DeepMind

Neural Algorithmic Reasoning

Petar Veličković

PHYS 7332 – Network Data Science 2
Northeastern University

2 April 2021

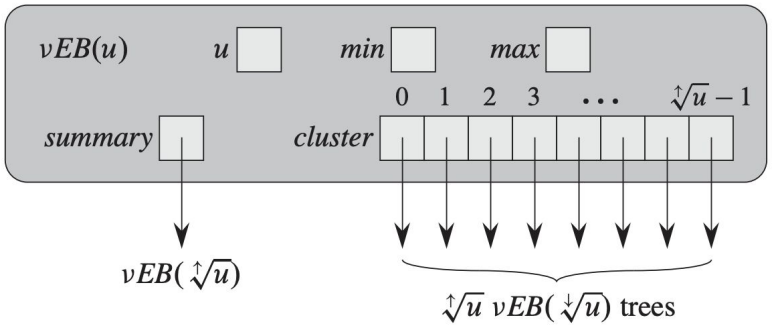
DeepMind

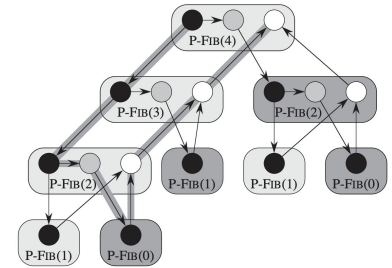
In this talk:
(Classical) **Algorithms**

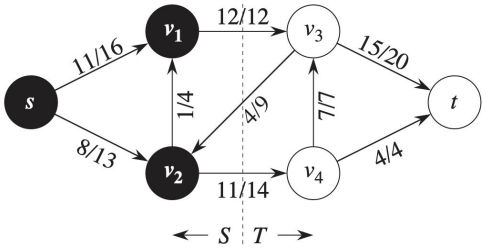
DeepMind

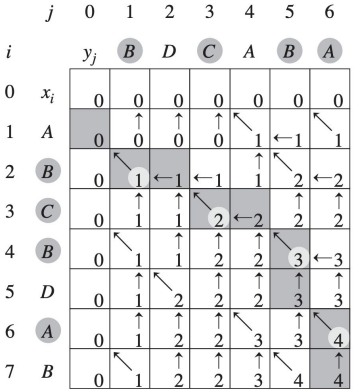
In this talk: (Classical) Algorithms

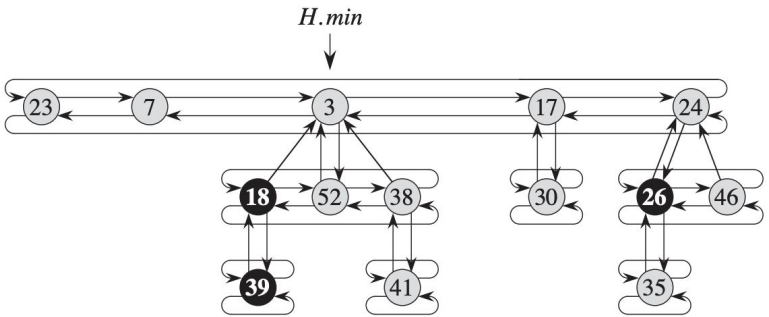
```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

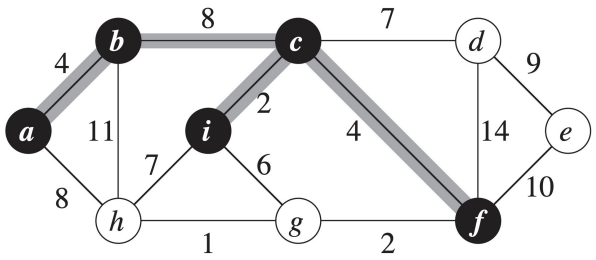


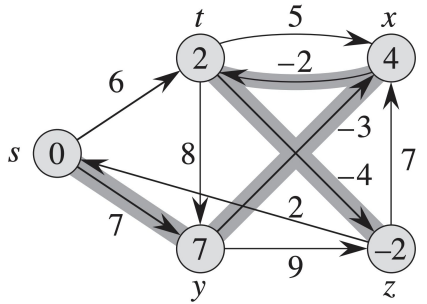




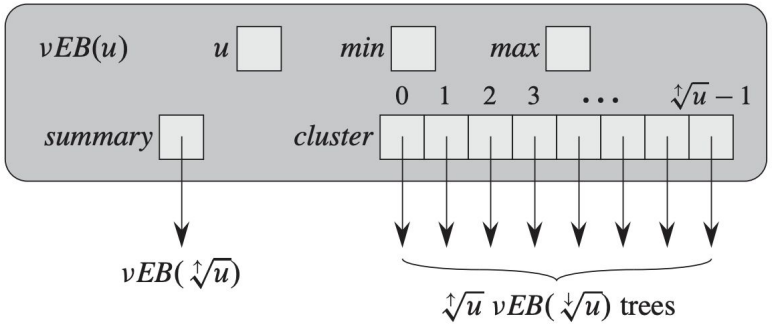




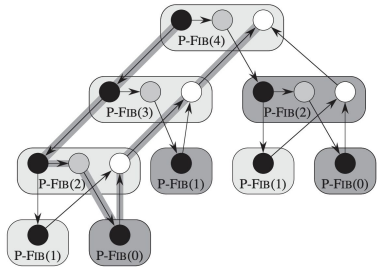




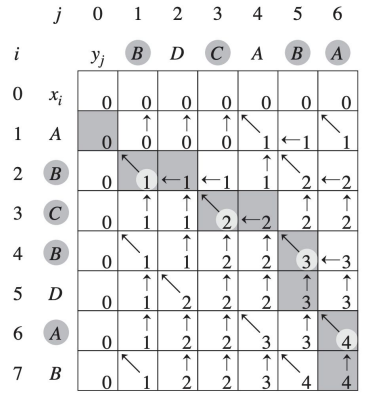
Algorithm figures: Cormen, Leiserson, Rivest and Stein. **Introduction to Algorithms.**

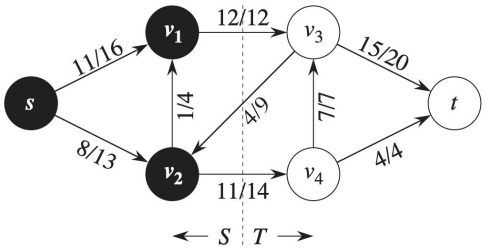


DeepMind

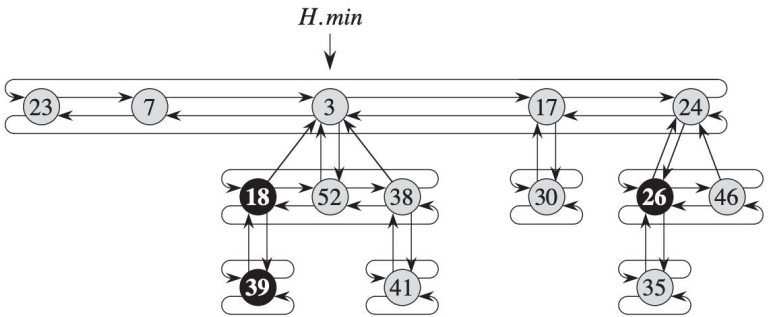


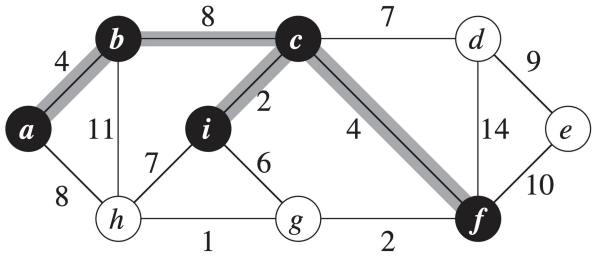

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

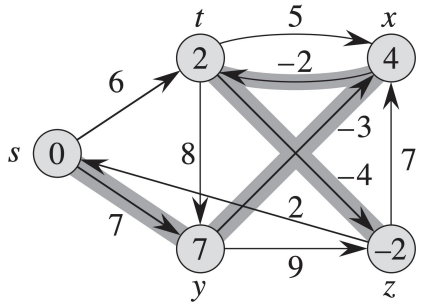




In this talk:
(Classical) **Algorithms**
(with a bit of neural spice)







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

Overview

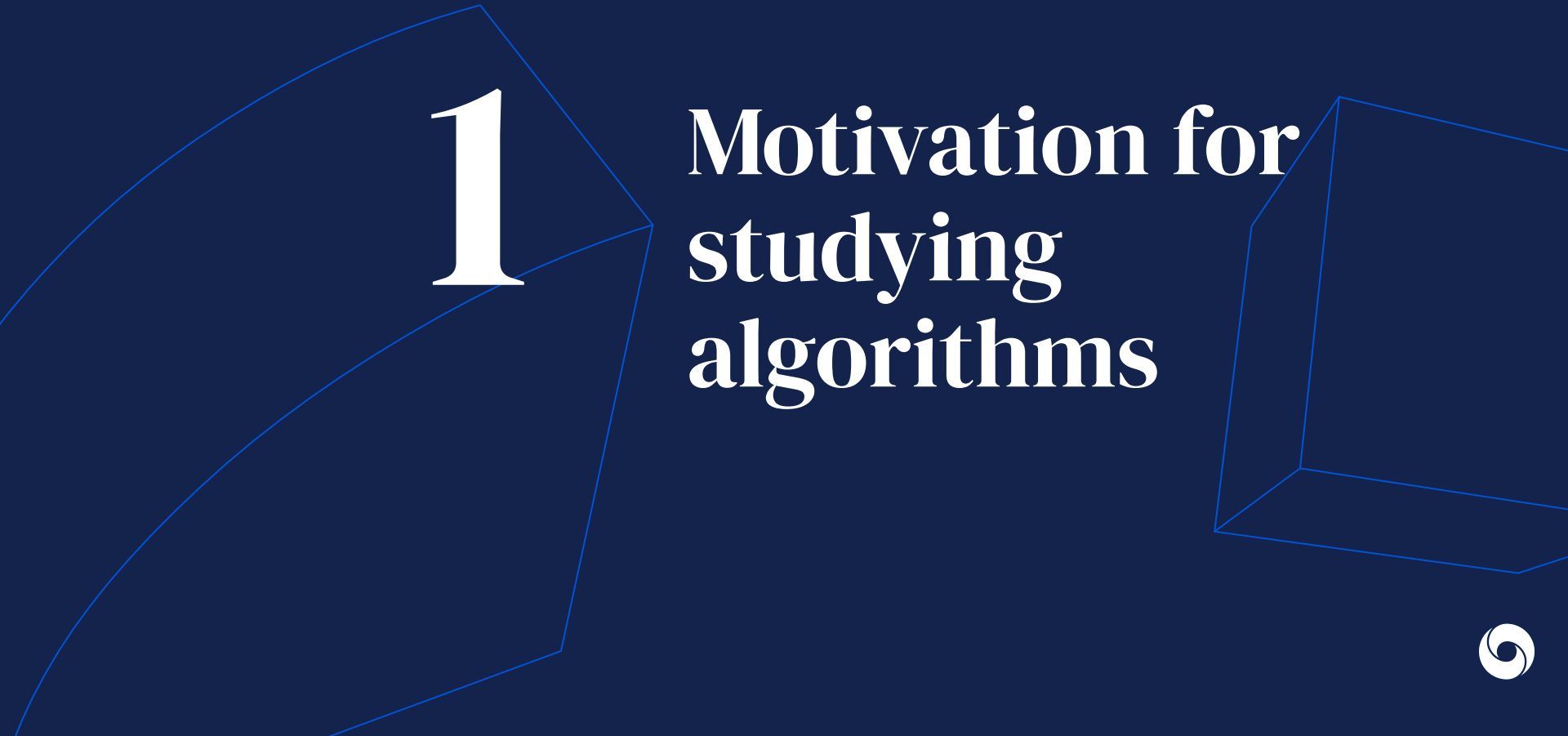
Our aim is to address **three** key questions: (roughly ~20min for each)

- Why should we, as deep learning practitioners, study **algorithms**?
  - Further, why might it be beneficial to make *'algorithm-inspired'* neural networks?
- How to **build** neural networks that behave algorithmically?
  - And why am I even telling you this in a *"Graph Machine Learning"* course?
- Do algorithmic neural networks actually **work** when deployed?
  - If so, how are they *actually* being used?

Hopefully, also some ideas on ***where*** *you might be able to* ***apply*** the ideas above :)

DeepMind

1 Motivation for studying algorithms



Why algorithms?

- Essential **“pure”** forms of combinatorial **reasoning**
  - ‘Timeless’ principles that will remain regardless of the model of computation
  - Completely decoupled from any form of **perception***

**though perception itself may also be expressed in the language of algorithms*

Why algorithms?

- Essential **“pure”** forms of combinatorial **reasoning**
  - ‘Timeless’ principles that will remain regardless of the model of computation
  - Completely decoupled from any form of **perception***

- **Favourable** properties
  - Trivial **strong** generalisation
  - **Compositionality** via *subroutines*
  - Provable **correctness** and **performance** guarantees
  - Interpretable **operations** / *pseudocode*

Why algorithms?

- Essential **“pure”** forms of combinatorial **reasoning**
  - ‘Timeless’ principles that will remain regardless of the model of computation
  - Completely decoupled from any form of **perception***

- **Favourable** properties
  - Trivial **strong** generalisation
  - **Compositionality** via *subroutines*
  - Provable **correctness** and **performance** guarantees
  - Interpretable **operations** / *pseudocode*

- Hits *close to home*
  - Algorithms and competitive programming are how I got into Computer Science

DeepMind

2 Maximum flow and the Ford-Fulkerson algorithm

Maximum flow problem

- **Flow network**: graph G = (V, E), augmented with a *capacity function*, *c*: V x V → $\mathbb{R}^+$
  - Capacity c_{uv} denotes how much **flow** is allowed on (*u, v*) edge

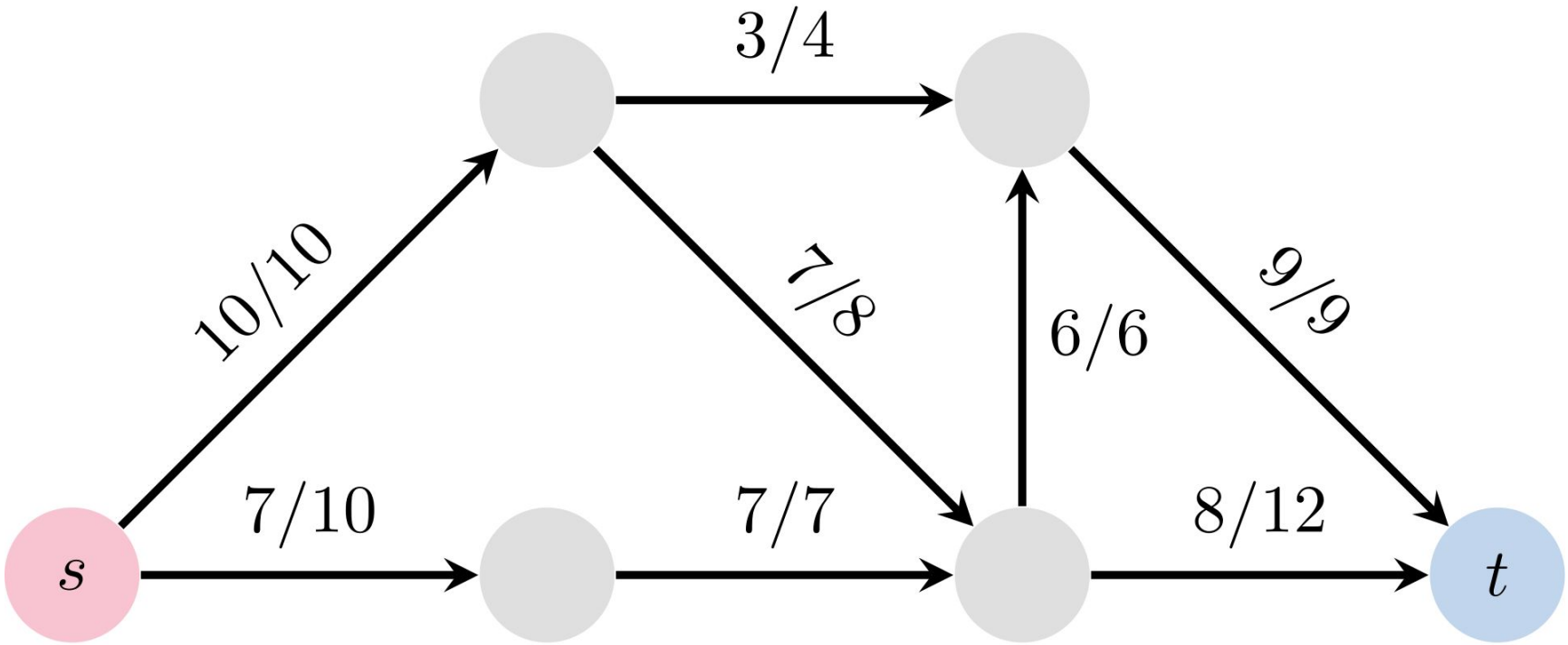
- Two special nodes: source, **s**, and sink, **t**
  - Source unleashes “infinite” capacity, sink receives “infinite” capacity

- A **flow** in G is any mapping *f*: V x V → $\mathbb{R}^+$, such that:

$$\begin{array}{lrcl} \forall u, v \in V & f_{u,v} & \leq & c_{u,v} \\ \forall u \in V \setminus \{s, t\} & \sum_{v \in V} f_{v,u} & = & \sum_{v \in V} f_{u,v} \end{array}$$

- The **value** of a flow is the total flow emanating from the source: $\sum_{v \in V} f_{s,v} - \sum_{v \in V} f_{v,s}$
  - We are interested in **maximising** it!

Max-flow example ($f = 17$)



Ford-Fulkerson's Algorithm

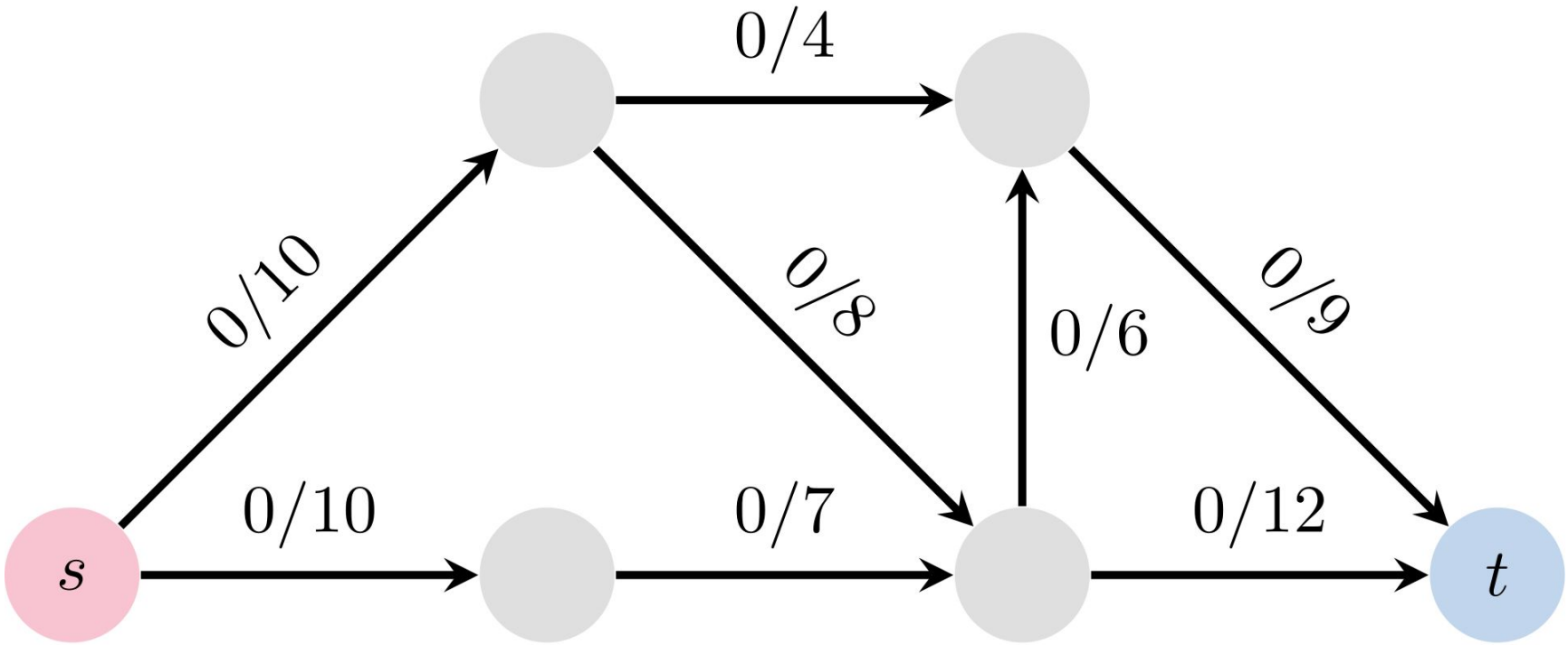
- Such a **rigorously** defined problem often admits remarkably **elegant** and **provably correct** algorithm blueprint!

```
FORD-FULKERSON-METHOD(G, s, t)
1  initialize flow f to 0
2  while there exists an augmenting path p in the residual network G_f
3      augment flow f along p
4  return f
```

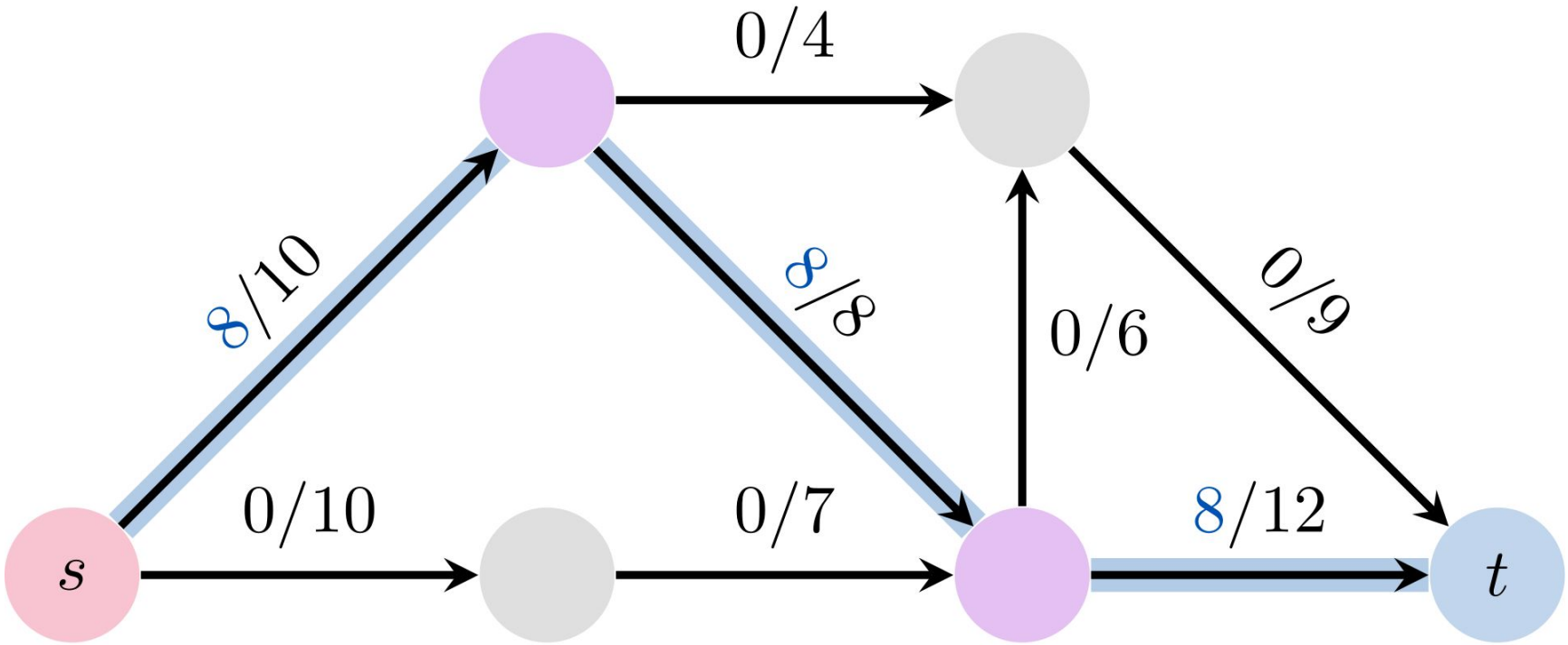
*representing the capacities that remain after applying f

- Many specific ways to find *p* yield different algorithms (e.g. Edmonds-Karp, Dinitz, etc...)
  - This can be proven to terminate with correct solution

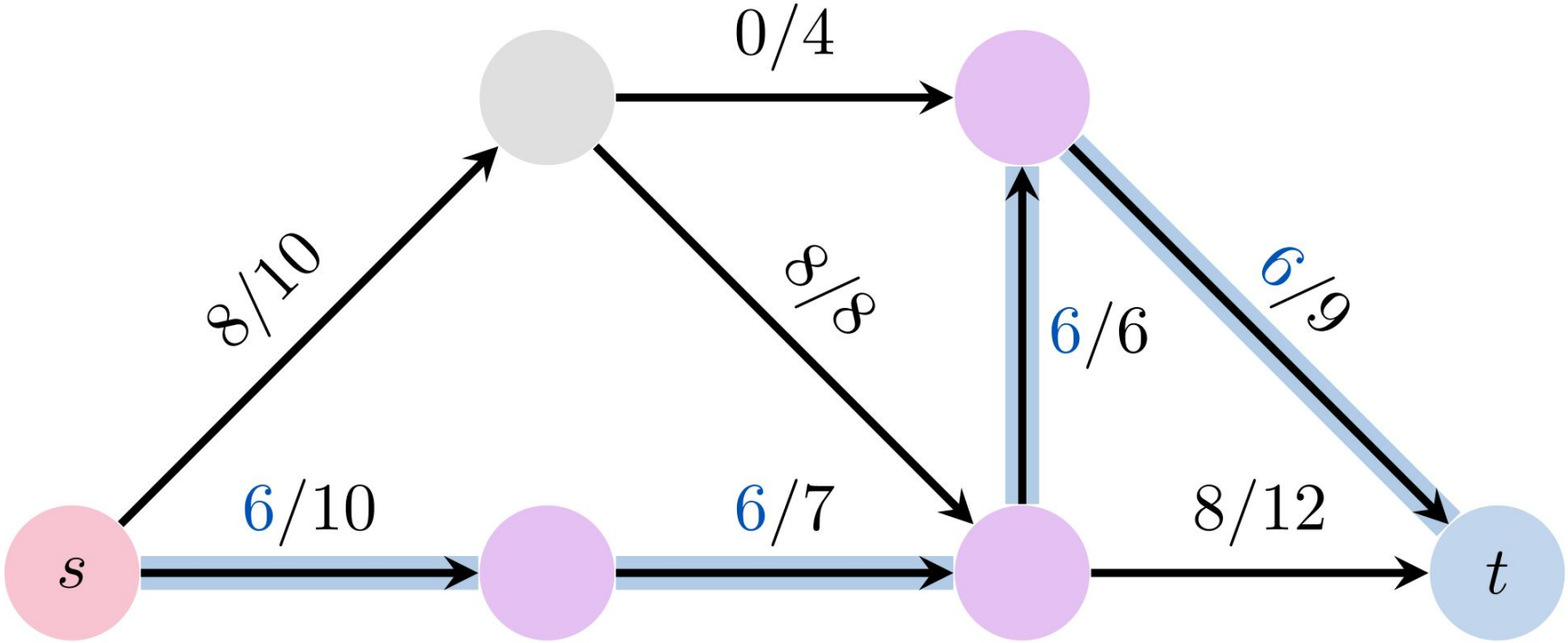
Ford-Fulkerson in action



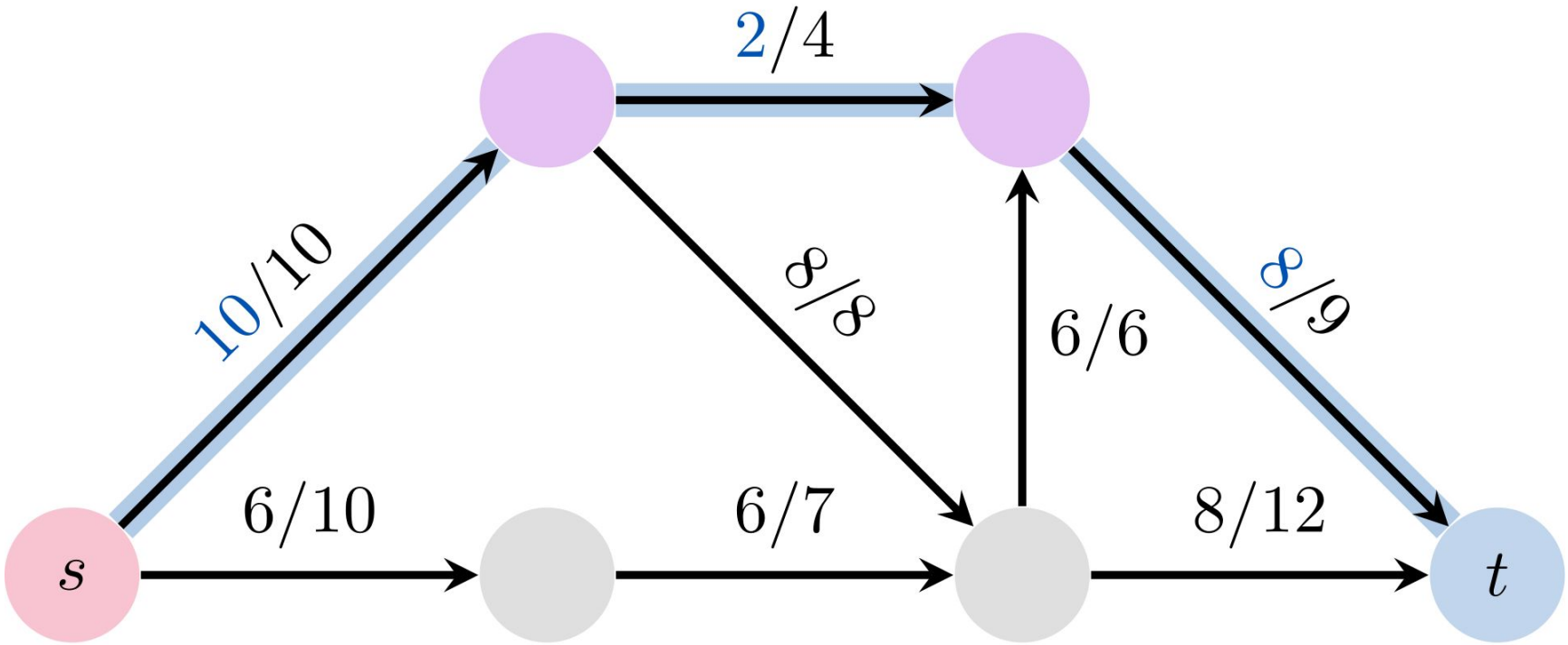
Ford-Fulkerson in action



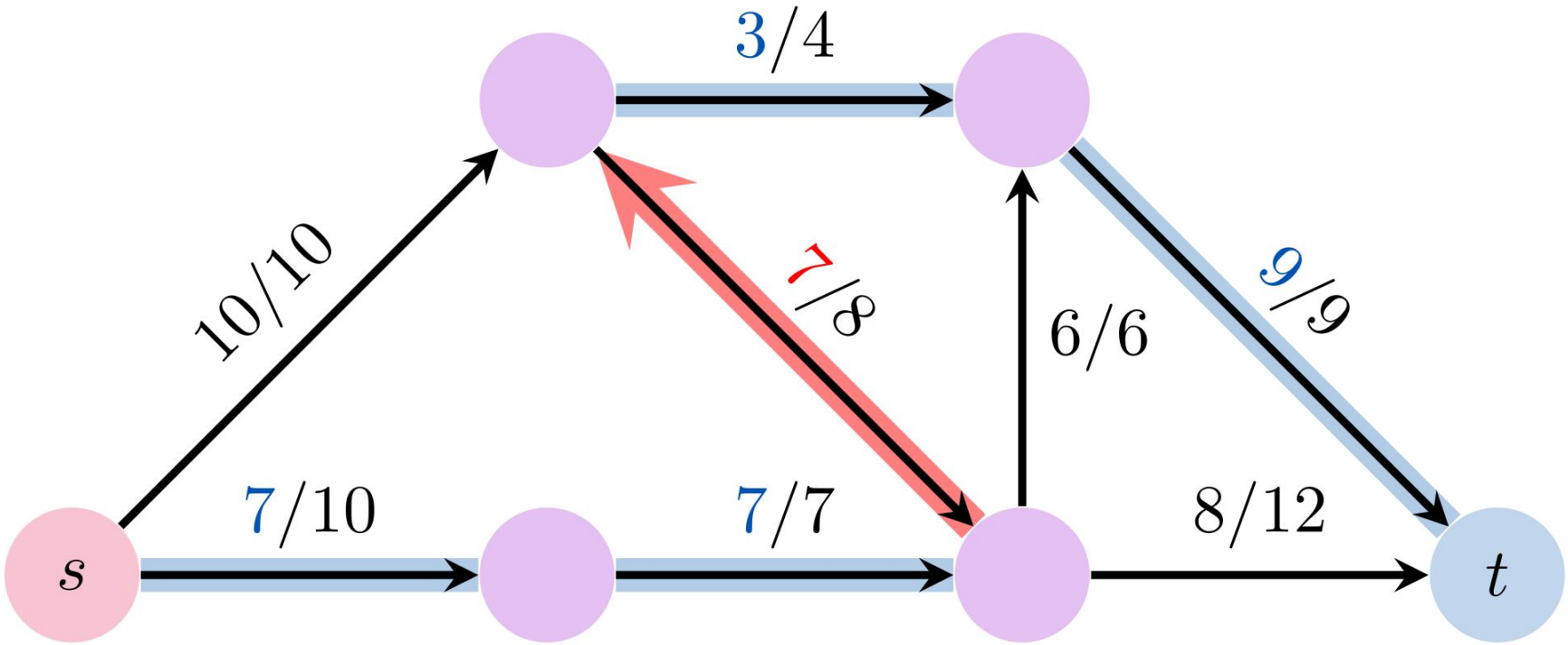
Ford-Fulkerson in action



Ford-Fulkerson in action

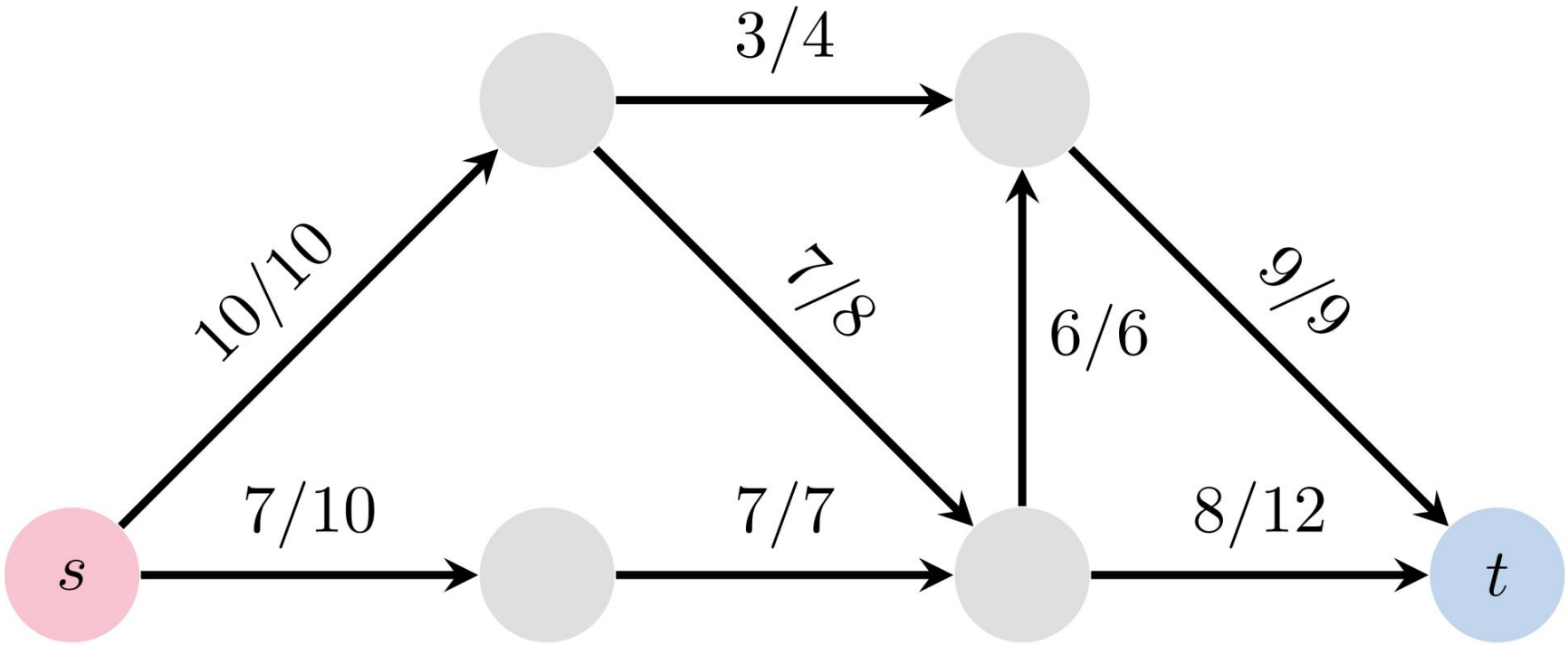


Ford-Fulkerson in action



(the flow may also be **returned**!)

Final solution!

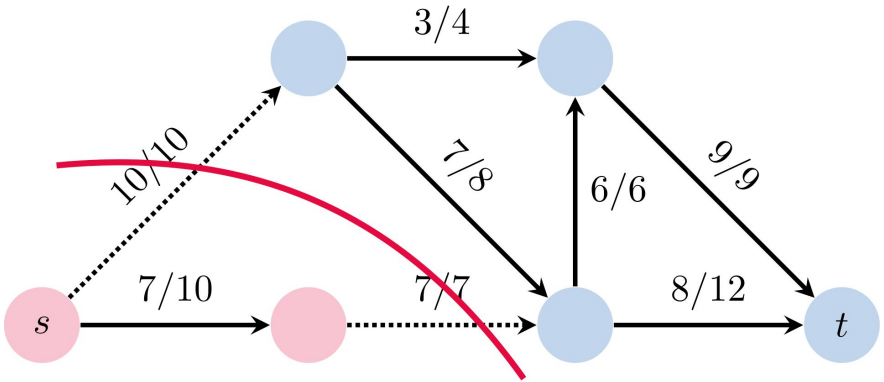


Max-flow Min-cut theorem

Observing data in this way, also yields easy observation of **connections**, hence **theorems**!

Theorem 26.6 (Max-flow min-cut theorem)
If f is a flow in a flow network $G = (V, E)$ with source s and sink t, then the following conditions are equivalent:

1. f is a maximum flow in G.
2. The residual network G_f contains no augmenting paths.
3. $|f| = c(S, T)$ for some cut (S, T) of G.



DeepMind

3

“Fundamentals of a method for evaluating rail net capacities”

(Harris & Ross, 1955)

The core problem

- Classical algorithms are designed with *abstraction* in mind, enforcing their inputs to conform to stringent **preconditions**.
  - Keeping the inputs constrained enables an uninterrupted focus on “reasoning”
  - Easily certify the resulting procedure’s correctness, i.e., stringent **postconditions**

- However, we must never forget **why** we design algorithms!

- Unfortunately, this is at **timeless odds** with the way they are designed
  - Let’s study an example from the 1950s.

Original interest in flows

SECRET

U. S. AIR FORCE

PROJECT RAND

RESEARCH MEMORANDUM

FUNDAMENTALS OF A METHOD FOR EVALUATING
RAIL NET CAPACITIES (U)

T. E. Harris
F. S. Ross

RM-1573

October 24, 1955 Copy No. 137

SUMMARY

Air power is an effective means of interdicting an enemy's rail system, and such usage is a logical and important mission for this Arm.

As in many military operations, however, the success of interdiction depends largely on how complete, accurate, and timely is the commander's information, particularly concerning the effect of his interdiction-program efforts on the enemy's capability to move men and supplies. This information should be available at the time the results are being achieved.

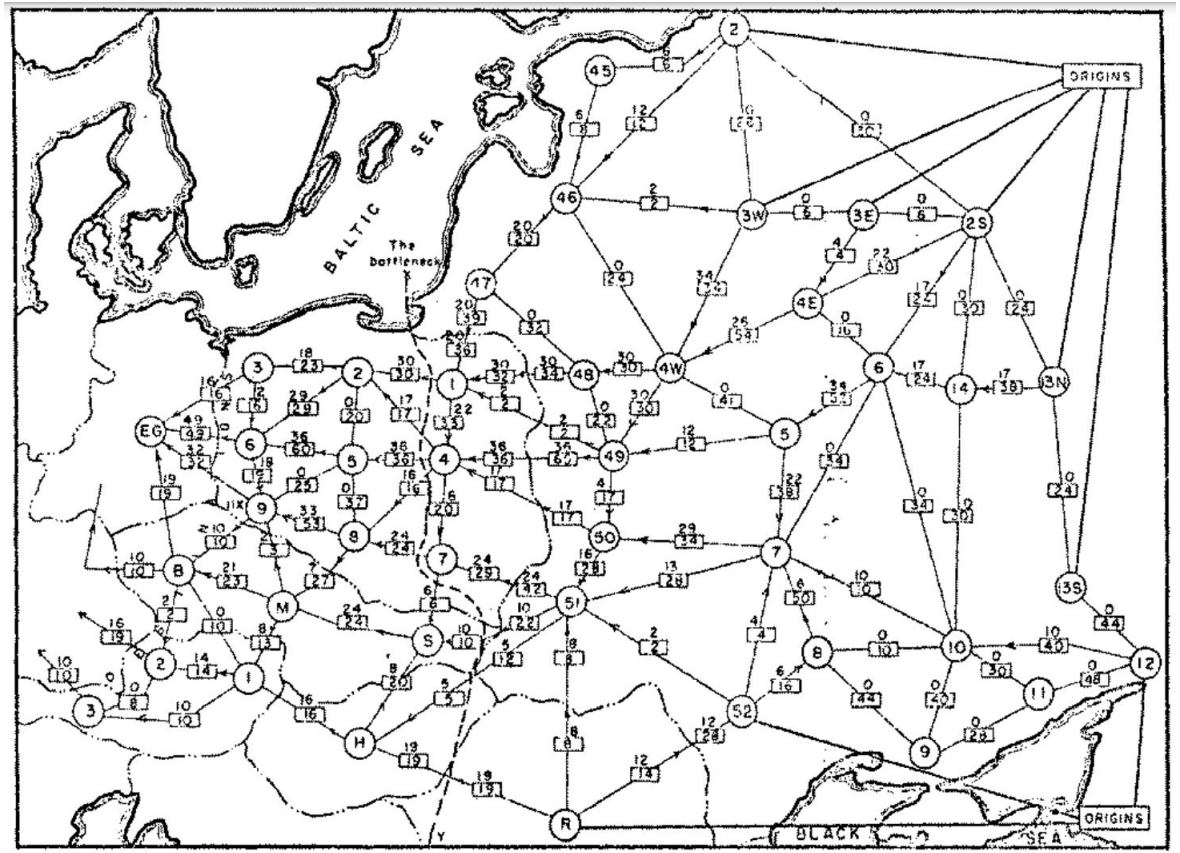
https://apps.dtic.mil/dtic/tr/fulltext/u2/093458.pdf

The Warsaw Pact railway network

Find *"the bottleneck"*, i.e. the **minimum cut**.

As we saw, this is directly related to computing the **maximum flow**.

(this was *intuitively* assumed by Harris & Ross as well)



The core problem, as seen in 1955

II. THE ESTIMATING OF RAILWAY CAPACITIES

The evaluation of both railway system and individual track capacities is, to a considerable extent, an art. The authors know of no tested mathematical model or formula that includes all of the variations and imponderables that must be weighed.* Even when the individual has been closely associated with the particular territory he is evaluating, the final answer, however accurate, is largely one of judgment and experience.

An important issue for the community

- The “core problem” plagues applications of classical combinatorial algorithms to this day!
- Satisfying their preconditions necessitates converting inputs into an **abstractified** form
- If done manually, this often implies *drastic* **information loss**
  - Combinatorial problem no longer accurately portrays the dynamics of the real world.
  - Algorithm will give a **perfect** solution, but in a ***useless*** environment
- The data we need to apply the algorithm may be only **partially** observable
  - This can often render the algorithm completely inapplicable.
- An issue of high interest for *both* combinatorial and operations research communities.

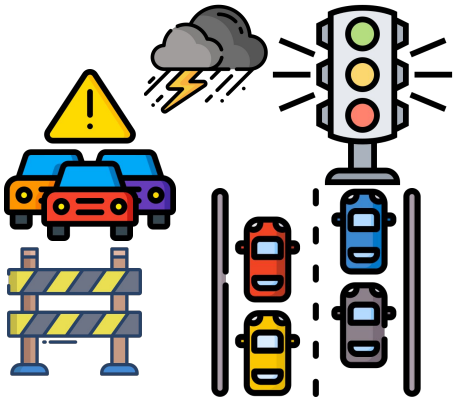
DeepMind

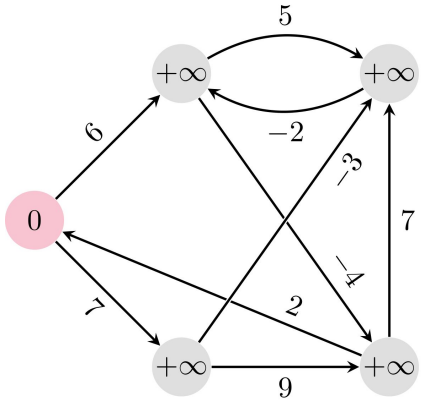
4

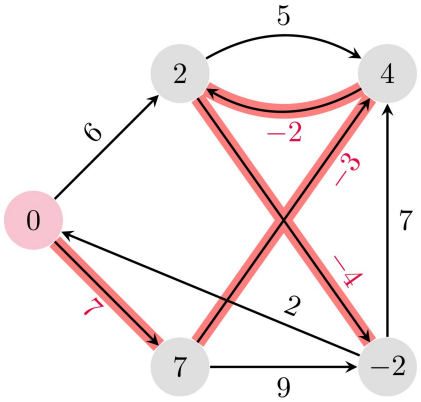
Towards a *neurally* spiced solution

Abstractifying the core problem

- Assume we have *real-world* inputs, but our algorithm only admits *abstract* inputs
  - For now, we assumed **manually** converting from one input to another

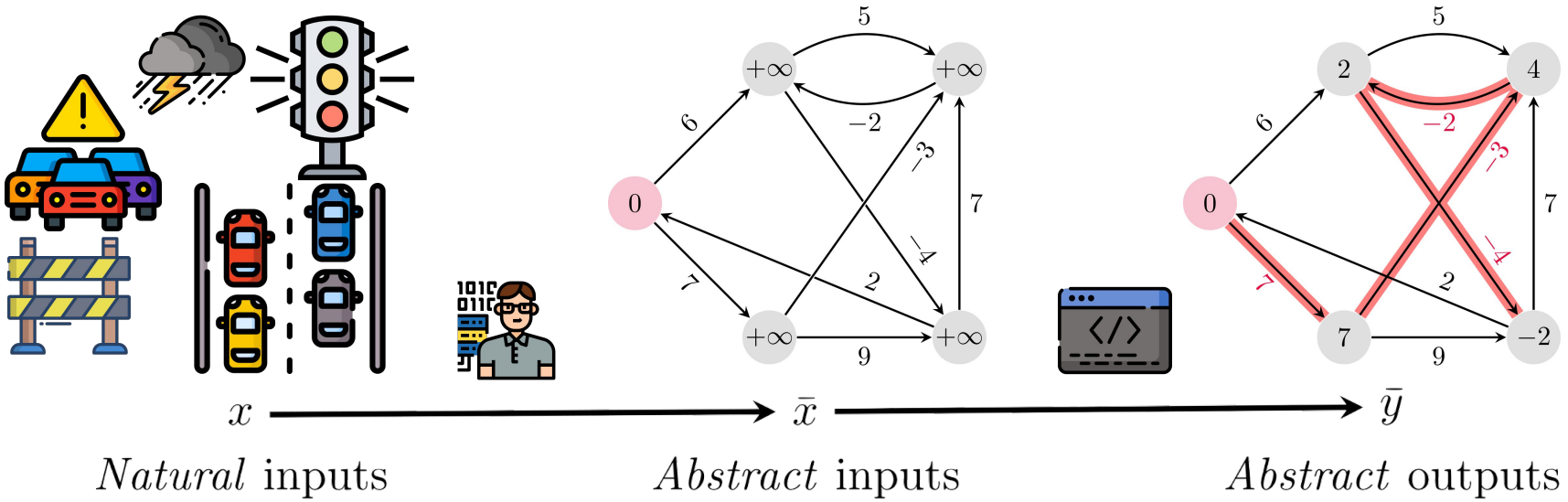






Abstractifying the core problem

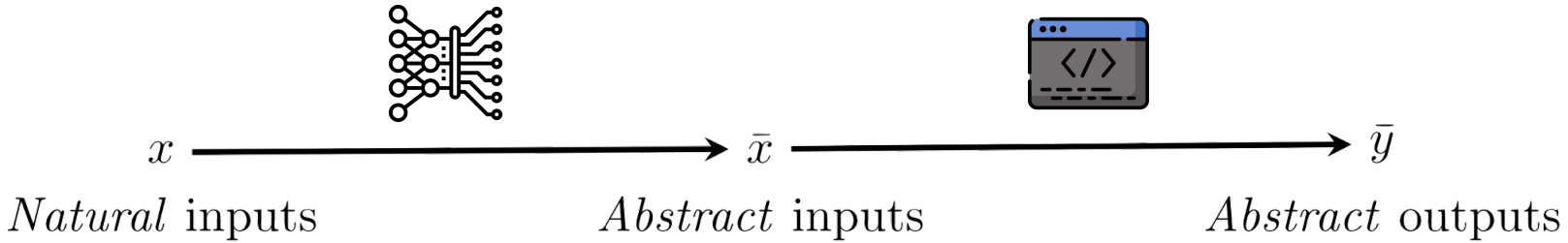
- Assume we have *real-world* inputs, but our algorithm only admits *abstract* inputs
  - For now, we assumed **manually** converting from one input to another



- Whenever we have **manual** feature engineering of **raw** data, **neural nets** are attractive!

Attacking the core problem

- First point of attack: *"good old deep learning"*
  - Replace human feature extractor with **neural network**
  - Still apply the same combinatorial algorithm



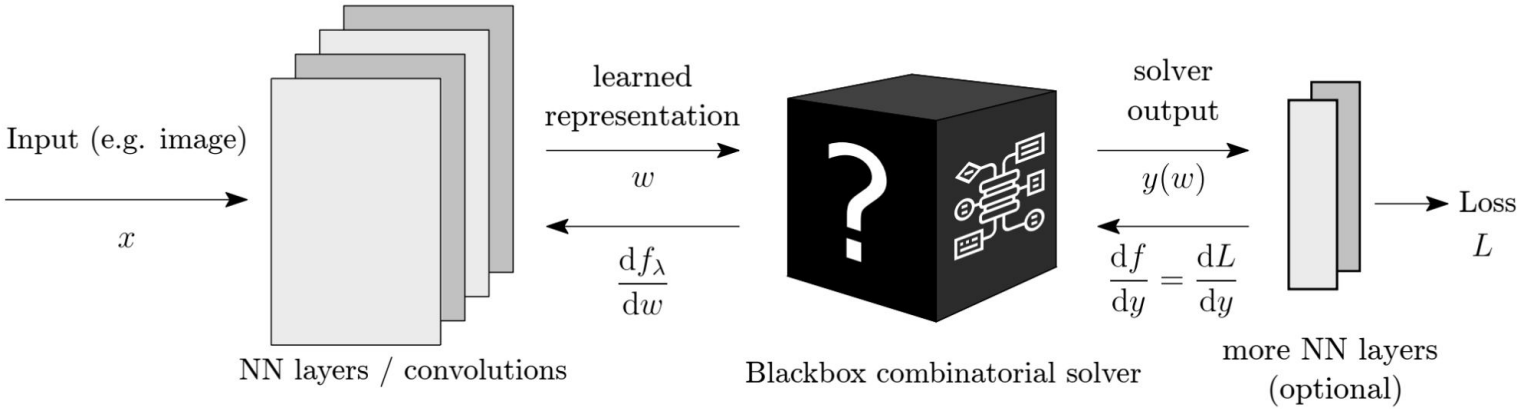
- **First issue**: algorithms typically perform **discrete optimisation**
  - This does not play nicely with gradient-based optimisation that neural nets require.

Backpropagating through classical algorithms

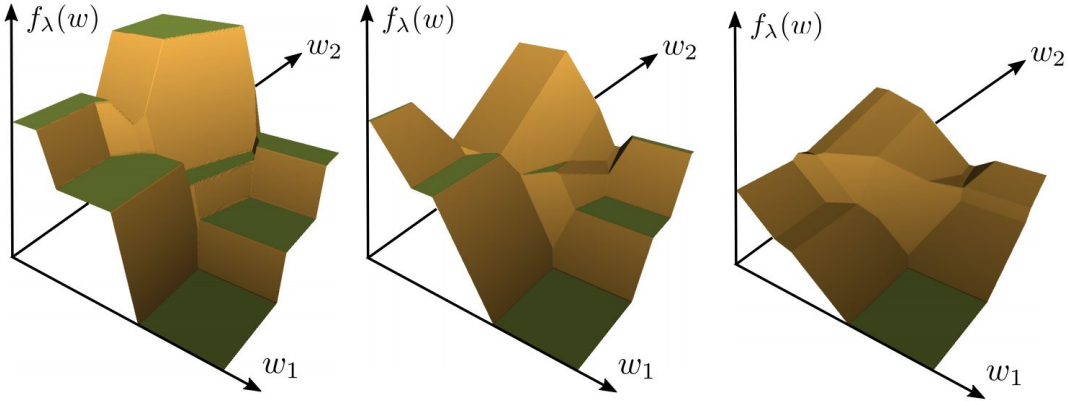
Vlastelica *et al.* (ICLR'20) provide a great approach for differentiating CO solver outputs

Differentiation of Blackbox Combinatorial Solvers

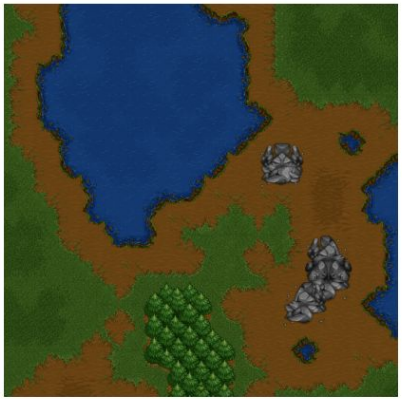
Marin Vlastelica[1*], Anselm Paulus[1*], Vít Musil[2], Georg Martius[1], Michal Rolínek[1]



Black-box backprop



Input



$\rightarrow$

Label

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

$k \times k$ indicator matrix of shortest path

Algorithmic ***bottleneck***

- Second (more fundamental) issue: ***data efficiency***
  - Real-world data is often incredibly *rich*
  - We still have to compress it down to **scalar values**

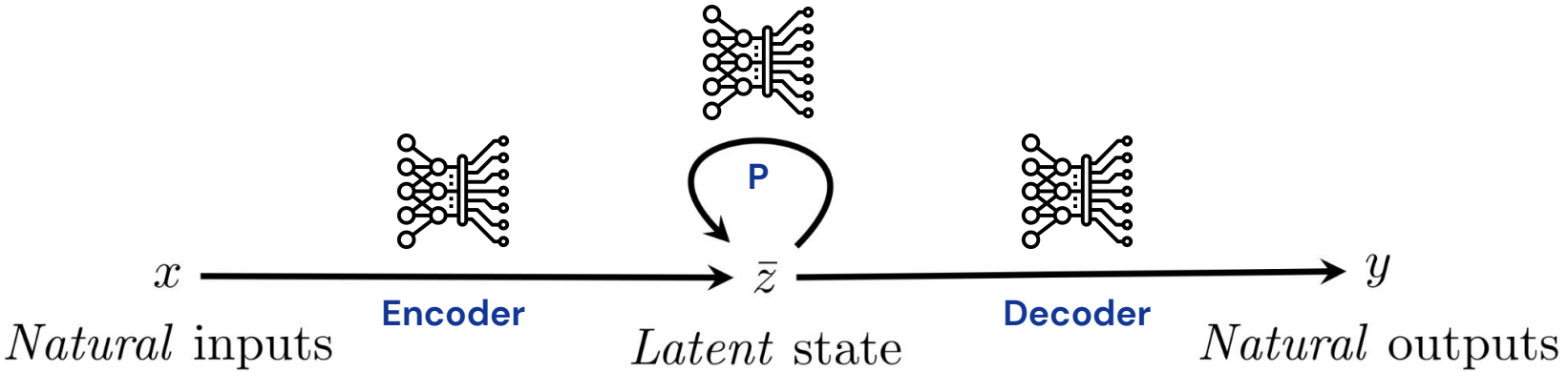
- The algorithmic solver:
  - **Commits** to using this scalar
  - Assumes it is **perfect**!

- If there are insufficient training data to properly estimate the scalars, we hit same issues!
  - Algorithm will give a **perfect** solution, but in a ***suboptimal*** environment

Breaking the bottleneck

- Neural networks derive great flexibility from their **latent** representations
  - They are inherently *high-dimensional*
  - If any component is poorly predicted, others can step in and compensate!

- To *break the bottleneck*, we replace the algorithm with a **neural network**!

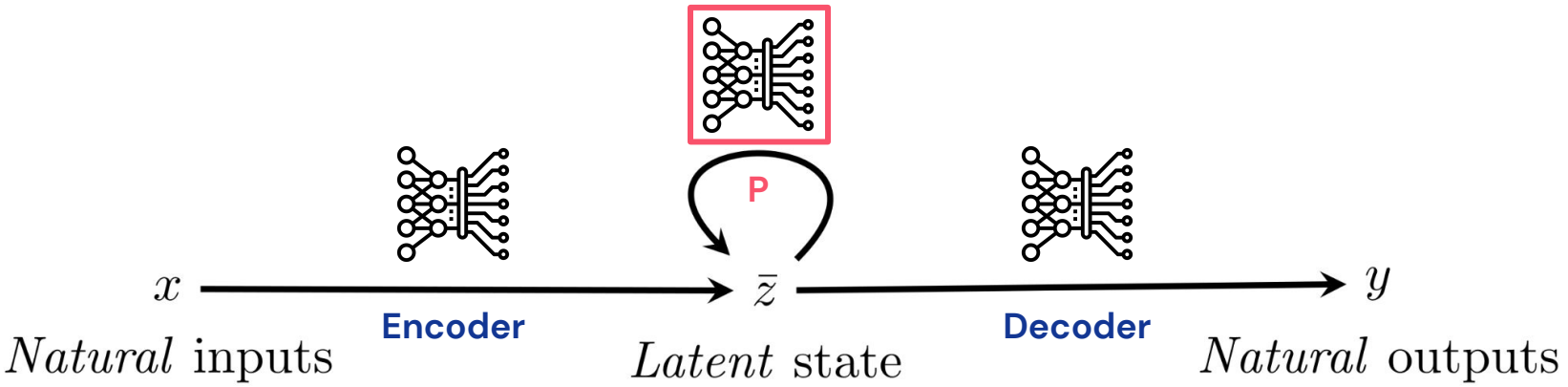


(The setting naturally aligns with *encode-**process**-decode* (Hamrick *et al.*, CSS'18))

Properties of this construction

- Assuming our **latent**-state NN *aligns* with the steps of an algorithm, we now have:
  - An **end-to-end** neural pipeline which is fully differentiable
  - No scalar-based bottlenecks, hence higher data efficiency.

- How do we obtain **latent-state neural networks** that **align** with algorithms?



DeepMind

(Have we answered Question 1?)

- Why should we, as deep learning practitioners, study **algorithms**?
  - Further, why might it be beneficial to make *'algorithm-inspired'* neural networks?

DeepMind

5 Algorithmic reasoning

Algorithmic reasoning

- The desiderata for our processor network **P** are slightly different than usual:
  - They are required to imitate the steps of the algorithm *faithfully*
  - This means they must **extrapolate**!
  - (*Related*: *how to best decide the **weights** of **P** to **robustly** match the algorithm?*)

- Neural networks typically **struggle** in the extrapolation regime!

- ***Algorithmic reasoning*** is an emerging area that seeks to ameliorate this issue
  - Primarily through theoretical and empirical *prescriptions*
  - These guide the neural *architectures*, *inductive biases* and *featurisations* that are useful for extrapolating combinatorially

- This is a **very** active research area, with many key papers published only last year!
  - We will navigate it by an increasingly complex sequence of *toy algorithmic problems*

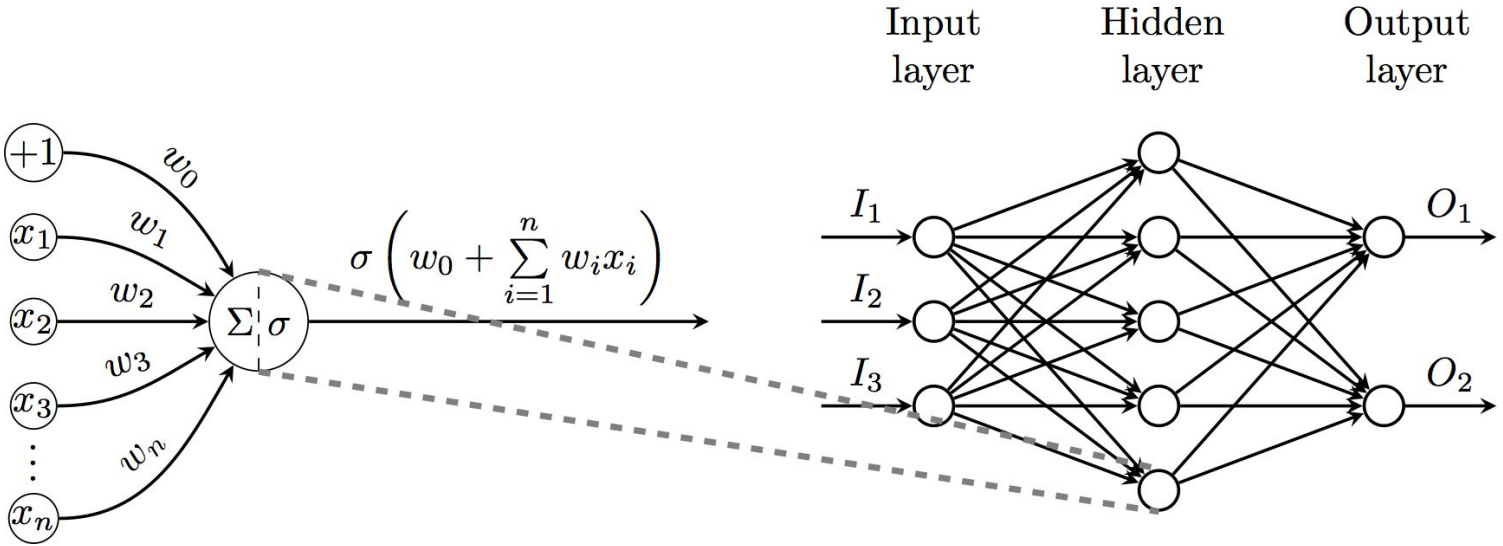
Starting simple

- **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?*)

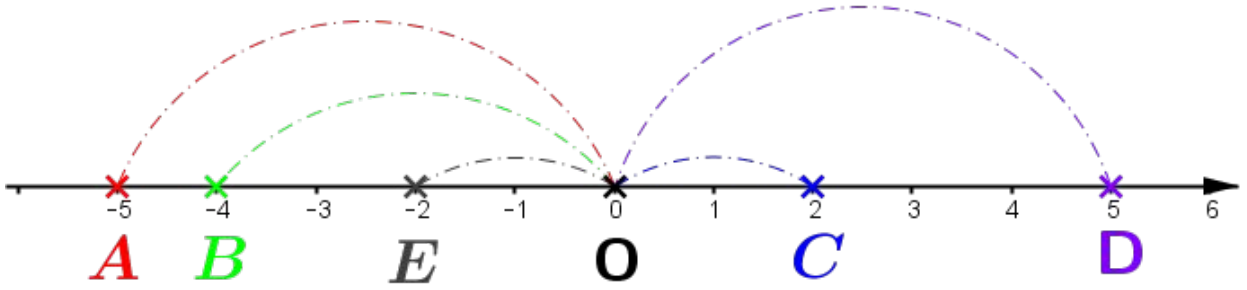
A canonical problem solvable by a *multilayer perceptron* (**MLP**).

- 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

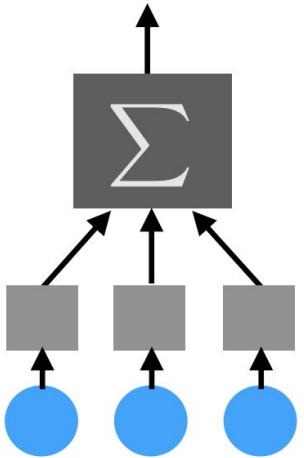
- **Input**: A **set** of 1D points, with features containing their coordinate and colour.
- **Output**: Some **aggregate** property of the set (e.g. the *furthest pairwise distance*).

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.

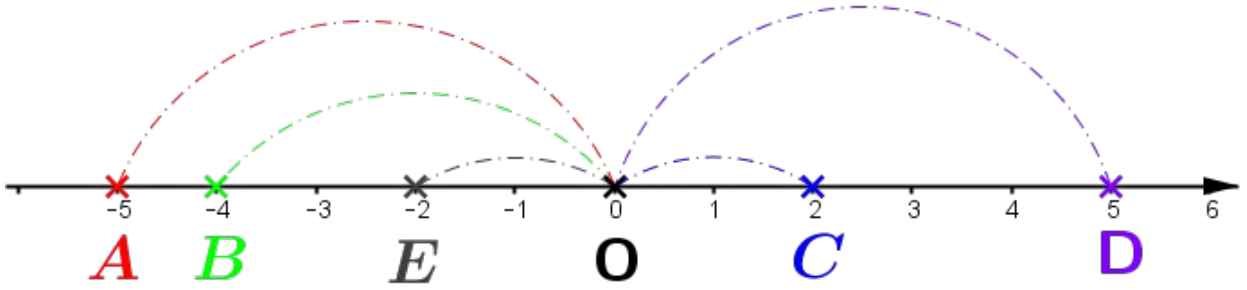
MLPs have no way of dealing with set boundaries!

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*)

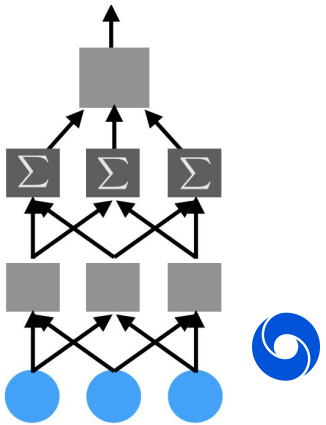
A **relational argmax** problem: requires **identifying** an optimising (pairwise) *relation*.

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*)

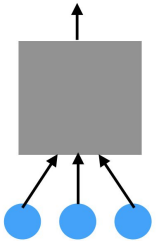
A **relational argmax** problem: requires **identifying** an optimising (pairwise) *relation*.

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*)

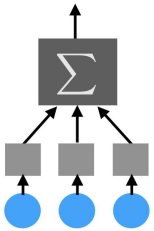
Architectures so far



MLPs

~ feature extraction

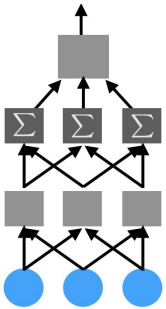
$$y = \mathrm{MLP}(\|_{s \in S} X_s)$$



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?

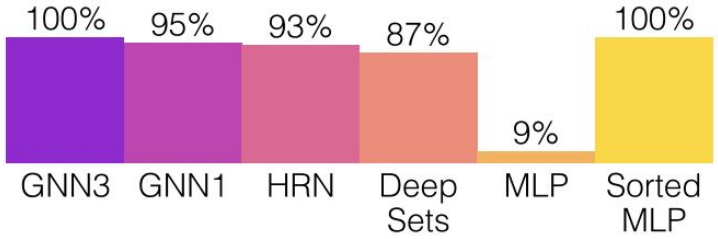
- **Theorem:** better *structural alignment* implies better *generalisation*!
  - GNNs ~ dynamic programming

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

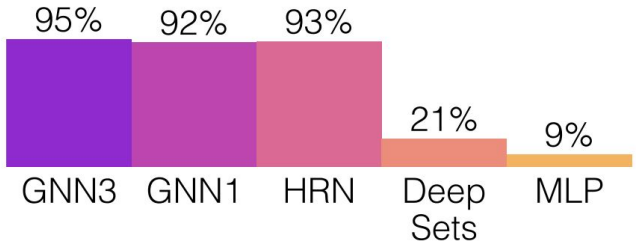
(Xu, Li, Zhang, Du, Kawarabayashi and Jegelka. ICLR 2020)

Empirical results

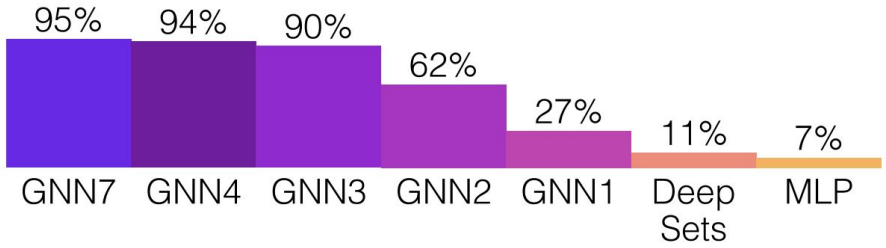
Summary statistics
What is the maximum value difference among treasures?



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

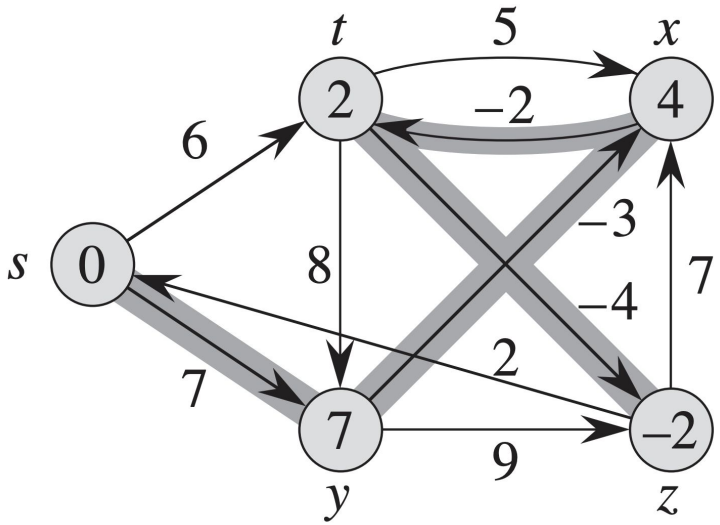


Dynamic programming
What is the cost to defeat monster X by following the optimal path?



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

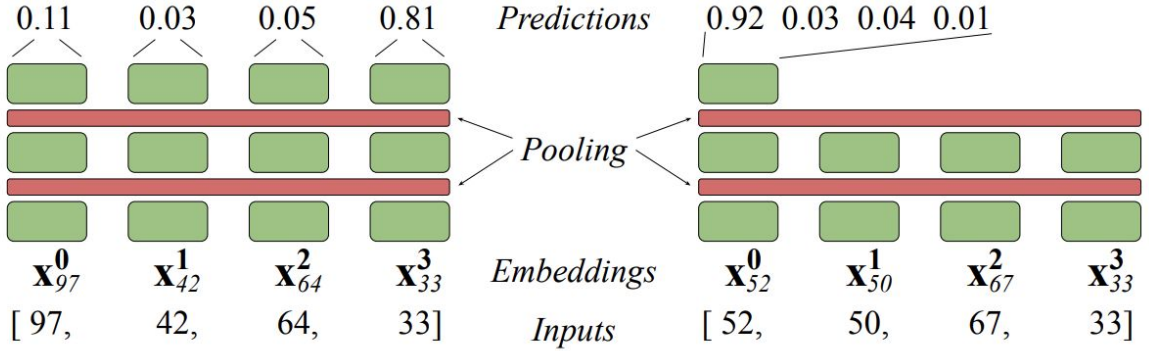
- A recent exploration of Transformers studies the effect of alignment on learning stability.

(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!

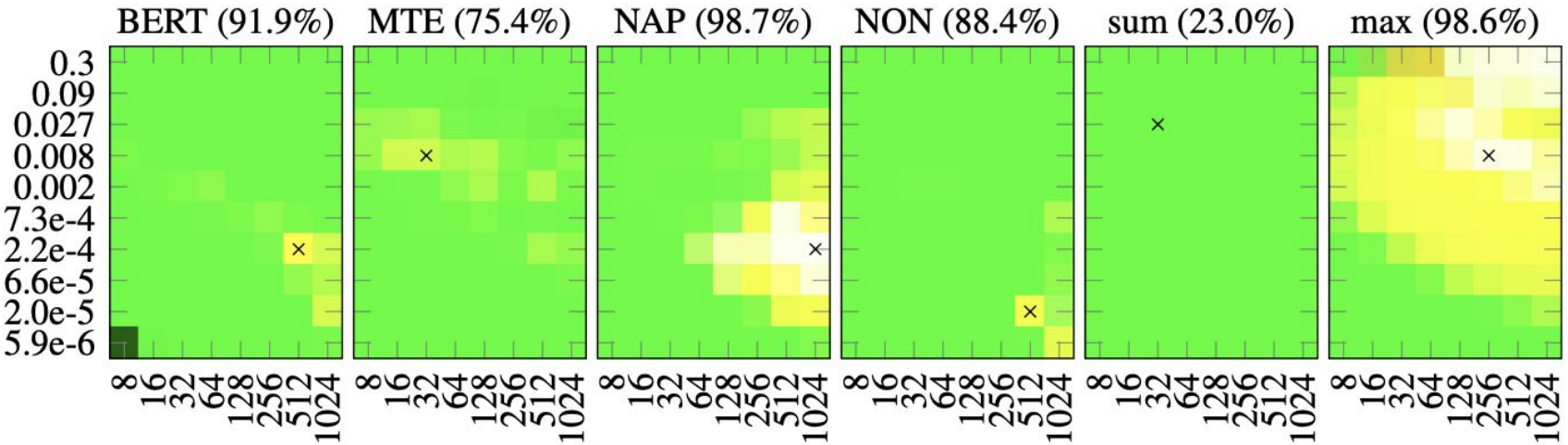


Figure 5: Learning rate (y-axis) vs. model dimension d (x-axis) on the case distinction task with output from the first token. RGB pixel values correspond to *argmin*-, *first*- and *argmax*-mean-case-accuracies. Crosses indicate the best mean accuracy, which we report behind the model name.

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

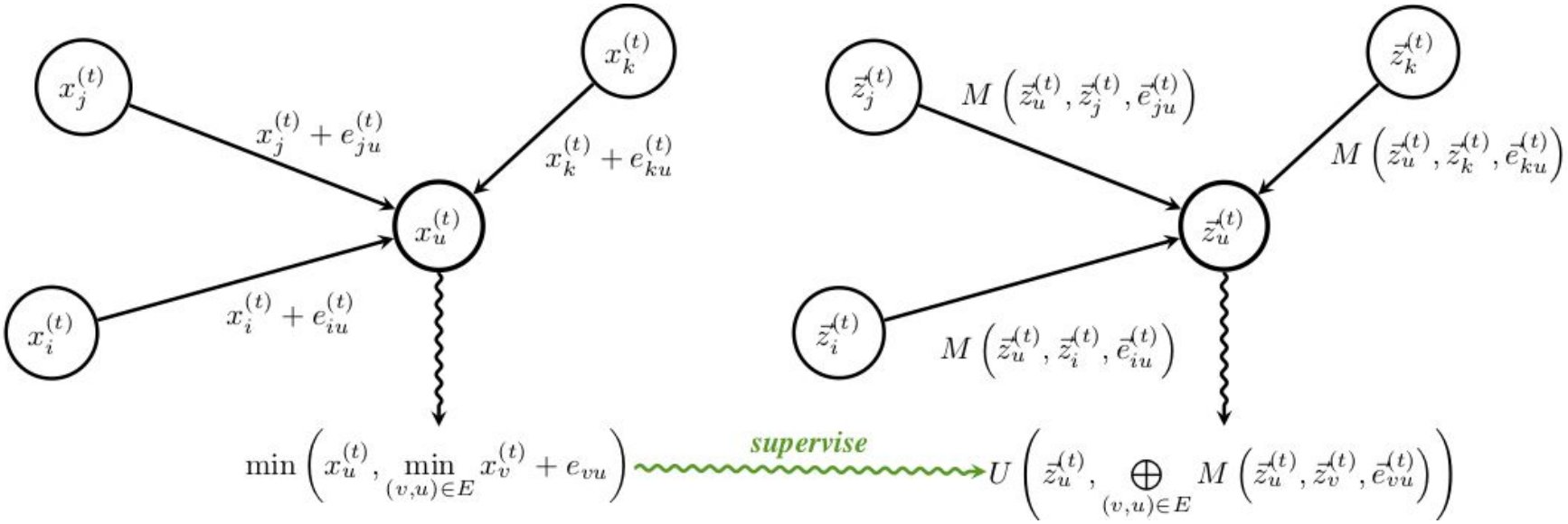
- Instruct the GNN computation to respect the **intermediate outputs** of the algorithm!

(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!

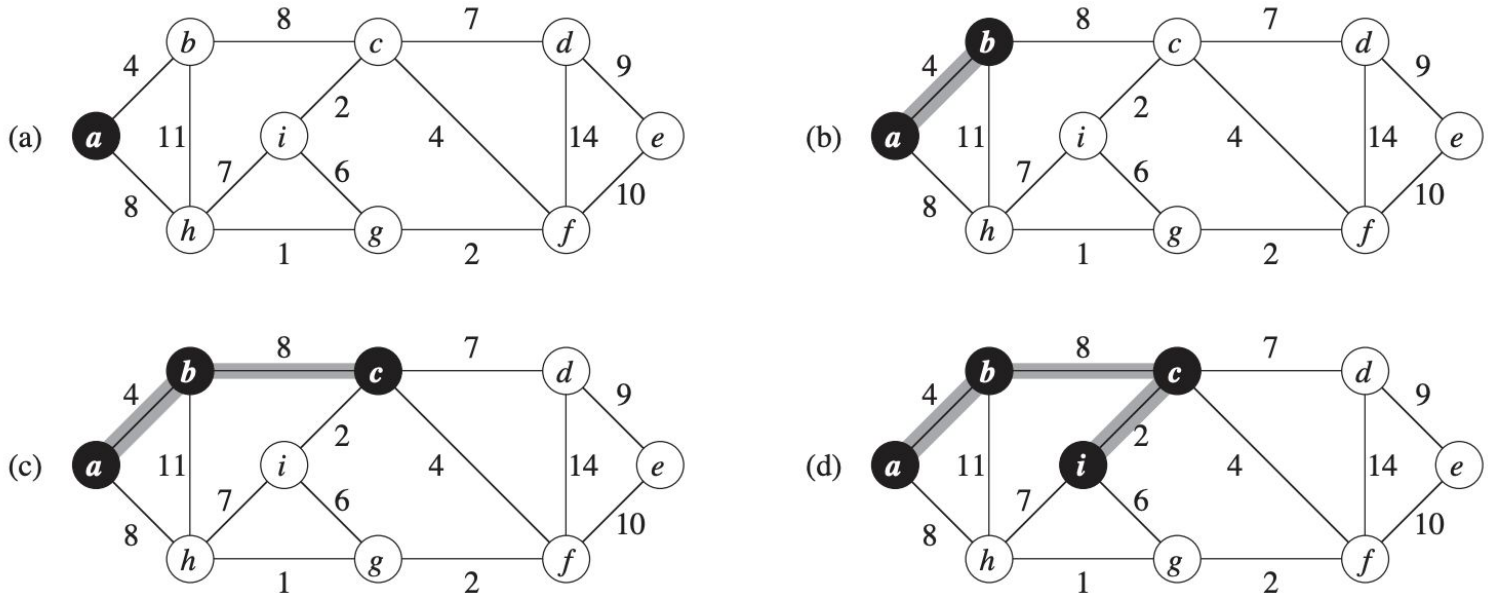
Aggregators other than max

Trained without step-wise supervision

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

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)

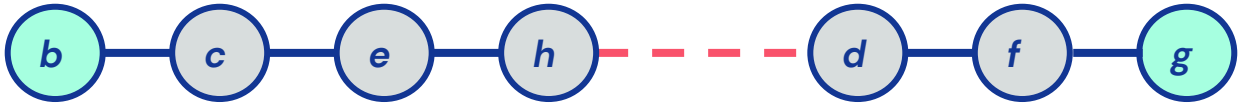
Evaluation: Sequential execution

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**!

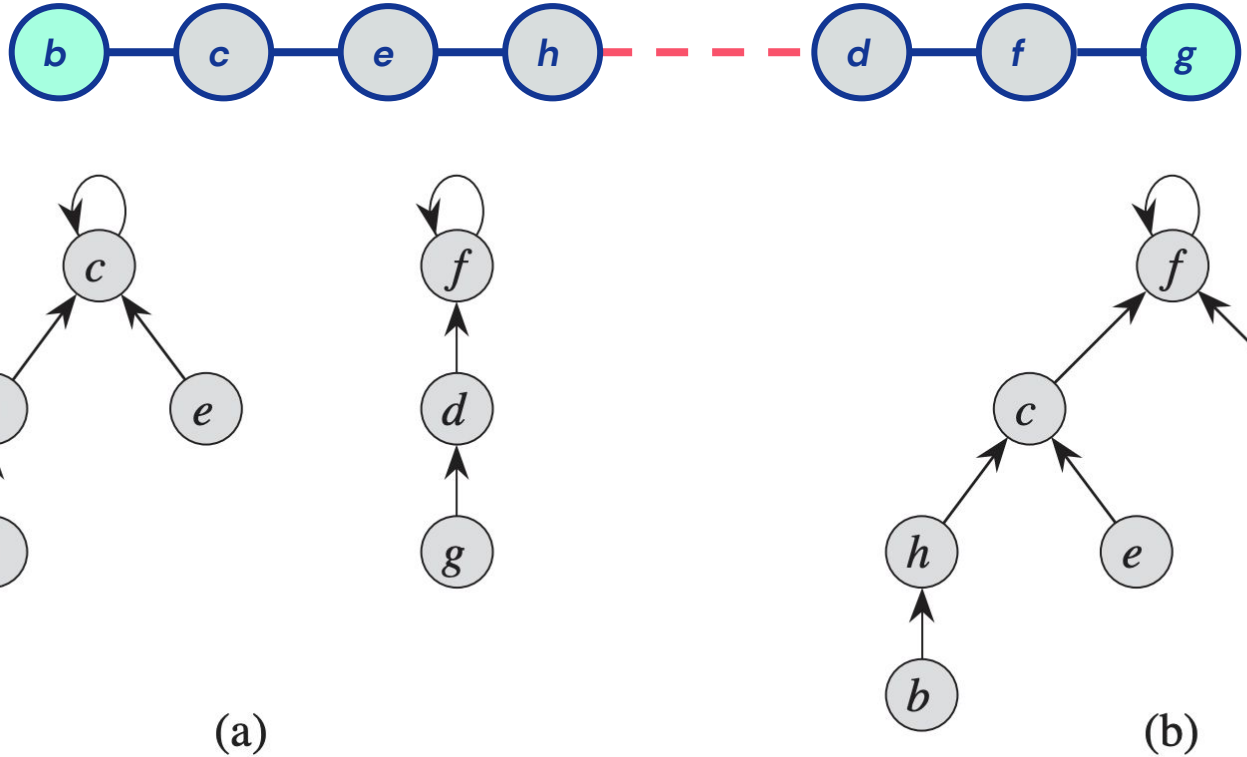
Incremental connectivity task

- **Input:** (u, v) representing an edge to add
- **Queries:** are nodes (i, j) **connected**?
- Are the input graph edges **most relevant**?
  - (relatedly: what to do when there is ***no*** graph?)
- 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. NeurIPS 2020)

Pointer Graph Networks

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**:

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

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

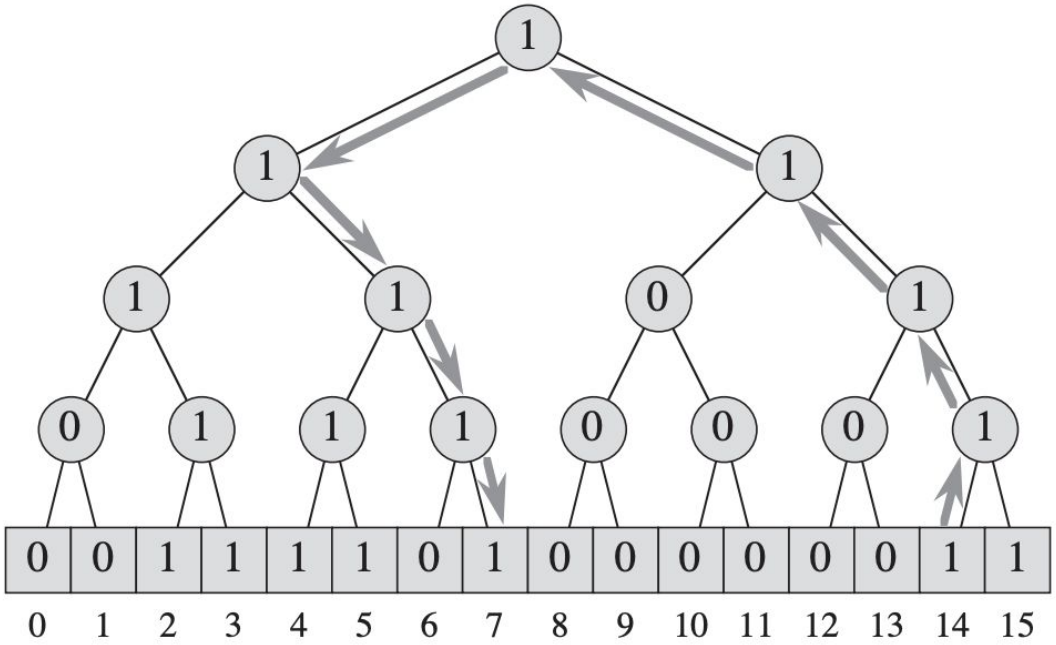
$$\vec{h}_i^{(t)} = U\left(\vec{z}_i^{(t)}, \max_{\boldsymbol{\Pi}_{ji}^{(t-1)}=1} M\left(\vec{z}_i^{(t)}, \vec{z}_j^{(t)}\right)\right)$$

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

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

Masking inductive bias

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



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**.

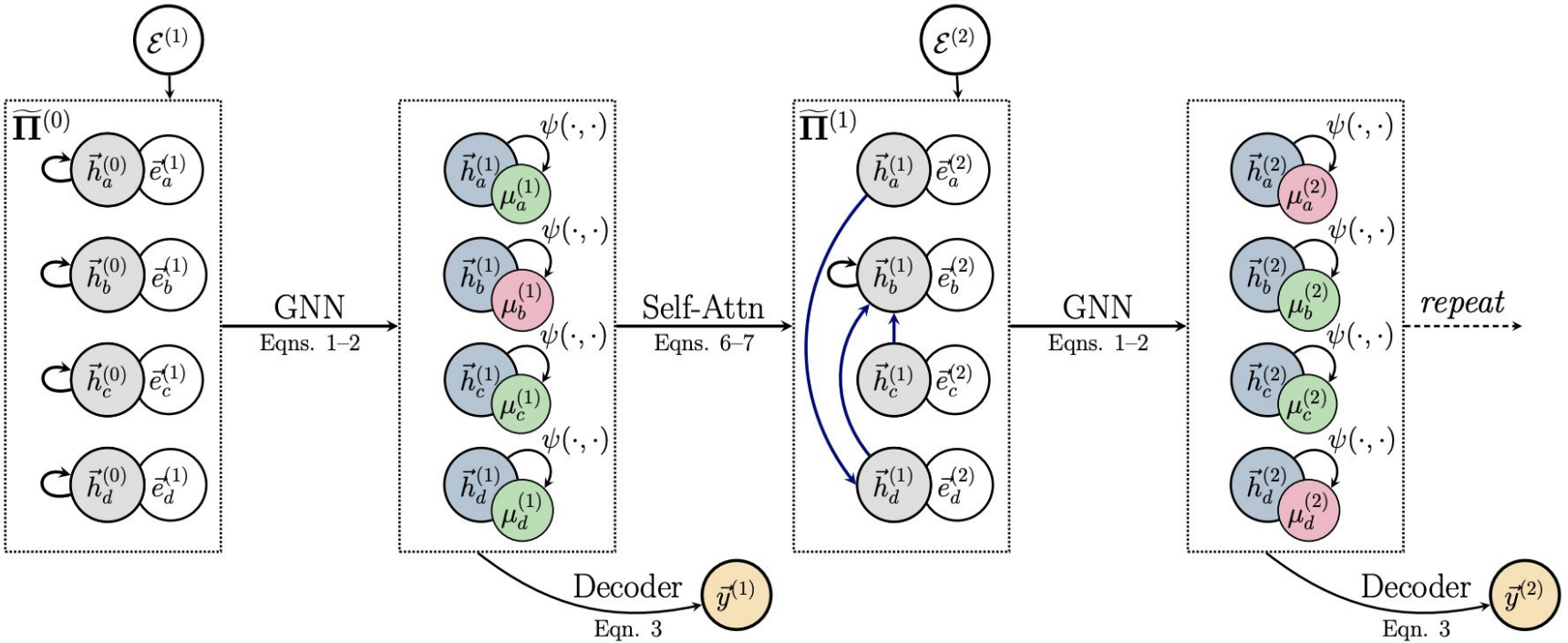
$$\mathbb{P}\left(\mu_i^{(t)} = 1\right) = \psi\left(\vec{z}_i^{(t)}, \vec{h}_i^{(t)}\right)$$

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

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

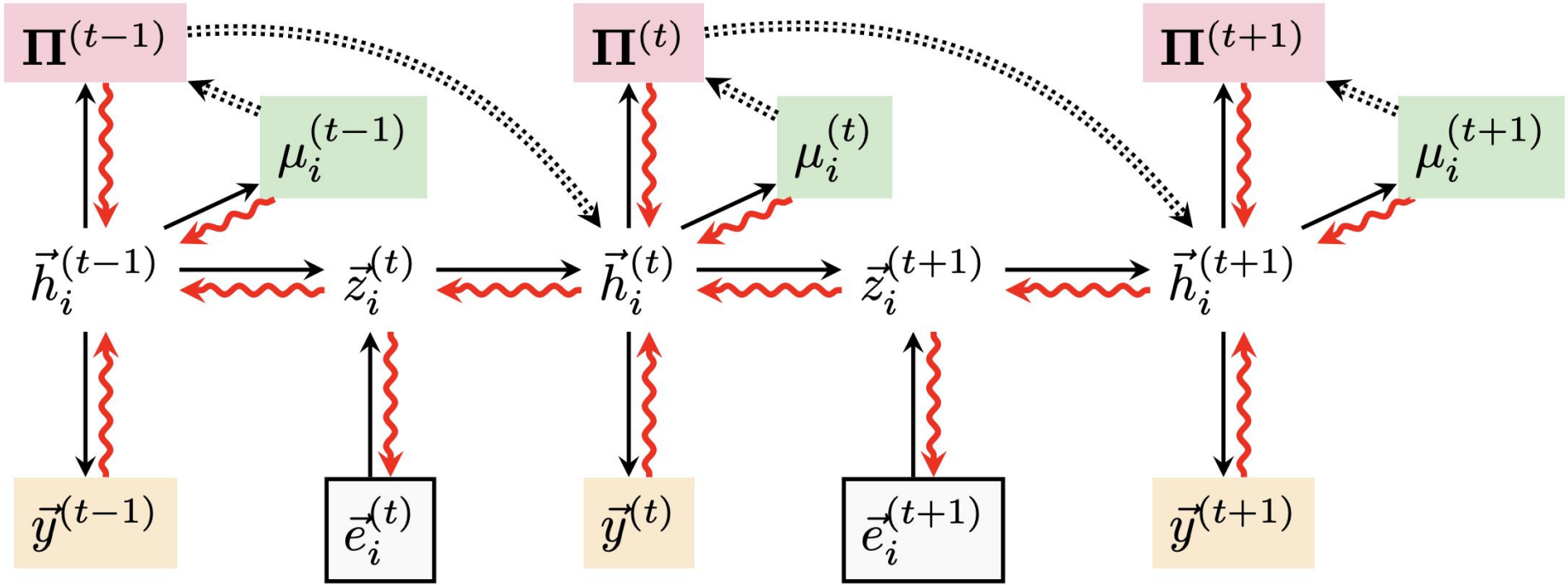
Pointer Graph Network (PGN)

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



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

Overall PGN dataflow



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 $n = 20$, 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	$0.944 \pm .006$	$0.964 \pm .007$	$0.968 \pm .013$	$0.776 \pm .011$	$0.744 \pm .026$	$0.636 \pm .065$

Incremental connectivity

Fully dynamic connectivity

Trained without masking objective

Trained on ground-truth pointer graphs

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%

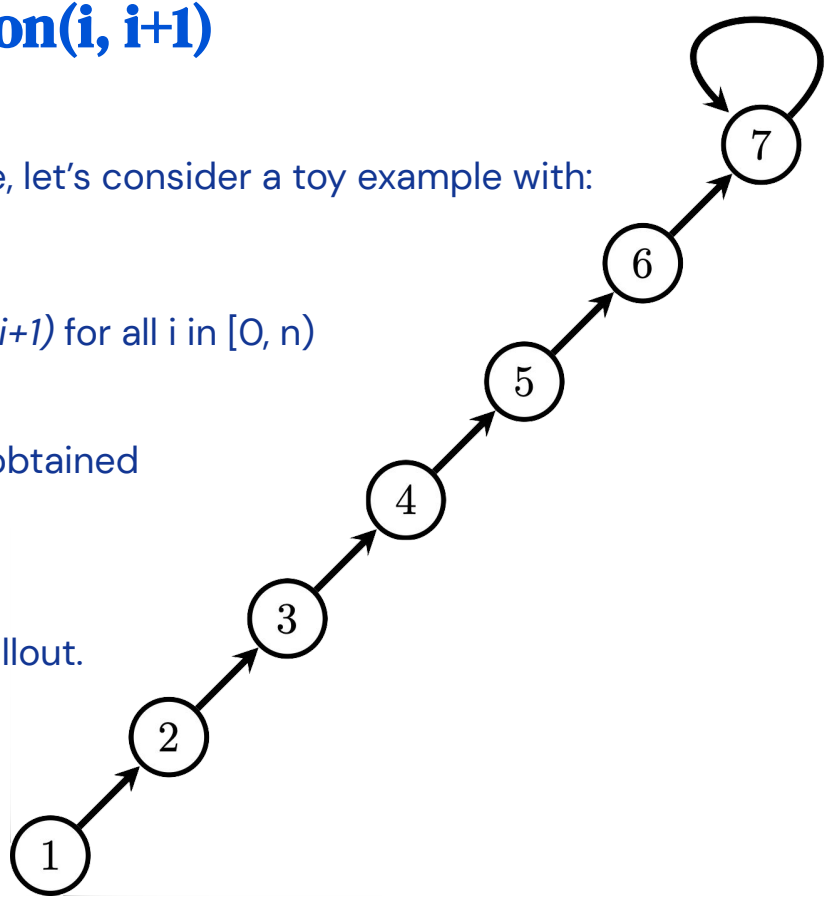
- It appears that our learnt data structure substantially **deviates** from ground-truths!
- What did it learn to do?

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)

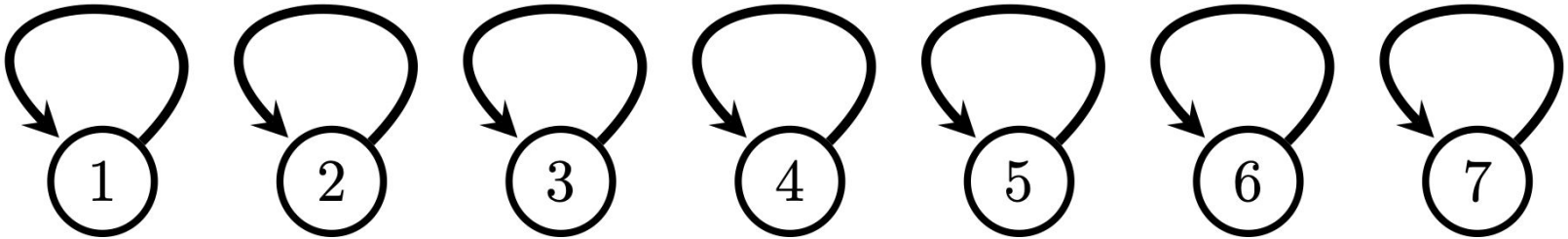
- The ground-truth DSU pointers obtained form a "worst-case"* scenario:

- We will perform a trained PGN rollout.



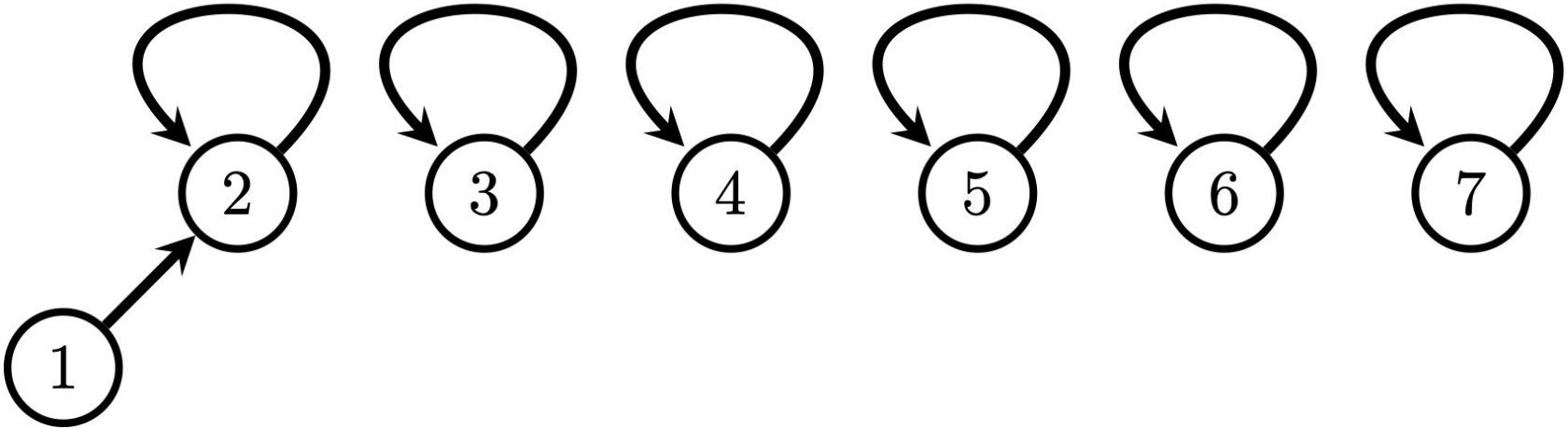
*not really damaging for DSU, but potentially troublesome for GNNs (**large diameter**).

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



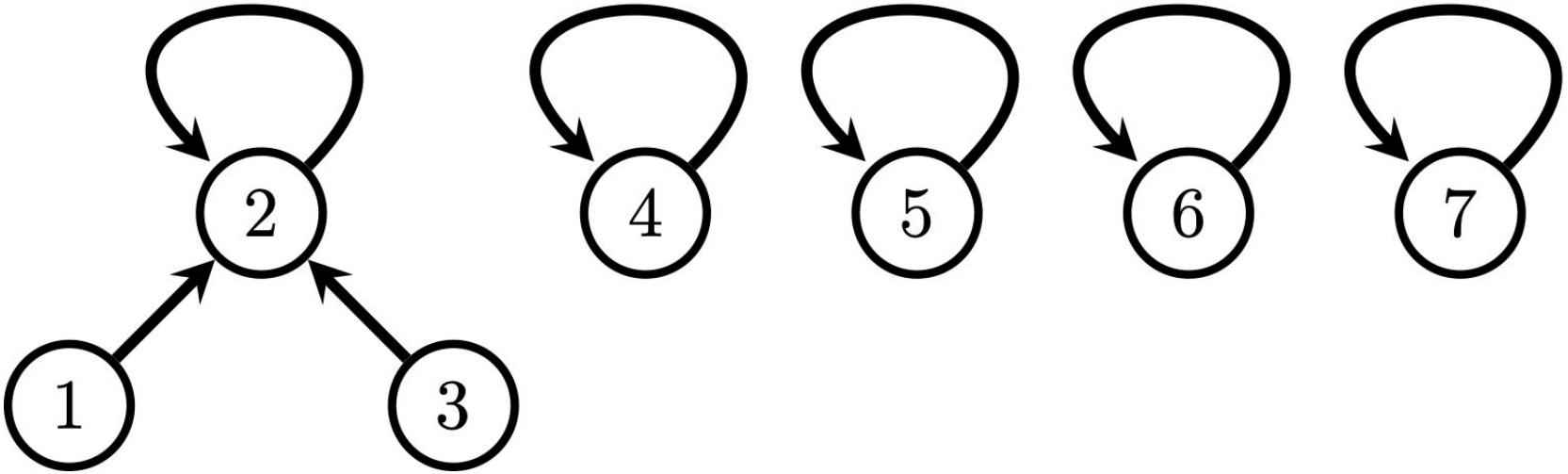
(Veličković, Buesing, Overlan, Pascanu, Vinyals and Blundell. NeurIPS 2020)

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



So far, so good....

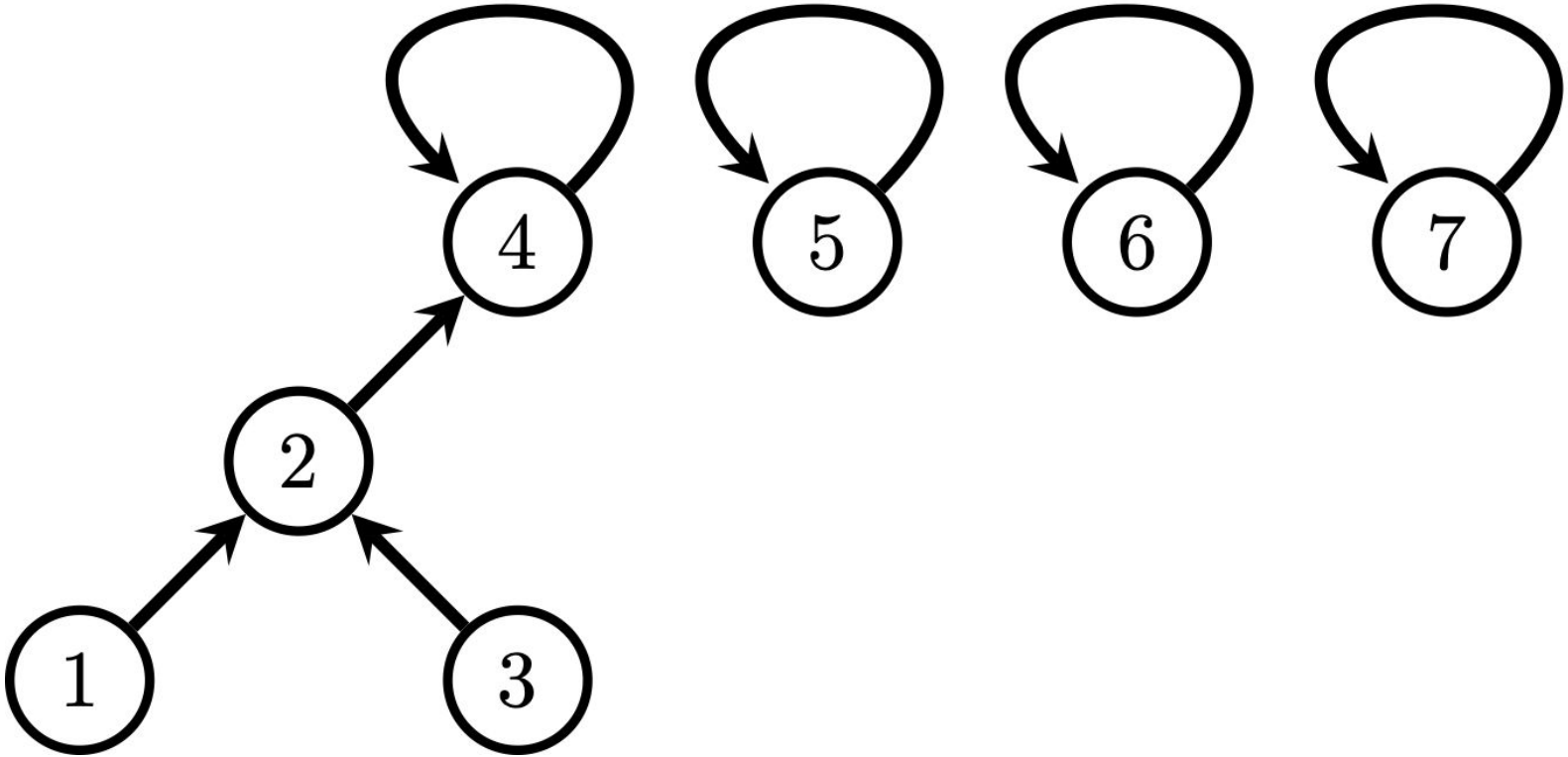
PGN iterations on **Union(i, i+1)**: *(2, 3)*



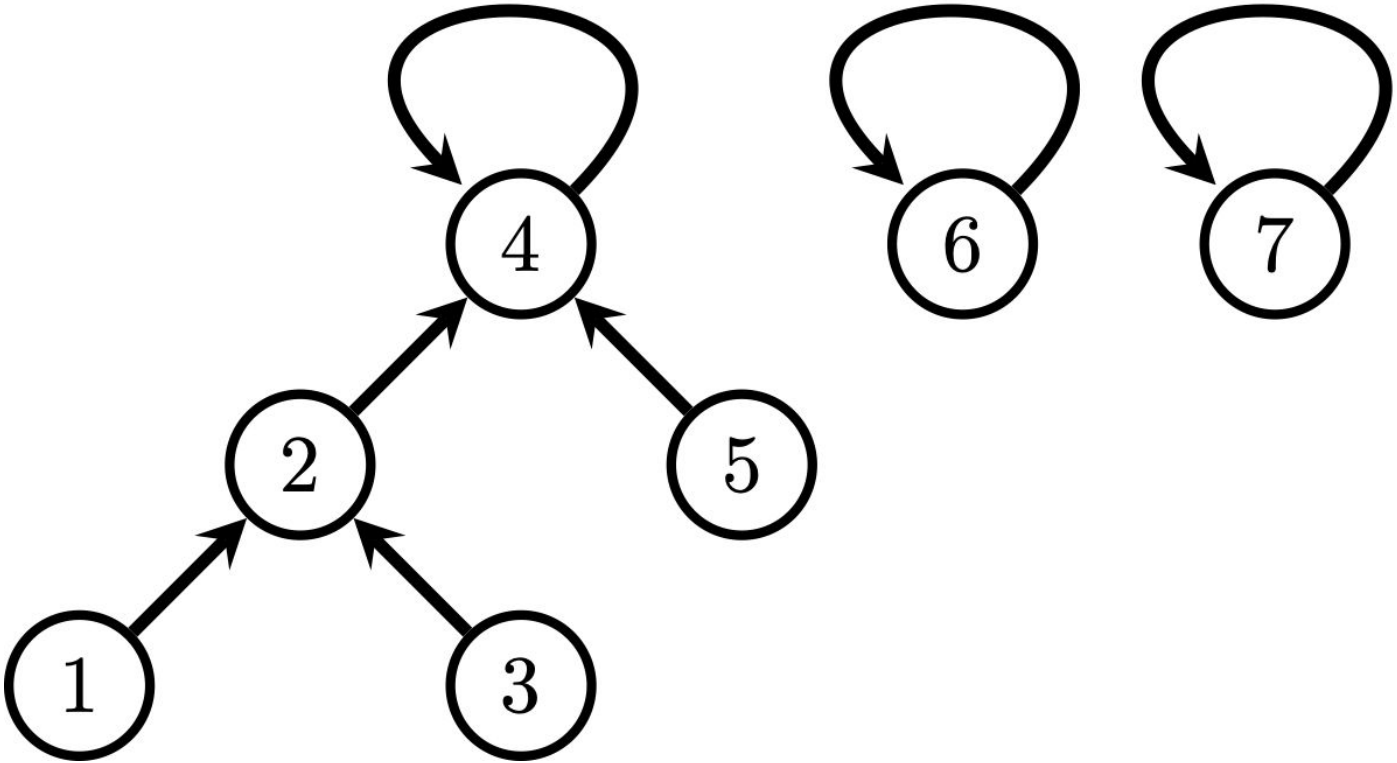
Differs from ground-truth already -- and **shallower**!

Continuing from here...

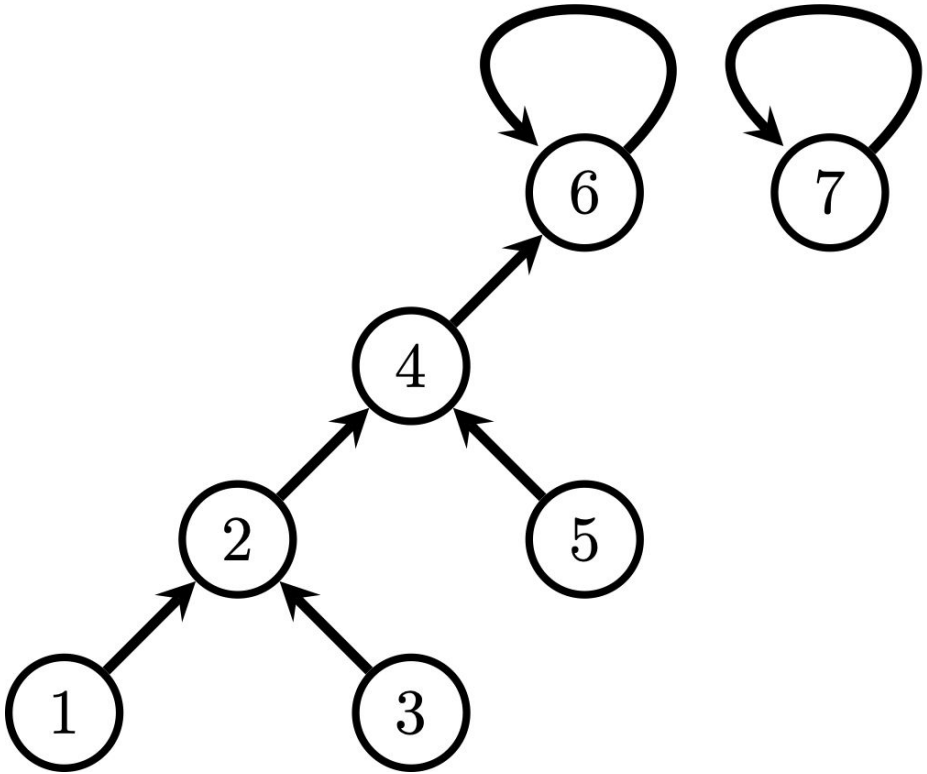
PGN iterations on **Union(i, i+1)**: *(3, 4)*



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



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



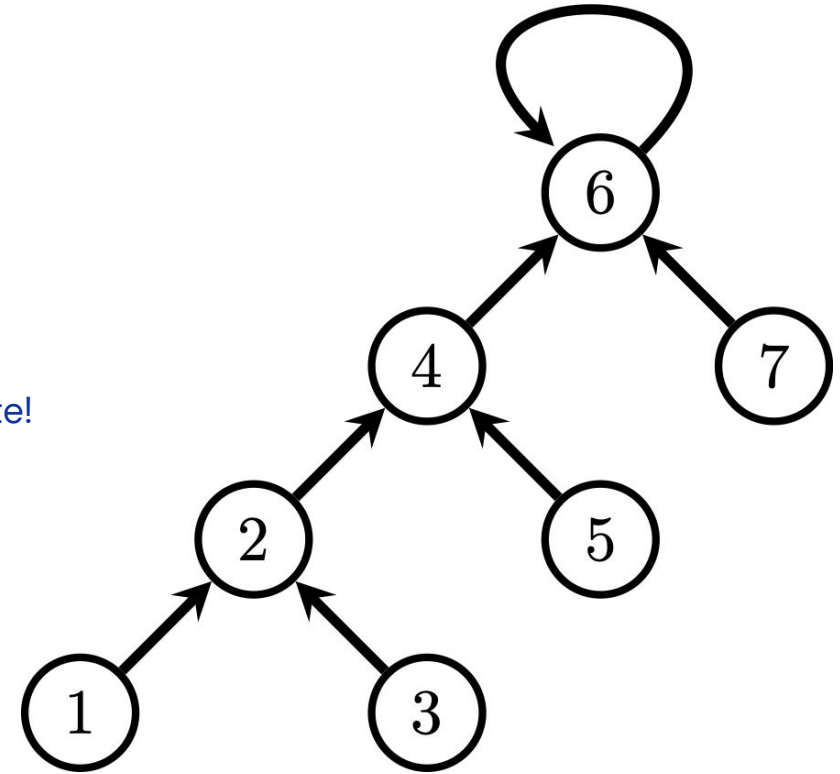
PGN iterations on **Union(i, i+1)**: *(6, 7)*

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:

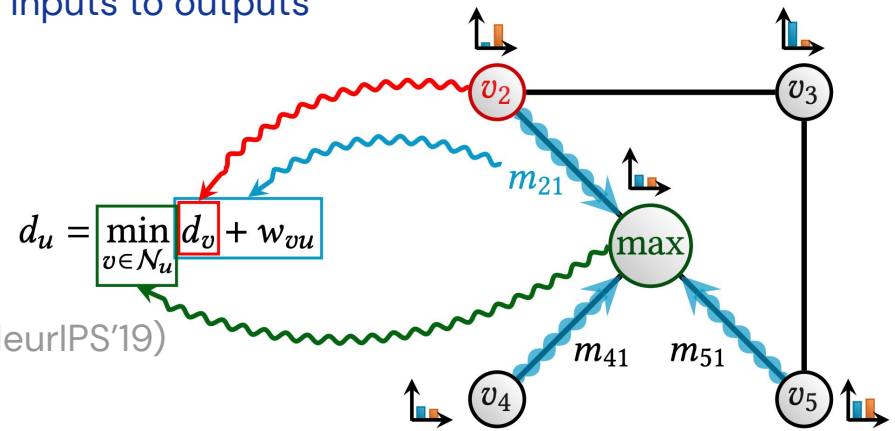
- *What is the kind of reasoning procedure I'd like my neural network to perform?*
- *How to **constrain** the network to compute (intermediate) results in this manner?*

tl;dr of algorithmic reasoning

- *Graph neural networks* (GNNs) *align* well with **dynamic programming** (Xu *et al.*, ICLR'20)

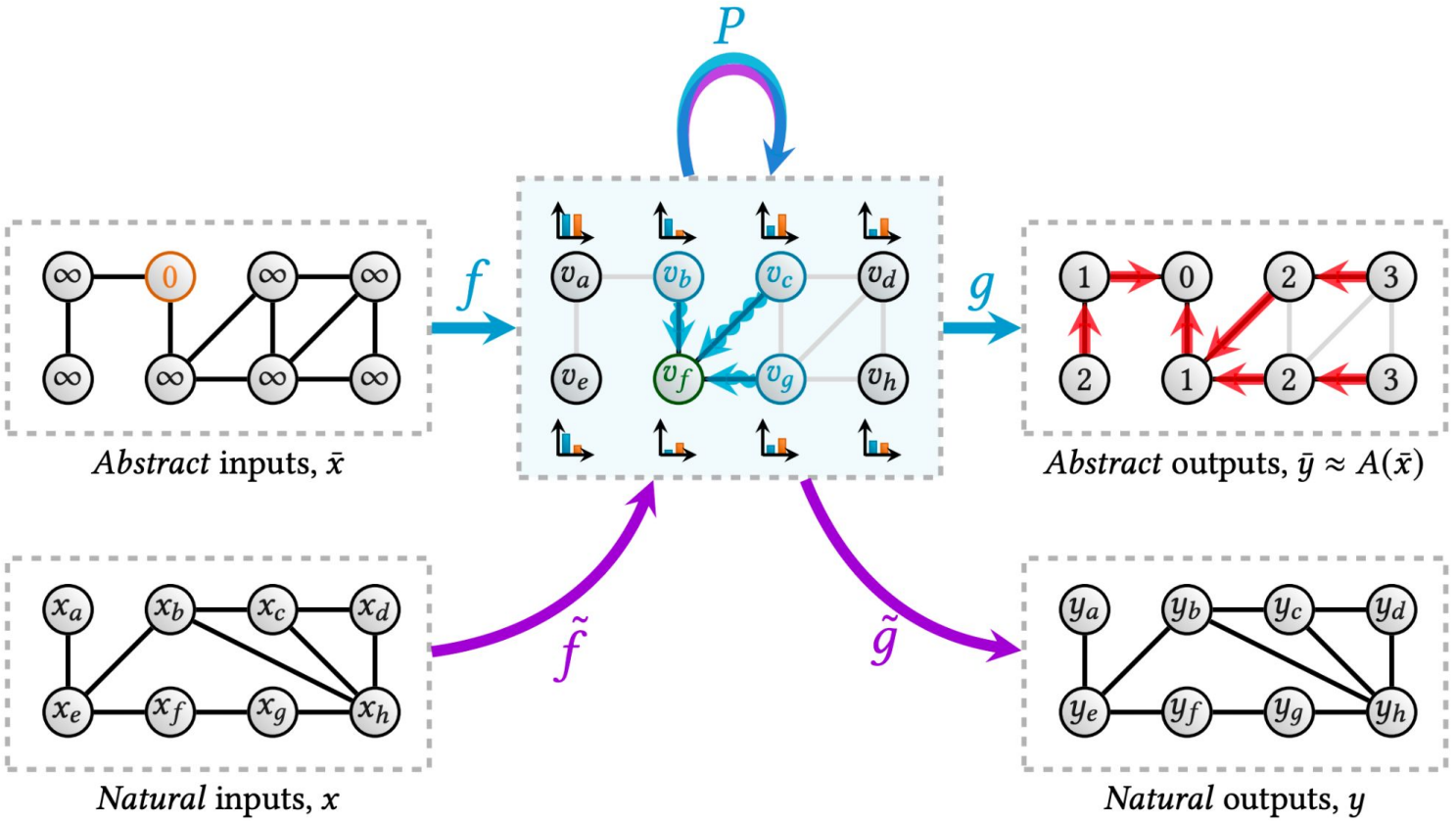
- Interesting **inductive biases** explored by Veličković *et al.* (ICLR'20):
  - Encode-***process***-decode from abstract inputs to outputs
  - Favour the **max** aggregation
  - **Strong** supervision on trajectories

- Further interesting work:
  - IterGNNs (Tang *et al.*, NeurIPS'20)
  - Shuffle-exchange nets (Freivalds *et al.*, NeurIPS'19)
  - PGN (Veličković *et al.*, NeurIPS'20)
  - PMP (Strathmann *et al.*, ICLR'21 SimDL)



- Latest insights: ***linear*** *algorithmic alignment* is highly beneficial (Xu *et al.*, ICLR'21)

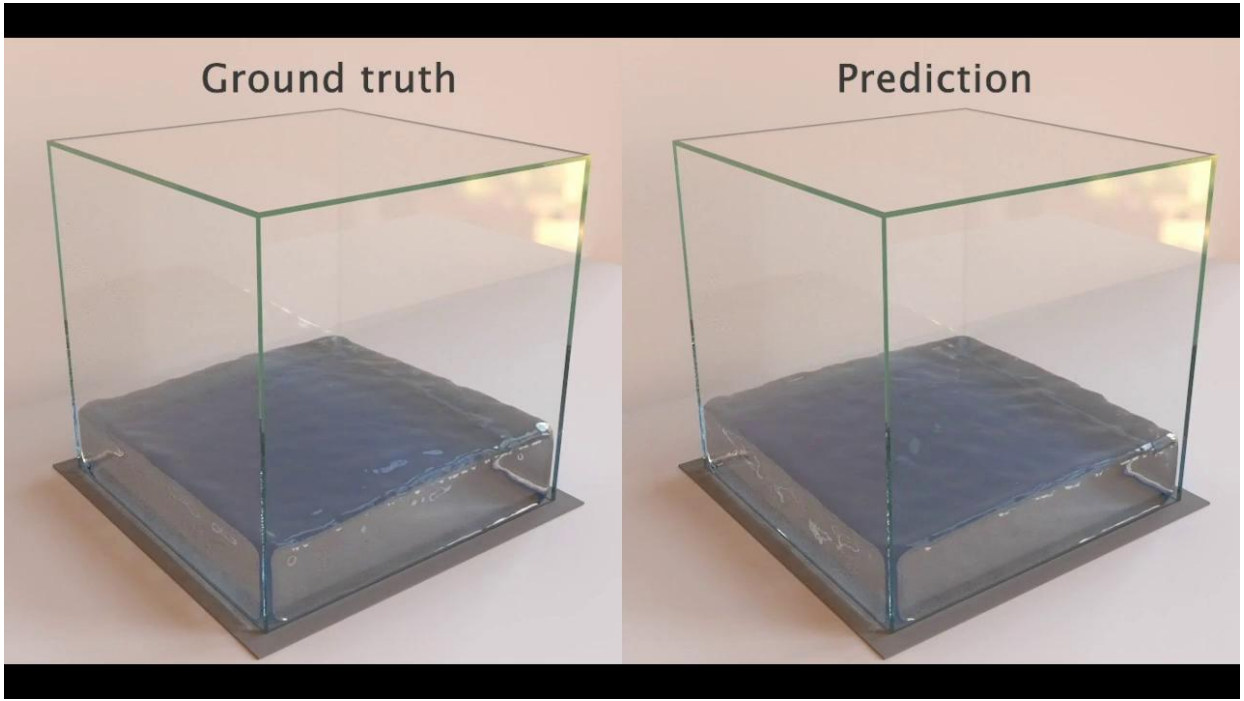
Blueprint of algorithmic reasoning



"Learning to Simulate Complex Physics with Graph Networks" (ICML 2020)
Video page: sites.google.com/view/learning-to-simulate

Aside: Connection to *simulations*

You might've come across a very exciting recent body of work for simulating **physics** with GNNs



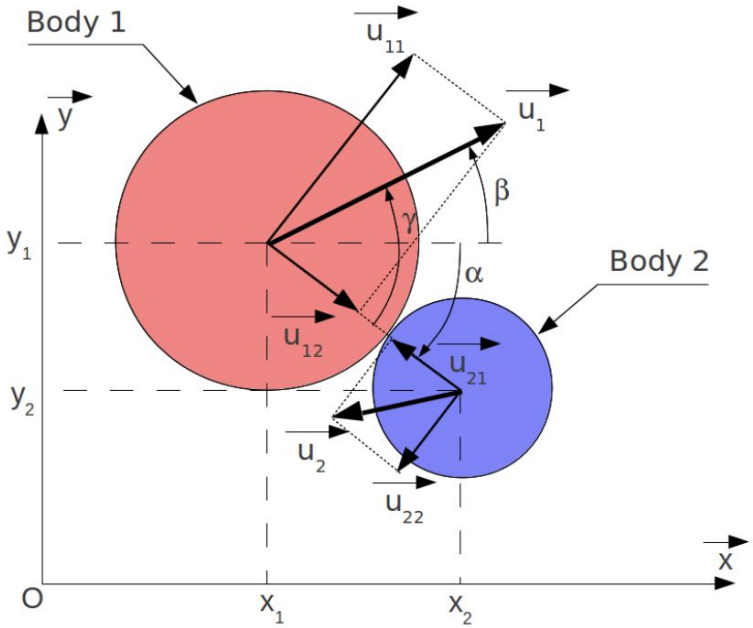
DeepMind

Why are *simulations* of interest for algorithms?

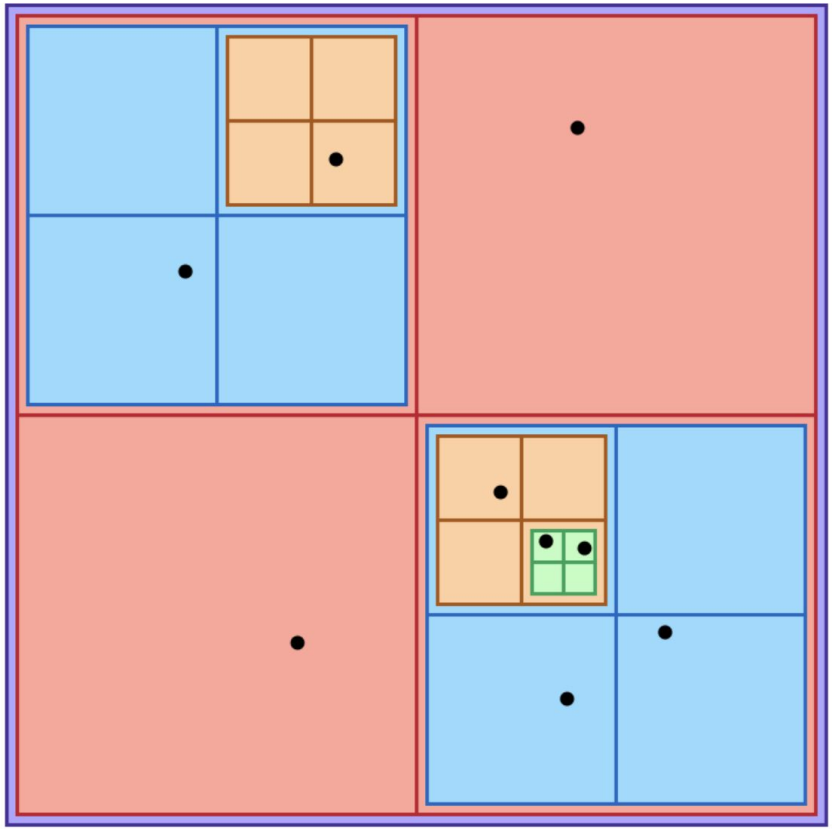
(a few examples)

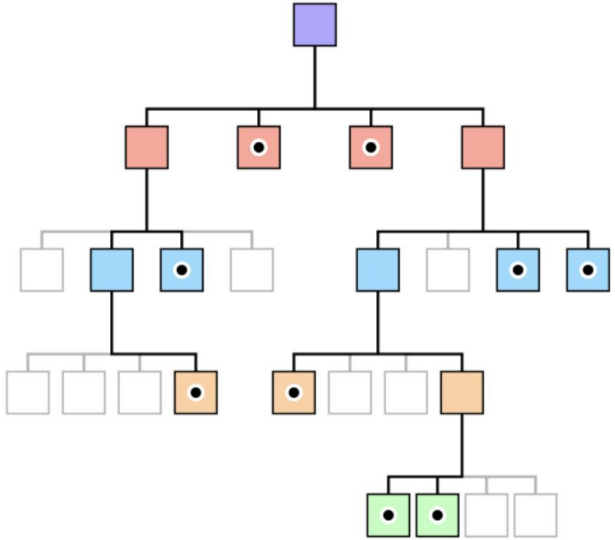
Physics simulations / collision detectors have *algorithms...*

```
dt = 1.0 / 60.0
a = 0.0
t1 = current_time_seconds()

while true:
    t2 = current_time_seconds()
    a += t2 - t1
    t1 = t2

    while a >= dt:
        prevState = state
        step(state, dt)
        a -= dt

    alpha = a / dt
    render(interpolate(prevState, state, alpha))
```



… and *data structures*! (Quadtree / k-d tree)

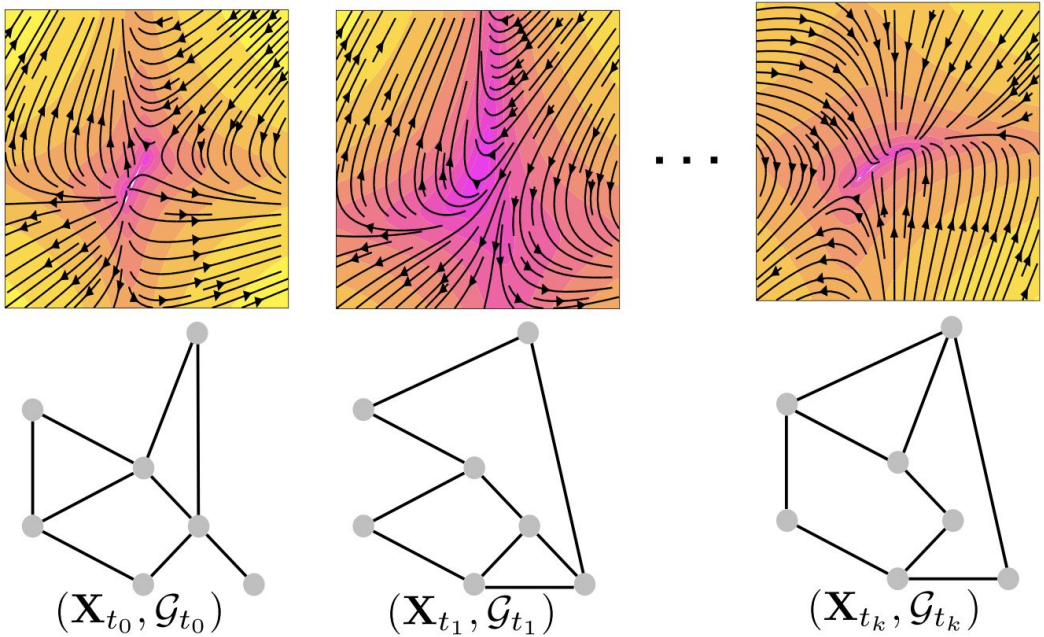




Numerical integrators? (hence ODEs)

Ultimately, computers cannot *really* do **real numbers**...

$$\dot{\mathbf{H}}_t = \mathbf{F}(t, \mathbf{H}_t, \mathbf{\Theta})$$




```
1. Start

2. Define function f(x,y)

3. Read values of initial condition(x0 and y0),
    number of steps (n) and calculation point (xn)

4. Calculate step size (h) = (xn - x0)/n

5. Set i=0

6. Loop

        k1 = h * f(x0, y0)
        k2 = h * f(x0+h/2, y0+k1/2)
        k3 = h * f(x0+h/2, y0+k2/2)
        k4 = h * f(x0+h, y0+k3)

        k = (k1+2*k2+2*k3+k4)/6

        yn = y0 + k

        i = i + 1
        x0 = x0 + h
        y0 = yn

    While i < n
```

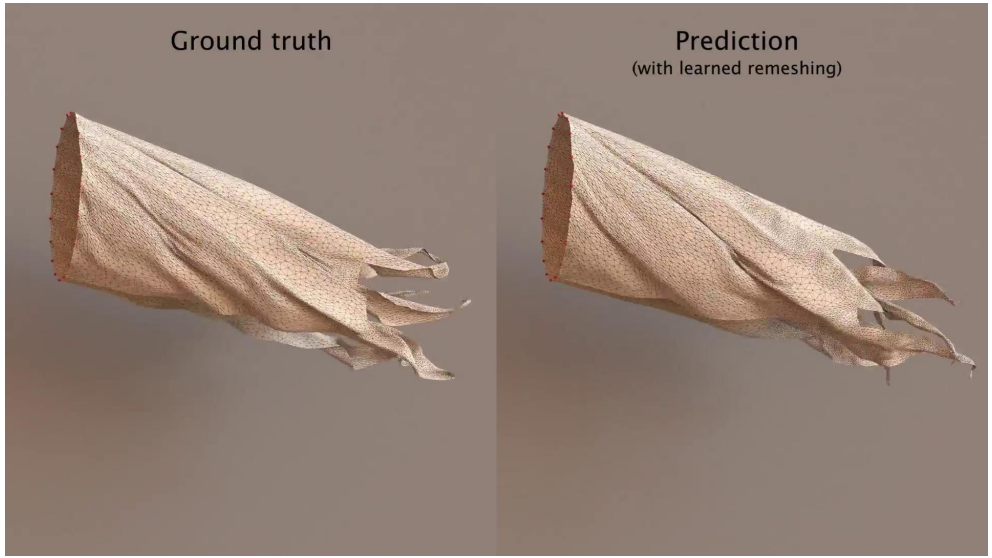
Simulations of life (Gillespie)

```
   //Initialize:
01 FOR i=1,...,N
02   x[i] = S //set node states to S
03 ENDFOR
04 x[root] = I //set state of root node to I
05 m_I = [root] //list of infected nodes
06 N_I = 1 //number of infected nodes
07 N_R = 0 //number of recovered nodes
08 Mu = mu //cumulative recovery rate
09 tau = randexp(1) //draw tau ~ Exp(1)

   //Run through the time-steps:
10 FOR t=0,1,...,T_simulation-1
     //Update list of possible S->I transitions:
11   CLEAR m_SI //S nodes in contact with I nodes
12   FOR contact in contactLists[t]
13     (i,j) = contact
14     IF (x[i],x[j])==(S,I)
15       APPEND i to m_SI
16     ELSE IF (x[i],x[j])==(I,S)
17       APPEND j to m_SI
18     ENDIF
19   ENDFOR
20   M_SI = length of m_si
21   Beta = beta*M_SI //cumulative infection rate
22   Lambda = Mu+Beta //cumulative transition rate
//Check if a transition takes place:
23   IF Lambda<tau //no transition
24     tau -= Lambda
25   ELSE //at least one transition
26     xi = 1. //remaining fraction of time-step
27     WHILE xi*Lambda>=tau
28       DRAW z uniformly from [0,Lambda)
29       IF z<Beta //S->I transition
30         DRAW m at random from m_SI
31         x[m] = I
32         APPEND m to m_I
33         N_I += 1
34         Mu += mu
35       ELSE //I->R transition
36         DRAW m at random from m_I
37         x[m] = R
38         REMOVE m from m_I
39         N_I -= 1
40         N_R += 1
41         Mu -= mu
42       ENDIF
43       xi -= tau/Lambda //update remaining fraction
         //Update list of S->I transitions and rates:
44       REDO lines 11-22
45       tau = randexp(1) //draw new tau
46     ENDWHILE
47   ENDIF
     //Read out the desired quantities:
48   WRITE N_I, N_R, ...
49 ENDFOR
```

Pseudocode 1. **Pseudocode for an SIR process with constant and homogeneous transition rates.** C++ code for homogeneous and heterogeneous populations is found at `https://github.com/CLVestergaard/TemporalGillespieAlgorithm`.

A very deep connection

It should come as **no surprise** that physics-simulating GNNs are largely recommending the same inductive biases as **algorithmic reasoning** :)



(See also: **Kyle Cranmer**'s recent guest lecture at USI Lugano)

DeepMind

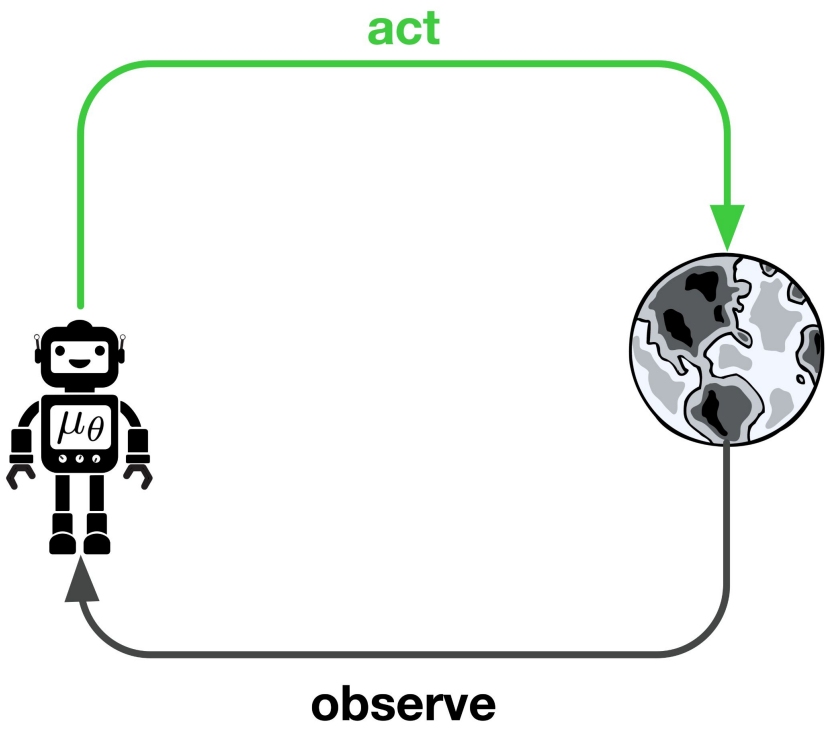
(Have we answered Question 2?)

- How to **build** neural networks that behave algorithmically?
  - And why am I even telling you this in a *“Graph Machine Learning”* course?

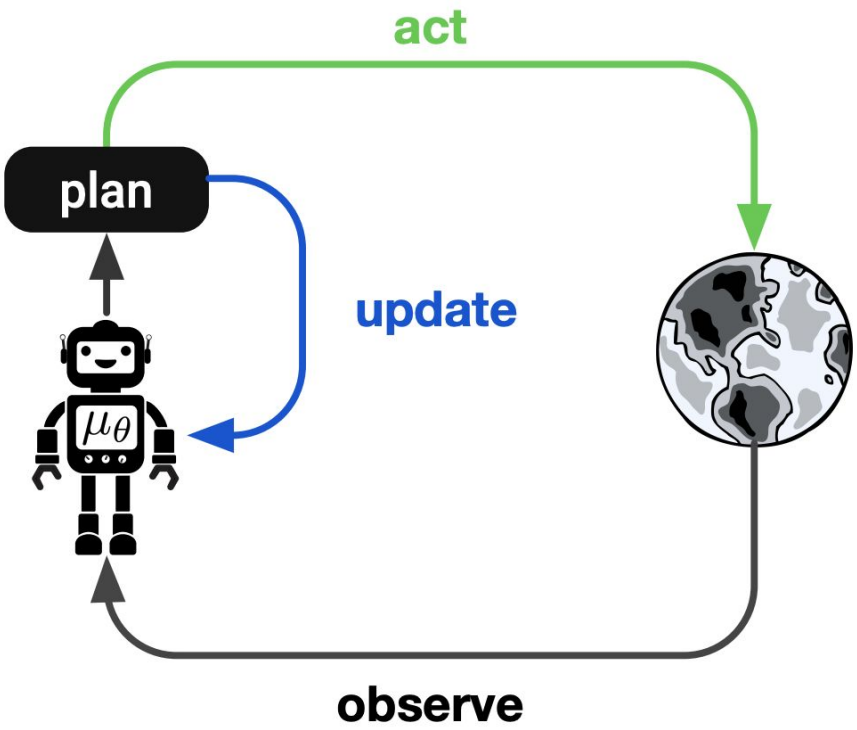
DeepMind

6 An algorithmic *implicit planner*

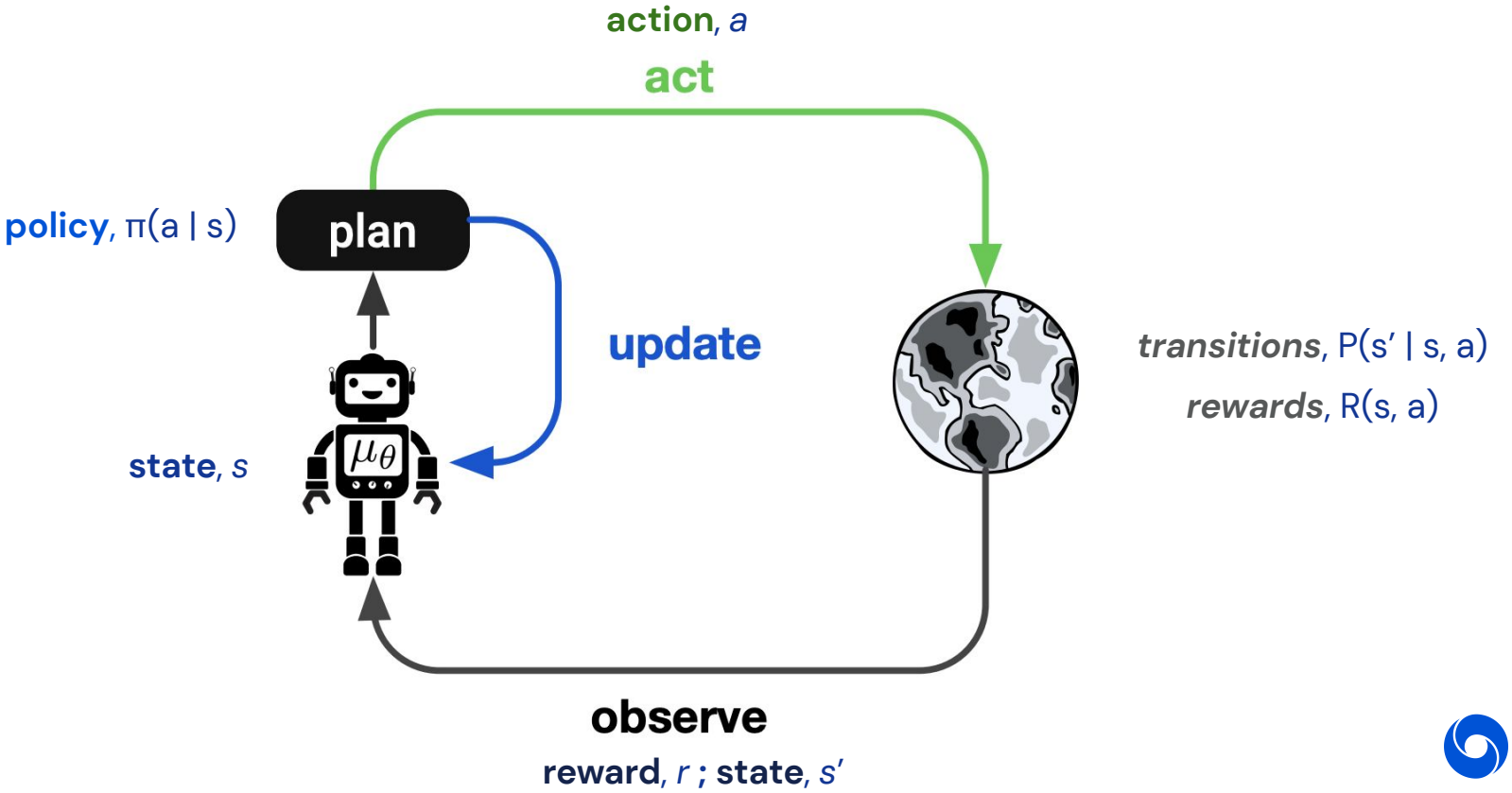
Reinforcement learning (RL) setting



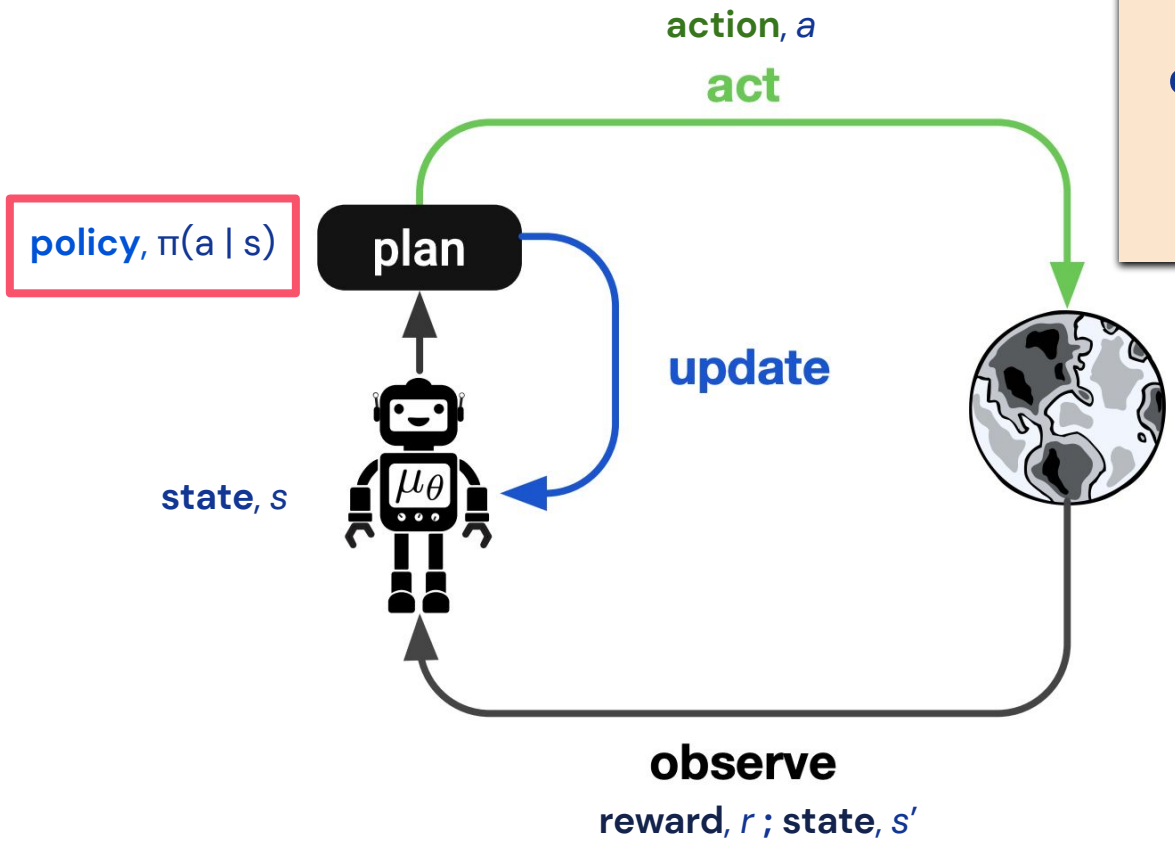
Reinforcement learning (RL) setting (with *planning*)



Reinforcement learning (RL) setting (variables)



Reinforcement learning (RL) setting



Want to optimise:

Discounted cumulative reward

$$G = \sum_{t \geq 0} \gamma^t r_t$$

Intro to value iteration

- Value Iteration: *dynamic programming* algorithm for **perfectly** solving an RL environment

$$v^{(t+1)}(s) = \max_{a \in \mathcal{A}_s} r(s,a) + \gamma \sum_{s' \in \mathcal{S}} p(s'|s,a) v^{(t)}(s')$$

  where v(*s*) corresponds to the **value** of state *s*.

- **Guaranteed** to converge to *optimal* solution (fixed-point of Bellman optimality equation)!

$$V^\star(s) = \max_{a \in \mathcal{A}} \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a) V^\star(s') \right)$$

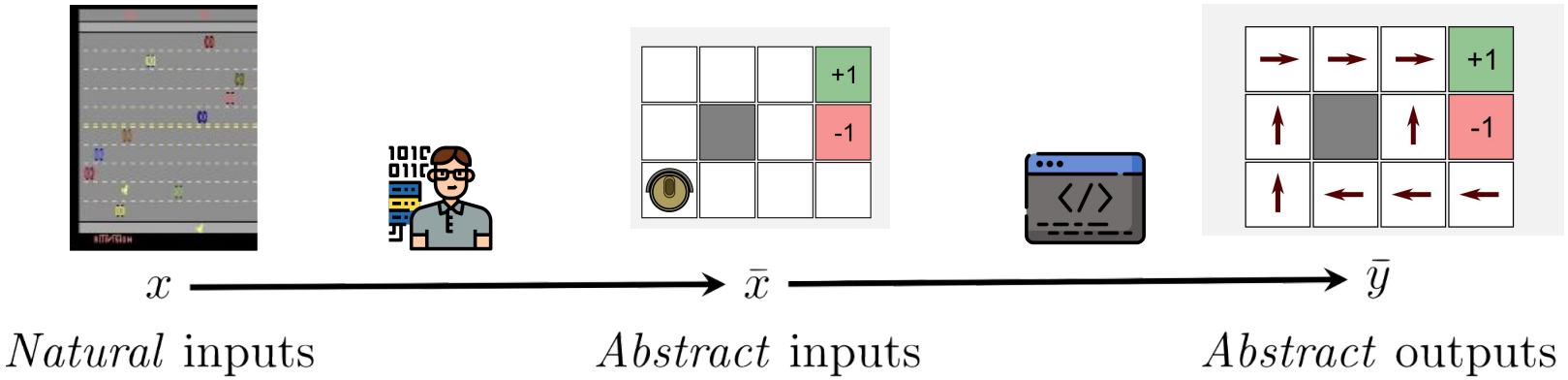
  Optimal policy takes actions that **maximise** *expected value*: $\text{argmax}_a \sum_{s'} V^*(s') P(s' \mid s, a)$

- BUT requires **full knowledge** of underlying MDP (P / R)
  - Prime target for our previously studied blueprint :)

Algorithmic reasoning over Value Iteration

- How would a human feature engineer make VI applicable?
  - Looking back to our blueprint example...

- As before, we will try to automate away the **manual** feature extraction



Latent-space transition models

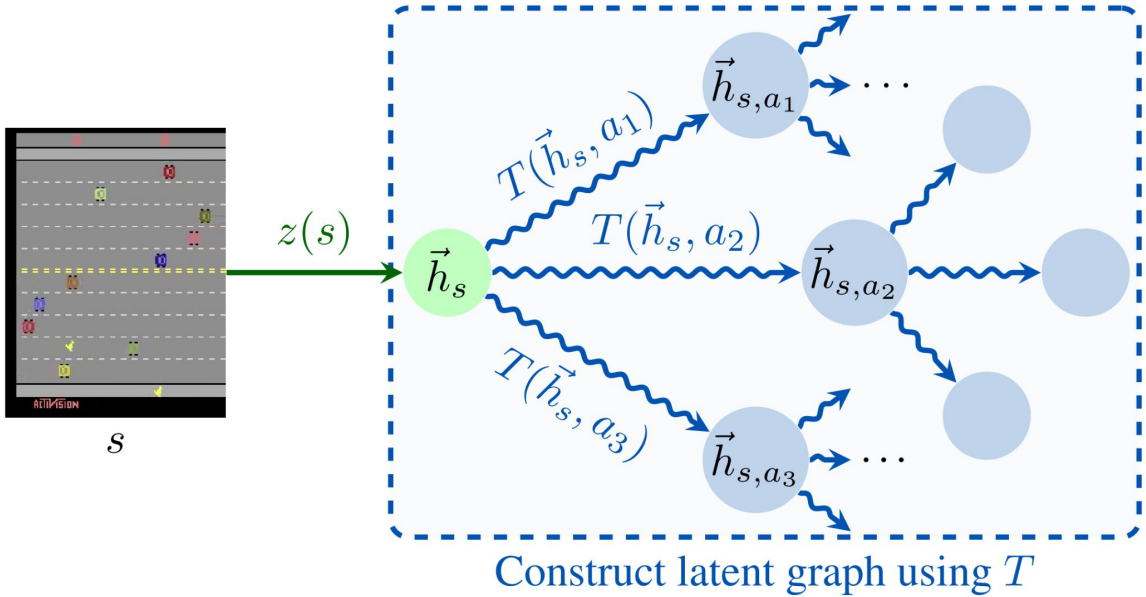
- Assume we have encoded our state (e.g. with a NN) into **embeddings**, $z(s) \in \mathbb{R}^k$
- To expand a *"local MDP"* we can apply VI over, we can then use a *transition model*, T
  - It is then of the form $T : \mathbb{R}^k \times A \rightarrow \mathbb{R}^k$
  - Optimised such that $T(z(s), a) \approx z(s')$
- Many popular methods exist for learning T in the context of *self-supervised learning*
- **Contrastive** learning methods try to discriminate (s, a, s') from *negative* pairs $(s, a, s^{\sim})$

Using a transition model to **expand**

We can use a learned transition model on **every** action, to be exhaustive (*~breadth-first search*)

Doesn't **scale** with large action spaces / thinking times; $O(|A|^K)$

Can find more interesting *rollout policies*, e.g. by **distilling** well-performing **model-free** ones.



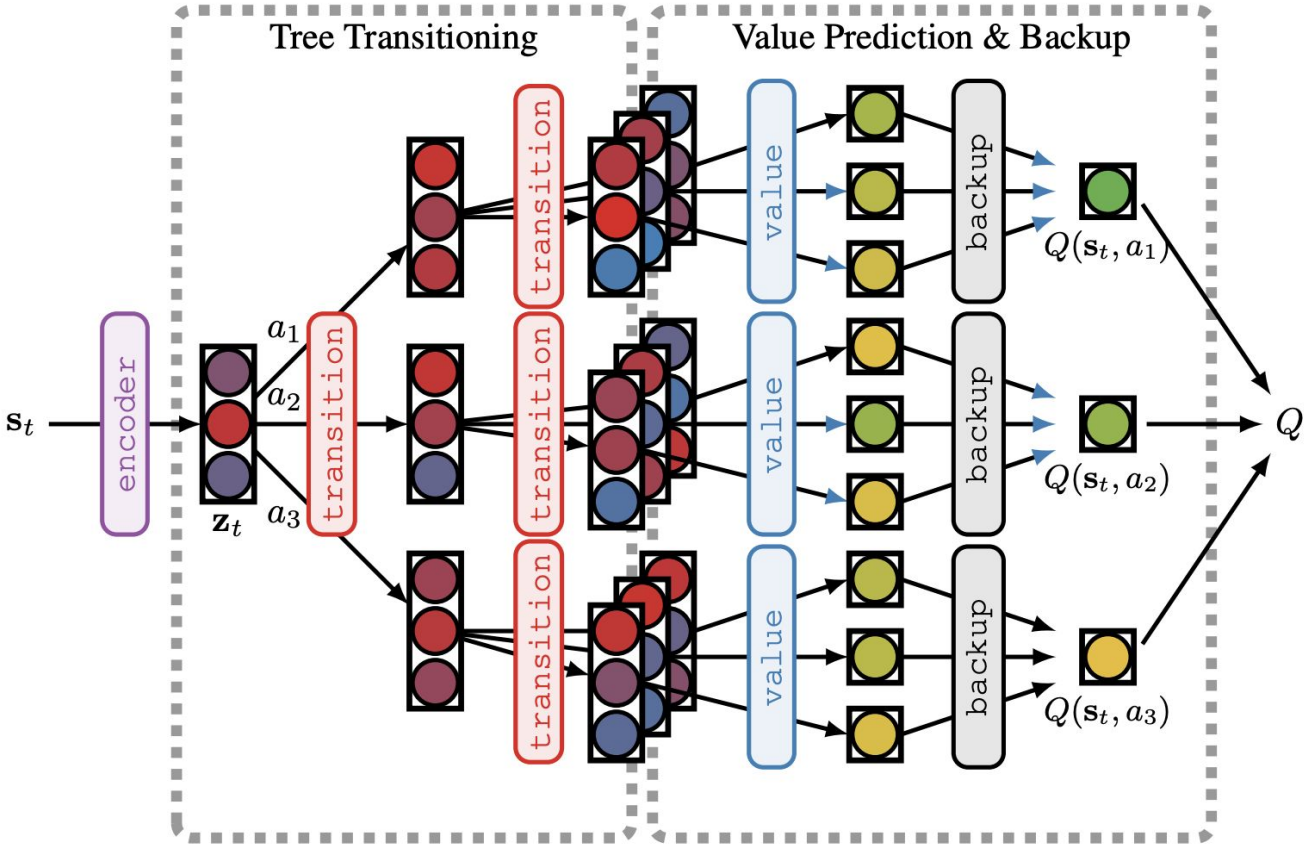
TreeQN / ATreeC

- Assume that we have reward/value models, giving us scalar **values** in every expanded node
- We can now **directly** apply a VI-style update rule!

$$Q(\mathbf{z}_{l|t}, a_i) = r(\mathbf{z}_{l|t}, a_i) + \begin{cases} \gamma V(\mathbf{z}^{a_i}_{d|t}) & l = d-1 \\ \gamma \max_{a_j} Q(\mathbf{z}^{a_i}_{l+1|t}, a_j) & l < d-1 \end{cases}$$

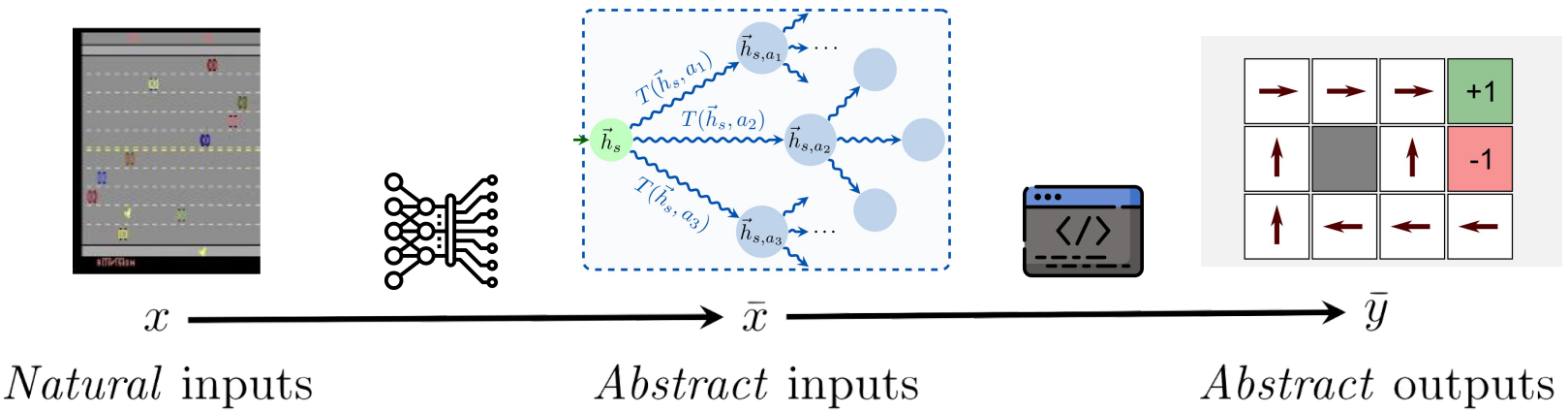
- Can then use the computed Q-values **directly** to decide the policy
- Exactly as leveraged by models like TreeQN / ATreeC (Farquhar *et al.*, ICLR'18)
  - Also related: Value Prediction Networks (Oh *et al.*, NeurIPS'17)

TreeQN / ATreeC in action



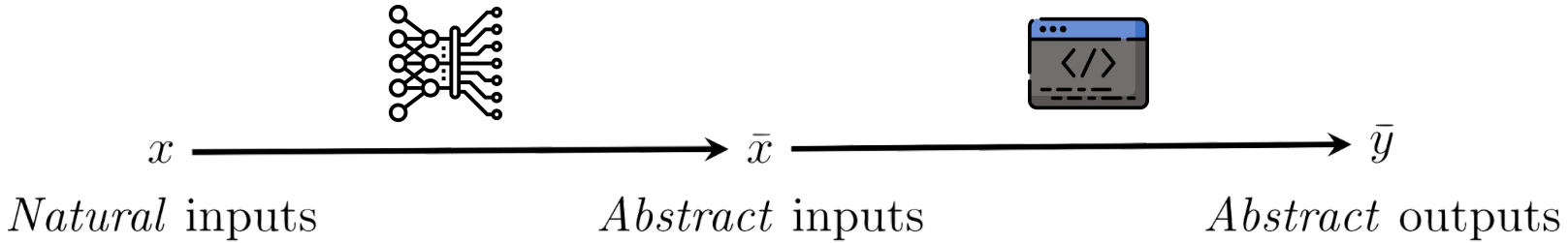
High-level view

- It's good to take a recap and realise what we have done so far



High-level view

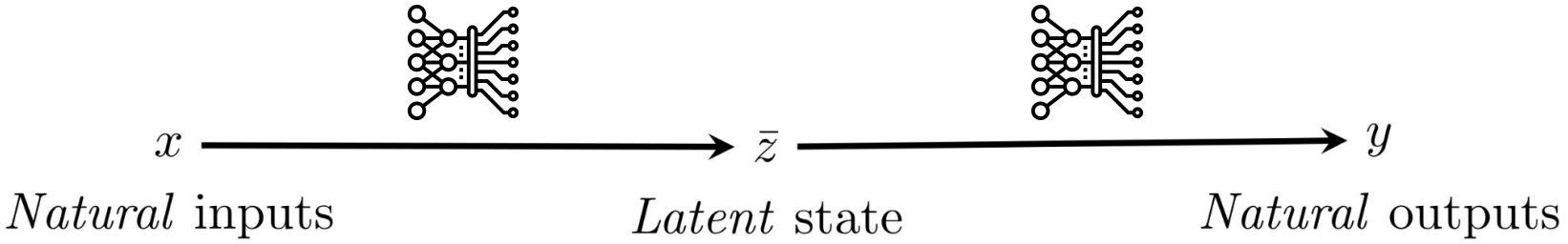
- It's good to take a recap and realise what we have done so far
  - We mapped our **natural** inputs (e.g. pixels) to the space of abstract inputs
  - (local MDP + reward values in every node)
  - This allowed us to execute VI-style algorithms **directly** on the abstract inputs



- The VI update is differentiable, and hence so is our entire implicit planner.

Breaking the bottleneck

- We hit bottleneck-based ***data efficiency*** issues again!
  - If there are insufficient training data to properly estimate the scalars…
  - Algorithm will give a **perfect** solution, but in a ***suboptimal*** environment



- To *break the bottleneck*, we replace the VI update with a **neural network**!

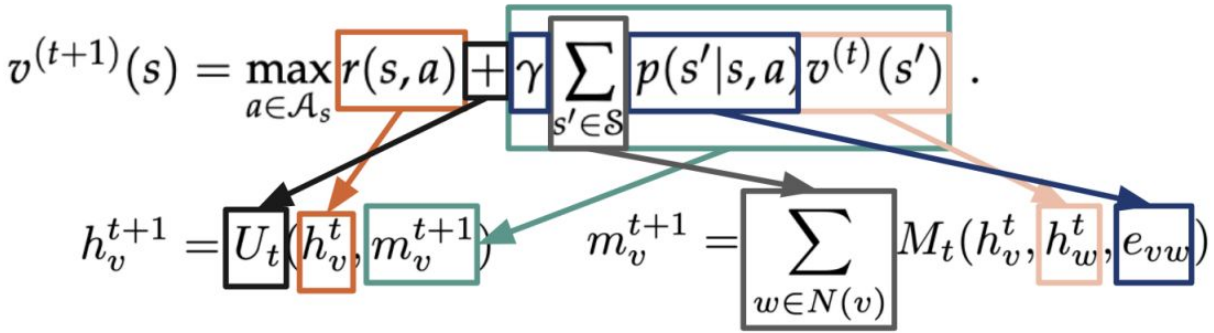
- As before, we can use **graph neural networks** to perform VI-aligning computations.

Algorithmic reasoning

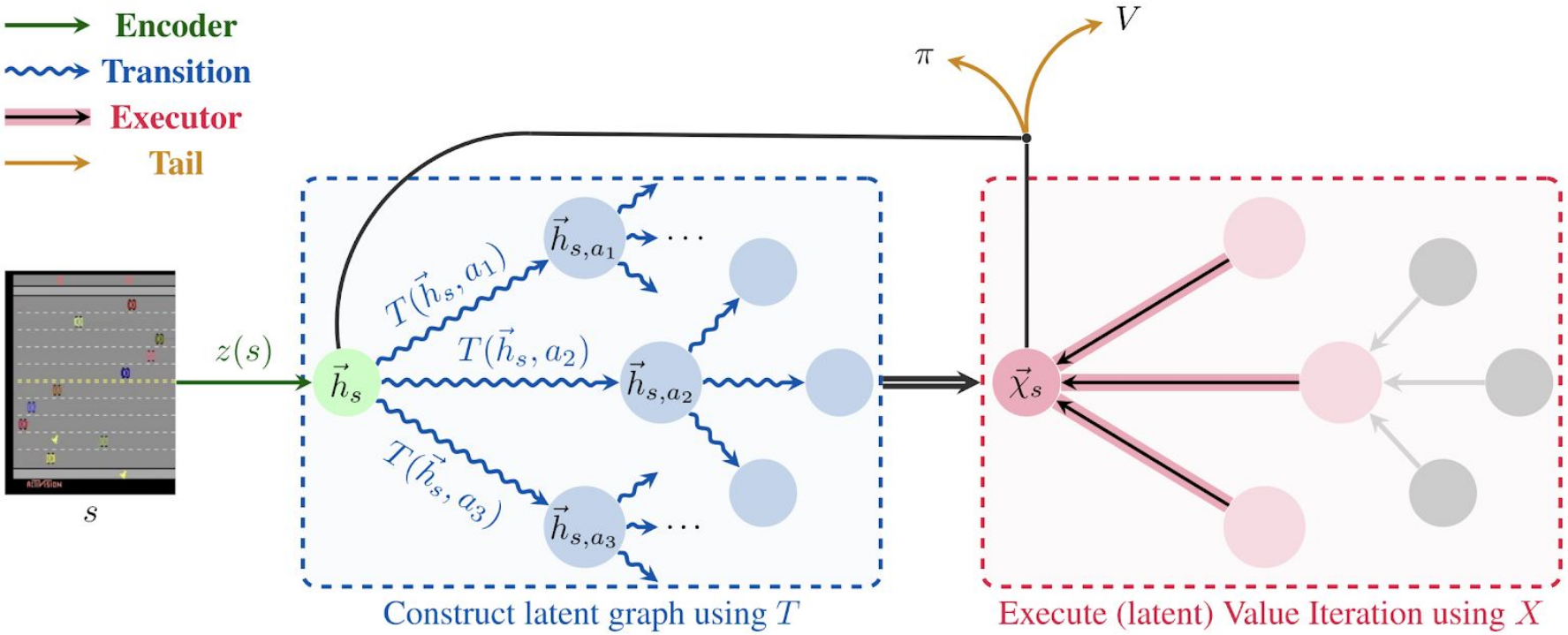
- GNN over state representations *aligns* with VI, but may put **pressure** on the planner
  - Same gradients used to *construct* correct graphs **and** make VI computations

- To alleviate this issue, we choose to **pre-train** the GNN to perform value iteration-style computations (over many **synthetic** MDPs), then deploying it within our planner

- This exploits, once again, the concept of *algorithmic alignment* (Xu *et al.*, ICLR'20)



Putting it all together!



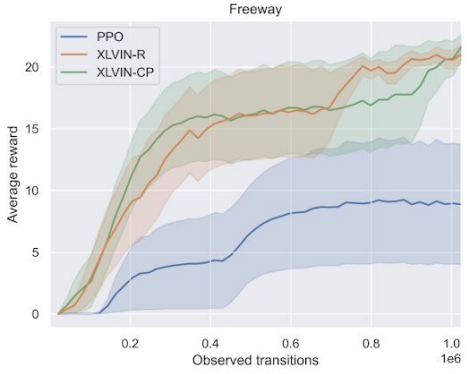
XLVIN (Deac *et al.*, NeurIPS'20 DeepRL)

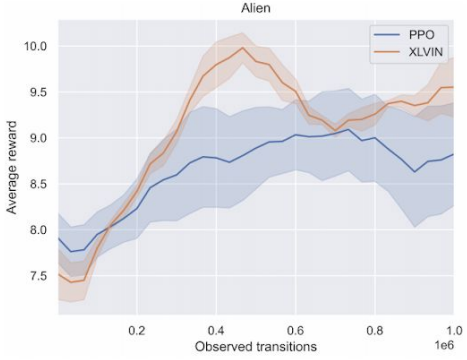
XLVIN Components

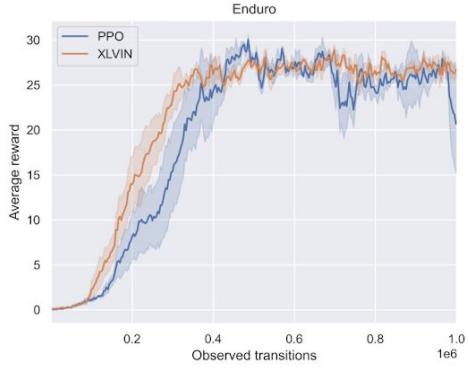
- **Encoder** (**z**: $S \to \mathbb{R}^k$) provides state representations
- **Transition** (**T**: $\mathbb{R}^k \times A \to \mathbb{R}^k$) simulates effects of actions in *latent* space
  - **Pre-trained & Fine-tuned** on the TransE loss (observed trajectories)
- **Executor** (**X**: $\mathbb{R}^k \times \mathbb{R}^{|A| \times k} \to \mathbb{R}^k$) simulates a planning algorithm (Value Iteration) in *latent* space
  - **Pre-trained** to execute VI on synthetic MDPs of interest, then **frozen**
- **Policy / Value Head**, computing action probabilities and state-values given embeddings
  - Use **PPO** as the policy gradient method

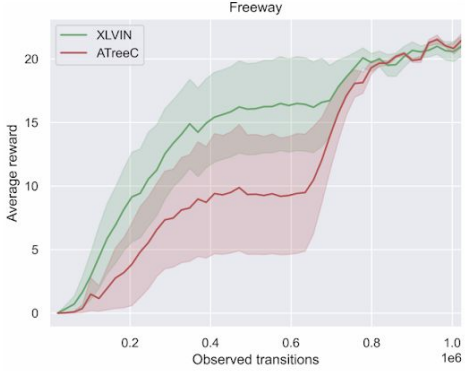
The entire procedure is end-to-end differentiable, does not impose any assumptions on the structure of the underlying MDP, and has the capacity to perform computations directly aligned with value iteration. Hence our model can be considered as a generalisation of VIN-like methods to settings where the MDP is not provided or otherwise difficult to obtain.

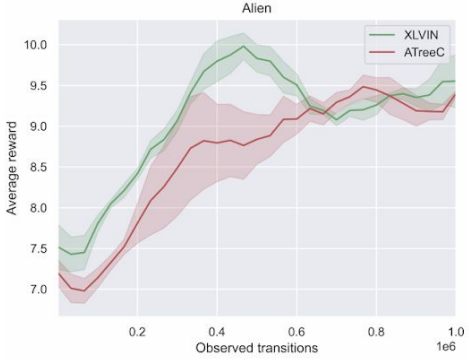
Results on *low-data* Atari

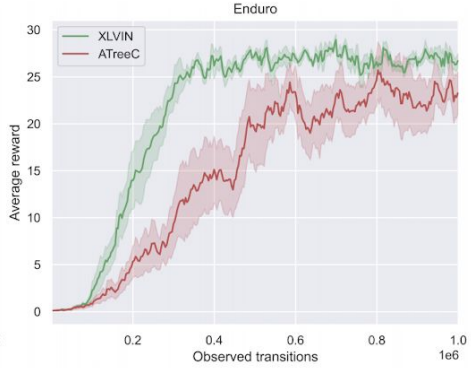












...why did it work?

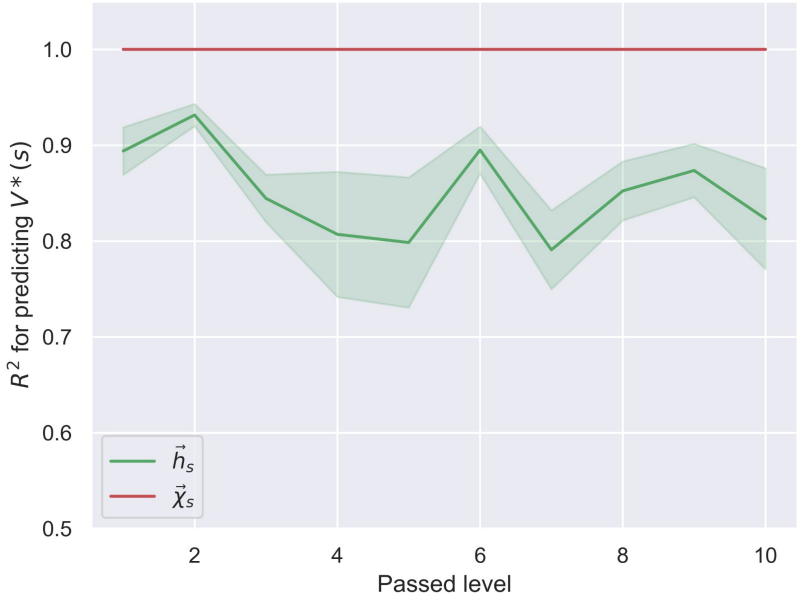
- Recall, our executor network was pre-trained and **frozen**
- The pixel-level encoder needed to learn to map **rich** inputs into the executor's latent space
  - Analogous to a human who tries to map real-world problems to algorithmic inputs!
- We set out to investigate to what extent it succeeded.

Grid-world qualitative study

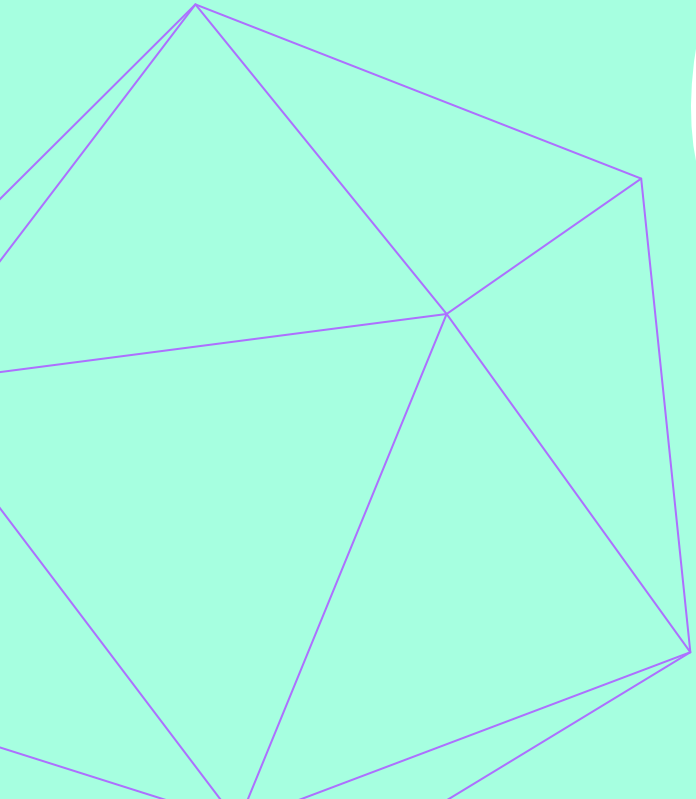
- We evaluate the quality of the embeddings **before** and **after** applying the executor, in a *grid-world* environment
  - Here we can compute optimal V*(s)
  - Evaluate linear decodability by linear regression!

- Results verify our hypothesis!
  - Input values are already predictive
  - But the executor consistently **refines** them!

- Our encoder learnt to correctly *map* the input to the latent algorithm! :)



DeepMind



(Have we answered Question 3?)

- Do algorithmic neural networks actually **work** when deployed?
  - If so, how are they *actually* being used?

DeepMind

7 Summary and conclusions

Overview, *revisited*

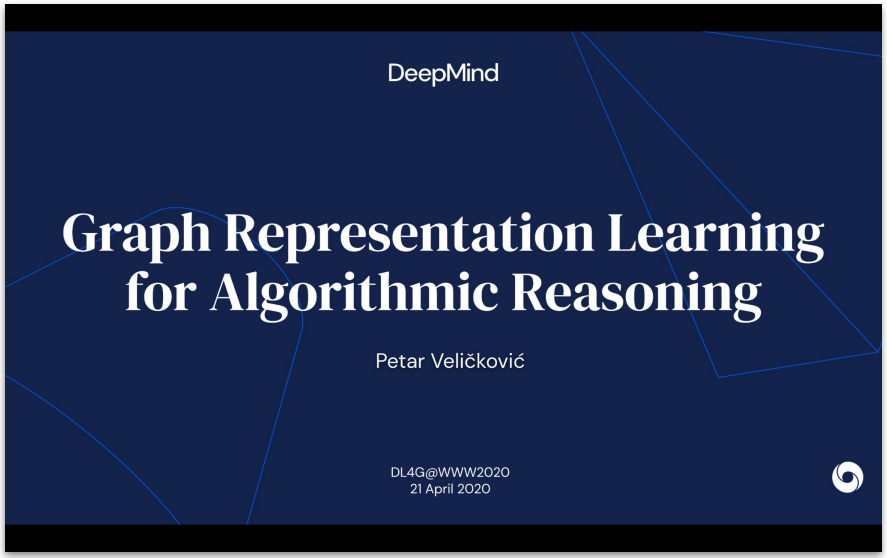
Our aim ~~is~~ was to address **three** key questions: (roughly ~20min for each)

- Why should we, as deep learning practitioners, study **algorithms**?
  - Further, why might it be beneficial to make *'algorithm-inspired'* neural networks?
- How to **build** neural networks that behave algorithmically?
  - And why am I even telling you this in a *"Graph Machine Learning"* course?
- Do algorithmic neural networks actually **work** when deployed?
  - If so, how are they *actually* being used?

Hopefully, also some ideas on ***where*** *you might be able to* ***apply*** the ideas above :)

Further insight: Algorithmic reasoning

If you would like to know more details about constructing good processor networks:



https://www.youtube.com/watch?v=IPQ6CPoluok

https://drive.google.com/file/d/1_EQ9Yu7VEkvrHaVHl_WbT5ABvxrSNY-s/view?usp=sharing

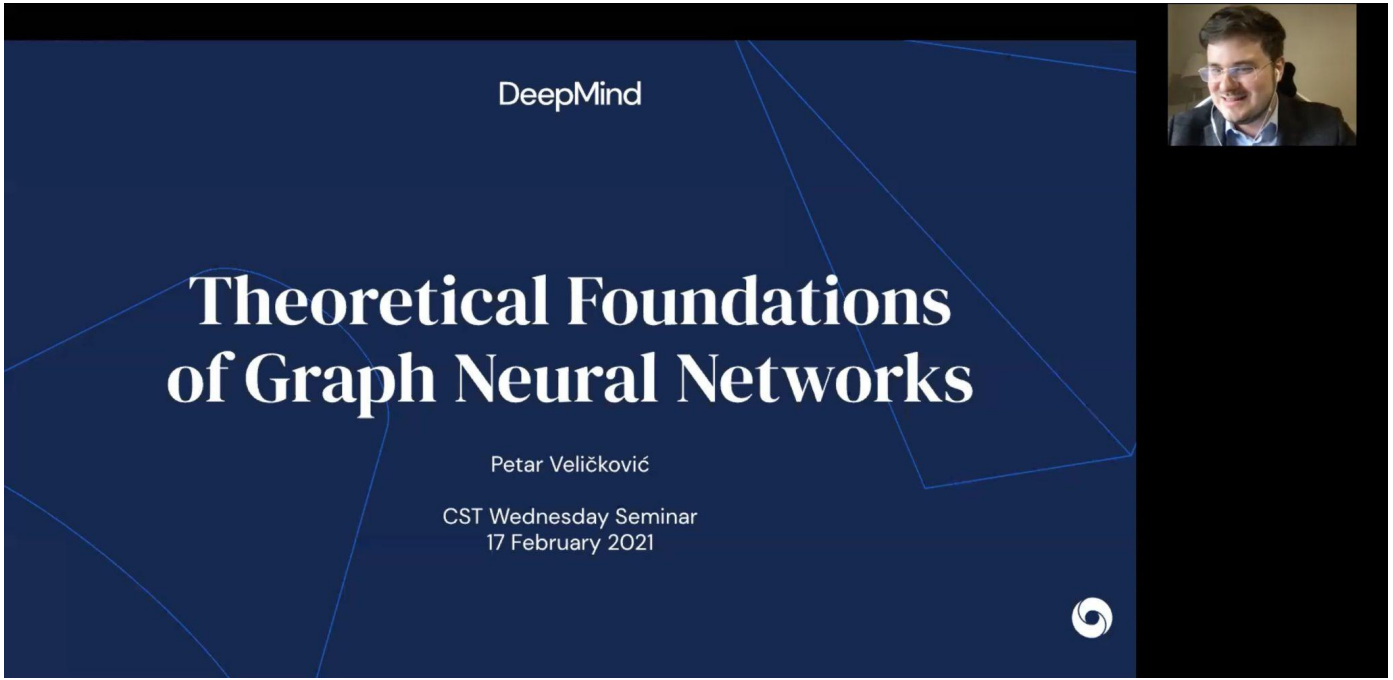
Further insight: Algorithmic implicit planning

If you would like to know more details about implicit planning and XLVIN:

https://www.youtube.com/watch?v=mGw9ewL8wCU

Further insight: graph representation learning

If GNNs are new(ish) to you, I recently gave a useful talk on **theoretical GNN foundations**:
https://www.youtube.com/watch?v=uF53xsT7mjc



Want to know more?

Combinatorial optimization and reasoning with graph neural networks

Quentin Cappart[1], Didier Chételat[2], Elias Khalil[3], Andrea Lodi[2], Christopher Morris[2], and Petar Veličković[*4]

[1]Department of Computer Engineering and Software Engineering, Polytechnique Montréal
[2]CERC in Data Science for Real-Time Decision-Making, Polytechnique Montréal
[3]Department of Mechanical & Industrial Engineering, University of Toronto
[4]DeepMind

Combinatorial optimization is a well-established area in operations research and computer science. Until recently, its methods have focused on solving problem instances in isolation, ignoring the fact that they often stem from related data distributions in practice. However, recent years have seen a surge of interest in using machine learning, especially graph neural networks (GNNs), as a key building block for combinatorial tasks, either as solvers or as helper functions. GNNs are an inductive bias that effectively encodes combinatorial and relational input due to their permutation-invariance and sparsity awareness. This paper presents a conceptual review of recent key advancements in this emerging field, aiming at both the optimization and machine learning researcher.

Our *43-page* survey on GNNs for CO!

https://arxiv.org/abs/2102.09544

Section 3.3. details algorithmic reasoning, with comprehensive references.

DeepMind

Thank you!

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

In collaboration with Charles Blundell, Raia Hadsell, Rex Ying, Matilde Padovano, Andreea Deac, Ognjen Milinković, Pierre-Luc Bacon, Jian Tang, Mladen Nikolić, Christopher Morris, Quentin Cappart, Elias Khalil, Didier Chételat, Andrea Lodi, Lovro Vrček, Mile Šikić, Lars Buesing, Matt Overlan, Razvan Pascanu and Oriol Vinyals