DeepMind

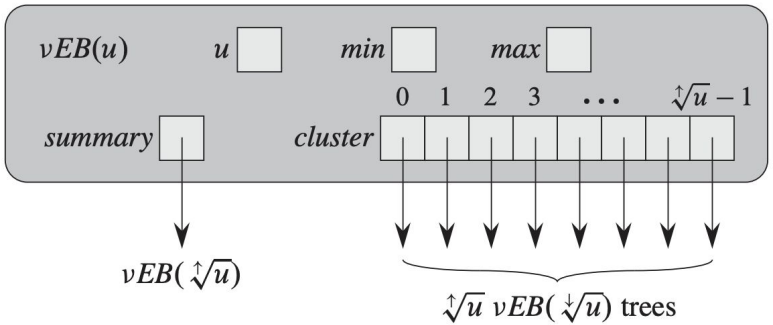
Algorithmic Inductive Biases

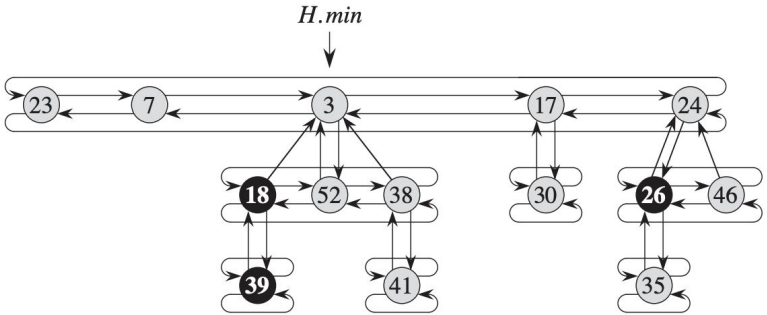
Petar Veličković

Mila Tea Talk Series
3 July 2020

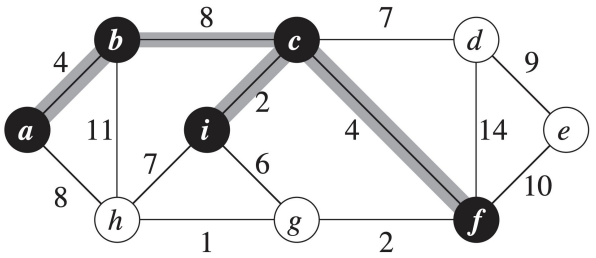
Introduction, motivation, and disclaimer

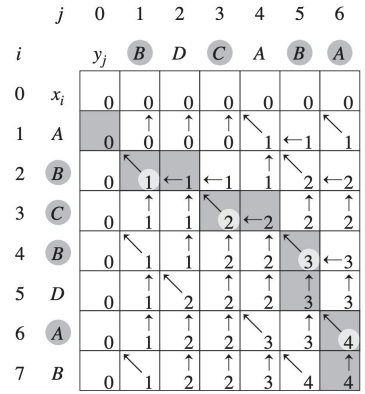
- We will investigate **structural inductive biases**, and when is the right time to impose them, through the lens of learning representations of **algorithms** and **data structures**.

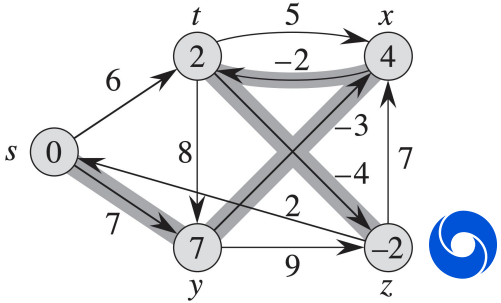





```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```







Algorithm figures: Cormen, Leiserson, Rivest and Stein. **Introduction to Algorithms.**

Introduction, motivation, and disclaimer

- We will investigate **structural inductive biases**, and when is the right time to impose them, through the lens of learning representations of **algorithms** and **data structures**.

- Algorithms are a **natural** ground for studying these rigorously:
  - Know **exactly** how the inputs and outputs are produced;
  - No **noise** involved;
  - Algorithm has **interpretable** subroutines and control flow;
  - **Credit assignment** can be directly investigated.

- In this talk, I will be dealing primarily with reasoning over sets of **flat** object representations.
  - Assume perception / "System 1" is either *solved* or *not present* in the problem

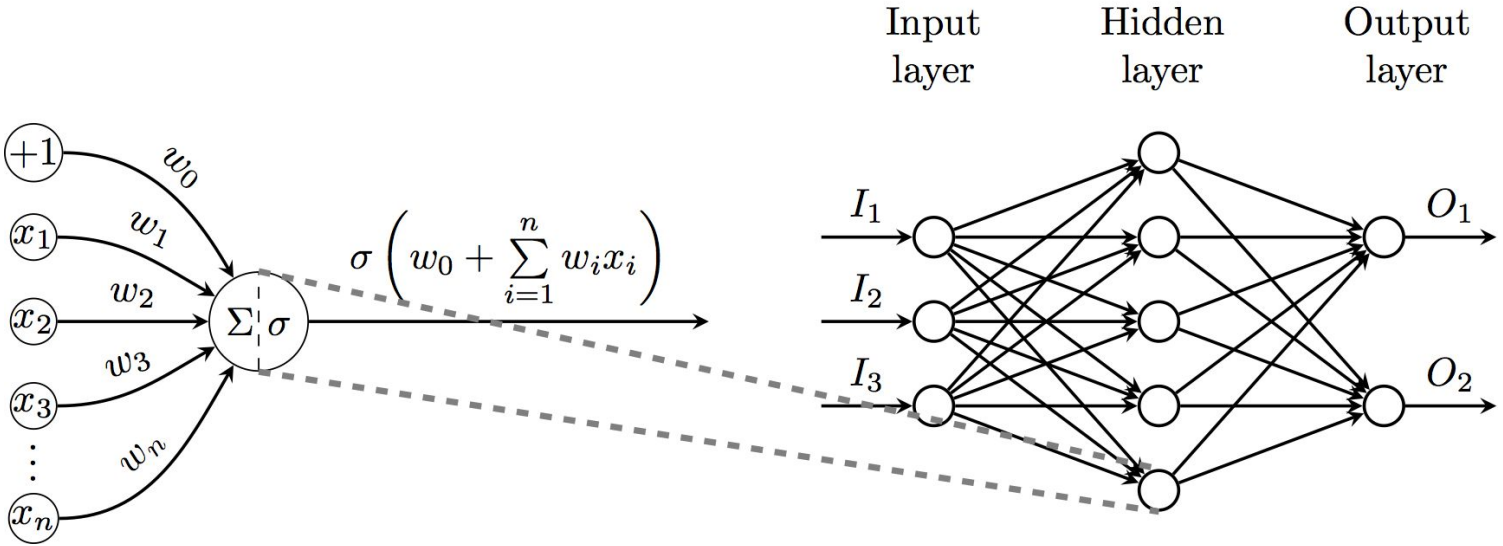
Starting simple

- **Input**: (flat) representation of an object's **features** (e.g. *position, shape, color…*)
- **Output**: some **property** of the object (e.g. *is it round and yellow?*)

Starting simple

- **Input**: (flat) representation of an object's **features** (e.g. *position, shape, color…*)
- **Output**: some **property** of the object (e.g. *is it round and yellow?*)

A canonical problem solvable by a *multilayer perceptron* (**MLP**).



Starting simple

- **Input**: (flat) representation of an object's **features** (e.g. *position, shape, color…*)
- **Output**: some **property** of the object (e.g. *is it round and yellow?*)

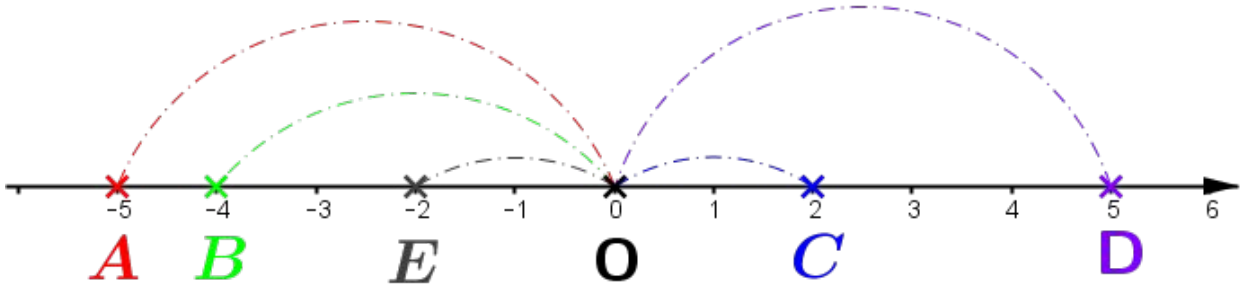
A canonical problem solvable by a *multilayer perceptron* (**MLP**).

- A simple *universal approximator* that makes **no** assumptions about its input structure.

- We will now gradually introduce **inductive biases** as we learn more about our problem.
  - Every step of the way, we will **validate** our choice theoretically or empirically.
  - **N.B.** _All_ architectures considered here will be universal approximators!
    - But proper choices of biases will drastically improve learning *generalisation*.

Summary statistics

- **Input**: A **set** of 1D points, with features containing their coordinate and colour.
- **Output**: Some **aggregate** property of the set (e.g. the *furthest pairwise distance*).



(Output: 10)

Summary statistics

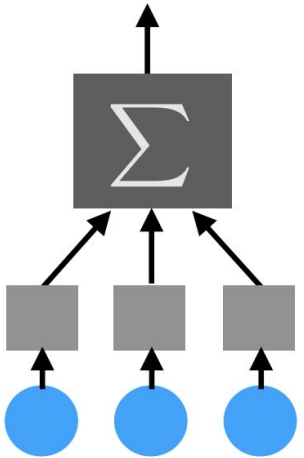
- **Input**: A **set** of 1D points, with features containing their coordinate and colour.
- **Output**: Some **aggregate** property of the set (e.g. the *furthest pairwise distance*).

A **summary statistic** problem: requires reasoning about set element boundaries, computing the *maximal* and *minimal* coordinate, and subtracting them.

MLPs have no way of dealing with set boundaries!

Introduce **Deep Sets.** (Zaheer *et al.*, NeurIPS 2017)

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



Summary statistics

- **Input**: A **set** of 1D points, with features containing their coordinate and colour.
- **Output**: Some **aggregate** property of the set (e.g. the *furthest pairwise distance*).

A **summary statistic** problem: requires reasoning about set element boundaries, computing the *maximal* and *minimal* coordinate, and subtracting them.

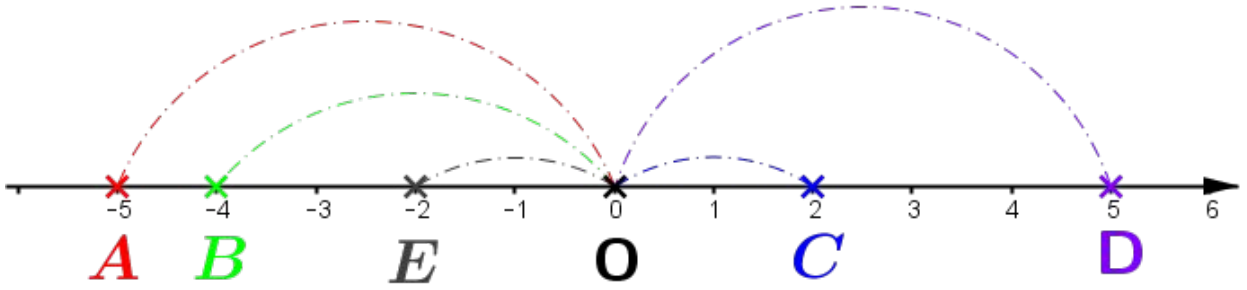
MLPs have no way of dealing with set boundaries!

Introduce **Deep Sets.** (Zaheer *et al.*, NeurIPS 2017)

- *Permutation-invariant* and *object-aware*!
- Can be extended to powerful variants aggregating over *subsets* at a time
  - See *Janossy pooling* (Murphy *et al.*, ICLR 2019)

Relational argmax

- **Input**: A **set** of 1D points, with features containing their coordinate and colour.
- **Output**: Some **relational** property of the set (e.g. the ***colours*** *of two furthest points*)



(Output: red and purple)

Relational argmax

- **Input**: A **set** of 1D points, with features containing their coordinate and colour.
- **Output**: Some **relational** property of the set (e.g. the ***colours*** *of two furthest points*)

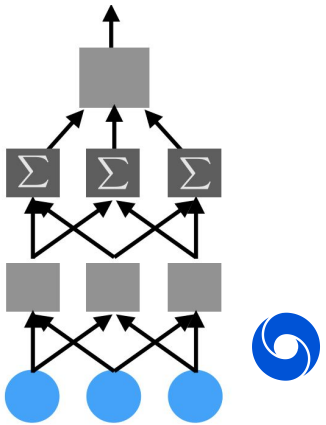
A **relational argmax** problem: requires **identifying** an optimising (pairwise) *relation*.

Deep Sets at a disadvantage: output MLP must **disentangle** all pairwise relations, imposing substantial **pressure** on its internal representations. (this will be a common and recurring theme)

Introduce **Graph Neural Networks** (GNNs). (Scarselli *et al.*, TNN 2009)

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

$$y = \mathrm{MLP}_2 \left(\sum_{s \in S} h_s^{(K)} \right)$$



Relational argmax

- **Input**: A **set** of 1D points, with features containing their coordinate and colour.
- **Output**: Some **relational** property of the set (e.g. the ***colours*** *of two furthest points*)

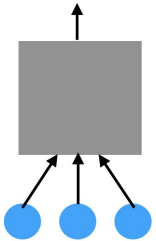
A **relational argmax** problem: requires **identifying** an optimising (pairwise) *relation*.

Deep Sets at a disadvantage: output MLP must **disentangle** all pairwise relations, imposing substantial **pressure** on its internal representations. (this will be a common and recurring theme)

Introduce **Graph Neural Networks** (GNNs). (Scarselli *et al.*, TNN 2009)

- Permutation-invariant, object-aware, and **relation-aware**!
- Directly provides "pairwise embeddings" within its inductive bias
- (**Powerful** paradigm: higher-order relations can be decomposed into *multi-step pairwise*)

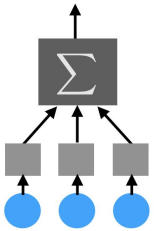
Architectures so far



MLPs

~ feature extraction

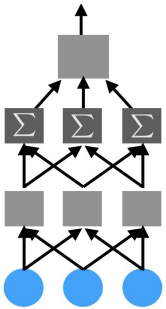
$$y = \mathrm{MLP}(\|_{s \in S} X_s)$$



Deep Sets (Zaheer *et al.*, NeurIPS 2017)

~ summary statistics

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



GNNs (Scarselli *et al.*, TNN 2009)

~ (pairwise) relations

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

$$y = \mathrm{MLP}_2 \left(\sum_{s \in S} h_s^{(K)} \right)$$

Algorithmic alignment

- At each step, we progressively made **stronger** assumptions about what kind of reasoning our problem needed, leading to stronger *inductive biases*.

- Under this, "noise-free", *algorithmic reasoning* lens, can we formalise what it means for an inductive bias to be favourable, and **prove** that it is favourable in some circumstance?

- **Yes!**

(Xu, Li, Zhang, Du, Kawarabayashi and Jegelka. ICLR 2020)

WHAT CAN NEURAL NETWORKS REASON ABOUT?

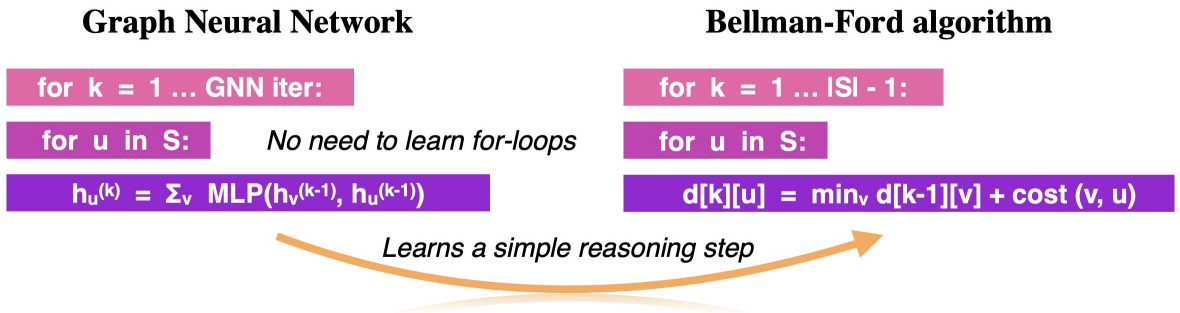
- I will only present this proof **intuitively**.

  (tl;dr: it relies on PAC-like frameworks, using *sample complexity* as a notion of favourability)

(Xu, Li, Zhang, Du, Kawarabayashi and Jegelka. ICLR 2020)

What Can Neural Networks Reason About?

- Which networks are best suited for certain types of **reasoning**?
  - **Theorem**: better *structural alignment* implies better *generalisation*!
  - GNNs ~ dynamic programming



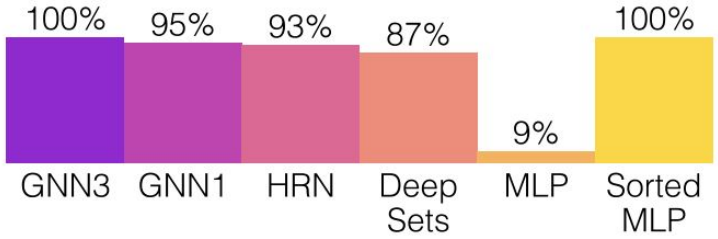
$$\text{Answer}[k][i] = \text{DP-Update}(\{\text{Answer}[k-1][j],\ j = 1 \dots n\})$$

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

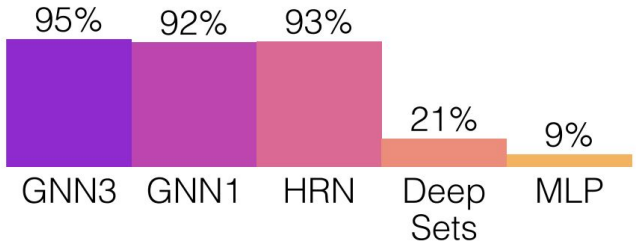
(Xu, Li, Zhang, Du, Kawarabayashi and Jegelka. ICLR 2020)

Empirical results

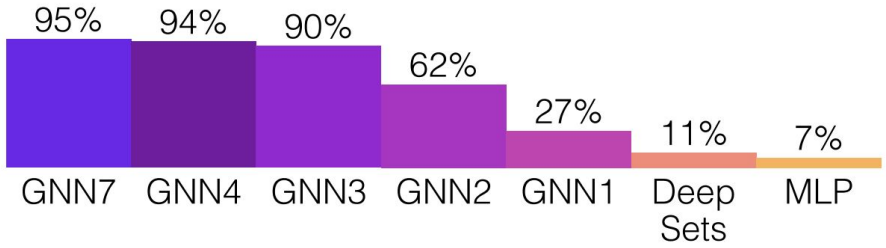
Summary statistics
What is the maximum value difference among treasures?



Relational argmax
What are the colors of the furthest pair of objects?

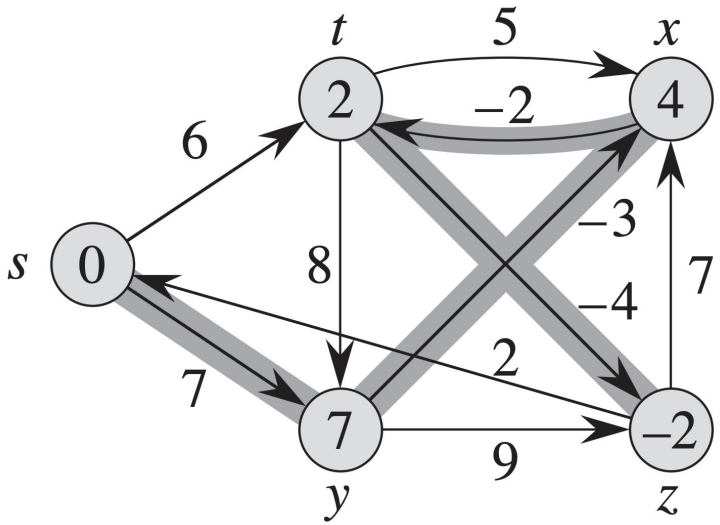


Dynamic programming
What is the cost to defeat monster X by following the optimal path?



Dynamic programming

- **Input**: A weighted **graph** with a provided **source** node
- **Output**: All **shortest paths** out of the source node (*shortest path tree*)



Dynamic programming

- **Input**: A weighted **graph** with a provided **source** node
- **Output**: All **shortest paths** out of the source node (*shortest path tree*)

Standard computer science task, solvable by dynamic programming methods (e.g. Bellman-Ford). Note that, at each step, Bellman-Ford **selects** an optimal neighbour in each node.

So far, we used the **sum** aggregator to aggregate GNN neighbourhoods. It is universal, but does not *align* with the task (and can lead to exploding signals)!

Introduce the **max** aggregator.

$$\vec{h}_i' = U\left(\vec{h}_i, \max_{j \in \mathcal{N}_i} M\left(\vec{h}_i, \vec{h}_j, \vec{e}_{ij}\right)\right)$$

Dynamic programming

- **Input**: A weighted **graph** with a provided **source** node
- **Output**: All **shortest paths** out of the source node (*shortest path tree*)

Standard computer science task, solvable by dynamic programming methods (e.g. Bellman-Ford). Note that, at each step, Bellman-Ford **selects** an optimal neighbour in each node.

So far, we used the **sum** aggregator to aggregate GNN neighbourhoods. It is universal, but does not *align* with the task (and can lead to exploding signals)!

Introduce the **max** aggregator.

Naturally aligns with many **search-like** reasoning procedures, has explicit **credit assignment**, and is more robust to **larger-size out-of-distribution** tests!

Empirical validation into max aggregation

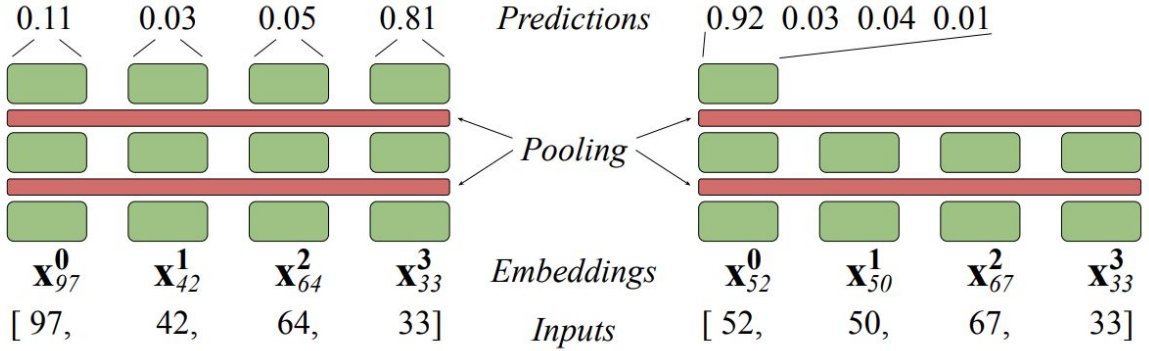
- A recent exploration of Transformers studies the effect of alignment on learning stability.

(Richter and Wattenhofer. 2020)

Normalized Attention Without Probability Cage

- Specify a *case distinction* task that clearly aligns with max.

```
// Case distinction task data generator
inputs ← random integer sequence
if 64 in inputs                 // argmin case
    label ← argmin(inputs)
else if 50 in inputs            // first case
    label ← 0
else                            // argmax case
    label ← argmax(inputs)
return (inputs, label)
```



(Richter and Wattenhofer. 2020)

Max is stable under *most* hyperparameters!

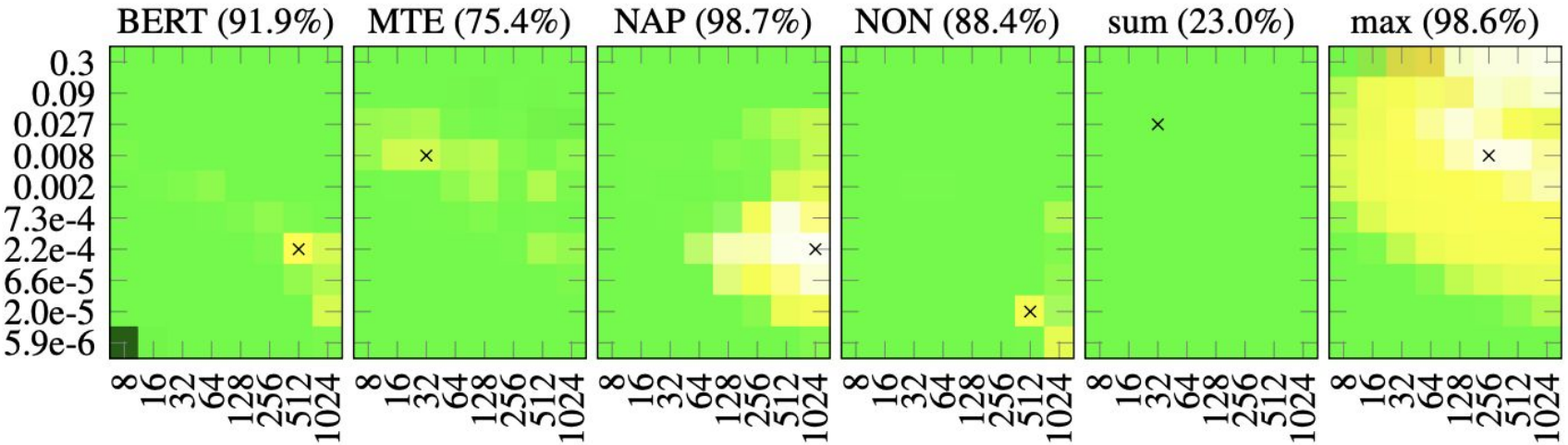


Figure 5: Learning rate (y-axis) vs. model dimension d (x-axis) on the case distinction task with output from the first token. RGB pixel values correspond to *argmin*-, *first*- and *argmax*-mean-case-accuracies. Crosses indicate the best mean accuracy, which we report behind the model name.

Shortest paths, *cont'd*

- The GNN will still struggle on the shortest-path task when generalising **out-of-distribution**!
  - A critical component of proper *reasoning* systems.

- It can *overfit* to the distribution of inputs of a particular (training) size, side-stepping the actual procedure it is attempting to imitate.

- Introducing **step-wise supervision**.

(Veličković, Ying, Padovano, Hadsell and Blundell. ICLR 2020)

NEURAL EXECUTION OF GRAPH ALGORITHMS

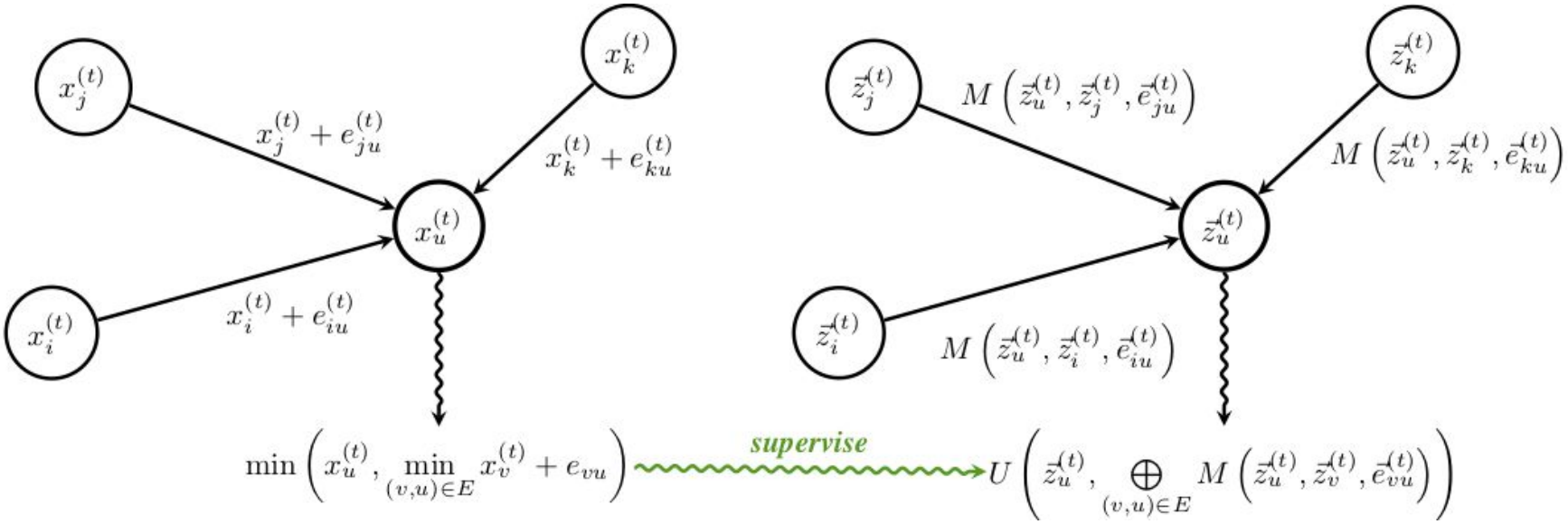
- Instruct the GNN computation to respect the **intermediate outputs** of the algorithm!

(other aspects, such as algorithm multi-task learning, are out-of-scope of this talk)

(Veličković, Ying, Padovano, Hadsell and Blundell. ICLR 2020)

Neural Execution of Graph Algorithms

Supervise on appropriate output values **at every step.**



Bellman-Ford algorithm

Message-passing neural network

(Veličković, Ying, Padovano, Hadsell and Blundell. ICLR 2020)

Evaluation: Shortest paths (+ Reachability)

Model	Predecessor (mean step accuracy / last-step accuracy)		
	20 nodes	*50 nodes*	*100 nodes*
LSTM (Hochreiter & Schmidhuber, 1997)	47.20% / 47.04%	36.34% / 35.24%	27.59% / 27.31%
GAT* (Veličković et al., 2018)	64.77% / 60.37%	52.20% / 49.71%	47.23% / 44.90%
GAT-full* (Vaswani et al., 2017)	67.31% / 63.99%	50.54% / 48.51%	43.12% / 41.80%
MPNN-mean (Gilmer et al., 2017)	93.83% / 93.20%	58.60% / 58.02%	44.24% / 43.93%
MPNN-sum (Gilmer et al., 2017)	82.46% / 80.49%	54.78% / 52.06%	37.97% / 37.32%
MPNN-max (Gilmer et al., 2017)	**97.13% / 96.84%**	**94.71% / 93.88%**	**90.91% / 88.79%**
MPNN-max (*curriculum*)	95.88% / 95.54%	91.00% / 88.74%	84.18% / 83.16%
MPNN-max (*no-reach*)	82.40% / 78.29%	78.79% / 77.53%	81.04% / 81.06%
MPNN-max (*no-algo*)	78.97% / 95.56%	83.82% / 85.87%	79.77% / 78.84%

Trained on **20-node** graphs!

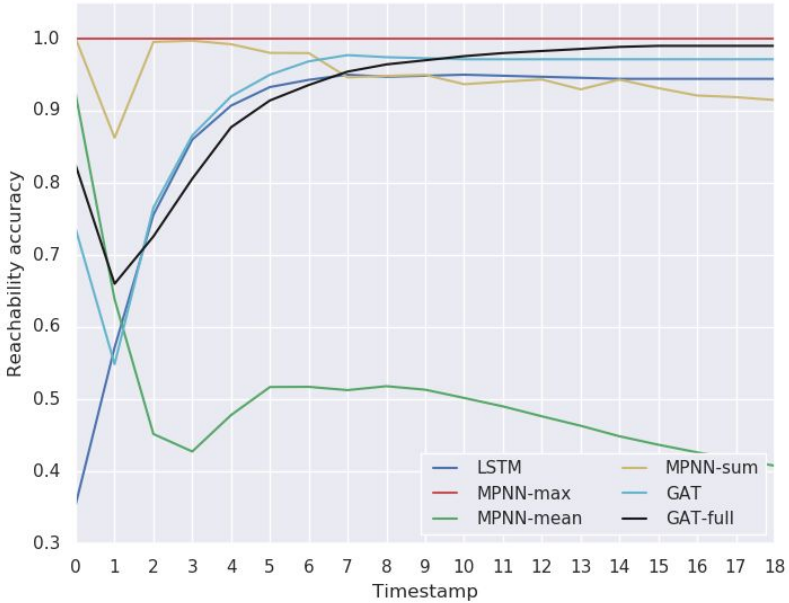
Aggregators other than max

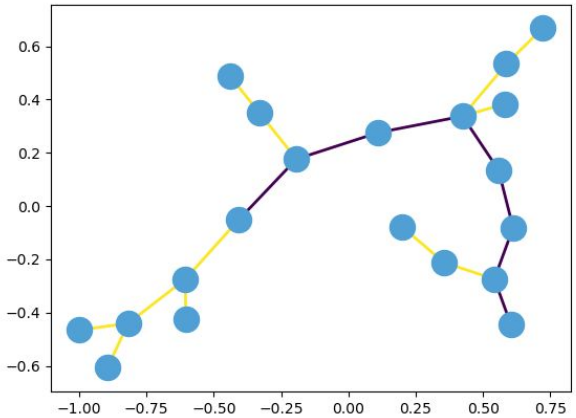
Trained without step-wise supervision

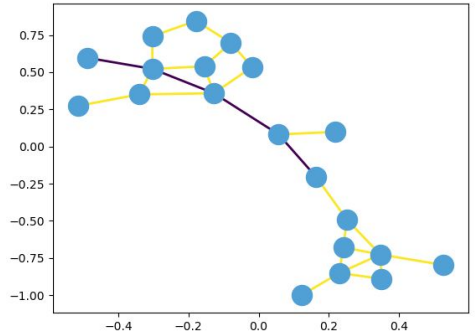
(Veličković, Ying, Padovano, Hadsell and Blundell. ICLR 2020)

Credit assignment, and catching "cheating" executors

- We directly illustrate how to verify if the learned function is "useful" on the **reachability** task, plotting **step-wise accuracy** and **explaining GNN outputs**. (Ying et al., NeurIPS 2019)



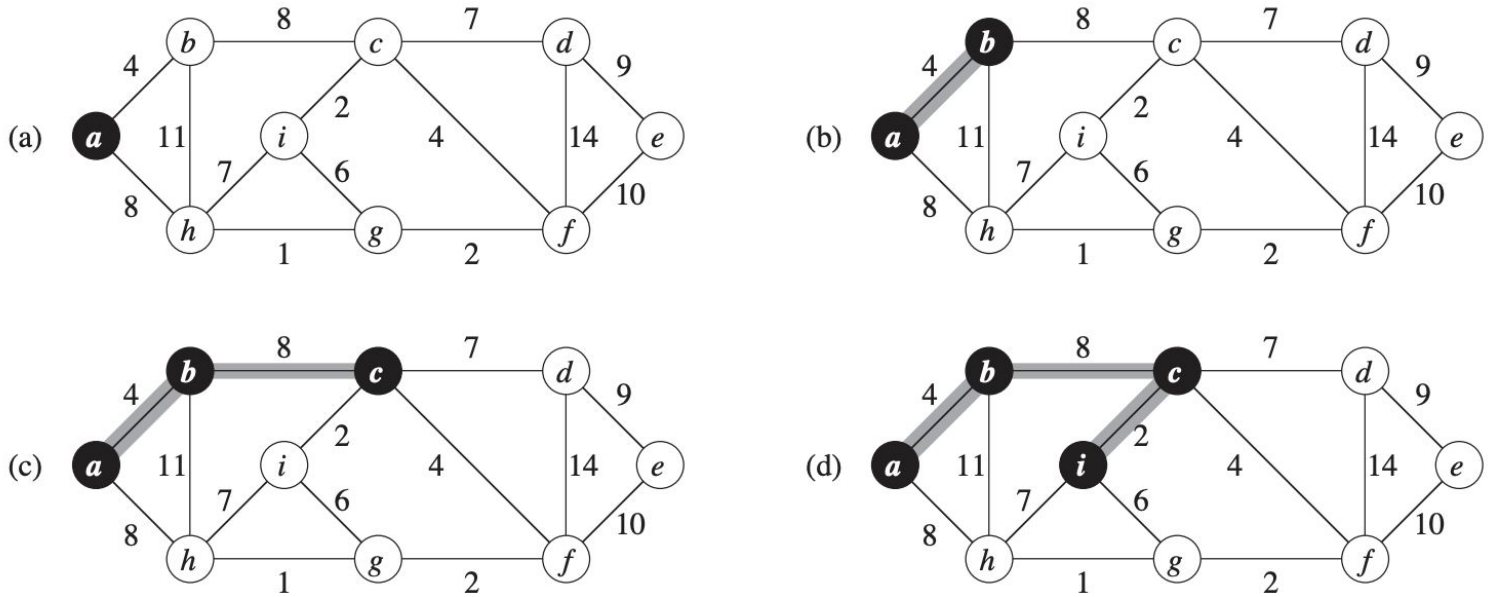




(Veličković, Ying, Padovano, Hadsell and Blundell. ICLR 2020)

Sequential algorithms

- Now consider algorithms such as **Prim**'s algorithm for minimum spanning trees (MST).
- This algorithm is inherently **sequential**: it adds *one node at a time* to the (partial) MST.



Sequential algorithms

- Now consider algorithms such as **Prim**'s algorithm for minimum spanning trees (MST).

- This algorithm is inherently **sequential**: it adds *one node at a time* to the (partial) MST.

Our previous model was forced to produce outputs for **every** node at **every** step. But in most cases, these outputs *don't change*, making the system vulnerable to overfitting.

Introduce a **sequential** inductive bias:

- At each step, select exactly **one** node to update, leaving all others unchanged.
- Can assign a score to every node by shared network, and choose **argmax**.
    - Optimise using cross-entropy on algorithm trajectories.

(Veličković, Ying, Padovano, Hadsell and Blundell. ICLR 2020)

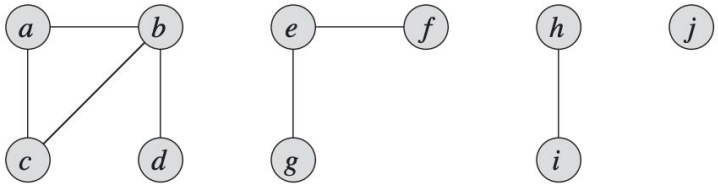
Evaluation: Sequential execution

Model	Accuracy (next MST node / MST predecessor) *20 nodes*	*50 nodes*	*100 nodes*
LSTM (Hochreiter & Schmidhuber, 1997)	11.29% / 52.81%	3.54% / 47.74%	2.66% / 40.89%
GAT* (Veličković et al., 2018)	27.94% / 61.74%	22.11% / 58.66%	10.97% / 53.80%
GAT-full* (Vaswani et al., 2017)	29.94% / 64.27%	18.91% / 53.34%	14.83% / 51.49%
MPNN-mean (Gilmer et al., 2017)	**90.56% / 93.63%**	52.23% / 88.97%	20.63% / 80.50%
MPNN-sum (Gilmer et al., 2017)	48.05% / 77.41%	24.40% / 61.83%	31.60% / 43.98%
MPNN-max (Gilmer et al., 2017)	87.85% / 93.23%	**63.89% / 91.14%**	**41.37% / 90.02%**
MPNN-max (*no-algo*)	— / 71.02%	— / 49.83%	— / 23.61%

The sequential inductive bias is very **helpful**!

Incremental connectivity task

- **Input:** (u, v) expressed by a 1 in nodes u and v, 0 otherwise
- **Output:** has adding / removing edge (u, v) resulted in one component less?
- **Very** challenging for (fully-connected) GNNs
  - Hidden state must usefully remember *everything* added so far
  - Vulnerable to *oversmoothing*!

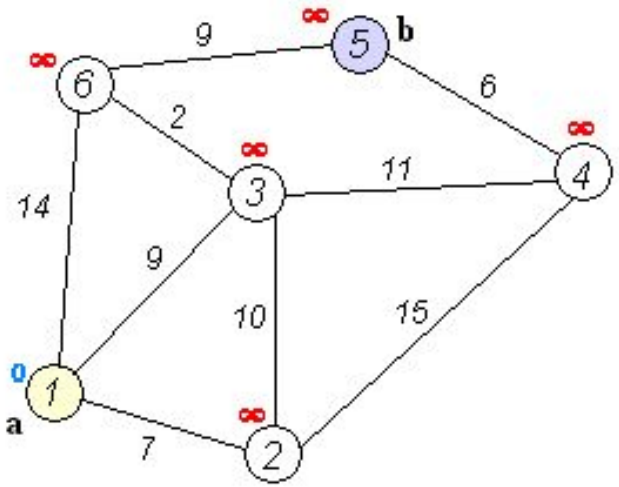


(a)

Edge processed	Collection of disjoint sets									
initial sets	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b,d)	{a}	{b,d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e,g)	{a}	{b,d}	{c}		{e,g}	{f}		{h}	{i}	{j}
(a,c)	{a,c}	{b,d}			{e,g}	{f}		{h}	{i}	{j}
(h,i)	{a,c}	{b,d}			{e,g}	{f}		{h,i}		{j}
(a,b)	{a,b,c,d}				{e,g}	{f}		{h,i}		{j}
(e,f)	{a,b,c,d}				{e,f,g}			{h,i}		{j}
(b,c)	{a,b,c,d}				{e,f,g}			{h,i}		{j}

(b)

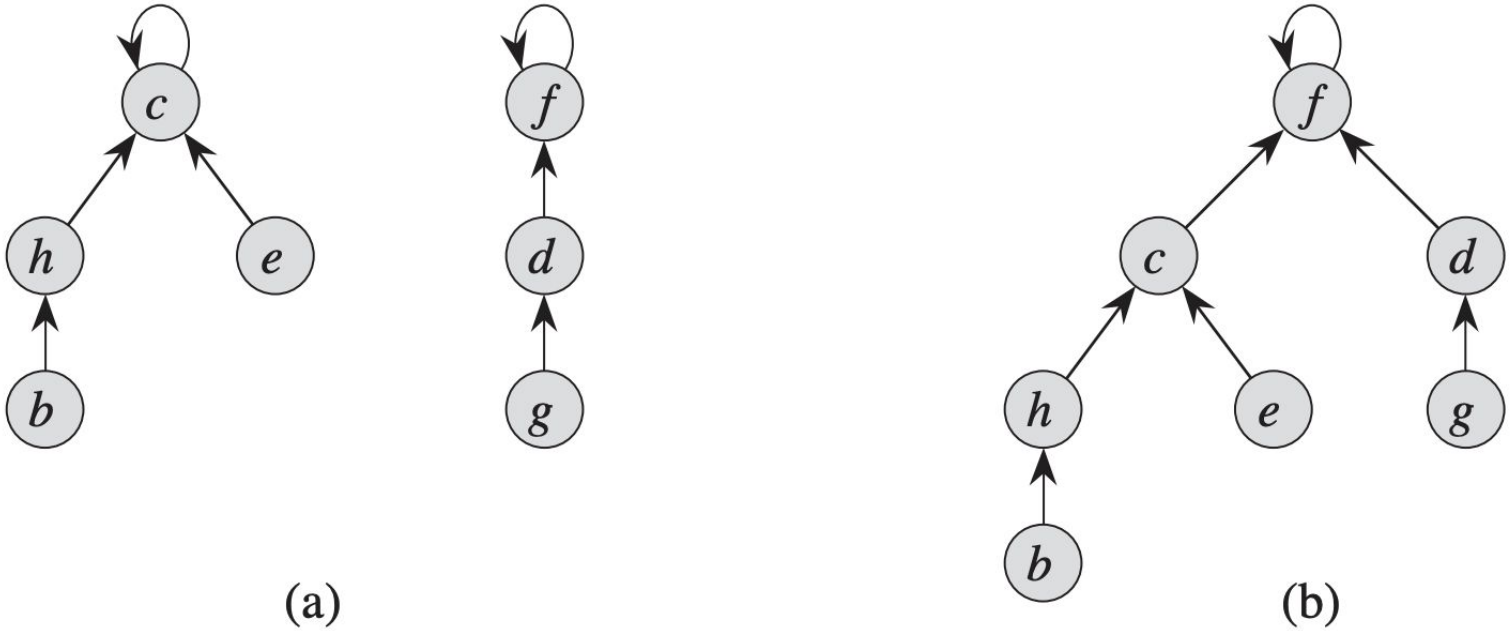
Graph refinement



- Are the input graph edges **most relevant**?
- Consider **Dijkstra**'s pathfinding algorithm (right):
  - Graph edges used for **relaxation**
  - But choosing which node to use next?
  - Relies on global state (or edges of a **heap**).
- Also useful to **add** edges which facilitate useful information propagation
- For the incremental connectivity queries, iterating only over the current graph's edges leads to **linear-time** query answering.

Connected components with disjoint-set unions

Maintaining a disjoint-set union (**DSU**) data structure allows answering such queries **sublinearly**!



GNNs with *supervised* pointer mechanisms

- Core idea: learn an (**auxiliary**) graph to be used for a GNN.
  - Derive based on the latent state.
  - A way to provide “global context”, or refine computational graph.

- Contrary to prior work, we let each node learn **one pointer** to another node.
  - Can model (**and supervise on!**) many influential *data structures*;
  - Preserves *sparsity* (O(V) edges used);
  - Relies on step-wise *predecessor predictions*, which we already covered.

(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

Pointer Graph Networks

(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

Pointers through Transformers

- Compute *queries*, *keys* and *attention coefficients* as usual

$$\vec{q}_i = \mathbf{W}_q \vec{h}_i \qquad \vec{k}_i = \mathbf{W}_k \vec{h}_i \qquad \alpha_{ij} = \mathrm{softmax}_{j \in V}\left(\vec{q}_i^T \vec{k}_j\right)$$

- Choose the largest coefficients as new pointers, forming the **pointer adjacency matrix**:

$$\mathbf{\Pi}_{ij} = \mathbb{I}_{j = \mathrm{argmax}_k(\alpha_{ik})}$$

- Use the (symmetrised) pointer adjacency matrix to form neighbourhoods for the GNN!

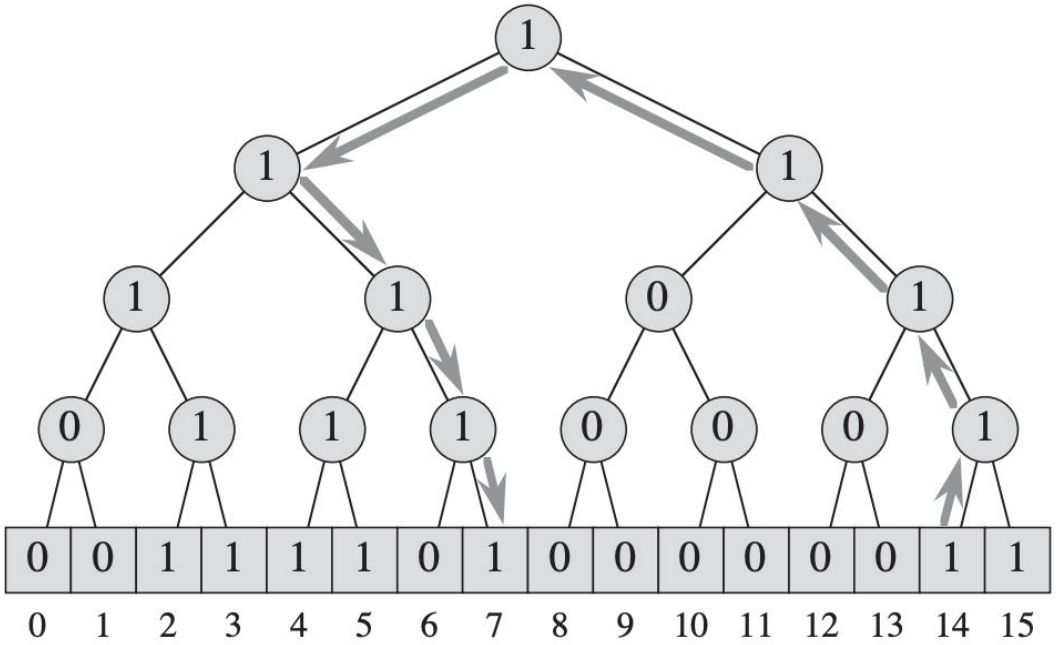
$$\vec{h}_i^{(t)} = U\left(\vec{z}_i^{(t)}, \max_{\mathbf{\Pi}_{ji}^{(t-1)}=1} M\left(\vec{z}_i^{(t)}, \vec{z}_j^{(t)}\right)\right)$$

- We optimise coefficients by using cross-entropy on ground-truth state of a *data structure*.

(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

Masking inductive bias

- Efficient data structures are sublinear because they only *modify* a **small** fraction of (e.g. logarithmically many) nodes at once!



(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

Masking inductive bias

- Efficient data structures are sublinear because they only *modify* a **small** fraction of (e.g. logarithmically many) nodes at once!

- Forcing to update all pointers at once is wasteful (and detrimental to performance!)
  - Let's revisit and generalise our **sequential** inductive bias!

- If we know the data structure will only update a subset of pointers at any point, we can learn to predict this **subset mask**, μ_i first -- then discard updates to other nodes.
  - This inductive bias proved **critical**.

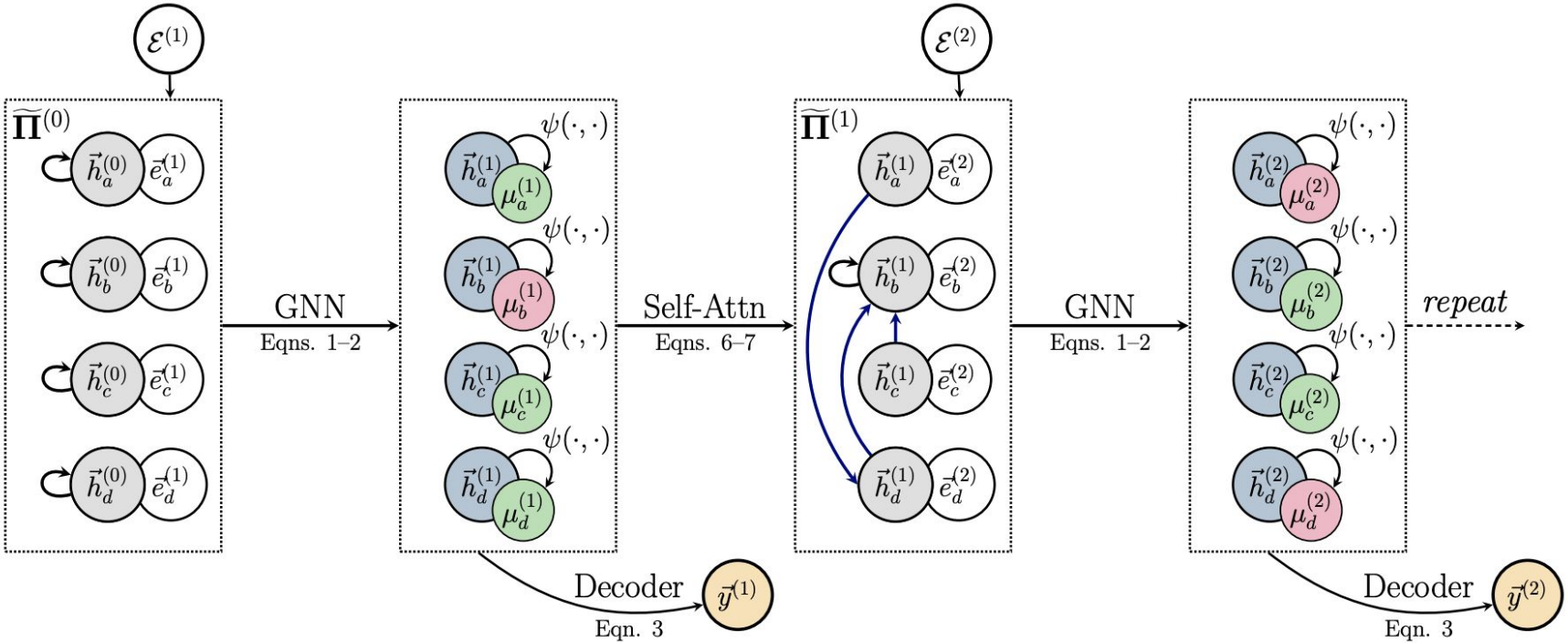
$$\mathbb{P}\left(\mu_i^{(t)} = 1\right) = \psi\left(\vec{z}_i^{(t)}, \vec{h}_i^{(t)}\right)$$

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

(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

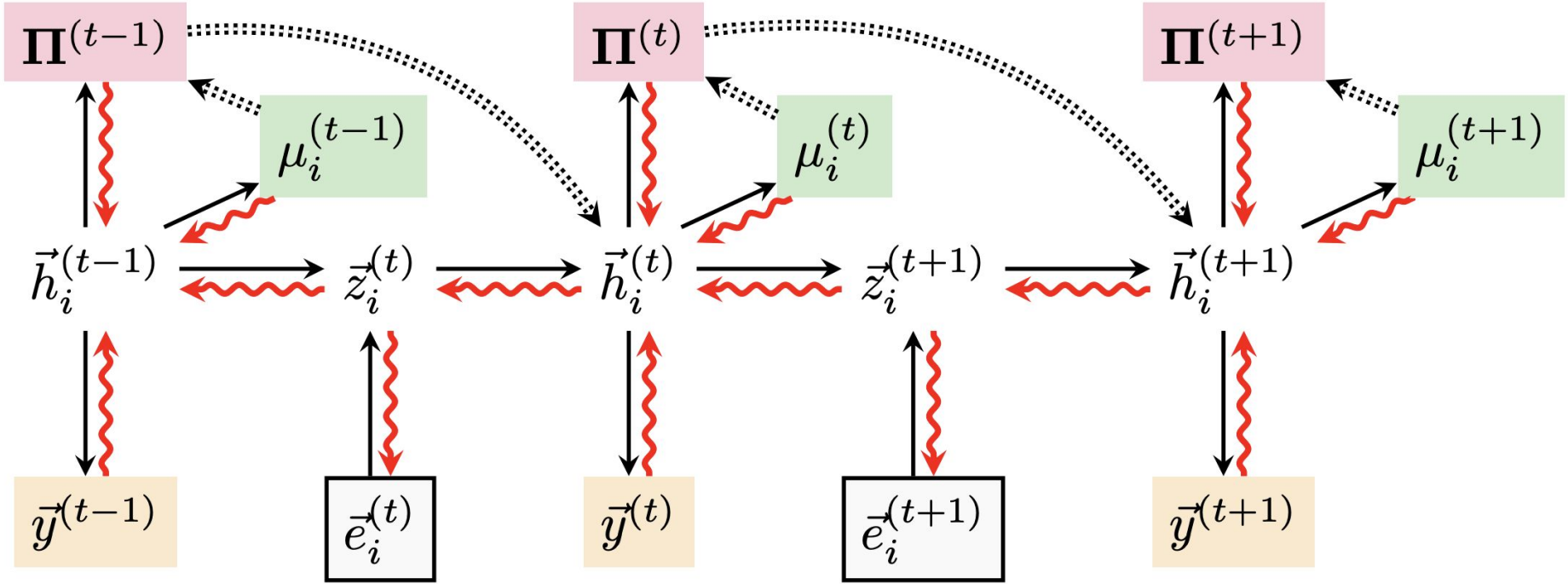
Pointer Graph Network (PGN)

- Further supervised to answer queries at every point in time.



(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

Overall PGN dataflow



(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

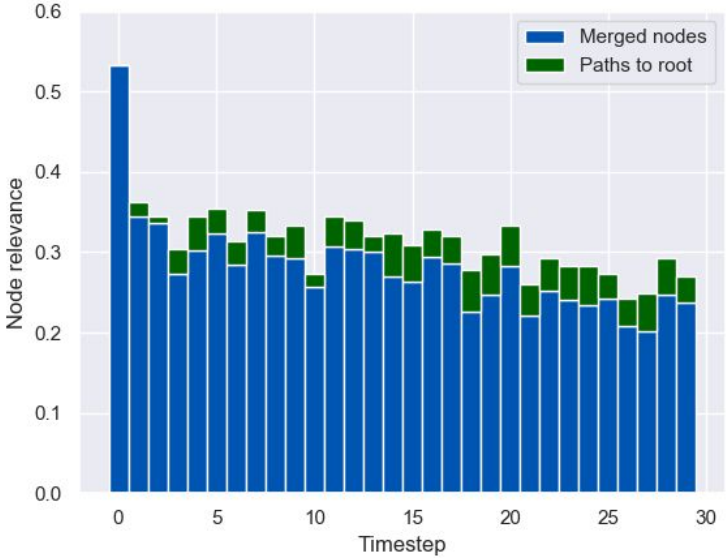
PGN Results

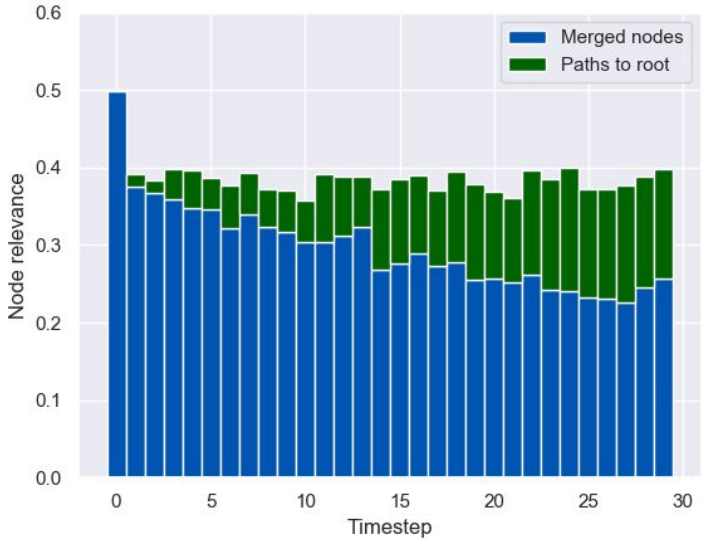
Table 1: F_1 scores on the dynamic graph connectivity tasks for all models considered, on five random seeds. All models are trained on $n = 20, \text{ops} = 30$ and tested on larger test sets.

Model	Disjoint-set union			Link/cut tree		
	$n = 20$ ops $= 30$	$n = 50$ ops $= 75$	$n = 100$ ops $= 150$	$n = 20$ ops $= 30$	$n = 50$ ops $= 75$	$n = 100$ ops $= 150$
GNN	$0.892 \pm .007$	$0.851 \pm .048$	$0.733 \pm .114$	$0.558 \pm .044$	$0.510 \pm .079$	$0.401 \pm .123$
Deep Sets	$0.870 \pm .029$	$0.720 \pm .132$	$0.547 \pm .217$	$0.515 \pm .080$	$0.488 \pm .074$	$0.441 \pm .068$
PGN-NM	$\mathbf{0.910} \pm .011$	$0.628 \pm .071$	$0.499 \pm .096$	$0.524 \pm .063$	$0.367 \pm .018$	$0.353 \pm .029$
PGN	$0.895 \pm .006$	$\mathbf{0.887} \pm .008$	$\mathbf{0.866} \pm .011$	$\mathbf{0.651} \pm .017$	$\mathbf{0.624} \pm .016$	$\mathbf{0.616} \pm .009$
PGN-Ptrs	$0.902 \pm .010$	$0.902 \pm .008$	$0.889 \pm .007$	$0.630 \pm .022$	$0.603 \pm .036$	$0.546 \pm .110$
Oracle-Ptrs	$0.944 \pm .006$	$0.964 \pm .007$	$0.968 \pm .013$	$0.776 \pm .011$	$0.744 \pm .026$	$0.636 \pm .065$

Which nodes *contribute* to the decision? ($|V|=20$)

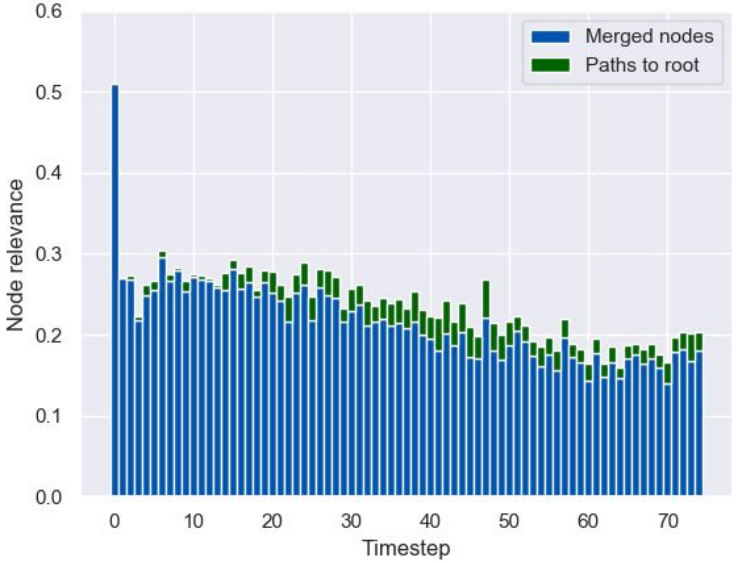
- Count how often is each node “used” in the final embedding that answer queries!
  - Check the two **merged nodes** and **nodes on path to root**
  - Comparing baseline GNN (**left**) and PGN (**right**) on **20-node test**

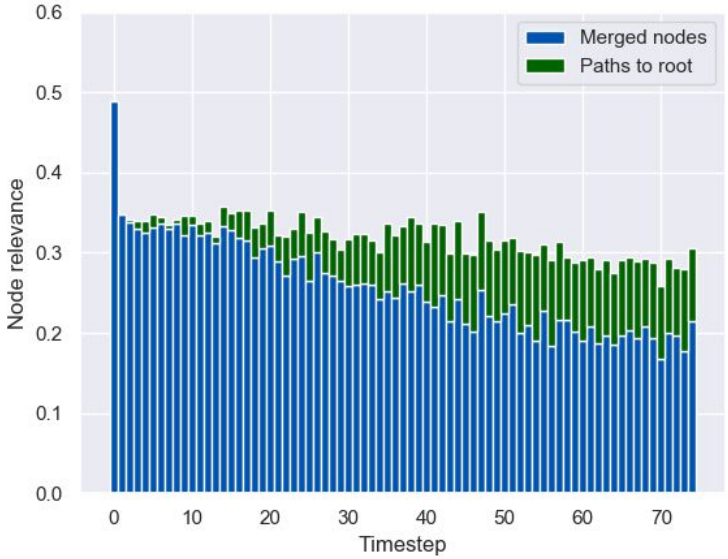




Which nodes *contribute* to the decision? ($|V|=50$)

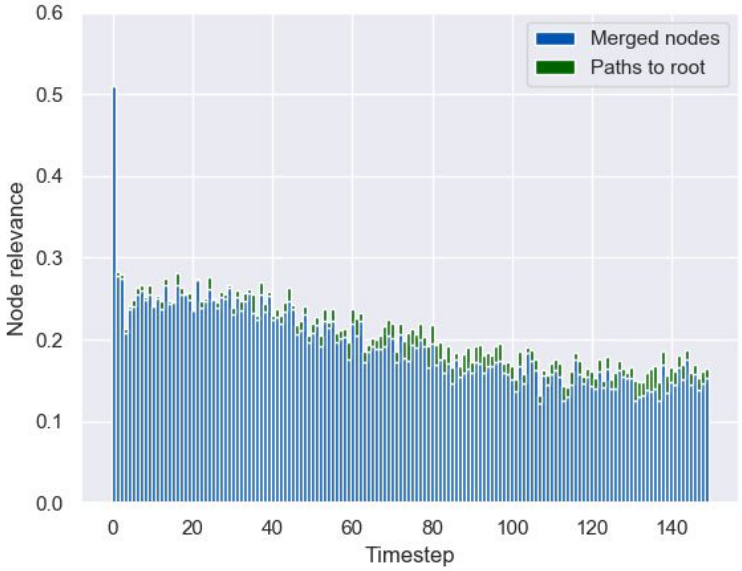
- Count how often is each node "used" in the final embedding that answer queries!
  - Check the two **merged nodes** and **nodes on path to root**
  - Comparing baseline GNN (**left**) and PGN (**right**) on **50-node test**

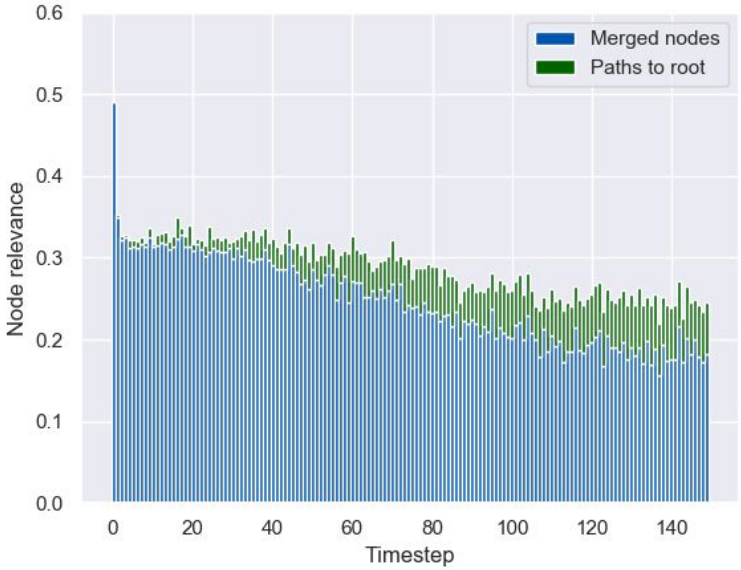




Which nodes *contribute* to the decision? ($|V|$=100)

- Count how often is each node “used” in the final embedding that answer queries!
  - Check the two **merged nodes** and **nodes on path to root**
  - Comparing baseline GNN (**left**) and PGN (**right**) on **100-node test**





(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

Pointer accuracies

Table 2: Pointer and mask accuracies of the PGN model w.r.t. ground-truth pointers.

Accuracy of	Disjoint-set union			Link/cut tree		
	$n = 20$ ops $= 30$	$n = 50$ ops $= 75$	$n = 100$ ops $= 150$	$n = 20$ ops $= 30$	$n = 50$ ops $= 75$	$n = 100$ ops $= 150$
Pointers (NM)	**80.3**±2.2%	32.9±2.7%	20.3±3.7%	**61.3**±5.1%	17.8±3.3%	8.4±2.1%
Pointers	76.9±3.3%	**64.7**±6.6%	**55.0**±4.8%	60.0±1.3%	**54.7**±1.9%	**53.2**±2.2%
Masks	95.0±0.9%	96.4±0.6%	97.3±0.4%	82.8±0.9%	86.8±1.1%	91.1±1.0%

- It appears that our learnt data structure substantially **deviates** from ground-truths!
- What did it learn to do?

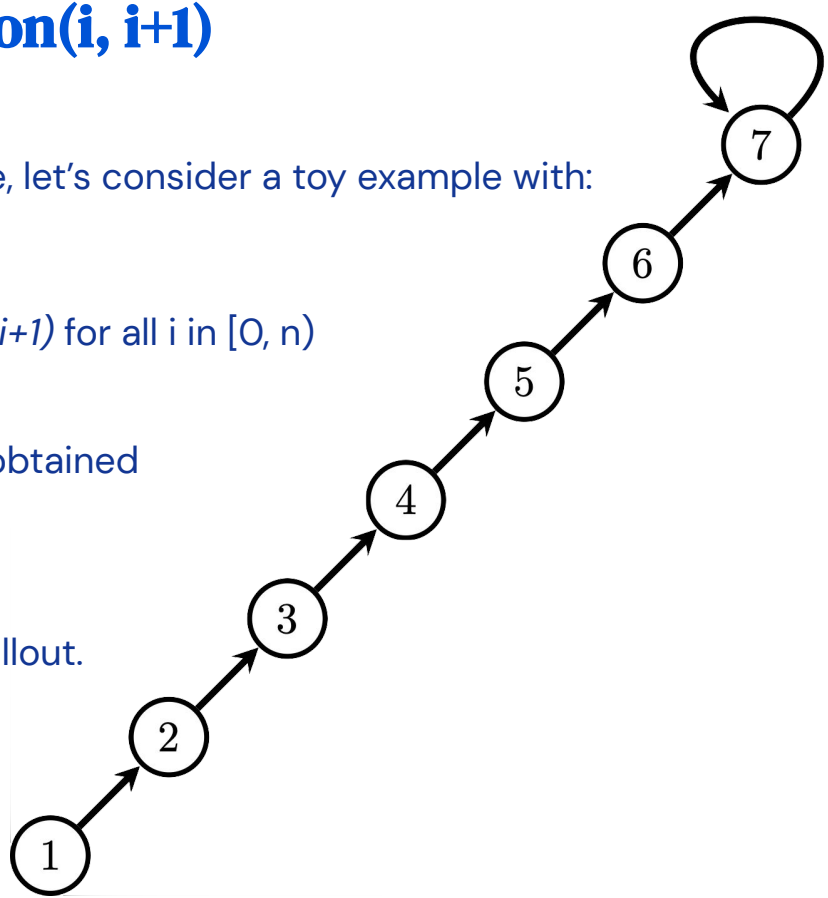
(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

Litmus test: repeated Union(i, i+1)

- To illustrate the pointer structure, let's consider a toy example with:
  - n = 7 nodes
  - Sorted ascending by rank
  - Repeatedly calling *Union(i, i+1)* for all i in [0, n)

- The ground-truth DSU pointers obtained form a "worst-case"* scenario:

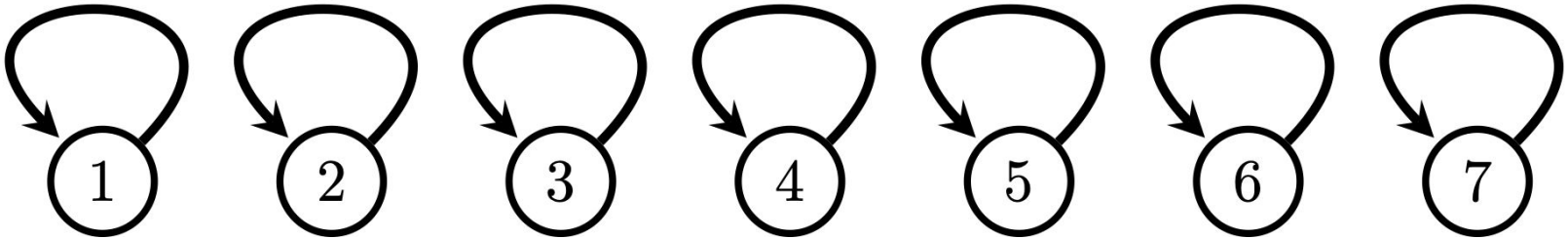
- We will perform a trained PGN rollout.



*not really damaging for DSU, but potentially troublesome for GNNs (**large diameter**).

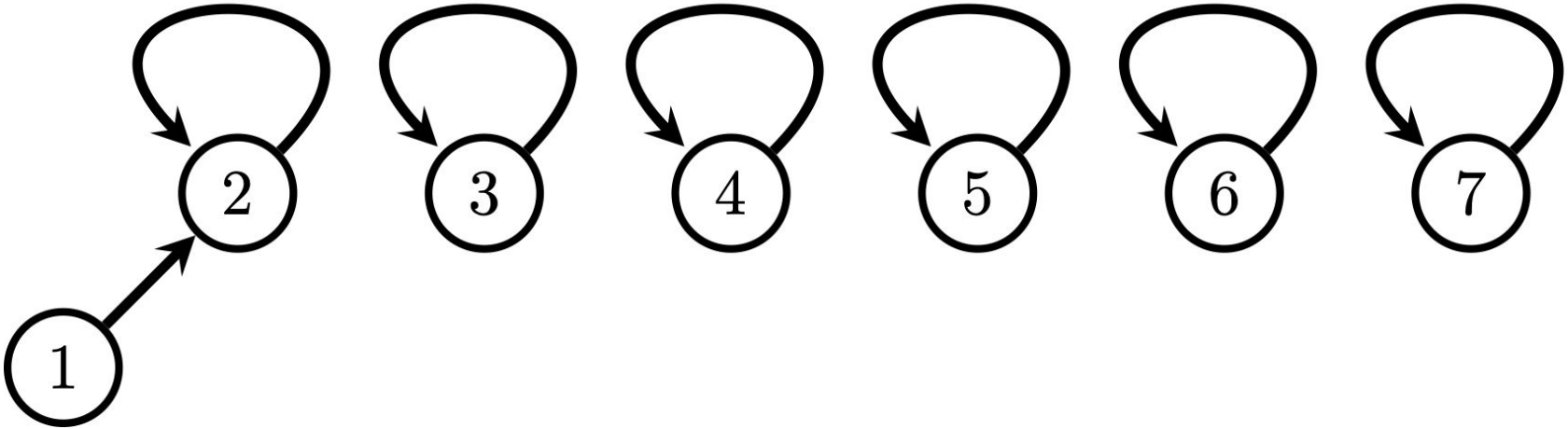
(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

PGN iterations on **Union(i, i+1)**: *initial state*



(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

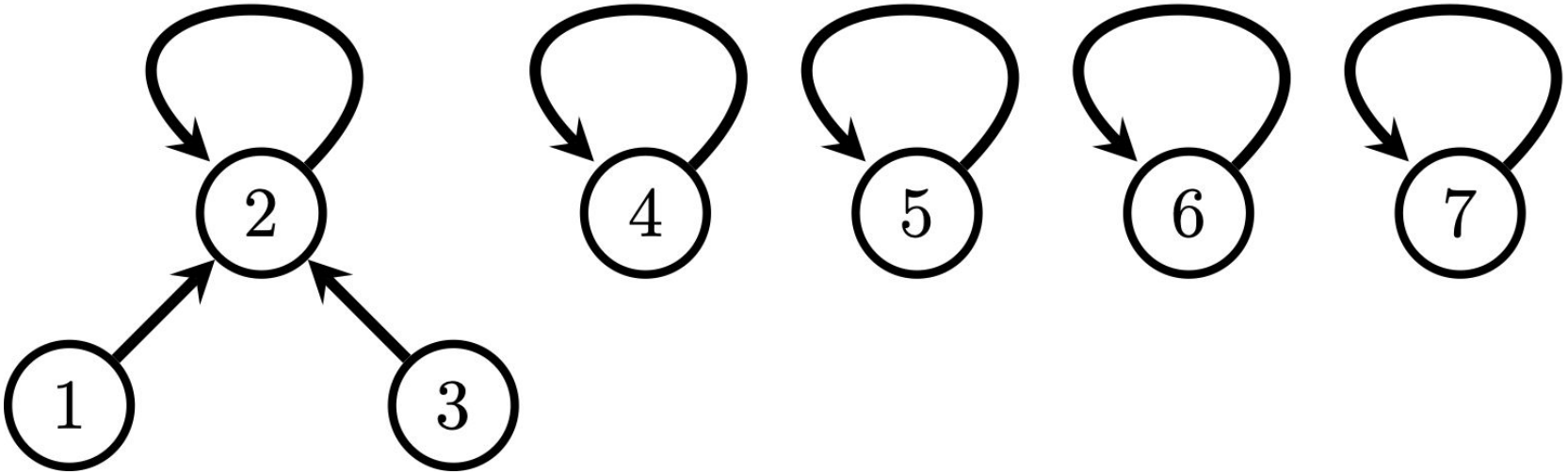
PGN iterations on **Union(i, i+1)**: *(1, 2)*



So far, so good....

(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

PGN iterations on **Union(i, i+1)**: *(2, 3)*

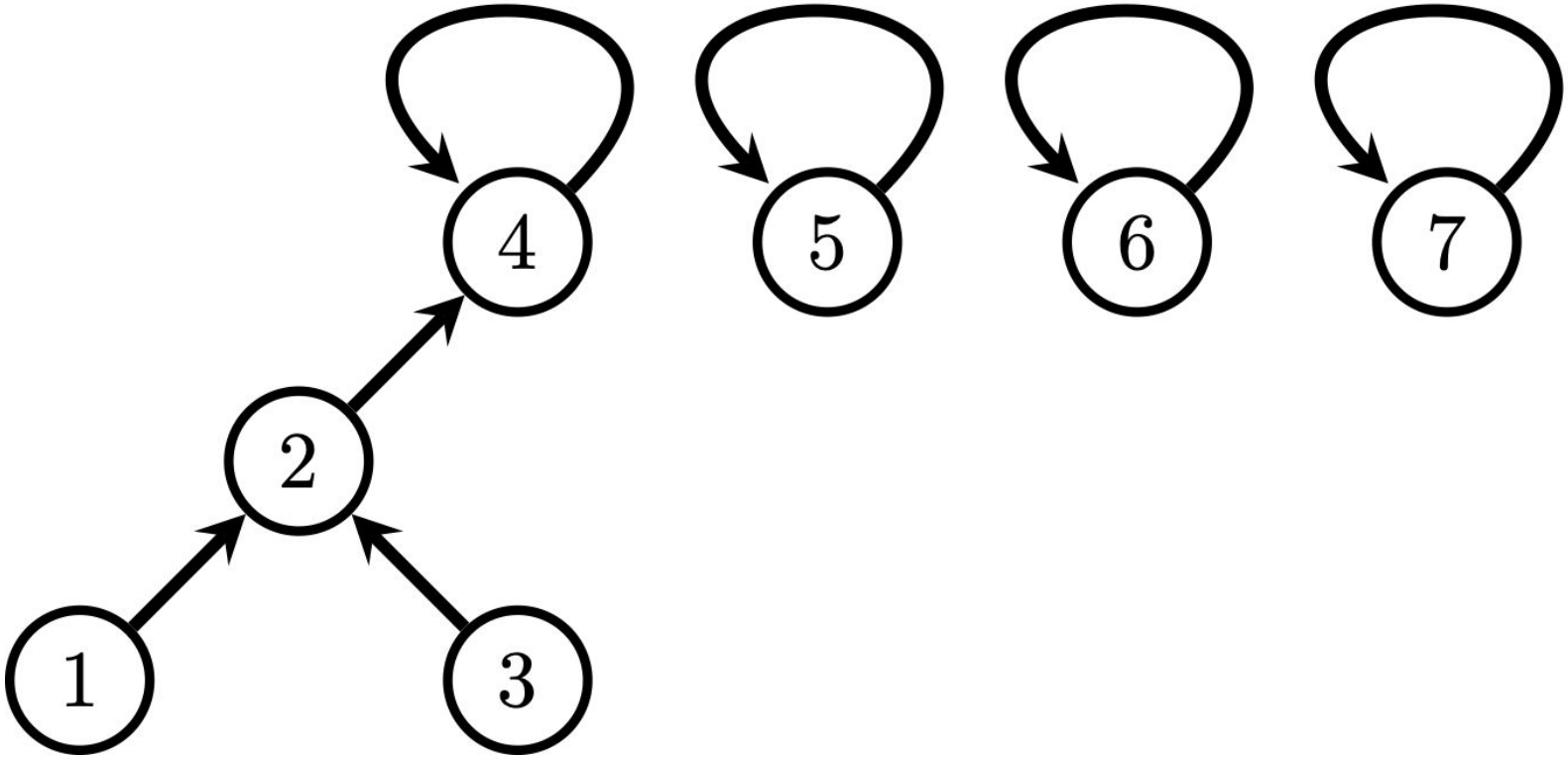


Differs from ground-truth already -- and **shallower**!

Continuing from here...

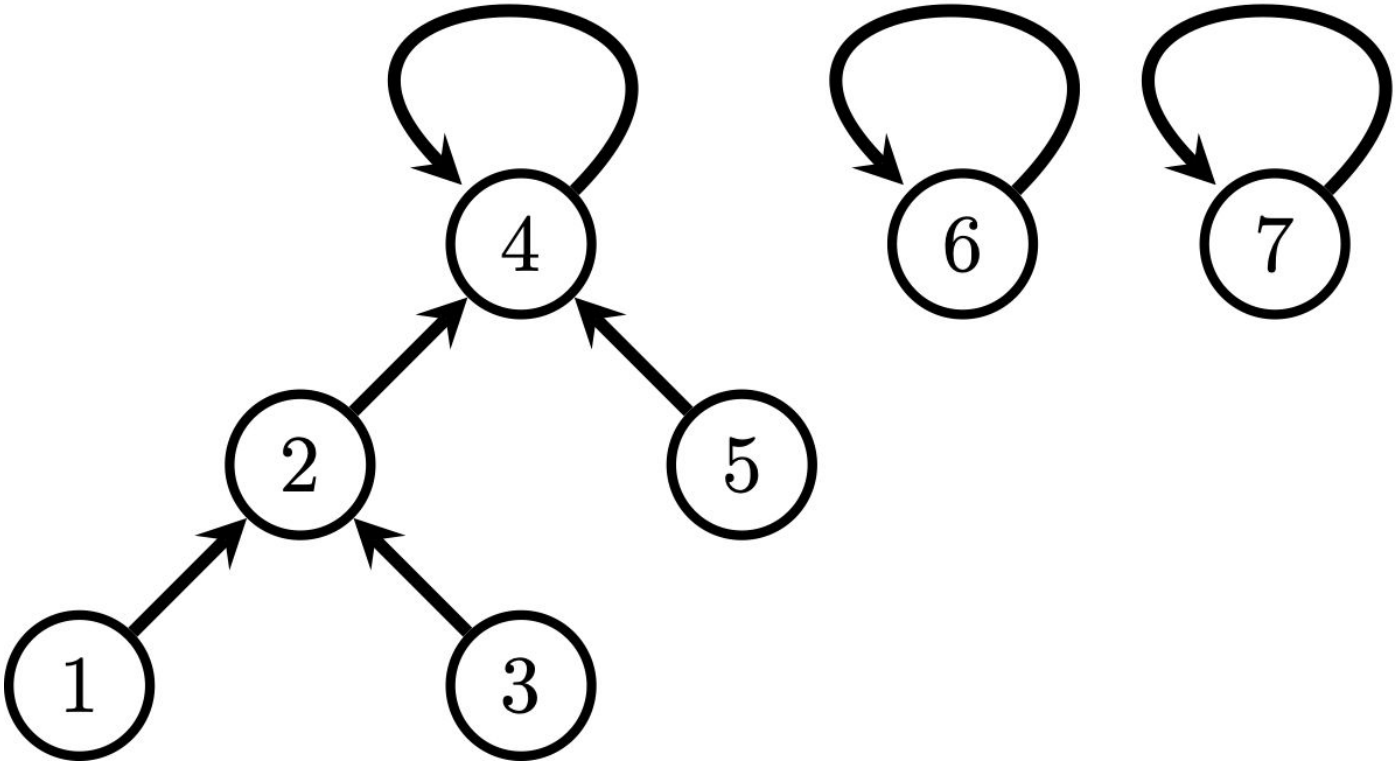
(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

PGN iterations on **Union(i, i+1)**: *(3, 4)*



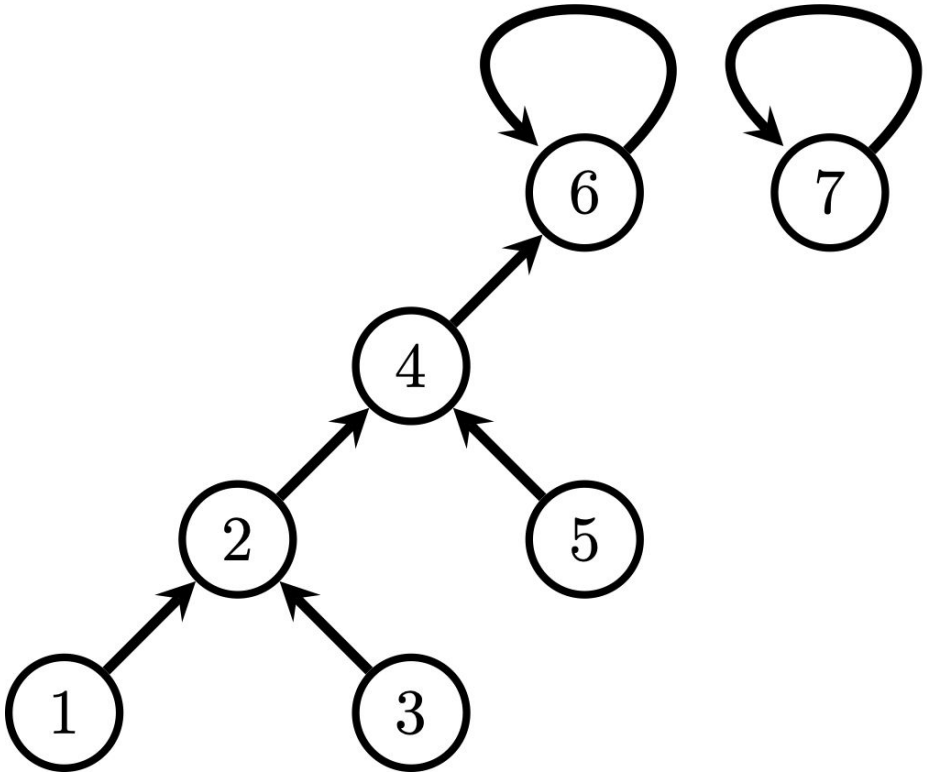
(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

PGN iterations on **Union(i, i+1)**: *(4, 5)*



(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. 2020)

PGN iterations on **Union(i, i+1)**: *(5, 6)*



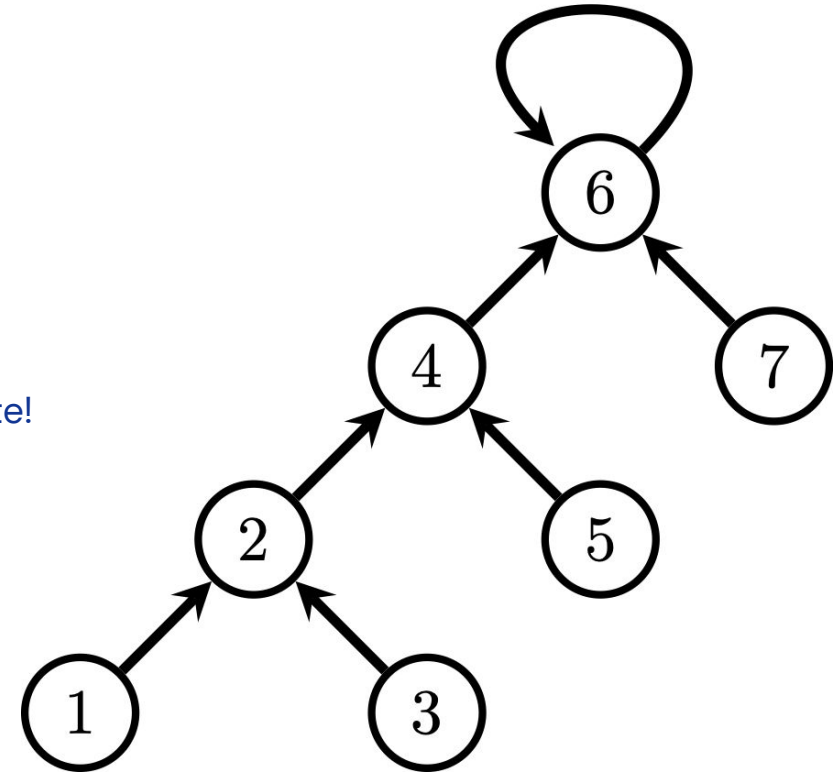
PGN iterations on **Union(i, i+1)**: *(6, 7)*

We recover a completely **valid** DSU tree...

...but one which cuts the diameter in **half**

⇒ more favourable for GNN!

Conditioned **entirely** through PGN's hidden state!



Summary

Let's think back to the **inductive biases** we've introduced, starting from a basic MLP:

- **DeepSets** ⇔ *object-level*;
- **GNNs** ⇔ *relational*;
- **Max aggregator** ⇔ *search-like*;
- **Step-wise imitation** ⇔ *algorithm-like*;
- **Sequential bias** ⇔ *one-object-at-a-time*;
- **Pointers** ⇔ *latent-graph-like;*
- **Masking** ⇔ *data-structure-like*.

For each bias, we had a clear motivation for why we introduced so, and an obvious means of doing either **theoretical** or **empirical** analysis.

None of the biases were too problem-specific.

In general, when solving a (reasoning) task, ask yourself:

- *What is the kind of reasoning procedure I'd like my neural network to perform?*
- *How to **constrain** the network to compute (intermediate) results in this manner?*

DeepMind

Thank you!

Questions?

petarv@google.com | https://petar-v.com

In collaboration with Charles Blundell, Raia Hadsell, Rex Ying, Matilde Padovano, Heiko Strathmann, Lars Buesing, Matt Overlan, Razvan Pascanu and Oriol Vinyals