

$$\mathbf{H}^{(t)} = P\left(\mathbf{Z}^{(t)}, \mathbf{\Pi}^{(t-1)}\right)$$

$$\vec{y}^{(t)} = g\left(\bigoplus_i \vec{z}_i^{(t)}, \bigoplus_i \vec{h}_i^{(t)}\right)$$

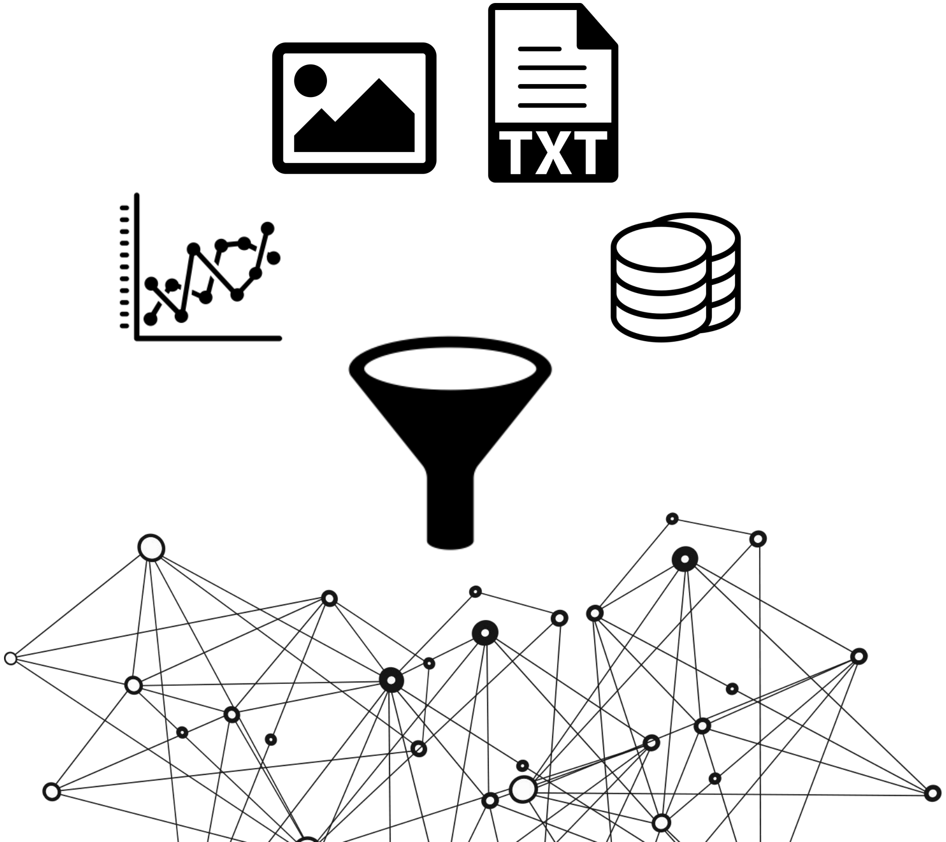
$$\mathbb{P}\left(\mu_i^{(t)} = 1\right) = \psi\left(\vec{z}_i^{(t)}, \vec{h}_i^{(t)}\right)$$

$$\vec{q}_i^{(t)} = \mathbf{W}_q \vec{h}_i^{(t)} \qquad \vec{k}_i^{(t)} = \mathbf{W}_k \vec{h}_i^{(t)} \qquad \alpha_{ij}^{(t)} = \mathrm{softmax}_j\left(\left\langle \vec{q}_i^{(t)}, \vec{k}_j^{(t)} \right\rangle\right)$$

$$\widetilde{\mathbf{\Pi}}_{ij}^{(t)} = \mu_i^{(t)} \widetilde{\mathbf{\Pi}}_{ij}^{(t-1)} + \left(1 - \mu_i^{(t)}\right) \mathbb{I}_{j = \mathrm{argmax}_k\left(\alpha_{ik}^{(t)}\right)} \qquad \mathbf{\Pi}_{ij}^{(t)} = \widetilde{\mathbf{\Pi}}_{ij}^{(t)} \vee \widetilde{\mathbf{\Pi}}_{ji}^{(t)}$$

Veličković et al., 2020

When *not* to use GNNs?



When *should* we use GNNs?

$$S = \{p \in P | M \text{ solves } p\}$$

"better than random guessing"

$$\text{ForE} = \frac{|S(\text{Edges}) \cup S(\text{Features})|}{|P|}$$

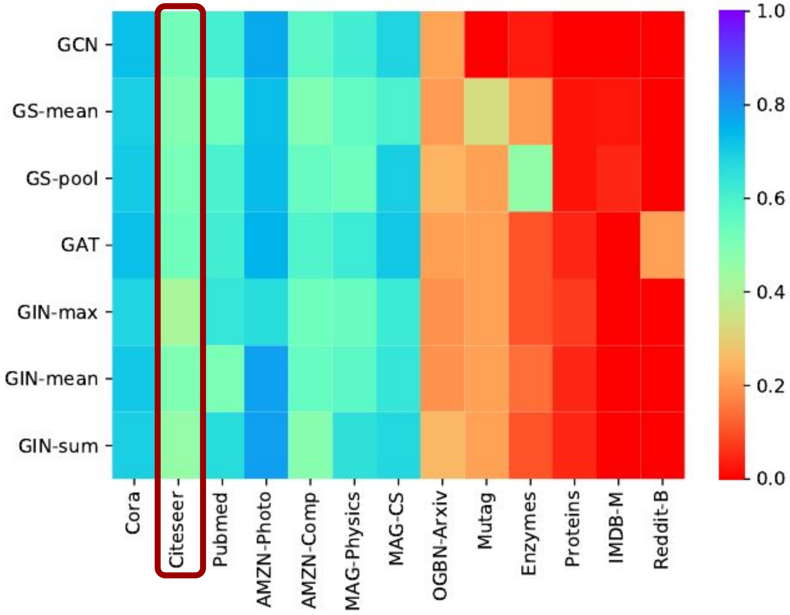
Dataset	Features	Edges	$\mathbb{E}$(FandE)	FandE	ForE	GNN
Cora	0.586	0.346	0.203	0.192	0.74	0.828
Citeseer	0.544	0.412	0.224	0.235	0.721	0.699
Pubmed	0.693	0.407	0.282	0.246	0.854	0.779
AMZN-Photo	0.777	0.286	0.222	0.172	0.891	0.909
AMZN-Comp	0.652	0.391	0.255	0.235	0.808	0.809
MAG-Physics	0.915	0.507	0.464	0.475	0.947	0.949
MAG-CS	0.924	0.136	0.126	0.129	0.932	0.933
OGBN-Arxiv	0.658	0.411	0.271	0.281	0.788	0.726
Mutag	0.45	0.55	0.248	0.45	0.55	0.55
Enzymes	0.4	0.333	0.133	0.2	0.533	0.65
Proteins	0.607	0.643	0.39	0.607	0.643	0.616
IMDB-M	0.26	0.293	0.076	0.24	0.313	0.287
Reddit-B	0.76	0.775	0.589	0.76	0.775	0.77

Faber et al., 2021

When *should* we use GNNs?

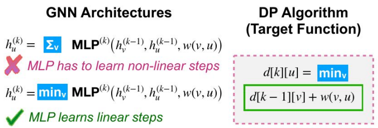
$$\frac{|S(\mathrm{GNN}) \cap \mathrm{U}|}{|\mathrm{U}|}$$

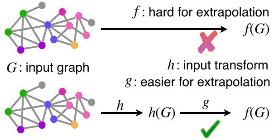
$$U = P \setminus (S(\mathrm{Features}) \cup S(\mathrm{Edges}))$$

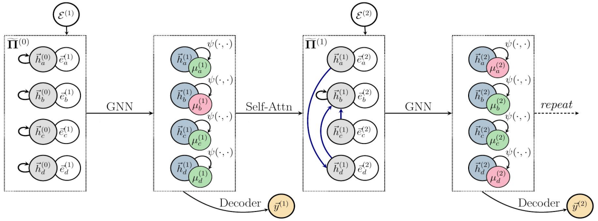


Faber et al., 2021

Famous last words



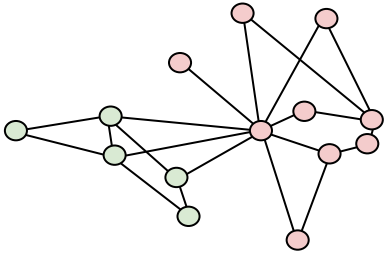




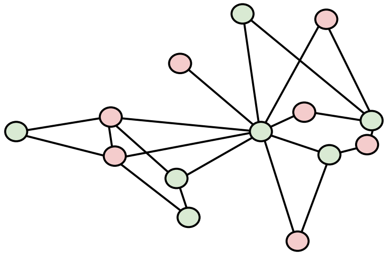
$$\text{ForE} = \frac{|S(\text{Edges}) \cup S(\text{Features})|}{|P|}$$

architectural overfitting to characteristics of evaluation data

graph homophily



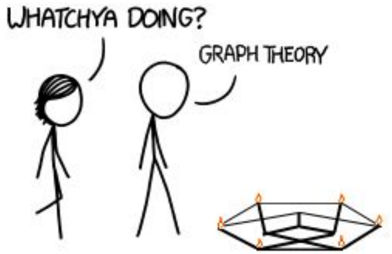
graph heterophily

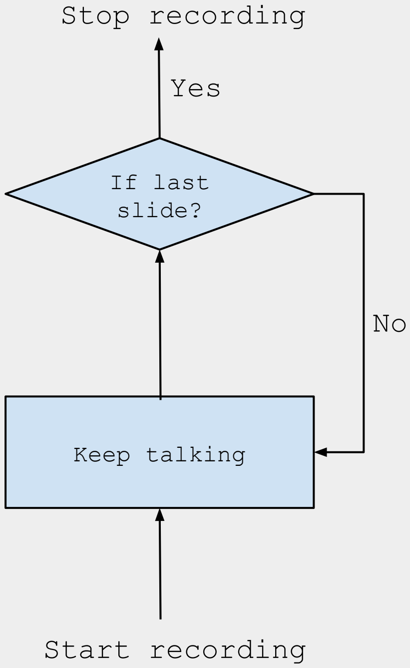


Zhu et al., 2020

Thank you for your attention!

Looking forward to the discussion!





References

- Battaglia et al., “*Relational inductive biases, deep learning and graph networks*”, 2018
- Trask et al., “*Neural Arithmetic Logic Units*”, 2018
- Xu et al., “*What can neural networks reason about?*”, 2019
- Veličković et al., “*Pointer Graph Networks*”, 2020
- Cranmer et al., “*Discovering symbolic models from deep learning with Inductive Biases*”, 2020
- Xu et al., “*How neural networks extrapolate: From feedforward to graph neural networks*”, 2020
- Veličković et al., “*Neural execution of graph algorithms*”, 2020
- Tang et al., “*Towards scale-invariant graph-related problem solving by iterative homogeneous graph neural networks*”, 2020
- Zhu et al., “*Graph Neural Networks with Heterophily*”, 2020
- Faber et al., “*Should graph neural networks use features, edges, or both?*”, 2021