DeepMind

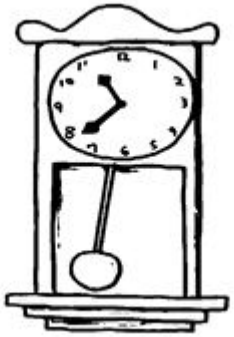
Neural Algorithmic Sketching

Petar Veličković

SketchDL Workshop
CVPR 2021

19 June 2021

DeepMind



In this talk:
Sketching

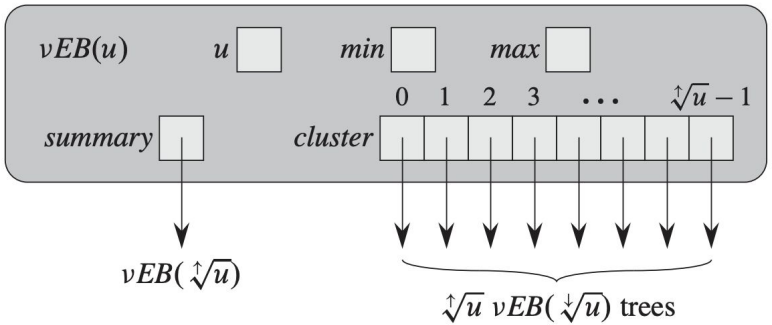
DeepMind

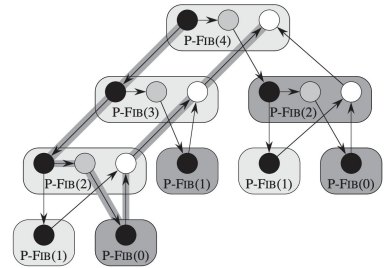
Also in this talk: (Classical) **Algorithms**

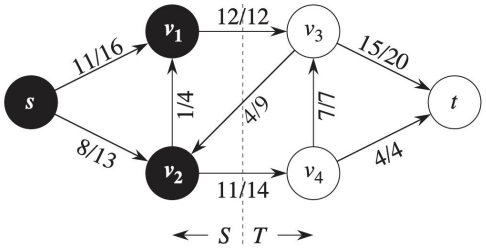
DeepMind

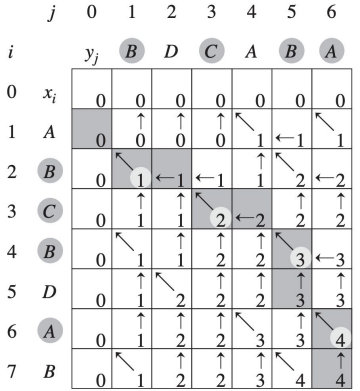
Also in this talk: (Classical) Algorithms

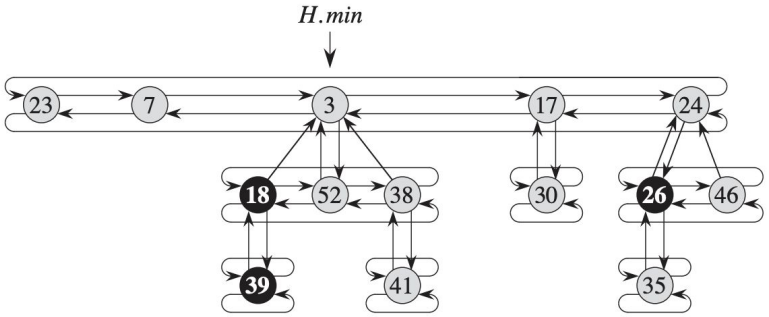
```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

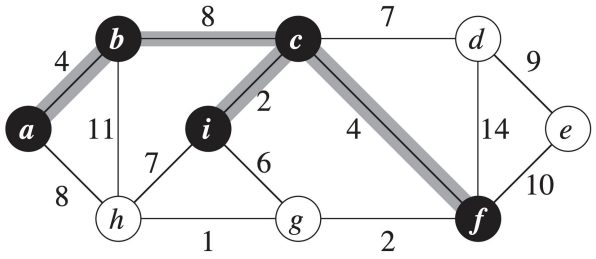


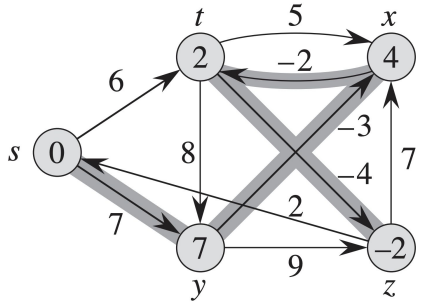












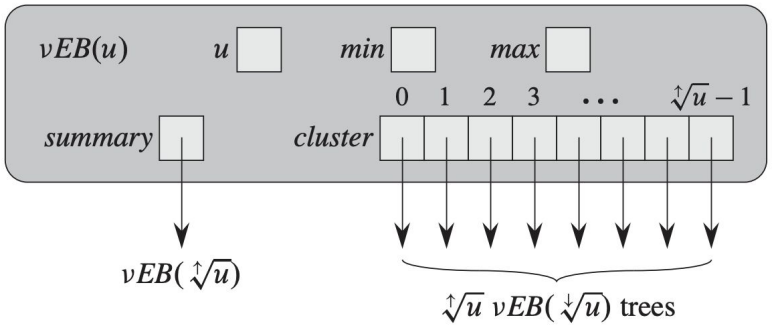
Algorithm figures: Cormen, Leiserson, Rivest and Stein. **Introduction to Algorithms.**

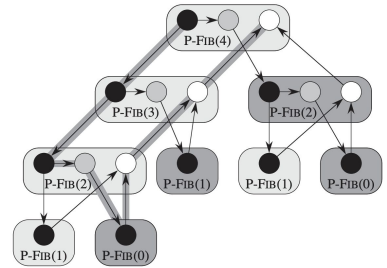
DeepMind

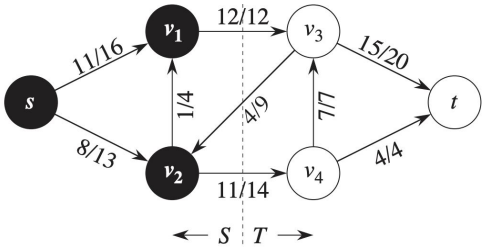
Also in this talk: (Classical) Algorithms

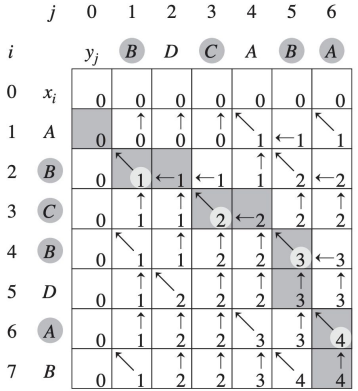
(with a bit of neural spice)

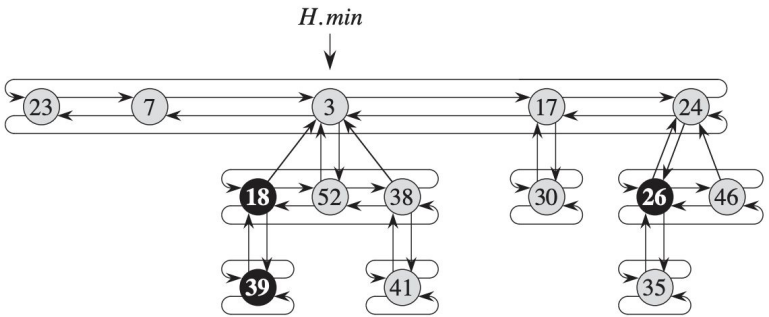
```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

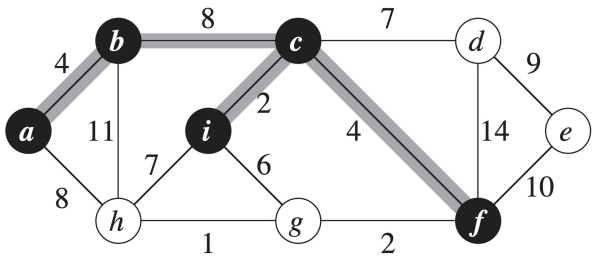


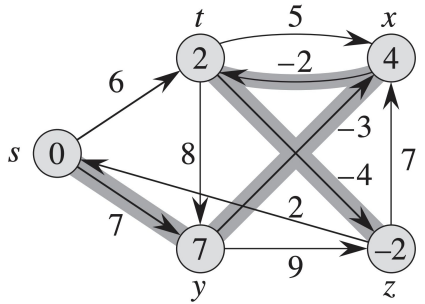












Algorithm figures: Cormen, Leiserson, Rivest and Stein. **Introduction to Algorithms.**

DeepMind

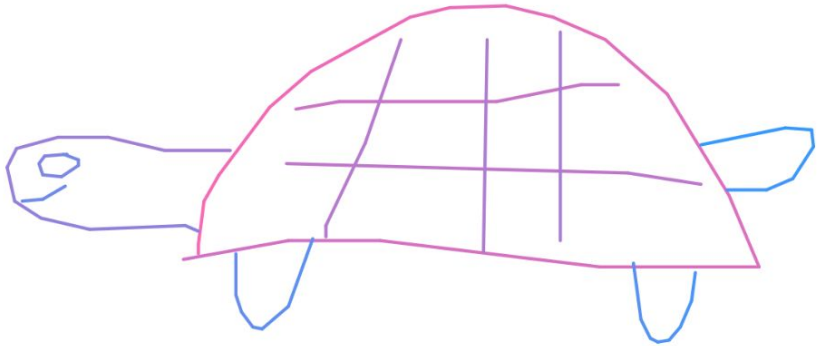
1 Sketches and algorithms?

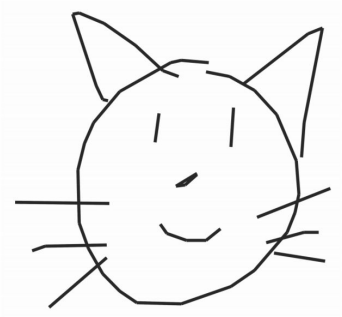
Sketches can be seen (roughly) analogous to graphs...

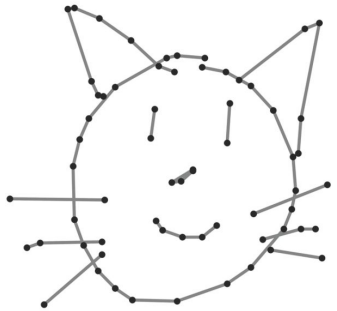
```
[ 0,   -5,  1,  0,  0 ]
[ 3,   -23, 1,  0,  0 ]
[ 8,   -14, 1,  0,  0 ]
[ 27,  -36, 1,  0,  0 ]
[ 22,  -19, 1,  0,  0 ]
[ 53,  -29, 1,  0,  0 ]
[ 21,  -5,  1,  0,  0 ]
[ 30,  -1,  1,  0,  0 ]
[ 25,  6,   1,  0,  0 ]
[ 28,  12,  1,  0,  0 ]
[ 33,  29,  1,  0,  0 ]
[ 33,  54,  1,  0,  0 ]
[ 16,  38,  1,  0,  0 ]
[ -85, 0,   1,  0,  0 ]
[ -117, -14, 1, 0,  0 ]
[ -49, 0,   1,  0,  0 ]
[ -56, 10,  0,  1,  0 ]
[ 60,  -80, 1,  0,  0 ]
[ 23,  -4,  1,  0,  0 ]
[ 84,  0,   1,  0,  0 ]
[ 45,  -9,  1,  0,  0 ]
[ 20,  0,   0,  1,  0 ]
[ -177, 42, 1,  0,  0 ]
[ 182, 5,   1,  0,  0 ]
[ 39,  6,   0,  1,  0 ]
[ -160, -77, 1, 0,  0 ]
```

```
[ -19, 55,  1,  0,  0 ]
[ -21, 44,  1,  0,  0 ]
[ 0,   6,   0,  1,  0 ]
[ 86,  -105, 1, 0,  0 ]
[ -2,  113, 0,  1,  0 ]
[ 41,  -117, 1, 0,  0 ]
[ 0,   111, 0,  1,  0 ]
[ -176, -48, 1, 0,  0 ]
[ -35, 0,   1,  0,  0 ]
[ -30, -7,  1,  0,  0 ]
[ -27, 0,   1,  0,  0 ]
[ -22, 6,   1,  0,  0 ]
[ -5,  10,  1,  0,  0 ]
[ 4,   18,  1,  0,  0 ]
[ 14,  9,   1,  0,  0 ]
[ 26,  6,   1,  0,  0 ]
[ 51,  -2,  1,  0,  0 ]
[ 7,   3,   0,  1,  0 ]
[ -70, -41, 1,  0,  0 ]
[ -12, 1,   1,  0,  0 ]
[ -3,  4,   1,  0,  0 ]
[ 2,   6,   1,  0,  0 ]
[ 10,  1,   1,  0,  0 ]
[ 9,   -6,  1,  0,  0 ]
[ 0,   -3,  1,  0,  0 ]
[ -8,  -3,  0,  1,  0 ]
```

```
[ -22, 25,  1,  0,  0 ]
[ 11,  -1,  1,  0,  0 ]
[ 12,  -7,  0,  1,  0 ]
[ 91,  36,  1,  0,  0 ]
[ 0,   22,  1,  0,  0 ]
[ 3,   9,   1,  0,  0 ]
[ 6,   8,   1,  0,  0 ]
[ 5,   1,   1,  0,  0 ]
[ 14,  -12, 1,  0,  0 ]
[ 13,  -36, 0,  1,  0 ]
[ 171, 13,  1,  0,  0 ]
[ 4,   30,  1,  0,  0 ]
[ 5,   10,  1,  0,  0 ]
[ 4,   2,   1,  0,  0 ]
[ 6,   -1,  1,  0,  0 ]
[ 6,   -7,  1,  0,  0 ]
[ 6,   -14, 1,  0,  0 ]
[ 2,   -15, 0,  1,  0 ]
[ 17,  -44, 1,  0,  0 ]
[ 21,  0,   1,  0,  0 ]
[ 14,  -6,  1,  0,  0 ]
[ 11,  -17, 1,  0,  0 ]
[ -1,  -9,  1,  0,  0 ]
[ -14, -1,  1,  0,  0 ]
[ -45, 9,   0,  1,  0 ]
[ 0,   0,   0,  0,  1 ]
```







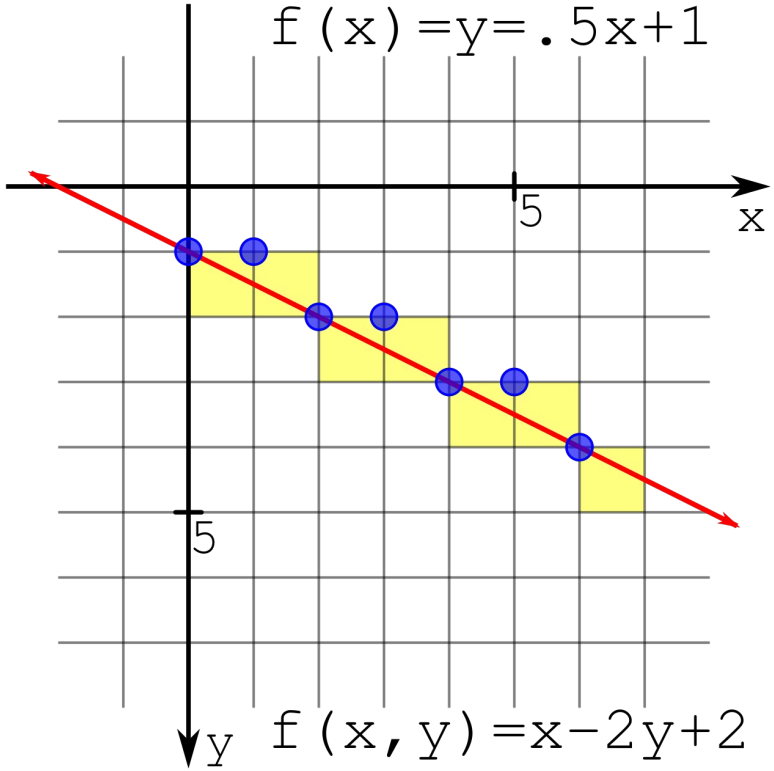
(a) original sketch

(b) 1-hop connected

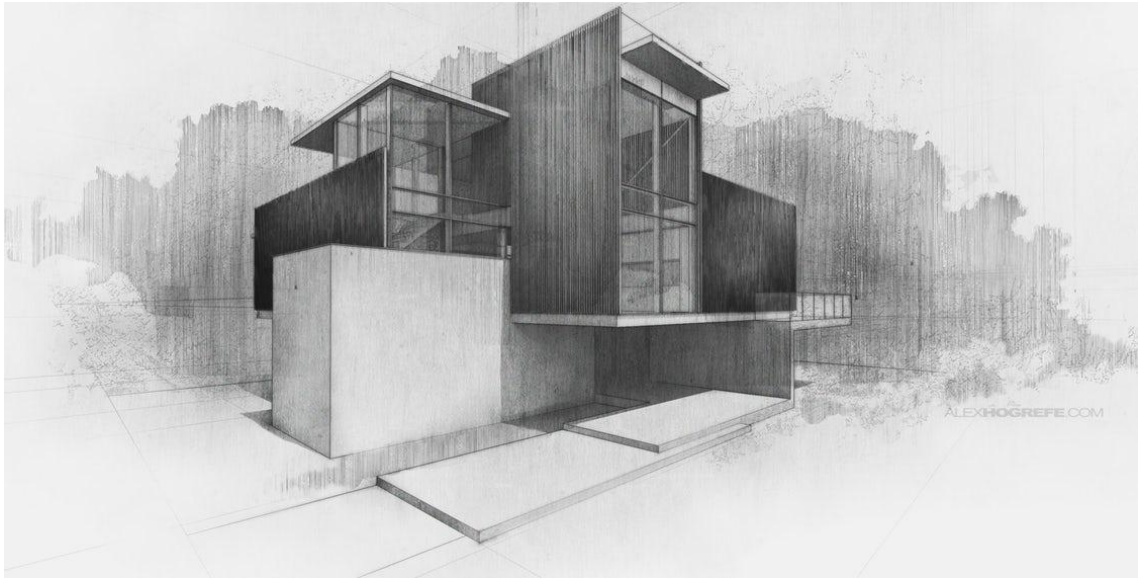
Visualising them requires a renderer! (algorithm :))

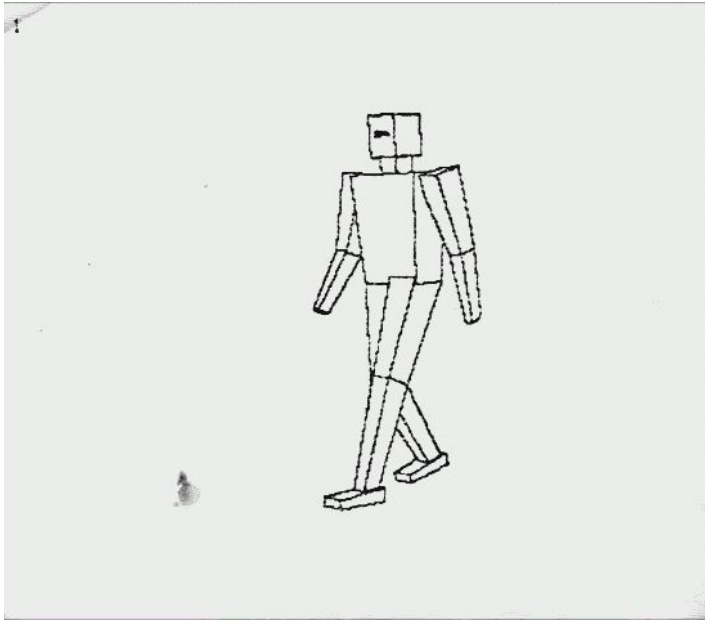
```
plotLine(x0, y0, x1, y1)
     dx = x1 - x0
     dy = y1 - y0
     D = 2*dy - dx
     y = y0

     for x from x0 to x1
          plot(x,y)
          if D > 0
               y = y + 1
               D = D - 2*dx
          end if
          D = D + 2*dy
```



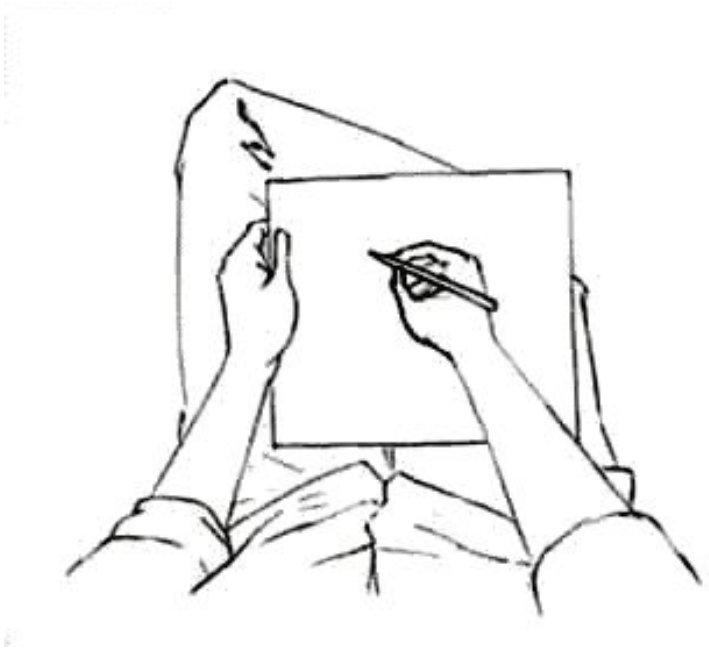
Rendering algorithms can be way more complex than this





The very process of sketching is often *algorithmic*

- Hierarchical
- Regular
- Sequential
- Sketching -> insight on generative factors!
- Taken to the extreme, we could get...



Algorithms enable **complex** sketching from **simple** patterns

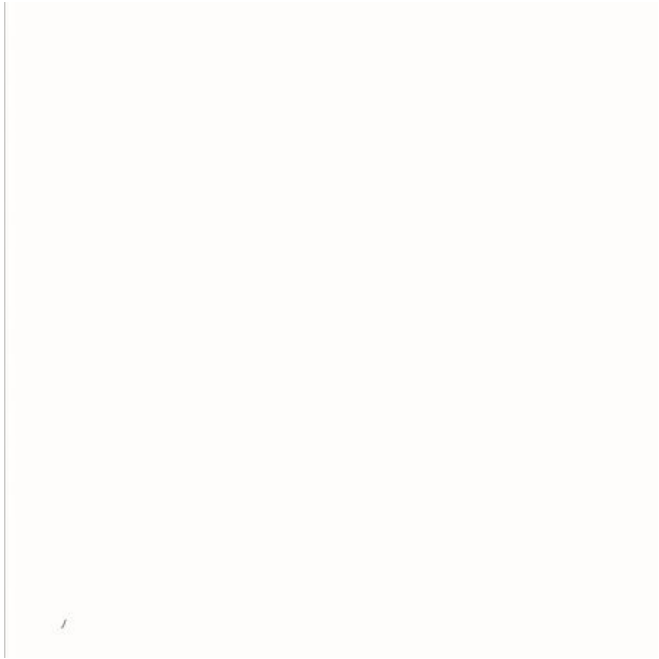
- L-systems allow for sketching complex, hierarchical shapes as rollouts of a CFG
  - **Very** amenable to algorithmic computation, e.g. parsing

$$f_1(x,y) = \begin{bmatrix} 0.00 & 0.00 \\ 0.00 & 0.16 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$f_2(x,y) = \begin{bmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.00 \\ 1.60 \end{bmatrix}$$

$$f_3(x,y) = \begin{bmatrix} 0.20 & -0.26 \\ 0.23 & 0.22 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.00 \\ 1.60 \end{bmatrix}$$

$$f_4(x,y) = \begin{bmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.00 \\ 0.44 \end{bmatrix}$$



Our aim for today

- Capture the “algorithmic” aspect of sketching within a (graph) neural network
- Illustrate neural algorithmic reasoning as one approach for doing so
  - We’ll use an algorithmic exposition for maximal rigour
- Outline some tasks that could benefit from this
  - Largely a **perspectives** talk!
  - Potential useful “playground”
- I should highlight I’m not active in sketch research!
  - Likely missing many rel. work
  - Hope to give interesting ideas!

DeepMind

2 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

“Fundamentals of a method for evaluating rail net capacities”

(Harris & Ross, 1955)

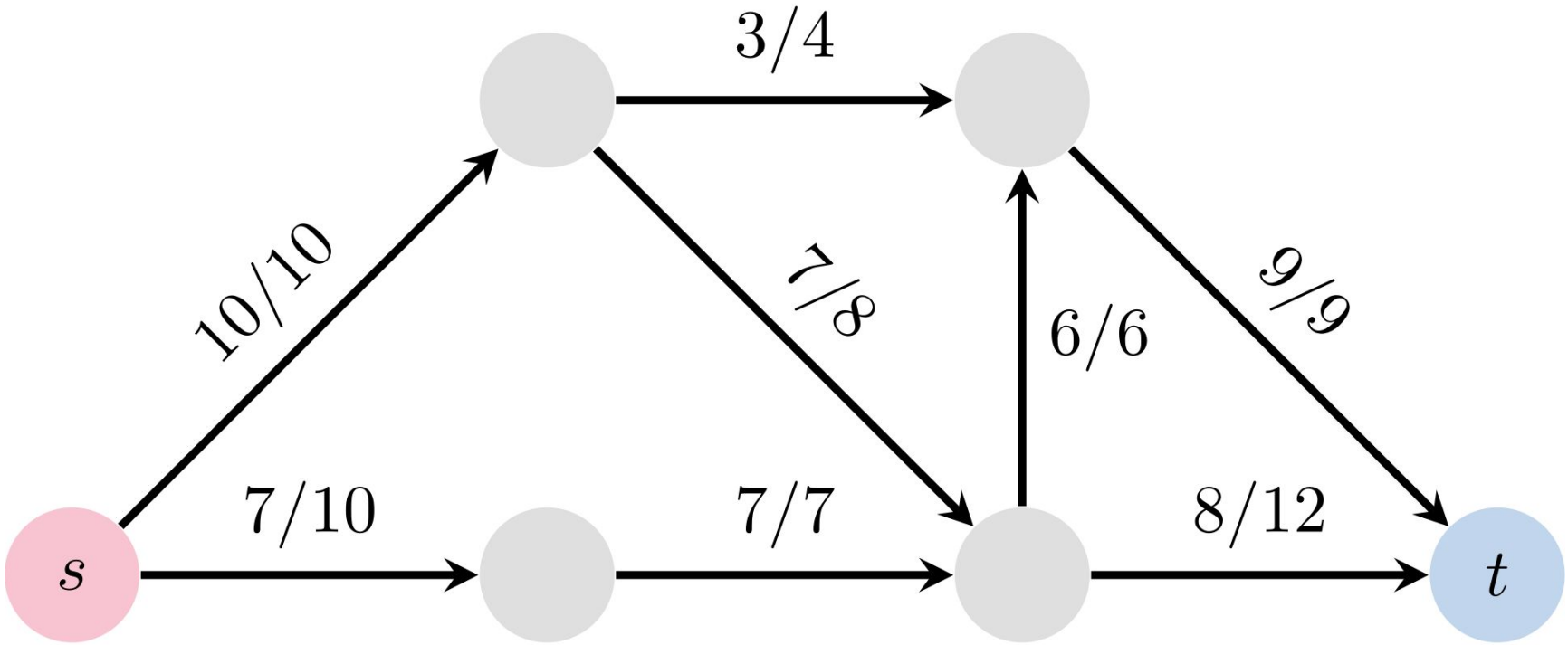
The core problem

- 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.

Max-flow example ($f = 17$)



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.

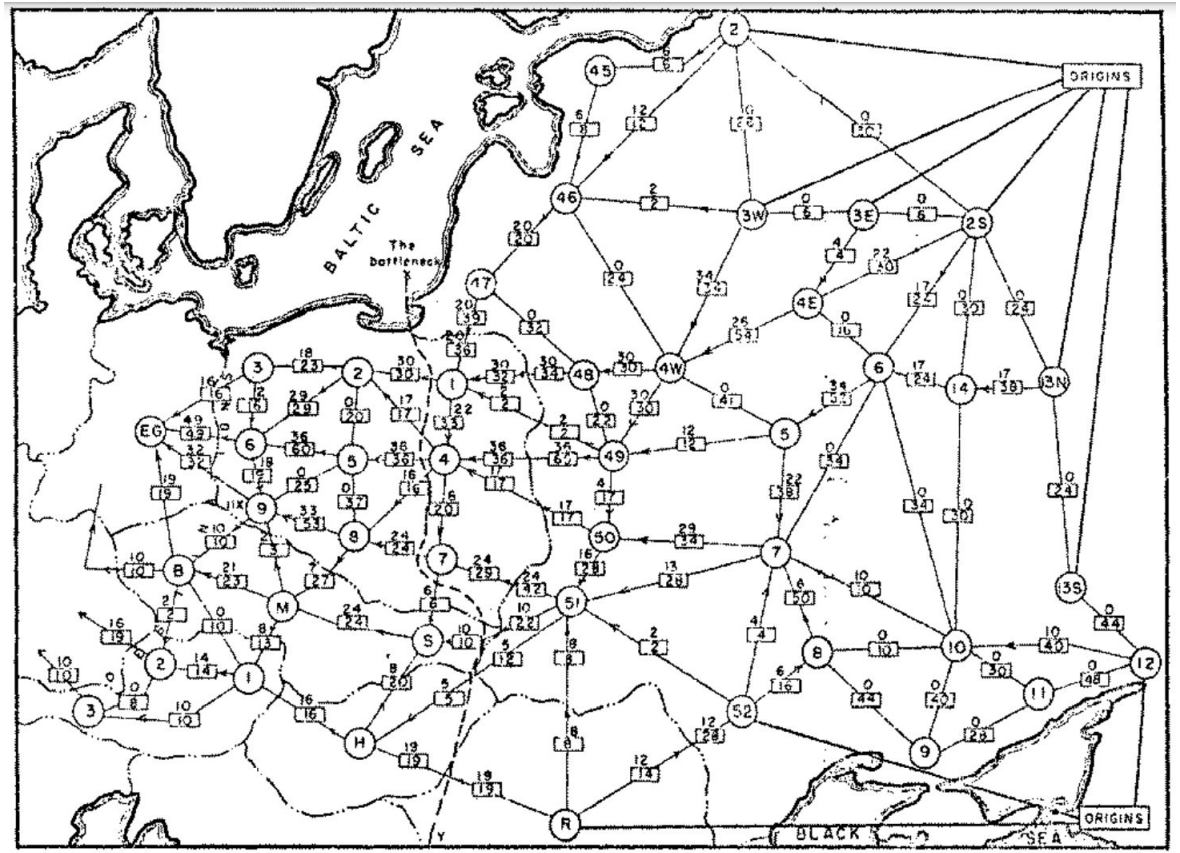
https://apps.dtic.mil/dtic/tr/fulltext/u2/093458.pdf

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

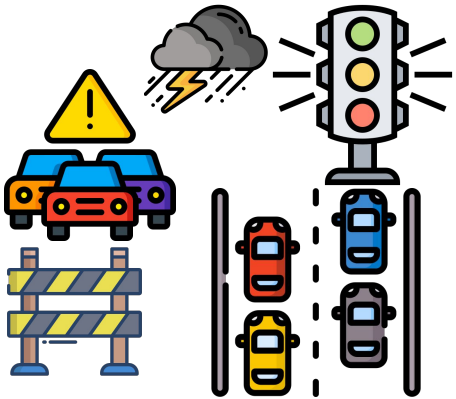
- The “core problem” plagues applications of classical combinatorial algorithms to this day!
- Satisfying their preconditions necessitates converting inputs into an **abstractified** form
- If done manually, this often implies *drastic* **information loss**
  - Combinatorial problem no longer accurately 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.

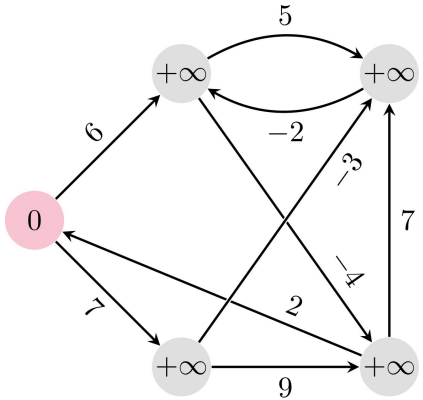
DeepMind

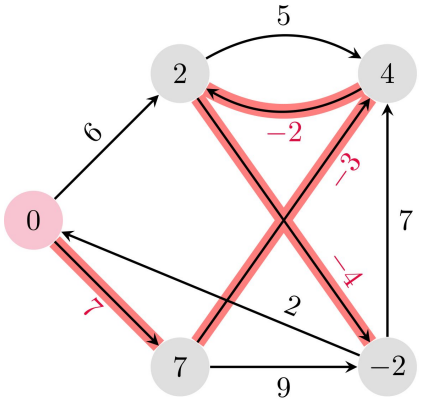
3 Towards a *neurally* spiced solution

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

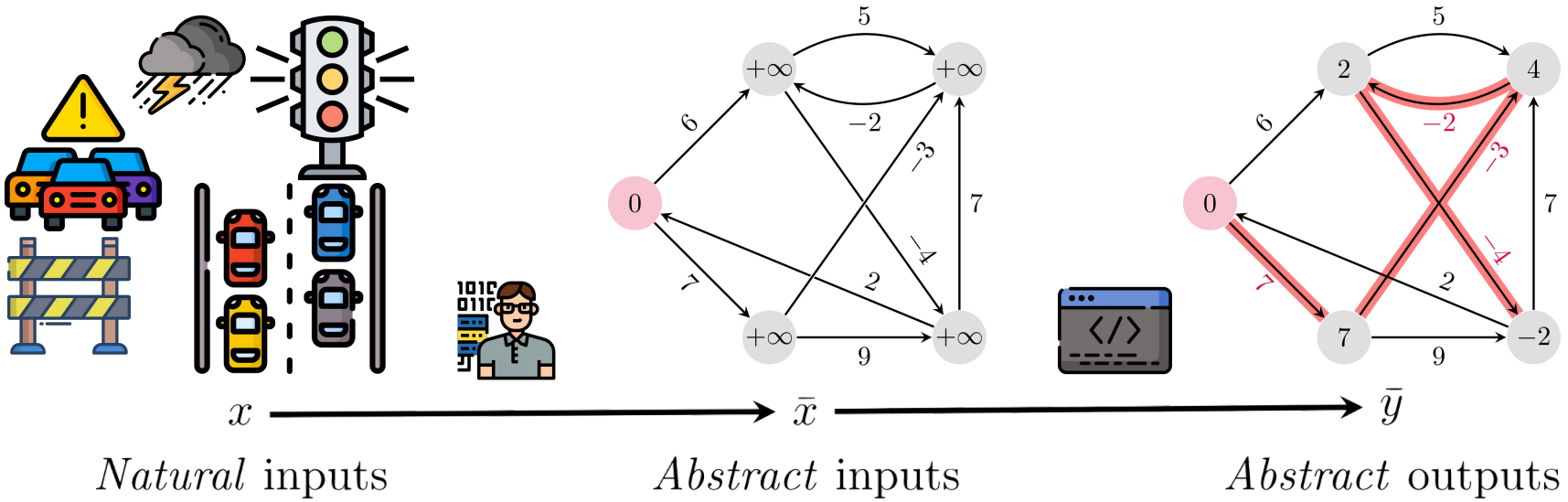






Abstractifying the core problem

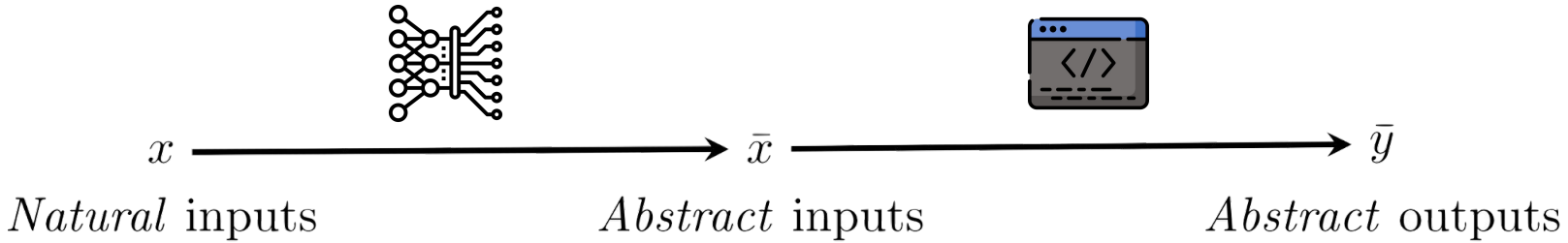
- Assume we have *real-world* inputs, but our algorithm only admits *abstract* inputs
  - For now, we assumed **manually** converting from one input to another



- 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.
  - Great solutions exist, however. See Vlastelica *et al.* (ICLR'20)

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**!

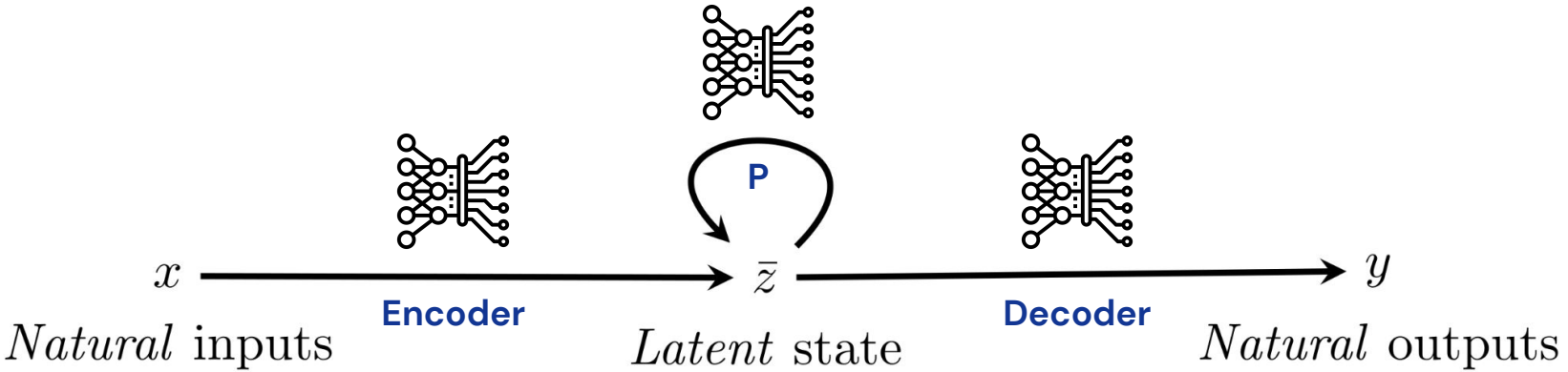
- If there are insufficient training data to properly estimate the scalars, we hit same issues!
  - Algorithm will give a **perfect** solution, but in a ***suboptimal*** environment

- Possible context for sketching -- low-data extraction of key points from an image?

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**!

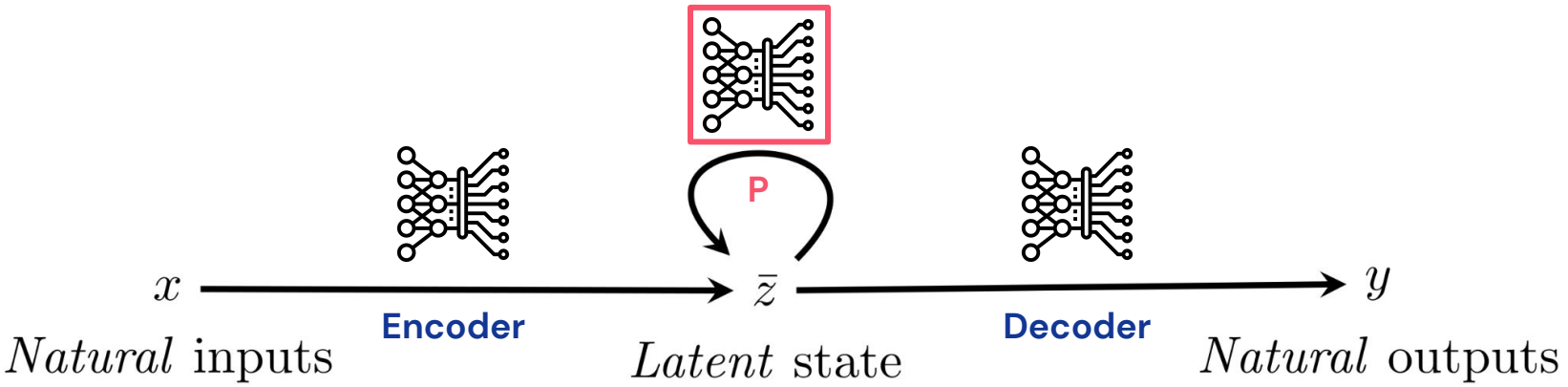


(The setting naturally aligns with *encode-**process**-decode* (Hamrick *et al.*, CSS'18))

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

4 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

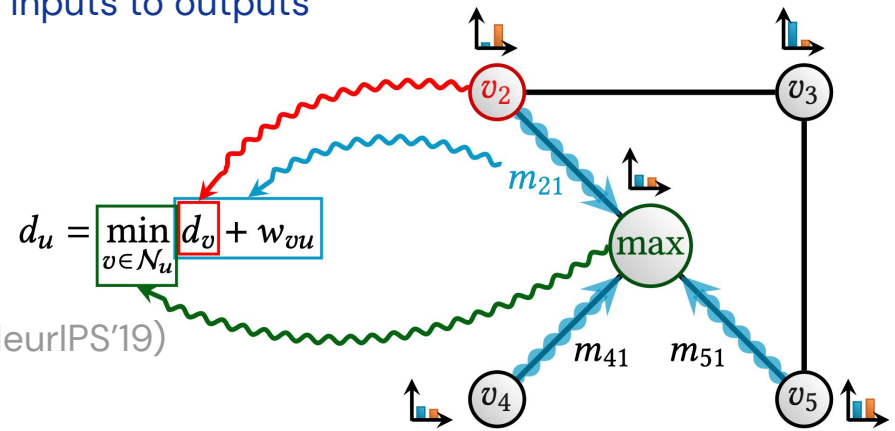
- This is a **very** active research area, with many key papers published only last year!

tl;dr of algorithmic reasoning

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

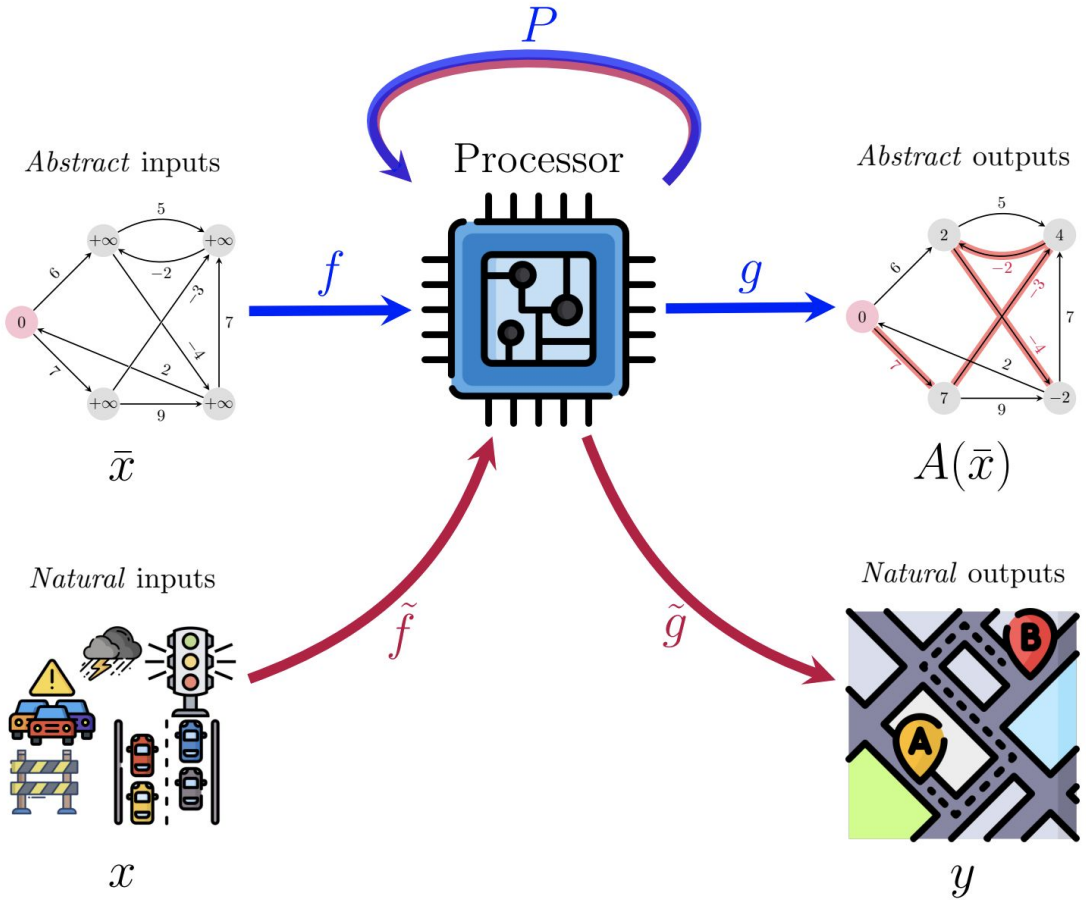
- Interesting **inductive biases** explored by Veličković *et al.* (ICLR'20):
  - Encode-***process***-decode from abstract inputs to outputs
  - Favour the **max** aggregation
  - **Strong** supervision on trajectories

- 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)

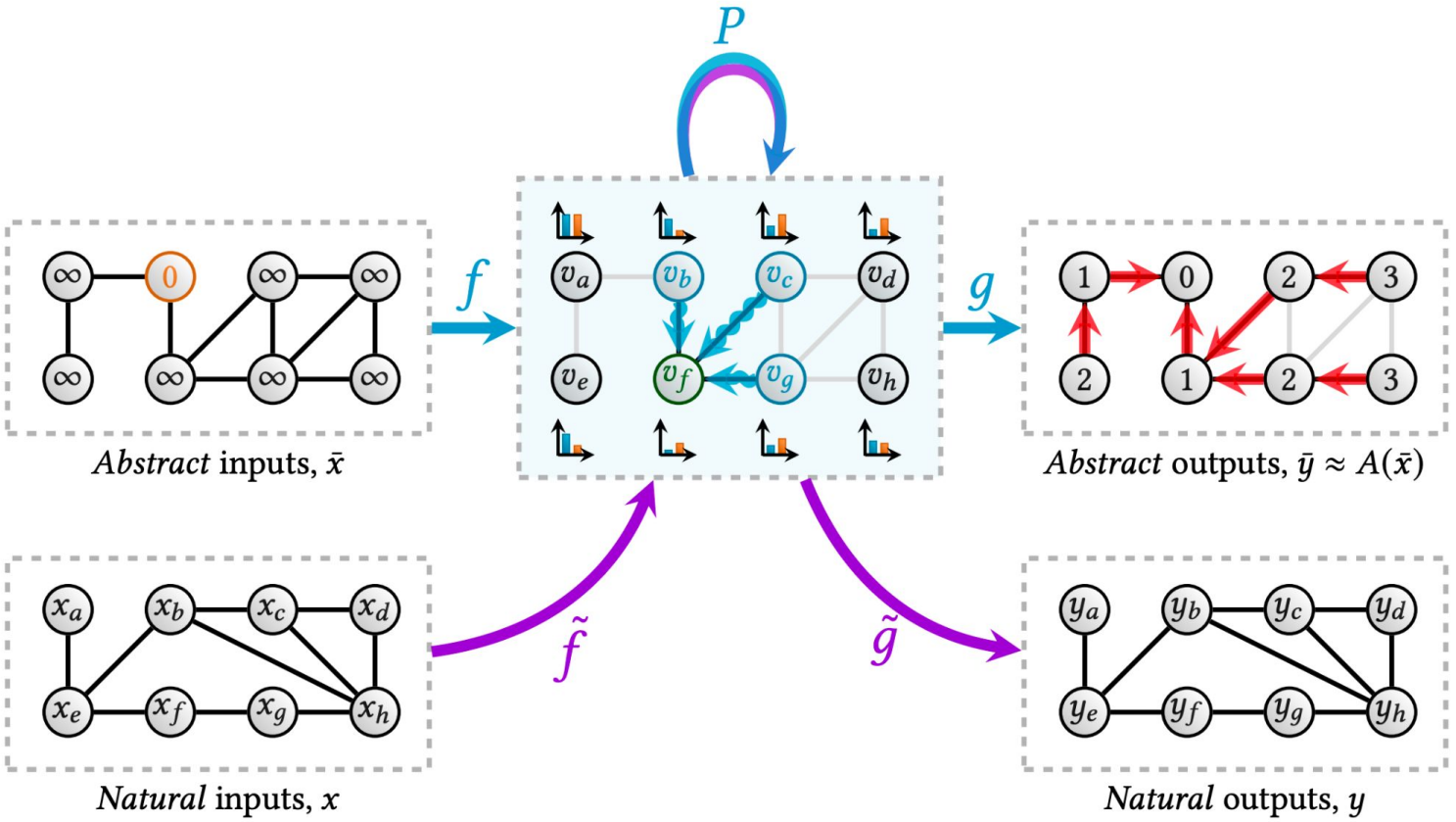


- Latest insights: ***linear*** *algorithmic alignment* is highly beneficial (Xu *et al.*, ICLR'21)

Blueprint of algorithmic reasoning



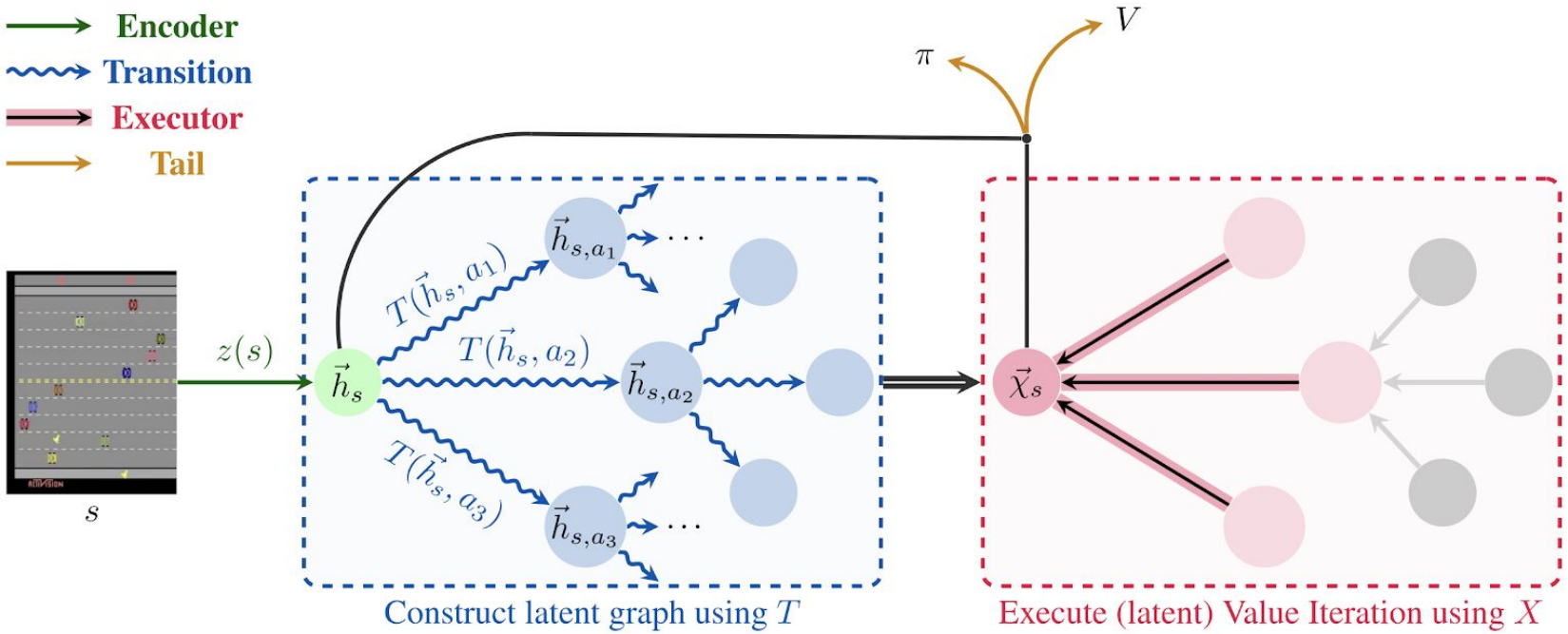
Blueprint of algorithmic reasoning, in-depth



We showed this works in RL

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

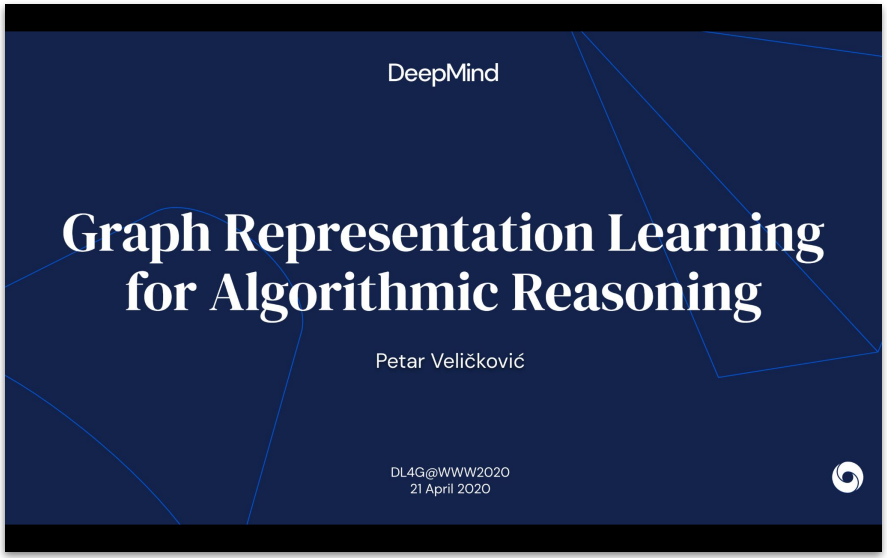
$$h_v^{t+1} = U_t(h_v^t, m_v^{t+1}) \qquad m_v^{t+1} = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw})$$



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

Further insight: Algorithmic reasoning

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

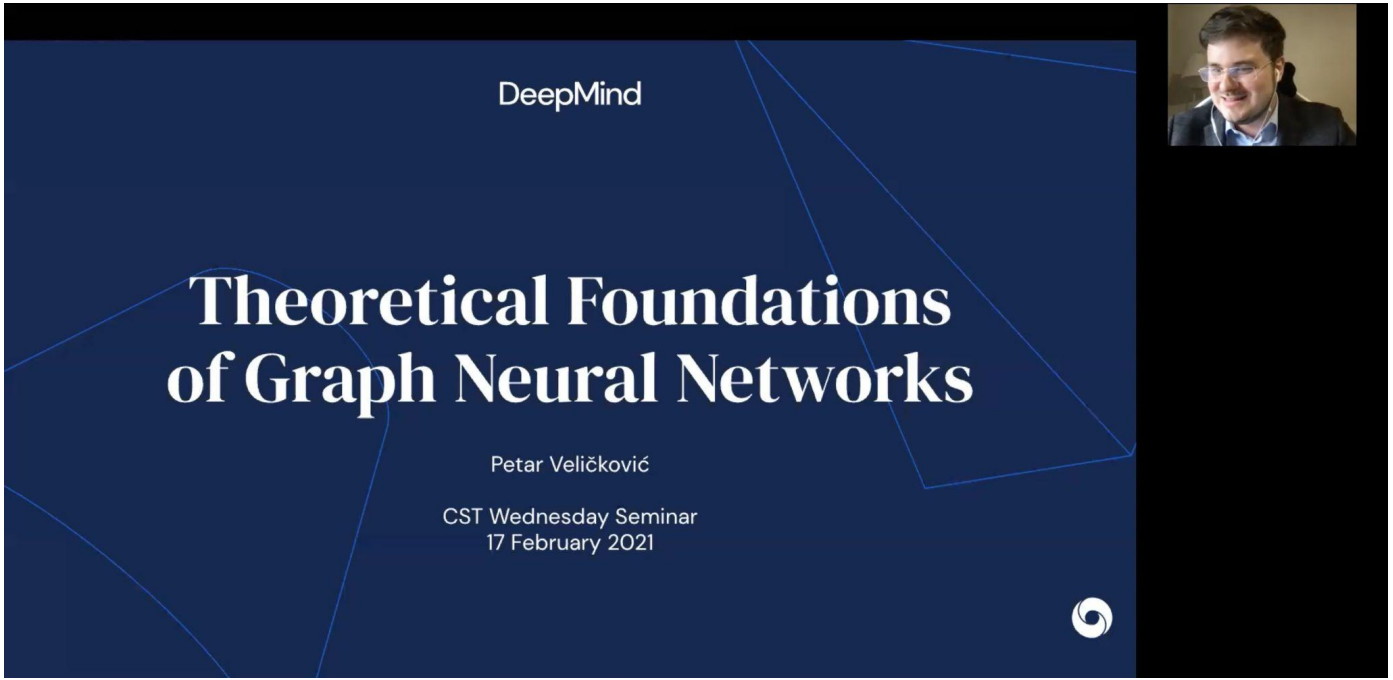


https://www.youtube.com/watch?v=IPQ6CPoluok

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

Further insight: 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?

Neural Algorithmic Reasoning

Petar Veličković and Charles Blundell

DeepMind

Abstract

Algorithms have been fundamental to recent global technological advances and, in particular, they have been the cornerstone of technical advances in one field rapidly being applied to another. We argue that algorithms possess fundamentally different qualities to deep learning methods, and this strongly suggests that, were deep learning methods better able to mimic algorithms, generalisation of the sort seen with algorithms would become possible with deep learning—something far out of the reach of current machine learning methods. Furthermore, by representing elements in a continuous space of learnt algorithms, neural networks are able to adapt known algorithms more closely to real-world problems, potentially finding more efficient and pragmatic solutions than those proposed by human computer scientists.

Here we present neural algorithmic reasoning—the art of building neural networks that are able to execute algorithmic computation—and provide our opinion on its transformative potential for running classical algorithms on inputs previously considered inaccessible to them.

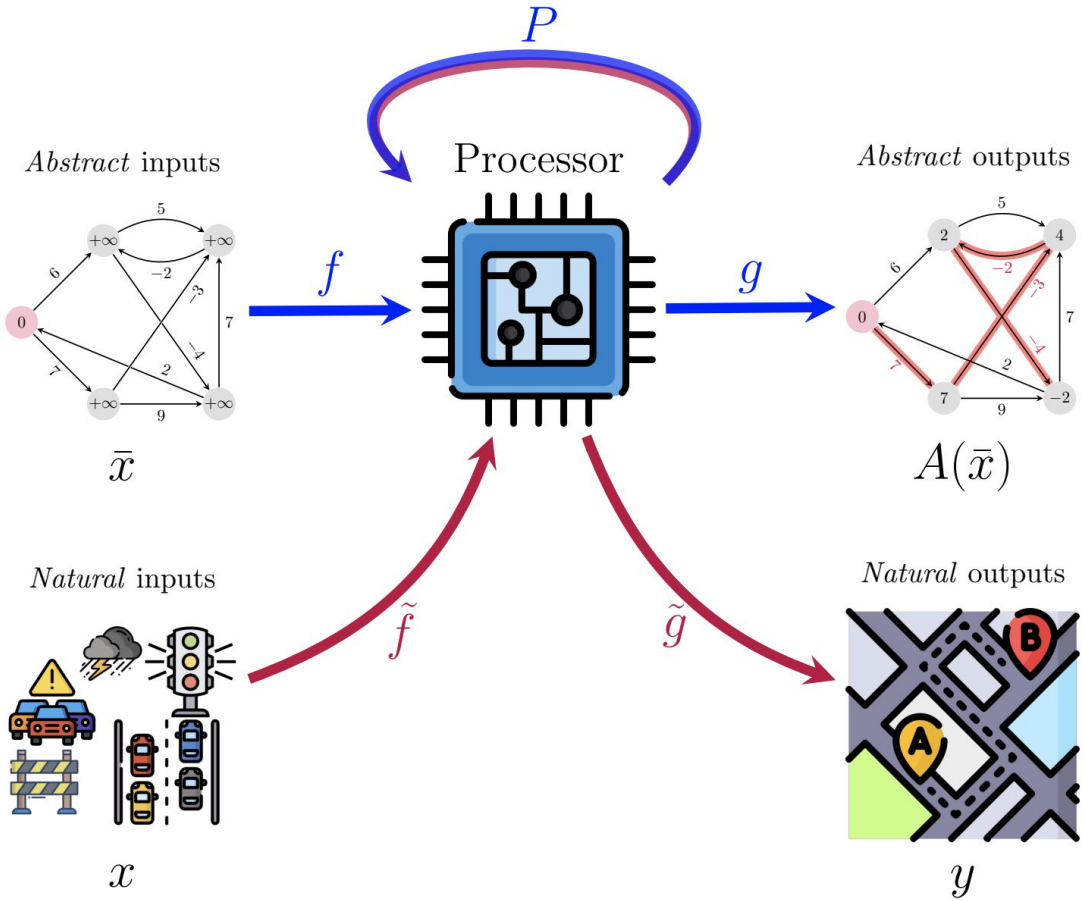
Our 7-page Opinion paper!

https://arxiv.org/abs/2105.02761

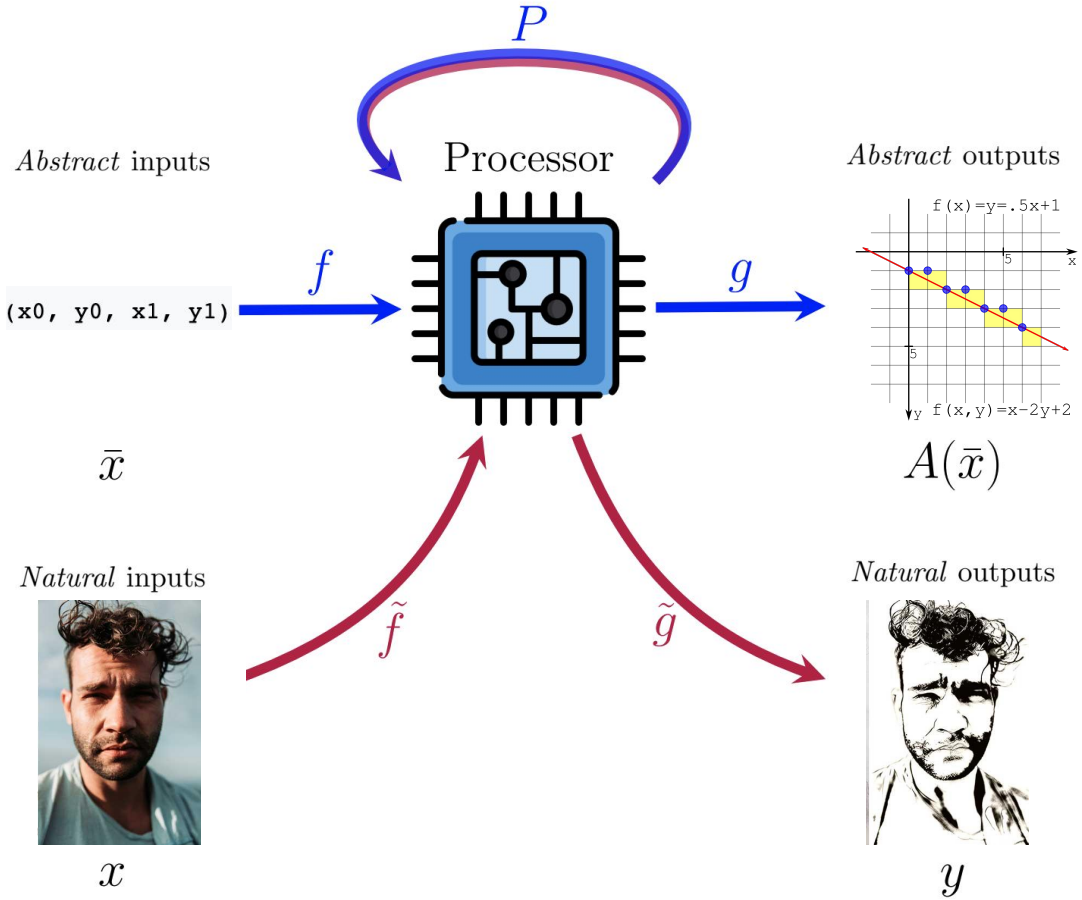
DeepMind

5 Implications of *algorithmic* sketching?

Blueprint of algorithmic reasoning



Backpropagating through the **drawing** algorithm

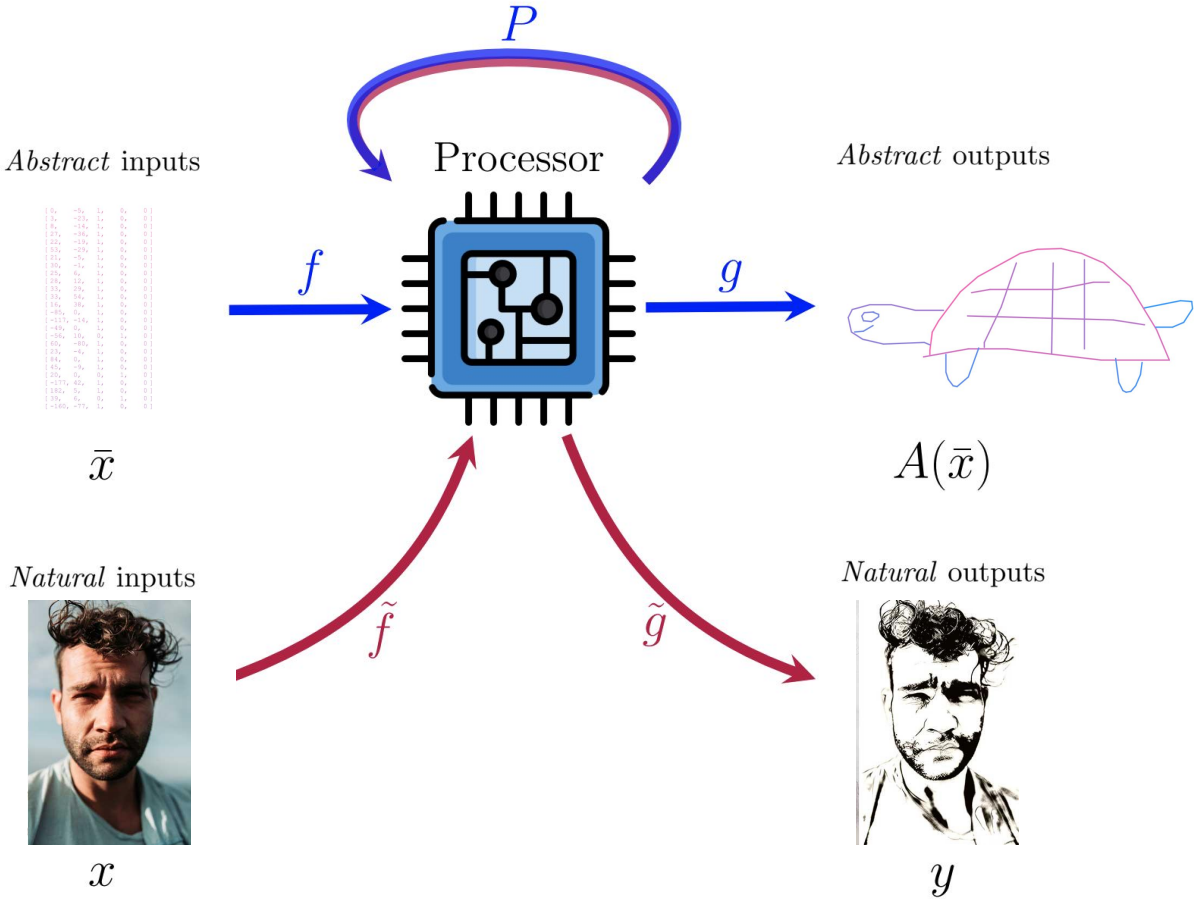


Now our "img2sketch" doesn't have to **re-learn** how to do **brush strokes**!

Can be taken to the extreme with *any* **rendering** algorithm.

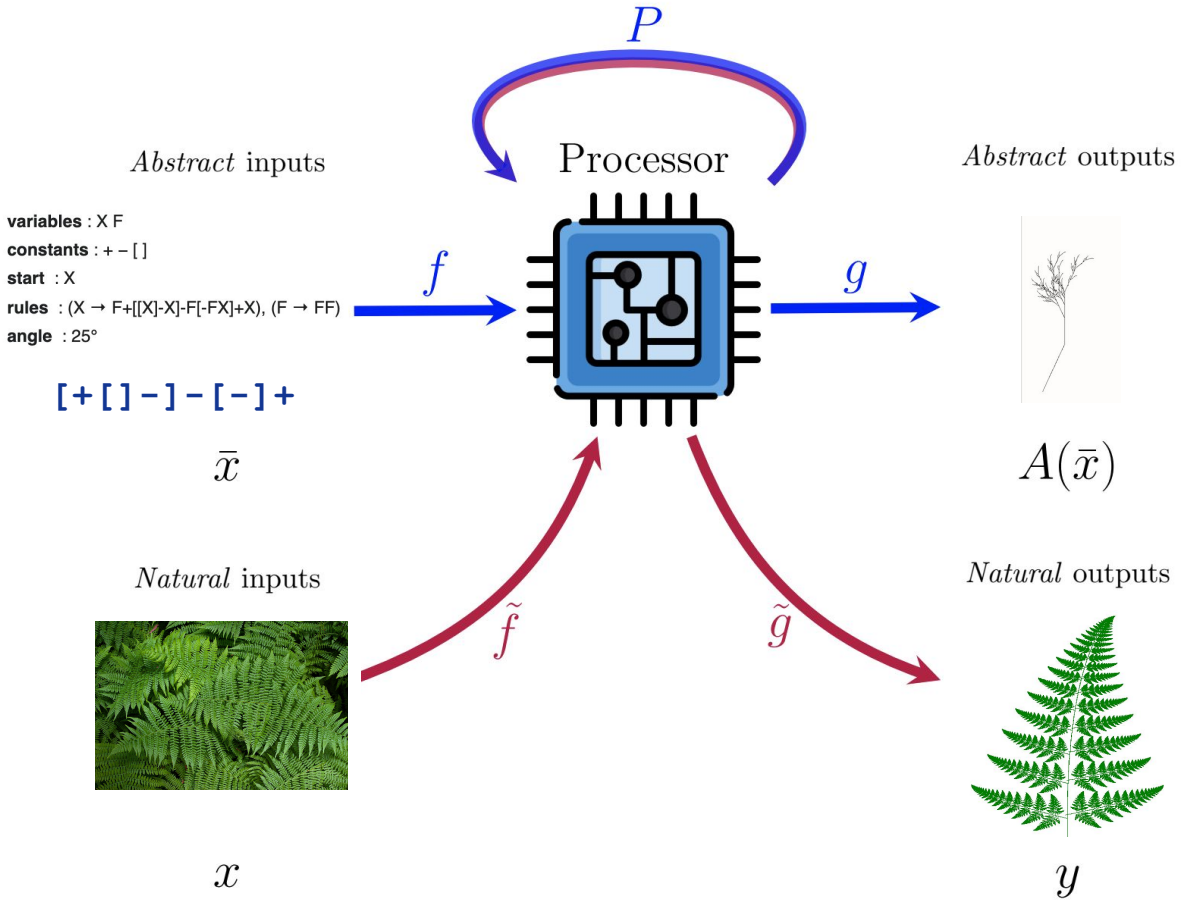
Backpropagating through the "sketcher"

We already know we can train abstract models of this kind (e.g. **SketchRNN**)!



Backpropagating through any parser

Extract the "hierarchy" within natural images by making them pass through a parser.



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