SYMBOLIC LANGUAGE MODELS FOR MATHEMATICAL DISCOVERY

FRANÇOIS CHARTON, CERMICS, ENPC, META AI

LANGUAGE MODEL - TRANSFORMERS

- An architecture for machine translation
- Represent language as sequences of tokens (words in a sentence)
- Learn, from examples only, to translate one language into another
  - Or complete a sentence (the prompt) by predicting the next token
- Became state of the art on a wide range of problems
  - Unsupervised translation
  - Vision
  - Speech/Music

LANGUAGE MODEL – TRANSFORMERS

- Require
  - Problems and solutions represented as sequences
  - Lots of training examples
  - Compute (GPU)
- Do not require
  - Problem-specific knowledge
  - Feature engineering
  - Specific classes of problems

LANGUAGE MODELS FOR MATHS

- Solving a problem is a translation task
- Learning to translating a problem into its solution
- Represented as sequences in some formal language
- From generated pairs of problems and solutions

LANGUAGE MODELS FOR MATHS

- Reasons to hope
  - Math is a language
  - Large sets of examples can be generated
  - Translation might work

- Reasons to doubt
  - Learning maths from examples ONLY?
  - Without rules and theory?
  - Prior work show that transformer struggle with arithmetic

SUPERVISED LEARNING IN ONE SLIDE

- A model is an obscenely overparametrized function (10s of millions of parameters), that mas a sequence of input tokens to the probability distributions of solution tokens
- The parameters (aka weights) are initialized randomly
  - Model predicts garbage
- Feed a group of inputs I (a mini-batch) into the model and record the (garbage) predictions P
- If O are the correct solutions (or one of the many possible soutions), compute a loss L(I, P, O, w)
  - For transformers, cross-entropy (are the tokens the correct ones?)
- Compute the gradient of L, with respect to the weights w
- Change model weights, so as to (slightly) reduce the loss
  - For cross-entropy: increase the probability of correct tokens
- Bring a new mini-batch, repeat the process: no problem-specific maths are involved during training

- Regularly (every 300 000 examples, an epoch), teste the model on held-out examples

DEEP LEARNING FOR SYMBOLIC MATHEMATICS

(Lample, Charton 2019, arXiv 1912.01412)

Two problems of symbolic mathematics

Symbolic integration, given f(x) find its anti-derivative

$$\frac{\cos(2x)}{\sin(x)} \longrightarrow \frac{\log(\cos(x)-1)}{2} - \frac{\log(\cos(x)+1)}{2} + 2\cos(x)$$

Differential equations, find y such that G(y(x), y'(x), x) = 0

$$162x\log(x)y' + 2y^3\log(x)^2 - 81y\log(x) + 81y = 0 \longrightarrow \frac{9\sqrt{x}}{\sqrt{c+2x}\sqrt{\log(x)}}$$

TWO PROBLEMS OF SYMBOLIC MATHEMATICS

- Advanced problems : taught at university level
- Non trivial, even for trained mathematicians
- Difficult for computer algebras
  - Risch algorithm for integration
- Involve pattern recognition
  - Neural nets have a chance

THREE STEPS

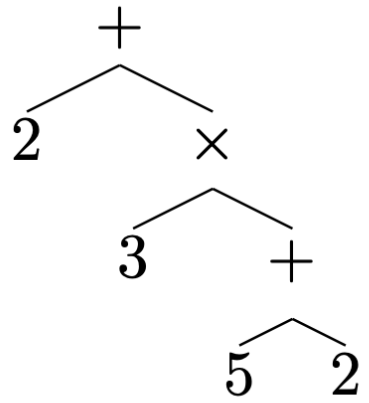
1. Represent problems and solutions as sequences of tokens
2. Generate datasets of problems and solutions
3. Train and evaluate models

REPRESENTING INPUT AND OUTPUT - TOKENIZATION

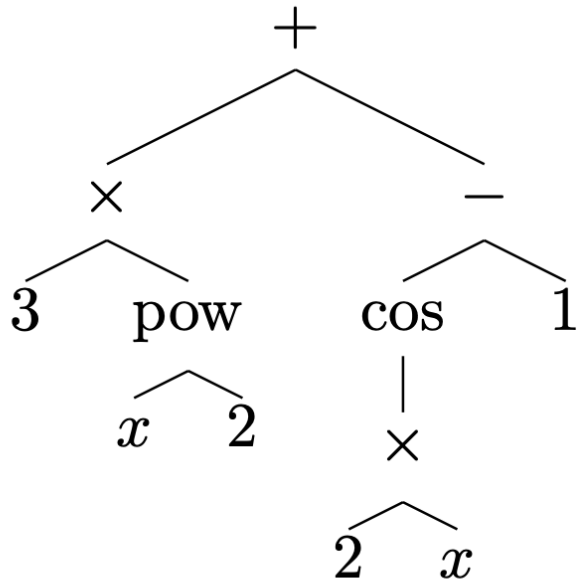
- Input and output are functions, i.e. mathematical expressions
  - For ODE, the "=0" part is dropped
- They can be represented as trees (abstract syntax trees)
- Which can then be enumerated (in prefix order), as sequences

EXPRESSIONS AS TREES

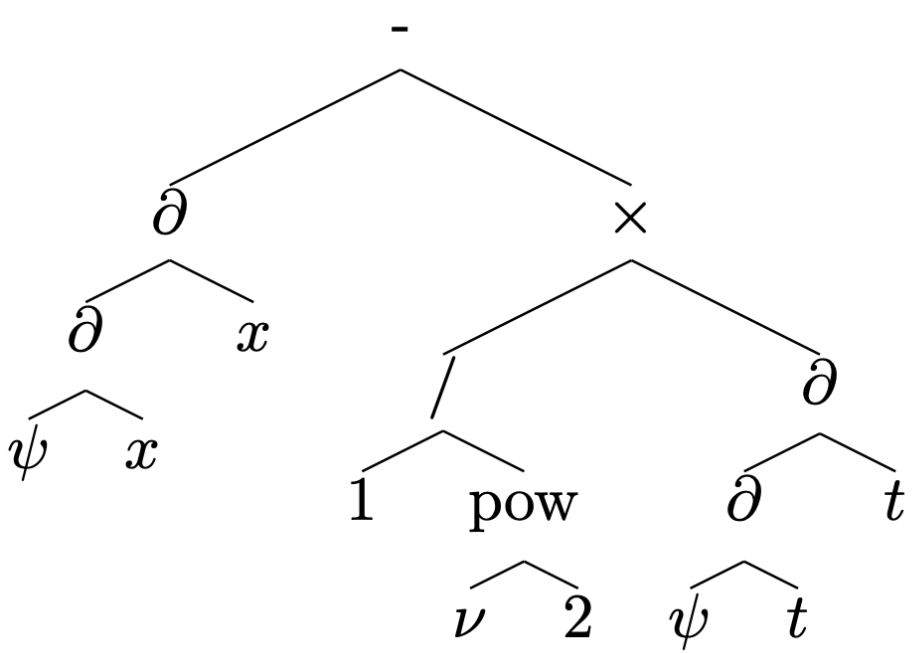
$2 + 3 \times (5 + 2)$



$3x^2 + \cos(2x) - 1$



$$\frac{\partial^2 \psi}{\partial x^2} - \frac{1}{\nu^2} \frac{\partial^2 \psi}{\partial t^2}$$

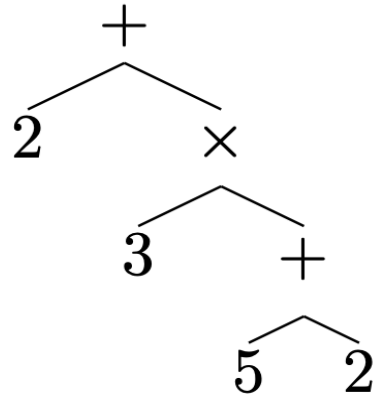


TREES AS SEQUENCES

Preorder enumeration, aka normal Polish notation

- begin from root
- parent before children
- left subtree before right subtree

$$2 + 3 \times (5 + 2)$$



+ 2 x 3 + 5 2

EXPRESSIONS AS SEQUENCES

$2 + 3 \times (5 + 2)$ + 2 * 3 + 5 2

$3x^2 + \cos(2x) - 1$ + * 3 pow x 2 - cos * 2 x 1

$\frac{\partial^2 \psi}{\partial x^2} - \frac{1}{\nu^2} \frac{\partial^2 \psi}{\partial t^2}$ - ∂ ∂ ψ x x * / 1 pow ν 2 ∂ ∂ ψ t t

Ready for the transformer!

GENERATING TRAINING DATA

- Forward method : the obvious idea
- Generate a random function f
  - By generating a random tree, and "decorating" it with operators for nodes, and small integers and the variable x for leaves
    - We use the four operations, and the Liouville elementary functions (exp, log, trigs and inverse, sqrt, pow, hyperbolic trigs and inverse) for operators
    - And the variable x and integers from -5 to 5 for leaves
- Compute its integral F, with an external tool (Sympy)
- Add the pair (f, F) to the training set

- Slow, limited to the integrals Sympy can solve

GENERATING TRAINING DATA

- Backward method: a faster approach
- Generate a random function F
- Compute its derivative f
- Add (f, F) to the training set

- Find the solutions of problems!

GENERATING TRAINING DATA

- Other methods are possible: Integration by part
- Generate random functions F and G
- Compute their derivatives f and g, and memorize them
- Check whether fG was already integrated, if so, we get Fg for free using $$\int Fg = FG - \int fG$$

- Inefficient, but will find its use
- Different generation techniques proovide different training (and test) sets

FORWARD GENERATION – SYMBOLIC INTEGRATION

Functions and their primitives generated with the forward approach (FWD)	
$\cos^{-1}(x)$	$x\cos^{-1}(x) - \sqrt{1-x^2}$
$x\left(2x + \cos\left(2x\right)\right)$	$\frac{2x^3}{3} + \frac{x\sin\left(2x\right)}{2} + \frac{\cos\left(2x\right)}{4}$
$\frac{x\left(x+4\right)}{x+2}$	$\frac{x^2}{2} + 2x - 4\log\left(x+2\right)$
$\frac{\cos\left(2x\right)}{\sin\left(x\right)}$	$\frac{\log\left(\cos\left(x\right)-1\right)}{2} - \frac{\log\left(\cos\left(x\right)+1\right)}{2} + 2\cos\left(x\right)$
$3x^2\sinh^{-1}\left(2x\right)$	$x^3\sinh^{-1}\left(2x\right) - \frac{x^2\sqrt{4x^2+1}}{6} + \frac{\sqrt{4x^2+1}}{12}$
$x^3\log\left(x^2\right)^4$	$\frac{x^4\log\left(x^2\right)^4}{4} - \frac{x^4\log\left(x^2\right)^3}{2} + \frac{3x^4\log\left(x^2\right)^2}{4} - \frac{3x^4\log\left(x^2\right)}{4} + \frac{3x^4}{8}$

BACKWARD GENERATION – SYMBOLIC INTEGRALS

Functions and their primitives generated with the backward approach (BWD)	
$\cos(x) + \tan^2(x) + 2$	$x + \sin(x) + \tan(x)$
$\frac{1}{x^2\sqrt{x-1}\sqrt{x+1}}$	$\frac{\sqrt{x-1}\sqrt{x+1}}{x}$
$\left(\frac{2x}{\cos^2(x)} + \tan(x)\right)\tan(x)$	$x\tan^2(x)$
$\frac{x\tan\left(\frac{e^x}{x}\right) + \frac{(x-1)e^x}{\cos^2\left(\frac{e^x}{x}\right)}}{x}$	$x\tan\left(\frac{e^x}{x}\right)$
$1 + \frac{1}{\log(\log(x))} - \frac{1}{\log(x)\log(\log(x))^2}$	$x + \frac{x}{\log(\log(x))}$
$-2x^2\sin\left(x^2\right)\tan(x) + x\left(\tan^2(x)+1\right)\cos\left(x^2\right) + \cos\left(x^2\right)\tan(x)$	$x\cos\left(x^2\right)\tan(x)$

INTEGRATION BY PART – SYMBOLIC INTEGRALS

Functions and their primitives generated with the integration by parts approach (`IBP`)	
$x\left(x+\log\left(x\right)\right)$	$\frac{x^2\left(4x+6\log\left(x\right)-3\right)}{12}$
$\frac{x}{\left(x+3\right)^2}$	$\frac{-x+\left(x+3\right)\log\left(x+3\right)}{x+3}$
$\frac{x+\sqrt{2}}{\cos^2\left(x\right)}$	$\left(x+\sqrt{2}\right)\tan\left(x\right)+\log\left(\cos\left(x\right)\right)$
$x\left(2x+5\right)\left(3x+2\log\left(x\right)+1\right)$	$\frac{x^2\left(27x^2+24x\log\left(x\right)+94x+90\log\left(x\right)\right)}{18}$
$\frac{\left(x-\frac{2x}{\sin^2\left(x\right)}+\frac{1}{\tan\left(x\right)}\right)\log\left(x\right)}{\sin\left(x\right)}$	$\frac{x\log\left(x\right)+\tan\left(x\right)}{\sin\left(x\right)\tan\left(x\right)}$
$x^3\sinh\left(x\right)$	$x^3\cosh\left(x\right)-3x^2\sinh\left(x\right)+6x\cosh\left(x\right)-6\sinh\left(x\right)$

GENERATING TRAINING DATA - ODE

- A backward method is needed
- Solutions of first order ODE are defined up to a constant *c*
- Starting from a solution $y = f(x, c)$
- Computing the function F such that $F(x, y) = c$ (solving in c)
- And differentiating w.r.t. *x* $$\frac{\partial F(x,y)}{\partial x} + y' \frac{\partial F(x,y)}{\partial y} = 0$$

- We have an equation that f(x,c) solves

GENERATING ODE

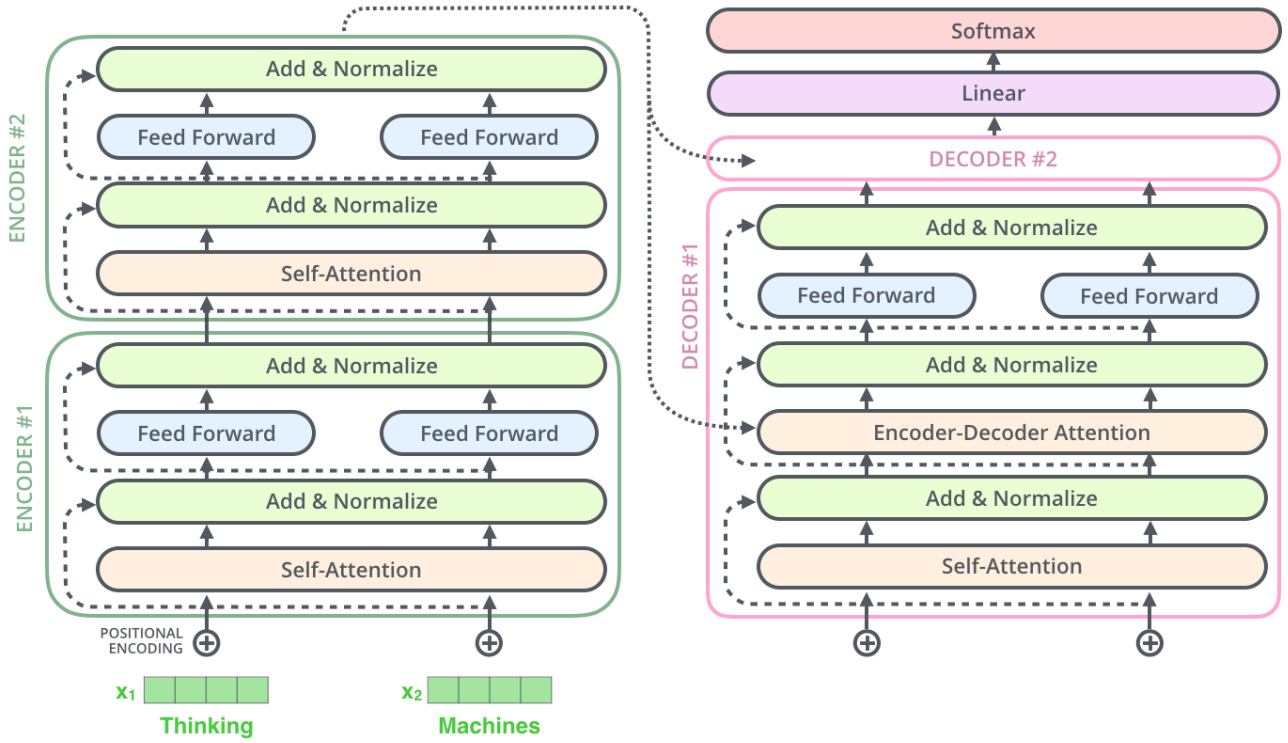
Generate a random function	$f(x,c) = x\log(\frac{c}{x})$ = y
Solve in c	$c = xe^{\frac{y}{x}} = F(x,y)$
Differentiate F wrt x	$e^{\frac{y}{x}}\left(1 + y' - \frac{y}{x}\right) = 0$
Simplify	$xy' - y + x = 0$

5 DATASETS

	Forward	Backward	Integration by parts	ODE 1	ODE 2
Training set size	20M	40M	20M	40M	40M
Input length	18.9±6.9	70.2±47.8	17.5±9.1	123.6±115.7	149.1±130.2
Output length	49.6±48.3	21.3±8.3	26.4±11.3	23.0±15.2	24.3±14.9
Length ratio	2.7	0.4	2.0	0.4	0.1
Input max length	69	450	226	508	508
Output max length	508	75	206	474	335

TRAIN AND EVALUATE MODELS

- An encoder-decoder architecture (Vaswani 2017)
- Two transformers
- A bidirectional encoder processes input sequences as a whole
- An auto-regressive transformer decodes output one token at a time
- From previously decoded output, and encoder output (via a cross-attention mechanism)
- 6 layers, 256 dimensions, 8 attention heads



DECODING THE OUTPUT

- The decoder computes a probability distribution for the next token, conditional to
  - The encoder output
  - The previously decoded sequence
- It is first fed a begin token [BOS], and outputs the most likely continuation (say [+])
- The decoder is then fed [BOS, +], and predicts the most likely continuation
- The process ends when the decoder outputs a specific end of sequence token (EOS).

BEAM SEARCH – ALLOWING SEVERAL GUESSES

- Instead of generating the most likely next token, generate the top k
- Feed the k predictions, generate k^2 continuations, and keep the k most likely
- Repeat until k guesses are computed

EVALUATING THE MODEL

- Trained models are tests on 5,000 held-out examples (never seen during training)
- Model output must be verified using an external tool, here Sympy
  - Because the solution is not unique
- For symbolic integration, compute the derivative of the solution, substract from the input, and reduce to verify that F' - f = 0
- For ODE, feed the solution into the equation, and reduce

RESULTS

	Integration (FWD)	Integration (BWD)	Integration (IBP)	ODE (order 1)	ODE (order 2)
Beam size 1	93.6	98.4	96.8	77.6	43.0
Beam size 10	95.6	99.4	99.2	90.5	73.0
Beam size 50	96.2	99.7	99.5	94.0	81.2

- Integration
  - Almost perfect results, even without beam
  - No matter how the data is generated, universal method
- ODE
  - Beam search to the rescue
  - Especially for order 2

BEATING MATHEMATICA...

	Integration (BWD)	ODE (order 1)	ODE (order 2)
Mathematica (30s)	84.0	77.2	61.6
Matlab	65.2	-	-
Maple	67.4	-	-
Beam size 1	98.4	81.2	40.8
Beam size 10	99.6	94.0	73.2
Beam size 50	99.6	97.0	81.0

GENERALIZATION ISSUES

- Models are tested on examples not seen during training: generalization
- But generated with the same method as the training set: in-domain generalization

- What if they were not?
- How dependent are we on the generating method?

- Out-of-domain generalization

GENERALIZATION - LOOKING BAD

Training data	Forward (FWD) Beam 1	Forward (FWD) Beam 10	Forward (FWD) Beam 50	Backward (BWD) Beam 1	Backward (BWD) Beam 10	Backward (BWD) Beam 50
FWD	93.6	95.6	96.2	10.9	13.9	17.2
BWD	18.9	24.6	27.5	98.4	99.4	99.7

- In domain generalization is good
- Out of domain is terrible (and asymmetric)

FORWARD GENERATION – SYMBOLIC INTEGRATION

Functions and their primitives generated with the forward approach (FWD)	
$\cos^{-1}(x)$	$x\cos^{-1}(x) - \sqrt{1-x^2}$
$x\left(2x + \cos\left(2x\right)\right)$	$\frac{2x^3}{3} + \frac{x\sin\left(2x\right)}{2} + \frac{\cos\left(2x\right)}{4}$
$\frac{x\left(x+4\right)}{x+2}$	$\frac{x^2}{2} + 2x - 4\log\left(x+2\right)$
$\frac{\cos\left(2x\right)}{\sin\left(x\right)}$	$\frac{\log\left(\cos\left(x\right)-1\right)}{2} - \frac{\log\left(\cos\left(x\right)+1\right)}{2} + 2\cos\left(x\right)$
$3x^2\sinh^{-1}\left(2x\right)$	$x^3\sinh^{-1}\left(2x\right) - \frac{x^2\sqrt{4x^2+1}}{6} + \frac{\sqrt{4x^2+1}}{12}$
$x^3\log\left(x^2\right)^4$	$\frac{x^4\log\left(x^2\right)^4}{4} - \frac{x^4\log\left(x^2\right)^3}{2} + \frac{3x^4\log\left(x^2\right)^2}{4} - \frac{3x^4\log\left(x^2\right)}{4} + \frac{3x^4}{8}$

BACKWARD GENERATION – SYMBOLIC INTEGRALS

Functions and their primitives generated with the backward approach (BWD)	
$\cos(x) + \tan^2(x) + 2$	$x + \sin(x) + \tan(x)$
$\frac{1}{x^2\sqrt{x-1}\sqrt{x+1}}$	$\frac{\sqrt{x-1}\sqrt{x+1}}{x}$
$\left(\frac{2x}{\cos^2(x)} + \tan(x)\right)\tan(x)$	$x\tan^2(x)$
$\frac{x\tan\left(\frac{e^x}{x}\right) + \frac{(x-1)e^x}{\cos^2\left(\frac{e^x}{x}\right)}}{x}$	$x\tan\left(\frac{e^x}{x}\right)$
$1 + \frac{1}{\log(\log(x))} - \frac{1}{\log(x)\log(\log(x))^2}$	$x + \frac{x}{\log(\log(x))}$
$-2x^2\sin\left(x^2\right)\tan(x) + x\left(\tan^2(x)+1\right)\cos\left(x^2\right) + \cos\left(x^2\right)\tan(x)$	$x\cos\left(x^2\right)\tan(x)$

INTEGRATION BY PART – SYMBOLIC INTEGRALS

Functions and their primitives generated with the integration by parts approach (`IBP`)	
$x\left(x+\log\left(x\right)\right)$	$\frac{x^2\left(4x+6\log\left(x\right)-3\right)}{12}$
$\frac{x}{\left(x+3\right)^2}$	$\frac{-x+\left(x+3\right)\log\left(x+3\right)}{x+3}$
$\frac{x+\sqrt{2}}{\cos^2\left(x\right)}$	$\left(x+\sqrt{2}\right)\tan\left(x\right)+\log\left(\cos\left(x\right)\right)$
$x\left(2x+5\right)\left(3x+2\log\left(x\right)+1\right)$	$\frac{x^2\left(27x^2+24x\log\left(x\right)+94x+90\log\left(x\right)\right)}{18}$
$\frac{\left(x-\frac{2x}{\sin^2\left(x\right)}+\frac{1}{\tan\left(x\right)}\right)\log\left(x\right)}{\sin\left(x\right)}$	$\frac{x\log\left(x\right)+\tan\left(x\right)}{\sin\left(x\right)\tan\left(x\right)}$
$x^3\sinh\left(x\right)$	$x^3\cosh\left(x\right)-3x^2\sinh\left(x\right)+6x\cosh\left(x\right)-6\sinh\left(x\right)$

GENERALIZATION – LOOKING BETTER

Training data	Forward (FWD)			Backward (BWD)			Integration by parts (IBP)		
	Beam 1	Beam 10	Beam 50	Beam 1	Beam 10	Beam 50	Beam 1	Beam 10	Beam 50
FWD	93.6	95.6	96.2	10.9	13.9	17.2	85.6	86.8	88.9
BWD	18.9	24.6	27.5	98.4	99.4	99.7	42.9	54.6	59.2
BWD + IBP	41.6	54.9	56.1	98.2	99.4	99.7	96.8	99.2	99.5

- On IBP test set, FWD and BWD trained models look much better
- Out-of-domain generalization is possible if training and test set are not too different
- (but we need a proper definition for this)

DEEP LEARNING FOR SYMBOLIC MATHS – TAKE AWAYS

- It works! Transformers can be trained just from examples, and achieve state-of-the-art performance
- Data generation matters: different methods exist.
- Out of domain generalization can be an issue.

LEARNING ADVANCED MATHEMATICAL COMPUTATIONS FROM EXAMPLES

(Charton, Hayat, Lample, 2020, 206.06462)

- Local stability of differential systems
- Systems of n first order differential equations in n variables $\frac{dx}{dt} = f(x) \quad x \in R^n$
- Equilibria happen when f(x)=0
- Spectral mapping theorem:

  let $\mathrm{J}(\mathrm{x}) = \nabla f(x)$ be the Jacobian of f, if all it (complex) eigenvalues have negative real part, the equilibrium is stable (and the largest real part is the speed of convergence).

LOCAL STABILITY OF DIFFERENTIAL SYSTEMS

- For the system

$$\frac{dx_1}{dt} = \cos(x_2) - 1 - \sin(x_1)$$
$$\frac{dx_2}{dt} = x_1^2 - \sqrt{1 + x_2}$$

- The Jacobian is

$$J(x) = \begin{pmatrix} -\cos(x_1) & -\sin(x_2) \\ 2x_1 & -(2\sqrt{1+x_2})^{-1} \end{pmatrix}$$

- Evaluate it at the equilibrium
- Compute real parts of eigenvalues: -1.031, -0.441
- Find the largest: -0.441, the system is stable

LOCAL STABILITY OF DIFFERENTIAL SYSTEMS

Table 1: **Prediction of local convergence speed (within 10%).**

	Degree 2	Degree 3	Degree 4	Degree 5	Degree 6	Overall
4 layers, dim 512	88.0	74.3	63.8	54.2	45.0	65.1
6 layers, dim 512	93.6	85.5	77.4	71.5	64.9	78.6
8 layers, dim 512	95.3	88.4	83.4	79.2	72.4	83.8
4 layers, dim 1024	91.2	80.1	71.6	61.8	54.4	71.9
6 layers, dim 1024	95.7	89.0	83.4	78.4	72.6	83.8
8 layers, dim 1024	**96.3**	**90.4**	**86.2**	**82.7**	**77.3**	**86.6**

CONTROLABILITY OF DIFFERENTIAL SYSTEMS

Let the differential system be overparametrized, with control variables u

$$\frac{dx}{dt} = f(x,u) \quad x \in R^n \quad u \in R^p$$

Can u be chosen so that $x(0) = x_0$ and $x(T) = x_1$?

Kalman condition : let $A = \partial_x f(x,u)$ $B = \partial_u f(x,u)$, the system is controllable if rank[B, AB, … $A^{n-1}B$] = n

CONTROLABILITY OF DIFFERENTIAL SYSTEMS

- For the system

$$\frac{dx_1(t)}{dt} = \sin(x_1^2) + \log(1 + x_2) + \frac{\operatorname{atan}(ux_1)}{1 + x_2}$$
$$\frac{dx_2(t)}{dt} = x_2 - e^{x_1 x_2},$$

- Differentiate in x

$$A(x,u) = \begin{pmatrix} 2x_1\cos(x_1^2) + \frac{u(1+x_2)^{-1}}{1+u^2x_1^2} & (1+x_2)^{-1} - \frac{\operatorname{atan}(ux_1)}{(1+x_2)^2} \\ -x_2 e^{x_1x_2} & 1 - x_1 e^{x_1x_2} \end{pmatrix}$$

- Differentiate in u

$$B(x,u) = \begin{pmatrix} x_1((1+u^2x_1^2)(1+x_2))^{-1} \\ 0 \end{pmatrix}$$

- Evaluate

$$A(x_e,u_e) = \begin{pmatrix} 1.50 & 0.46 \\ -0.64 & 0.36 \end{pmatrix}, \quad B(x_e,u_e) = \begin{pmatrix} 0.27 \\ 0 \end{pmatrix}$$

CONTROLABILITY OF DIFFERENTIAL SYSTEMS

- Evaluate $A(x_e, u_e) = \begin{pmatrix} 1.50 & 0.46 \\ -0.64 & 0.36 \end{pmatrix}, \quad B(x_e, u_e) = \begin{pmatrix} 0.27 \\ 0 \end{pmatrix}$
- Compute control matrix $C = [B, AB]((x_e, u_e)) = \begin{pmatrix} 0.27 & 0.40 \\ 0 & -0.17 \end{pmatrix}$
- Calculate the rank $\mathbf{rank}(C) = 2$

CONTROLABILITY OF DIFFERENTIAL SYSTEMS

Table 3: **Accuracy of autonomous control task over a balanced sample of systems with 3 to 6 equations.**

	Dimension 64	Dimension 128	Dimension 256	Dimension 512
1 layers	81.0	85.5	88.3	90.4
2 layers	82.7	88.0	93.9	95.5
4 layers	84.1	89.2	95.6	96.9
6 layers	84.2	90.7	96.3	**97.4**

- A large enough model achieves 95% accuracy
- Even very small models (that would fail on natural language tasks) achieve non-trivial performance

CONTROLABILITY OF DIFFERENTIAL SYSTEMS

- Bonus question: find K such that u=Kx. A possible answer is

$$K = -B^{tr}\left(e^{-AT}\left[\int_0^T e^{-At}BB^{tr}e^{-A^{tr}t}dt\right]e^{-A^{tr}T}\right)^{-1}$$

- In the previous example $K = (-22.8 \quad 44.0)$

CONTROLABILITY OF DIFFERENTIAL SYSTEMS

Table 5: **Prediction of feedback matrices - Approximation vs. correct mathematical feedback.**

	Degree 3	Degree 4	Degree 5	Degree 6	Overall
Prediction within 10%	50.0	9.3	2.1	0.4	15.8
Correct feedback matrix	87.5	77.4	58.0	41.5	66.5

- The feedback matrix K (from the previous slide) can be predicted with non-trivial accuracy, but preformance drops steeply as the model scale
- However, the model predicts correct feedback matrices in 66% of test cases
- Some maths were learned

MORE TAKE AWAYS

- Language models can predict numerical properties of symbolic systems
- They can perform (or approximate?) complex calculations, a mix of symbolic and numerical operations
- They seem to "understand" the underlying mathematics: the last result on controlability

- But so far, we are solving solved problems, could these models discover new maths?

DISCOVERING LYAPUNOV FUNCTIONS

(Alfarano, Charton, Hayat, 2024, 2410.08304)

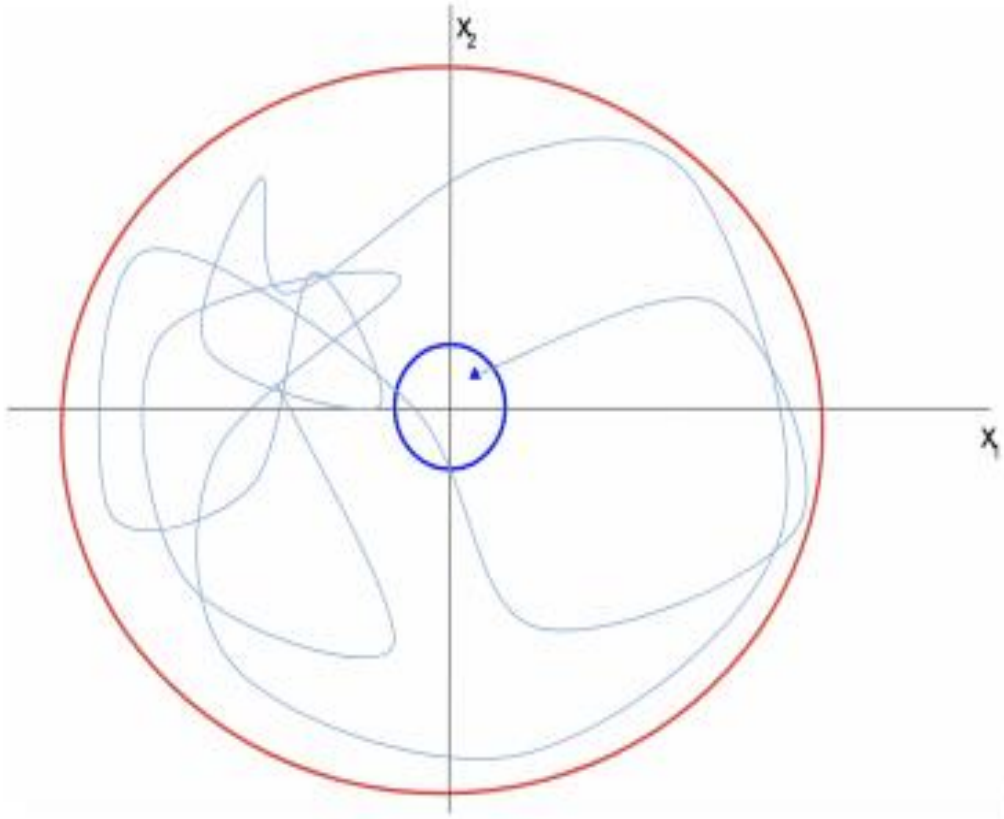
- Consider a dynamical system $\dot{x} = f(x), \quad x \in \mathbb{R}^n, \quad f \in C^1(\mathbb{R}^n)$
- The spectral mapping theorem takes care of the local stability at x_0, such $f(x_0) = 0$,
- But it says nothing of the global stability: what happens in a neighborhood of x_0, or over the whole configuration space.

DISCOVERING LYAPUNOV FUNCTIONS

- Global stability: behavior around a stable equilibrium (assuming $\dot{x}_0 = 0$)
- If $x(t) < \delta$ for $t = 0$, do we have $x(t) < B$ for any $t > 0$?

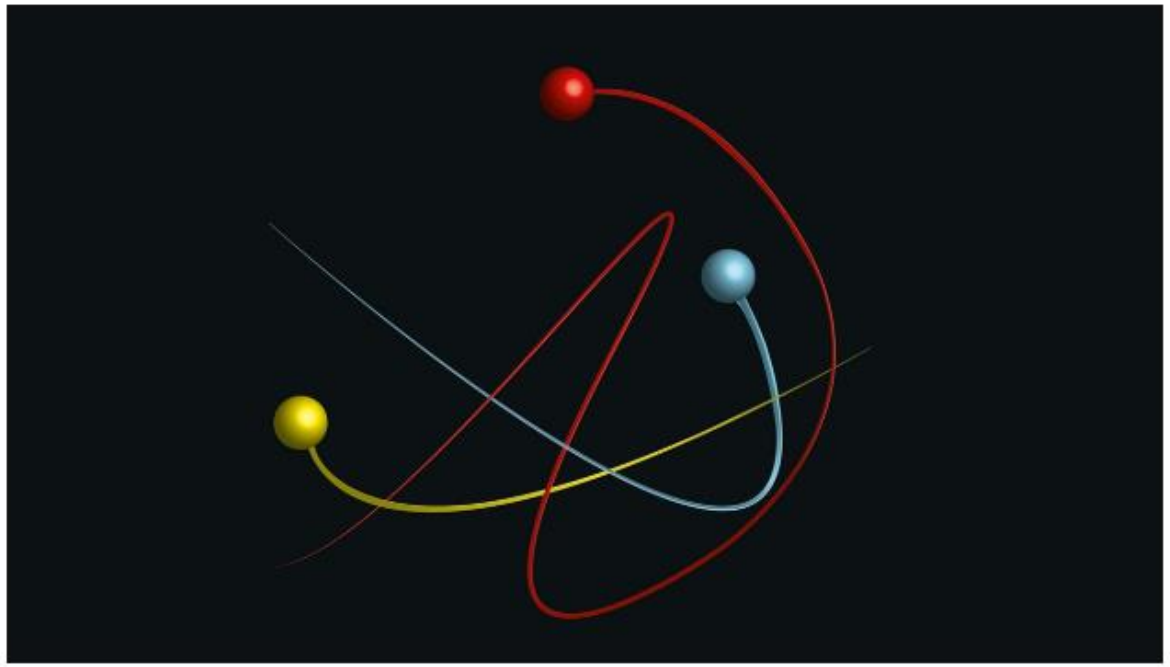
If I start close to the equilibrium
will I always remain close?

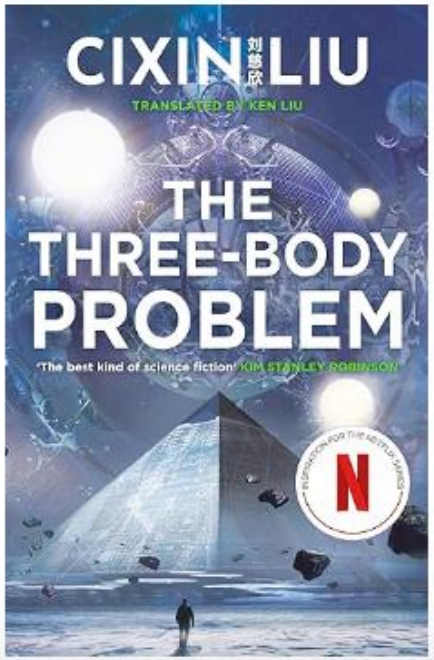
- A much harder problem



DISCOVERING LYAPUNOV FUNCTIONS

- Stability of the solar system: the N-body problem
- Of Laplace, Poincaré, King of Sweden and Netflix fame!





ENTERS LYAPUNOV

- Lyapunov (1892): if there exists $V \in C^1(\mathbb{R}^n, \mathbb{R})$, and for all $x \in \mathbb{R}^n$,

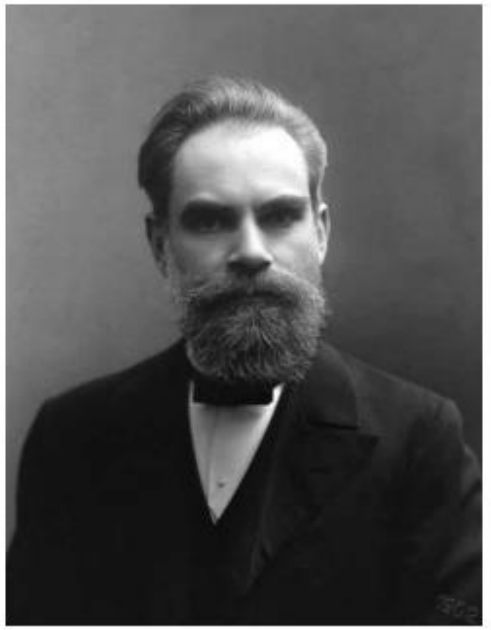
$V(x) > V(0)$ strict minimum in 0

$\lim_{|x|\to+\infty} V(x) = +\infty$ infinite at infinity

$\nabla V(x).f(x) \leq 0$ gradient points away from f

Then the system is globally stable

A sufficient condition, necessary in a number of cases



A. Lyapunov
(1857-1918)

BARRIER FUNCTIONS

- If we relax the first condition, asking for a non-strict minimum

$V(x) \geq V(0)$ strict minimum in 0

$\lim_{|x|\to+\infty} V(x) = +\infty$ infinite at infinity

$\nabla V(x).f(x) \leq 0$ gradient points away from f

Then the function V cuts the configuration space in two subspaces and trajectories are confined in one

V is a barrier Lyapunov functions (a weaker property)

LYAPUNOV FUNCTIONS

$$\begin{cases} \dot{x}_0 &= 2x_1^2 \\ \dot{x}_1 &= -10x_1 \end{cases}$$

Model inference: Our model recovers $V(x) = 10x_0^2 + 2x_0x_1^2 + 3x_1^4 + 6x_1^2$.

Clearly $V(0) = 0$ and $V(x) = 9(x_0)^2 + (x_0 + x_1^2)^2 + 2(x_1^2)^2 + 6x_1^2 > 0$ for all $x \neq 0$. Also $\nabla V \cdot f = -x_1^2(116x_1^2 + 120) \leq 0$.

LYAPUNOV FUNCTIONS

- No general method is known for constructing Lyapunov functions
- Methods exist for small polynomial systems with low degree, having a sum of square Lyapunov function (SOSTOOLS)
- But polynomial systems do not always have sum of square Lyapunov functions (Amadi 2011)

$$\begin{cases} \dot{x}_0 = -7x_0^5 - 4x_0^3x_1^2 - 5x_0^3 \\ \dot{x}_1 = 7x_0^4 - 3x_1 - 2x_2 \\ \dot{x}_2 = -8x_0^2 - 9x_2 \end{cases}$$

$$V(x) = 2x_0^4 + 2x_0^2x_1^2 + 3x_0^2 + 2x_1^2 + x_2^2$$

$$\begin{cases} \dot{x}_0 &= -x_0 + x_0x_1 \\ \dot{x}_1 &= -x_1 \end{cases}$$

$$V(x) = \ln(1 + 5x_0^2) + x_1^2$$

USING TRANSFORMERS FOR AN OPEN PROBLEM

- Generating training sets is the hard question
- Model predictions must be verified, here by checking the Lyapunov conditions, with optimization or SMT tools
  - These methods can fail, and cause false negatives: accuracies are always underestimated

GENERATING DATA IN A PERFECT WORLD

- The model is trained from examples, pairs of systems and associated Lyapunov functions
- In an ideal world, we would
  - Randomly sample stable systems S
  - Compute their Lyapunov function V
  - Add the pair (S, V) to the training set, repeat
- Forward generation
  - Only available for small polynomial systems

But wait what?

We cannot characterize a stable system without a Lyapunov function

We have no method for computing those

Quand les poules auront des dents

GENERATING DATA FOR OPEN PROBLEMS

- Backward generation: instead of finding the solutions of problems, find the problems of solutions
- Sample a random function V
  - A C^1 function
  - With a strict minimum in 0 $V(x) > V(0)$
  - Infinite at infinity $\lim_{|x|\to+\infty} V(x) = +\infty$
- Then find a random system f such that $\nabla V(x).f(x) \leq 0$

BACKWARD METHOD - TWO CAVEATS

- We want V, and f, as generic as possible
  - We should sample V from the class of continuous function with a strict minimum at 0, and infinite at infinity
  - For a given V, f should be sampled from all functions such that $\nabla V(x).f(x) \leq 0$
  - In any case, the training distribution will be different from the test distribution
- We want the model to learn to solve the Lyapunov problem not to reverse the generating procedure
  - Suppose we compute (exactly) the real roots of polynomial with integer coefficients
  - You could a polynomial from its roots: e.g. P(x)=(x-2)(x-5)(x-7), for roots 2,5,and 7
  - But the model, confronted with P(x)=(x-2)(x-5)(x-7), can "read" the roots in the problem, instead of solving it
  - If I provide the developed form P(x) = x^3-14x^2+59x-70, I am solving the hard problem.

GENERATING A LYAPUNOV FUNCTION

- V infinite at infinity, with a strict minimum at zero
  - No systematic way to sample functions with a strict minimum
- We rewrite $V = V_{proper} + V_{cross}$,
  - V_{proper} has a strict minimum in zero and is infinite at infinity,
  - V_{cross} is non-negative and bounded

GENERATING A LYAPUNOV FUNCTION

- Specifically, for V_{proper}, we
  - sample a positive polynomial $P(x) = \sum_{i,j=1}^{n} a_{i,j} x_i^{b_i} x_j^{b_j}$ with $a_{i,j}$ the entries of a random positive definite matrix,
  - apply a generic increasing function I (sampled in a large class)
  - multiply by random positive functions
- V_{cross} is a sum of squares of bounded functions

$$V(x) = \left[I\left(\sum_{i=1}^{n} \alpha_{i,j} x_i^{\beta_i} x_j^{\beta_j} \right) - f(0) \right] g\left(\sum_{i=1}^{q} \alpha_{\sigma(i),\sigma(j)} x_{\sigma(i)}^{\beta_{\sigma(i)}} x_{\sigma(j)}^{\beta_{\sigma(j)}} \right) + \sum_{i=1}^{m} b_k(\xi_k + p_k(x)),$$

GENERATING AN ASSOCIATED SYSTEM

We want $\nabla V(x).f(x) \leq 0$,

$f(x) = -\nabla V(x)$ is an obvious solution

But a very bad one: finding V from f, now amounts to integrating f, we are solving a different (and easier) problem!

We can modify this solutions by multiplying each coordinate by a random positive function

$$f(x) = -(h_i{}^2(x)(\nabla V(x))_i),$$

we still verify $\nabla V(x).f(x) \leq 0$, and integration of f no longer allows to recover V

GENERATING AN ASSOCIATED SYSTEM

$f(x) = -h^2(x)\nabla V(x)$ verifies $\nabla V(x).f(x) \leq 0$, but it is not generic enough

A better candidate is $f(x) = -h^2(x)\nabla V(x) + w(x)$, with $w(x).\nabla V(x) = 0$

i.e. adding a vector w from the normal hyperplane $\mathcal{H}_x = \{w | w.\nabla V = 0\}$,

generated as

$$w(x) = \sum_{i=0}^{n-1} g_i(x)e_i(x),$$

with g_i random functions and e_i base vectors of $\mathcal{H}_x$

(but don't compute the base vectors with Gram-Schmidt, or factor $1/||\nabla V(x)||$ will pop up in your system, and provide clues to the transformer)

TRAINING SETS (AT LAST!)

- We generate two backward datasets:
  - BPoly: 1M backward generated polynomial systems of 2 to 5 equations
  - BNonPoly: 1M backward generated non-polynomial systems: polynomials of general functions (e.g. trigonometric polynomials)

- And use SOSTOOLS to generate two polynomial forward sets
  - FLyap: 100,000 polynomial systems that SOSTOOLS can solve
  - FBarr: 300,000 polynomial systems for which SOSTOOLS can find a barrier Lyapunov function (V(x) is no longer strictly positive).

IN-DOMAIN PERFORMANCE

- Testing models on held-out sets from the same distribution as the training set
- Good results, but deceptive
  - The model might have learned to reverse the generation procedure

Backward datasets	Accuracy bs=1	Accuracy bs=50	Forward datasets	Accuracy bs=1	Accuracy bs=50
BPoly (polynomial)	99	100	FBarr (barrier)	93	98
BNonPoly (non-poly)	77	87	FLyap (Lyapunov)	81	88

Table 2: **In-domain accuracy of models**. Beam size (bs) 1 and 50.

OUT OF DISTRIBUTION PERFORMANCE

- Test backward on forward, and forward on backward
  - The model cannot cheat
- Forward models do not generalize
- Backward models generalize to polynomial systems, and even (but badly) to barrier systems (a slightly different problem)
- Generalization comes at price: the train and test sets have a different distribution: this hinders model performance

Backward datasets	FLyap	FBarr	Forward datasets	BPoly
BPoly (polynomial)	**73**	35	FBarr (barrier)	15
BNonPoly (non-poly)	**75**	24	FLyap (Lyapunov)	10

Table 3: **Out-of-domain accuracy of models**. Beam size 50. Columns are the test sets.

PRIMING FOR BETTER PERFORMANCE

- Priming (Jelassi et al 2023): to help generalize, add a tiny number of "in distribution" examples
  - Examples from the forward generated datasets
- Strong impact on performance
  - 300 examples (in a training set of one million!) are enough
  - The model learns a related task (barrier Lyapunov function) for (almost) free

Forward datasets	Examples added (1M in training set)	FLyap	FBarr
No mixture	0	73	35
FBarr	30	75	61
	300	83	89
	3,000	85	93
	30,000	89	95
FLyap	10	75	25
	100	82	29
	1,000	83	37
	10,000	86	38

BEATING SOTA

Test sets	SOSTOOL `findlyap`	Existing AI methods Fossil 2	ANLC	LyzNet	Models PolyMixture	FBarr	FLyap	BPoly
FSOSTOOLS	-	32	30	46	**84**	80	53	54
FBarr	-	12	18	28	**93**	-	28	35
FLyap	-	42	32	66	84	**93**	-	73
BPoly	15	10	6	24	**99**	15	10	-

Table 5: **Performance comparison on different test sets**. Beam size 50. PolyMixture is BPoly + 500 FBarr.

While our models exceed the algorithmic state of the art, one might wonder **how they compare to human mathematicians**. To this effect, we proposed 75 problems from the FSOSTOOLS datasets (polynomial systems with 2 or 3 equations) as a examination for 25 first year Masters students in mathematics, following a course on the subject. Each student was given 3 systems chosen at random from FSOSTOOLS and had a total of 30 min. Their performance was 9.33%, significantly lower than our models (84%).

HAVE WE SOLVED AN OPEN PROBLEM?

- Models trained on backward datasets can generalize to generic polynomial systems (that SOSTOOLS can solve)
- Priming improves performance, and allows generalization to related tasks
- We do better than existing AI methods

- But we are only solving polynomial systems, that are already known to be stable (because SOSTOOLS can solve them)

- Tout ça pour ça?

INTO THE WILD

- Generate three test sets of random systems with a local minimum in zero, but not guaranteed to be stable
  - Poly3: polynomial systems of degree 2 and 3
  - Poly5: polynomial systems of degree 2 to 5
  - NonPoly: non polynomial systems
  - SOTA is pretty bad

Test set	Sample size	SOSTOOL `findlyap`	Existing AI methods Fossil 2	ANLC	LyzNet
Poly3	65,215	1.1	0.9	0.6	4.3
Poly5	60,412	0.7	0.3	0.2	2.1
NonPoly	19,746	-	1.0	0.6	3.5

- We expect bad performance: most of those systems will be unstable (we don't know how many)

INTO THE WILD

- Backward primed models achieve 10 to 12% accuracy on random systems (even non polynomials)
- We have no way of knowing how good this is, because we do not know how many systems have Lyapunov functions
- But this is encouraging

Test set	Sample size	SOSTOOL `findlyap`	Existing AI methods Fossil 2	ANLC	LyzNet	Forward model FBarr	Backward model PolyMixture	NonPolyMixture
Poly3	65,215	1.1	0.9	0.6	4.3	11.7	**11.8**	11.2
Poly5	60,412	0.7	0.3	0.2	2.1	8.0	**10.1**	9.9
NonPoly	19,746	-	1.0	0.6	3.5	-	-	**12.7**

Table 6: **Discovering Lyapunov comparison for random systems**. Beam size 50. PolyMixture is BPoly + 500 FBarr. NonPolyMixture is BNonPoly + BPoly + 500 FBarr.

INTO THE WILD

- We can bootstrap: use "wild examples" that the transformer can solve to prime the training set.
- Adding 1000 wild examples to Bpoly, and fine-tuning the model on this dataset (regenerating the test set to prevent contamination) brings performance to 13.5% (vs 11.7) on Poly3 and 11.9 (vs 9.6) on Poly5.

LYAPUNOV: WHAT HAVE WE LEARNED.?

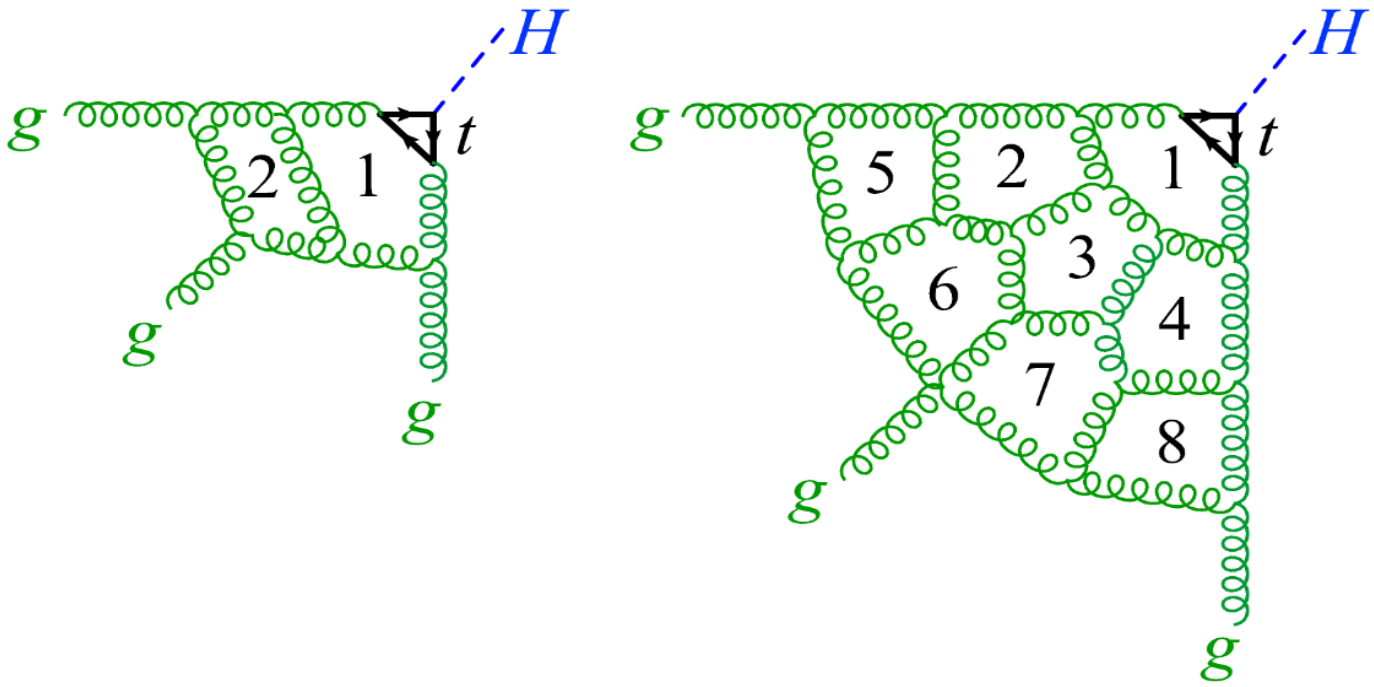
- We can discover solutions of an open problem
  - For random system
  - Next step is "interesting systems"
- Data generation is the hard part, lots of care is needed to prevent information from leaking into the train set
- Models trained on backward data MUST be tested out-of-distribution, in-distribution is irrelevant
- The litmus test for hard problems is "in the wild" test sets, even better: test sets of "interesting open problems"
- Priming, bootstrapping and other RLHF and DPO methods are an obvious next step

SCATTERING AMPLITUDES - TRANSFORMERS FOR THEORETICAL PHYSICS

- Scattering amplitudes: complex functions describing particle interactions
  - Their squared module are probabilities of outcomes
  - They serve as baselines for experimental results
    - $gg \rightarrow Hg$, the main process for Higgs production in the LHC
  - Need to be computed to high precision
- Computed by summing Feynman diagrams of increasing complexity
  - Low precision can be achieved with tree-level diagram
    - All interactions featuring 3 gluons and a Higgs (exactly)
  - For high precision, more complex interactions, where virtual particles are created and annihilated

SCATTERING AMPLITUDES

- Virtual particles result in loops in the Feynman diagram
- One more loop 10x precision on amplitude calculations



SCATTERING AMPLITUDES

- Unfortunately, each virtual particle introduces two latent variables in the amplitude calculation
- At loop 1, dilogarithms, transcendental functions, appear
- At higher loop, recursive polylogarithms, calculations soon become intractable
- At present, most QCD processes are known to two loops, a few to three loops
- Insufficient for future experiments: high luminosity experiments in LHC (a five-fold increase in Higgs production)

BOOTSTRAPPING AMPLITUDES

- Leverage algebraic properties of polylogarithms to predict the structure of the solution
- Amplitudes can be computed from symbols, algebraic structures
  - Known up to a (large) number of integer coefficients
  - That can be computed from symmetry, integrability, limit conditions

- In Planar N=4 supersymmetric Yang-Mills, $gg \rightarrow Hg$ can be computed to 8 loops

  Bootstrapping a stress-tensor form factor through eight loops, Dixon, Gurdogan, McLeod, Wilhelm, 2022, 2204.11901
  - A close relative of the standard model (share the same tree-level amplitudes)
  - Symbols: homogeneous polynomials in 6 non-commutative variables, with integer coefficients
  - Degree 2L at loop L
- Learn a mapping from sequences of 2L letters (abbcddbbff) onto $\mathbb{Z}$

THE THREE GLUON FORM FACTOR

- 3 gluons and a Higgs-like
- Symbols are polynomials in 6 (non commutative) variables a,b,c,d,e,f
  - Loop 3: -4 bccaff + 4 bcbaff + 8 bcafff + ...
- For loop L, 6^{2L} possible keys (ordered sequences of 2L letters)
  - Most coefficients are zero
- Integer coefficients verify relations
  - A giant sudoku game
  - A lot of regularities

L	number of terms
1	6
2	12
3	636
4	11,208
5	263,880
6	4,916,466
7	92,954,568
8	1,671,656,292

TABLE II. Number of terms in the symbol of $F_3^{(L)}$ as a function of the loop order L.

THE SIX LETTER GAME

- Coefficients are invariant by the dihedral symmetry: generated by (a,b,c), (d,e,f), (a,b), (d,e)
- Adjacencies: non-zero keys must
  - Begin with a,b, or c
  - End with d,e,or f
  - Not have adjacent a and d, b and e, c and f, d and e, d and f, e and f
- Relations exist between identical keys up to a few letters ($F^{a,b}$ is the coefficient of a key with a and b adjacent)

$$F^{a,b} + F^{a,c} - F^{b,a} - F^{c,a} = 0, \tag{3.6}$$

$$F^{c,a} + F^{c,b} - F^{a,c} - F^{b,c} = 0, \tag{3.7}$$

$$F^{d,b} - F^{d,c} - F^{b,d} + F^{c,d} + F^{e,c} - F^{e,a} - F^{c,e} + F^{a,e} + F^{f,a} - F^{f,b} - F^{a,f} + F^{b,f} + 4(F^{c,b} - F^{b,c}) = 0, \tag{3.8}$$

TRANSFORMERS FOR BOOTSTRAP

- Can a language model
  - Help predict missing coefficients, from a few that could be calculated?
  - Discover unknown regularities in the symbol? (which may suggest new properties of amplitudes)

- Train an encoder-decoder transformer to predict coefficients (sequences of digits in base 1000) from their keys (sequences of 2L letters)
- Learn from a fraction of a loop, predict the rest
  - Minimizing cross-entropy, a "pure letter game"

EXPERIMENT 1: PREDICTING ZEROES

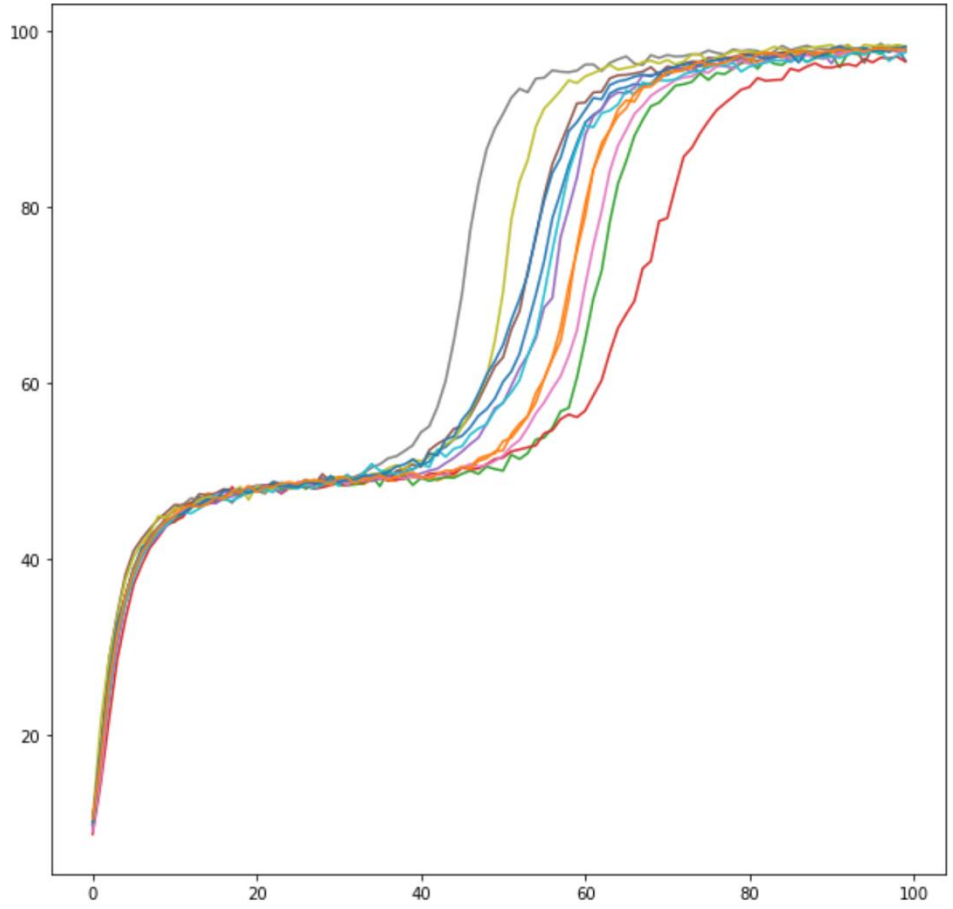
- Given a key, can we predict whether the coefficient is different from zero?
- From a 50/50 training sample of zero and non-zero keys (or the model will always predict zero)

- Loop 5: after training on 300,000 examples (57% of the non-zero keys and as many zero keys), the model predicts 99.96% of test examples (not seen during training)
- Loop 6 : after training on 600,000 examples (6% of the symbol), the model predicts 99.97% of test examples

- Zeroes are easy to predict... But adjacency rules might account for (some of) this

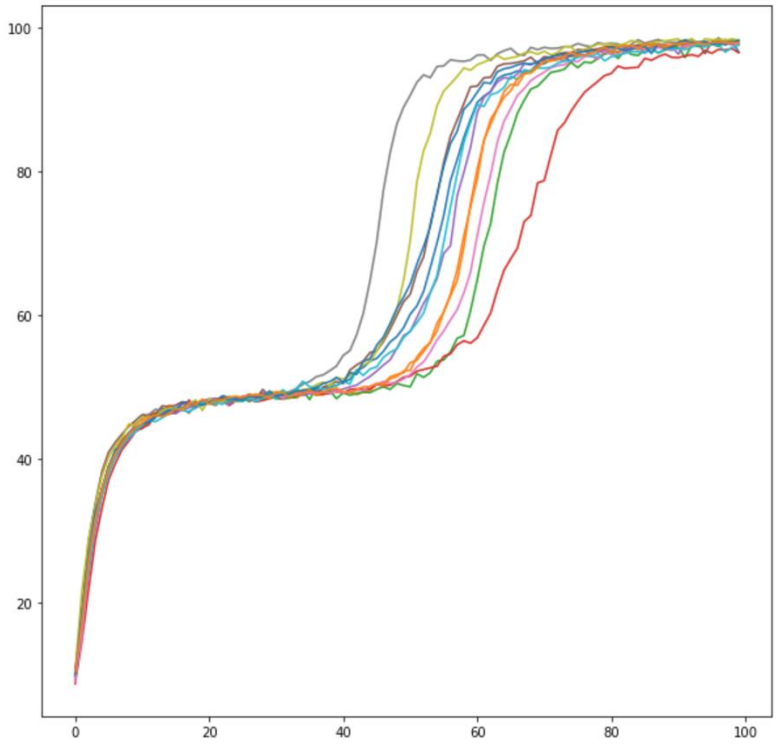
EXPERIMENT 2: PREDICTING NON-ZERO COEFFICIENTS

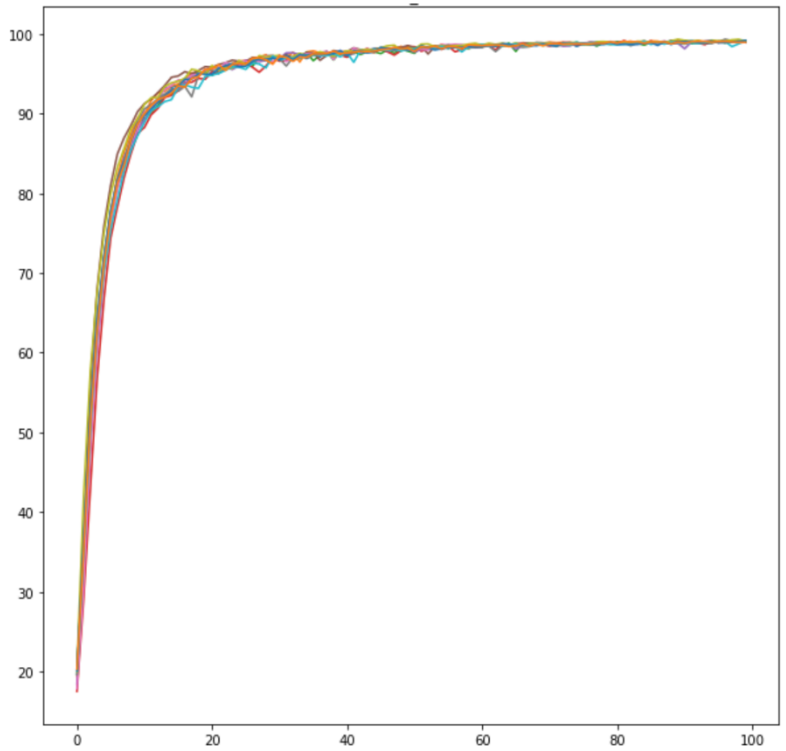
- From keys, sequences of 2L letters, predict coefficients, integers encoded in base 1000
- For loop 5, models trained on 164k examples (62% of the symbol), tested on 100k
  - 99.9% accuracy after 58 epochs of 300k examples
- For loop 6, models trained on 1M examples (20% of the symbol), tested on 100k
  - 98% accuracy after 120 epochs
  - BUT a two step learning curve

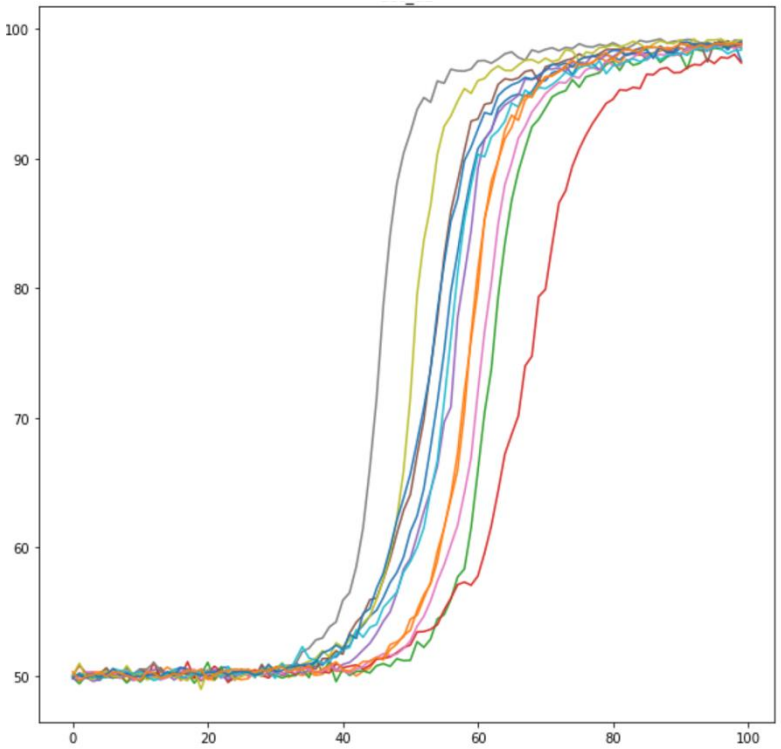


EXPERIMENT 2: PREDICTING NON-ZERO COEFFICIENTS

- Full prediction, magnitude and sign
  - The absolute value is easy to predict, the sign is not







EXPERIMENT 2: PREDICTING NON-ZERO COEFFICIENTS

- Models can learn to predict coefficients, with high accuracy
- Complete a partially calculated loop (provided we have a method for verifying model prediction)

- But known symmetries (dihedral) and relations may account for (some of) these results

EXPERIMENT 3: LEARNING THE NEXT LOOP

- Find a recurrence relation connecting coefficients from loop L-1, to coefficients from loop L
- A loop L key has 2L letters, we can associate it to loop L-1 "parents", by striking out two letters
  - The parents of K=aabd are ~~aa~~bd = bd , ~~aab~~d= ad, ~~aabd~~ = ab, ...
  - Call them P(K), there are L(2L-1) such parents
- Find a generalized recurrence linking the coefficient of K to it parents: E = f(P(K))
  - A generalized Pascal triangle/pyramid (in 6 non-commutative variables)

EXPERIMENT 3: LEARNING THE NEXT LOOP

- Predict loop 6 from loop 5:
  - From 66 integers: loop 5 coefficients
  - Predict 1 integer: the loop 6 coefficient
  - (NOT the keys: we already know the model can predict coefficients from keys)
- 98.1% accuracy, no difference between sign (98.4) and magnitude (99.6) accuracy
- A function f certainly exists (but do not know what it is)

EXPERIMENT 3: LEARNING THE NEXT LOOP

- We can learn about the unknown recurrence, by removing parents:
  - Only considering strike-outs of contiguous (or close apart) positions
  - k max distance for strike out : k=1 contiguous letters only, the smaller k; the less parents
  - Limited impact on performance for k larger than 1

	Accuracy	Magnitude accuracy	Sign accuracy
Strike two, all parents	98.1	98.4	99.6
Strike two, k=5	98.3	98.6	99.7
Strike two, k=3	98.4	98.7	99.7
Strike two, k=2	98.1	98.3	99.5
Strike two, k=1	94.3	95.2	98.5

EXPERIMENT 3: LEARNING THE NEXT LOOP

- Shuffling/sorting parents have little impact: the recurrence is almost permutation invariant
- Coupling between parent and child signs, and magnitudes

	Accuracy	Magnitude accuracy	Sign accuracy
Strike two, all parents	98.1	98.4	99.6
Strike two, k=5	98.3	98.6	99.7
Strike two, k=3	98.4	98.7	99.7
Strike two, k=2	98.1	98.3	99.5
Strike two, k=1	94.3	95.2	98.5
Shuffled parents	95.2	99.1	96.3
Shuffled parents, k=2	93.5	98.1	95.0
Sorted parents, k=5	93.9	95.4	97.9
Parent signs only	93.3	93.5	99.0
Parent magnitudes only	81.8	98.4	83.2

Table 2: **Global, magnitude and sign accuracy.** Best of four models, trained for about 500 epochs

THE SIX LETTER GAME REVISITED

- Since zeros are so easy to predict, there must be a general rule for adjacent zero keys
- Generalized end-rule: keys ending with a single letter d, e or f must be preceded with a run of a, b or c
  - * aaaaf can be non zero
  - * abbaf must be zero
  - Accounts for 92% of adjacent zeroes

THE SIX LETTER GAME REVISITED

- Since models can find relations between elements and their strike out parents exist, we could go looking for such empirical relations
  - Rays: sequences of keys of different loops, related by a "common strikeout pattern",
    - af, aaaf, aaaaaf, ..., or af, afff, afffff, ...
  - Closed recurrences can be found, coefficients of sequences ending with a variable length run of f verify

$$c_L(\mathtt{seq_7f\ldots f}) = p_L(\mathtt{seq_7}) \times (-1)^L 2^{2L-8}(2L-9)!!\,, \tag{5.5}$$

  - With

$$p_L(\mathtt{aaf\ldots f}) = 0, \tag{5.6}$$
$$p_L(\mathtt{caaf\ldots f}) = 32(L-2)(2L-5)(2L-7)\,, \tag{5.7}$$
$$p_L(\mathtt{caaaf\ldots f}) = \frac{16}{3}(4L-9)(2L-5)(2L-7)\,, \tag{5.8}$$
$$p_L(\mathtt{ccaaaaf\ldots f}) = -\frac{4}{5}(2L-7)(7L^2+22L-140)\,, \tag{5.9}$$
$$p_L(\mathtt{cccccaaf\ldots f}) = -\frac{8}{3}(L-4)(L^2-47L+135)\,, \tag{5.10}$$
$$p_L(\mathtt{bdddbbbf\ldots f}) = -\frac{2}{45}(163L^3-2220L^2+15977L-36660)\,. \tag{5.11}$$

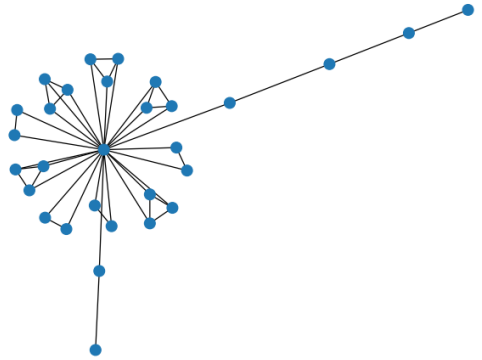
FINDING COUNTER-EXAMPLES IN GRAPH THEORY

Constructions in combinatorics via neural networks, Wagner 2021, 2104.14516

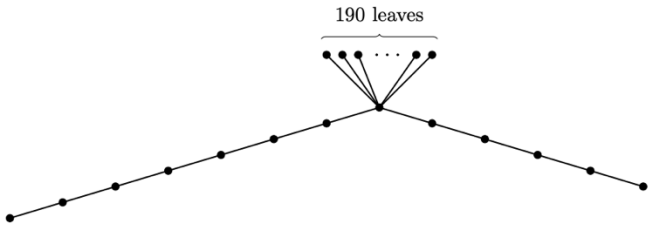
Conjecture (Aouchiche-Hansen 2011): Let G be a connected graph, with n ≥ 4 vertices, diameter (max distance between vertices) D, proximity (average distance between nodes) π and spectral distance (eigenvalues of distance matrix) $\partial_1 \geq \partial_2 \geq \ldots \geq \partial_n$,

Then $\pi + \partial \left\lfloor \frac{2D}{3} \right\rfloor > 0$

Train a model to find counter-examples, it fails, but all failed solutions follow a certain pattern



That a mathematician can turn into a valid counter-example



DISCOVERING OPTIMAL CONSTRUCTIONS

PatternBoost: constructions in mathematics with a little help from AI
Charton, Ellenberg, Wagner, Williamson 2024, 2411.00566

- Finding discrete objects that maximize a quantity:
  - Largest graphs with n nodes, but no cycle of 4
  - Largest set of points on a n^3 grid, with no 5 points on a sphere
  - Smallest subset of the d-dimensional hypercube, with diameter d
- By letting a model generate candidate solutions

PATTERNBOOST: THE INTUITION

- For a given problem, promising solutions share a number of features, or patterns, that a model could learn to imitate
  - Train a model to predict the next token of good solutions
  - Use the trained model to generate more good solutions
  - Feed the best of these generated solution into the model
- A risk: model collapse: the performance of models trained on their own output tend to degrade with time Training on generated data makes models forget, Shumailov 2023)
  - This can be prevented by adding verification to the generated data (Beyond Model Collapse, Feng 2024)
  - Use a local search algorithm, to improve model generated solutions

A GENETIC ALGORITHM, WITH A TRANSFORMER IN THE MIDDLE

- Randomly generate a population of candidates, improve via local search, keep the best
- Train a decoder only transformer: Makemore (Karpathy 2023) an implementation of GPT-2
- Generate a new population, improve via local search, keep the best
- Use these to fine-tune the transformer
- Rinse, repeat

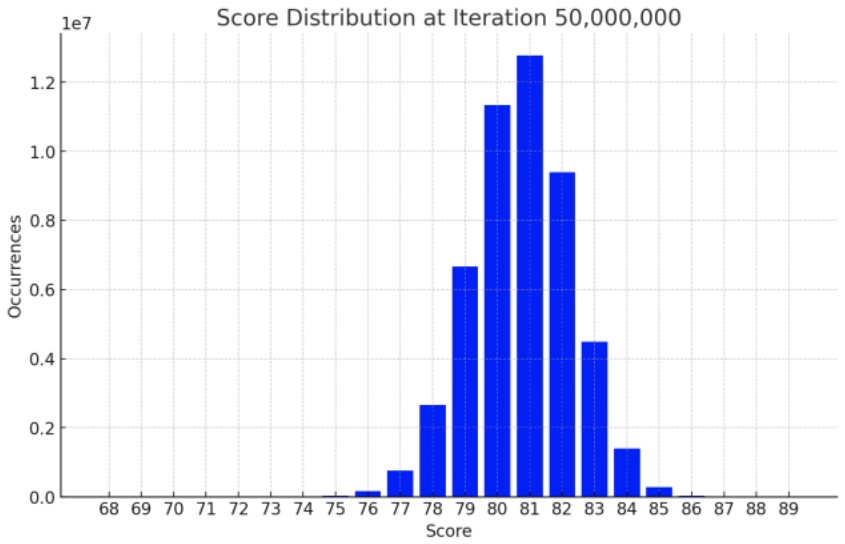
- A genetic algorithm, with the mutation/crossover operators replaced by a generative model

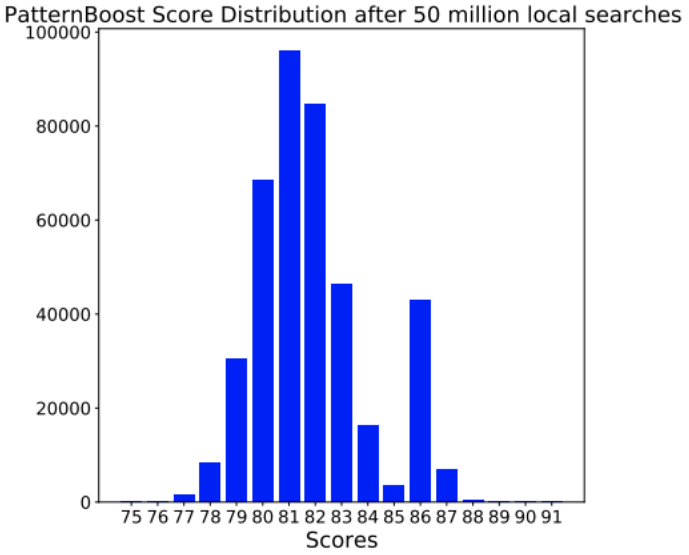
PATTERNBOOST: THE NO SQUARE PROBLEM

- Constructing graphs, with N vertices, without 4 cycles
- A well-studied problem, for which problem-specific methods have been designed
  - Solutions are known for N up to 40
- PatternBoost solves problems up to N=33 (best solution 96)

NO SQUARE PROBLEM

- After 50 million local searches, a "classic" local search only method generates a unimodal distribution of candidates, centered around 81
- With PatternBoost, a second mode, appears





NO SQUARE PROBLEM

- The best solutions are consistently found by the models that "learn best": generate the largest solutions
  - Not a lucky guess

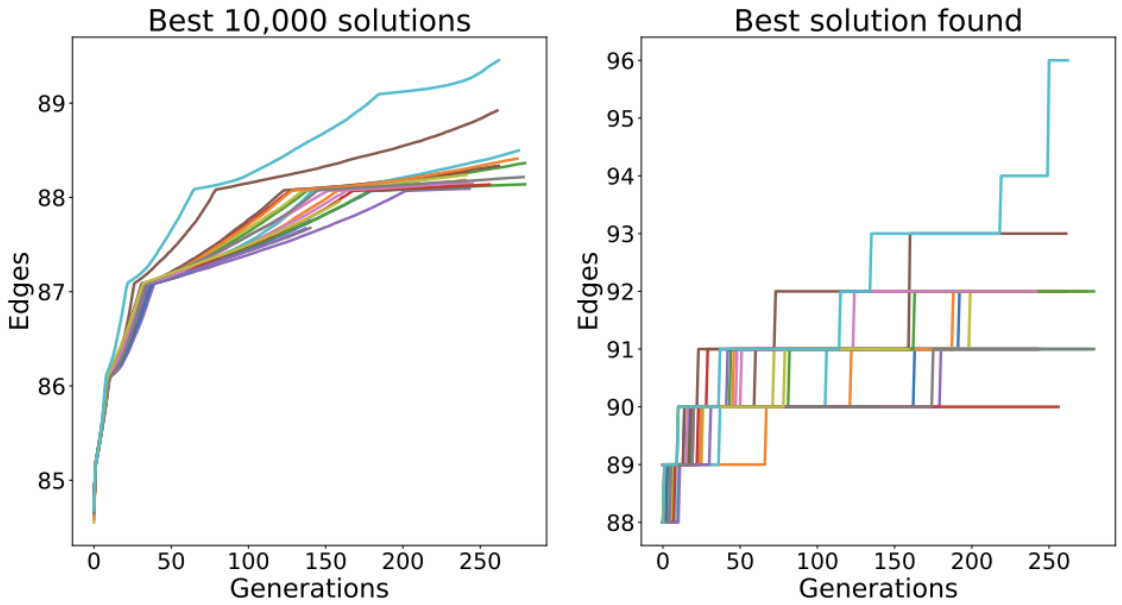


Figure 7: Evolution of scores over generations, for 20 different models. All models have 6 layers, and 128 dimensions, and only differ by their initialization. Left: average score of the 10,000 best solutions found so far. Right: best solution found so far.

NO SQUARE PROBLEM

- How graphs are tokenized, model architecture and size, play a role
  - Deserves further study

Edges	Local search	PatternBoost	
		2 layers 16 dimensions	4 layers 128 dimensions improved tokenizer
91	0	1	15
90	0	0	206
89	5	21	2,529
88	204	454	26,842
87	3,425	6,957	175,089

Table 2: **No-squares problem,** $n = 33$. Distribution of the number of edges of candidate solutions, after 50 million local searches.

DISCOVERING OPTIMAL CONSTRUCTIONS

- Competitive on hard problems, like no square graphs
- Found hitherto unknown no-sphere solutions for n=6 (best known was 17, we found 18)
- Solved a 30 years-old conjecture about d-hypercubes with diameter d

CONCLUSIONS

- Transformers can be used to solve hard problems of symbolic and discrete mathematics
- A developing field, where much remains to be discovered
- Data distribution matters, generalization can be an issue
- The future belongs to hybrid systems: a mixture of classical algorithms and language models