Problem solving as a translation task

François CHARTON, Meta AI

Mathematics as translation

- Train models to translate problems, encoded as sentences in some language, into their solutions
  - 7+9 => 16
  - x^2-x-1 => $\frac{1+\sqrt{5}}{2}, \frac{1-\sqrt{5}}{2}$

Maths as translation: learning GCD

- Two integers a=10, b=32, and their GCD gcd(a,b)=2
- Can be encoded as sequences of digits in base 10:
  - `‘+’, ‘1’, ‘0’`
  - `‘+’, ‘3’, ‘2’`
  - `‘+’, ‘2’`
- Translate `‘+’,‘1’,‘0’,‘+’,‘3’,‘2’` into ‘+’, ‘2’
  - From examples only
  - As a “pure language” problem: the model knows no maths

This works!

- Symbolic integration / Solving ODE:
    - Deep learning for symbolic mathematics (2020): Lample & Charton (ArXiv 1912.01412)
    https://arxiv.org/abs/1912.01412
- Dynamical systems:
    - Learning advanced computations from examples (2021) : Charton, Hayat & Lample (ArXiv 2006.06462)
    - Discovering Lyapunov functions with transformers (2023) : Alfarano, Charton, Hayat (3rd MATH&AI workshop, NeurIPS)
- Symbolic regression:
    - Deep symbolic regression for recurrent sequences (2022) : d'Ascoli, Kamienny, Lample, Charton (ArXiv 2201.04600)
    - End-to-end symbolic regression with transformers (2022) : Kamienny, d'Ascoli, Lample, Charton (ArXiv 2204.10532)
- Cryptanalysis of post-quantum cryptography:
    - SALSA: attacking lattice cryptography with transformers (2022): Wenger, Chen, Charton, Lauter (ArXiv 2207.04785)
    - SALSA PICANTE (2023) Li, Sotakova, Wenger, Mahlou, Garcelon, Charton, Lauter (ArXiv 2303.0478)
    - SALSA VERDE (2023) Li, Wenger, Zhu, Charton, Lauter (ArXiv 2306.11641)
- Theoretical physics
    - Transformers for scattering amplitudes (2023): Merz, Cai, Charton, Nolte, Wilhelm, Cranmer, Dixon (ML4PS Workshop, NeurIPS)
- Quantum computing
    - Using transformer to simplify ZX diagrams (2023) (3rd MATH&AI Workshop, NeurIPS)

Why do this?

- A challenge: like Go, like Chess
  - "When computers can do XXX, we will have artificial intelligence"
- A pre-requisite for AI for Science
  - No maths no science
- A framework for understanding transformers
  - and deep learning
  - and perhaps some science

Problem solving as translation

I. Symbolic integration: an initial example

II. Scattering amplitudes: an application to theoretical physics

III. Eigenvalues and GCD: robutsness and explainability

Deep learning for symbolic mathematics (2019)

- Train transformer to compute symbolic integrals

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

- Undergrad mathematics
- Not trivial for mathematicians
- Hard for machines (Risch algorithm)

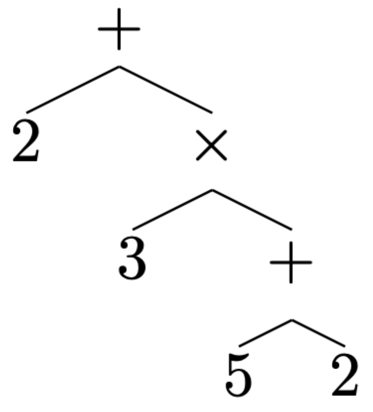
Deep learning for symbolic mathematics (2019)

- Three steps

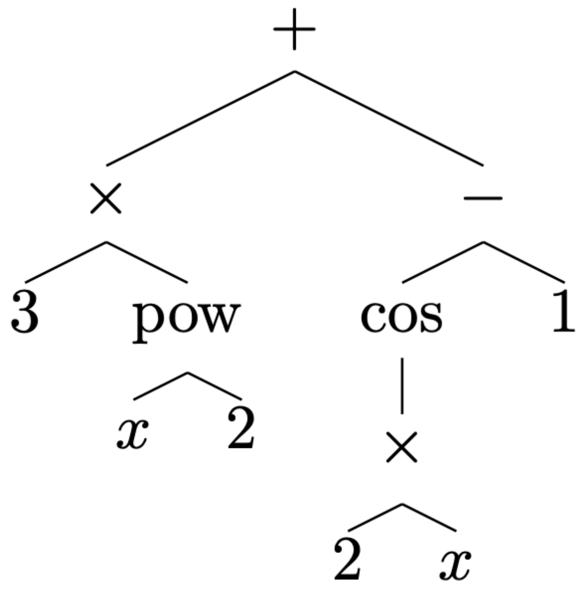
- Represent problems and solutions as sequences
- Generate large sets of problems and solutions
- Train transformers to translate problems into solutions

Expressions as trees

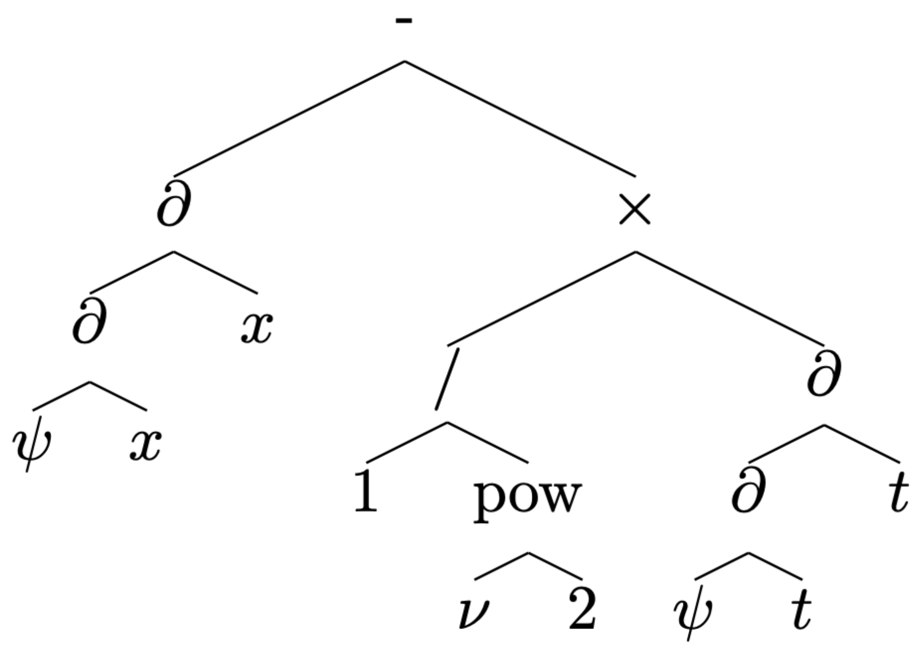
$$2 + 3 \times (5 + 2)$$



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



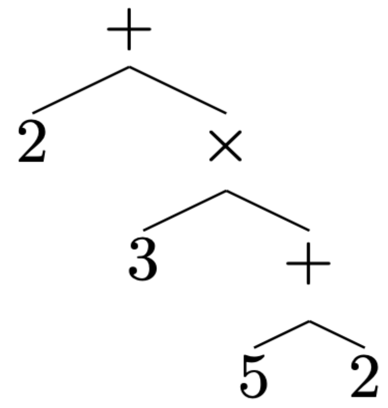
$$\frac{\partial^2 \psi}{\partial x^2} - \frac{1}{\nu^2}\frac{\partial^2 \psi}{\partial t^2}$$



Trees as sequences

- Polish notation (aka pre-order enumeration)
  - Begin from root
  - Parent before child
  - Children from left to right

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



+ 2 x 3 + 5 2

Expressions as sequences

Ready for the transformer!

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

Generating data – three approaches

- Forward
  - Generate a random function f
  - Compute its integral F
- Backward
  - Generate a random function F
  - Compute its derivative f
- Integration by part
  - Generate random functions F and G
  - Compute their derivative f and g
  - If fG is in the dataset, we get Fg for free using $\int Fg = FG - \int fG$

Generating data

- How to generate a random function?
  - Generate a random tree
  - Randomly select operators for its nodes
  - Constants and small integers for its leaves
- Why three training sets?
  - Different generating procedures explore different parts of the problem space

Generating data

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

Generating data

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

Generating data

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

Three training sets

- Expressions with up to 15 operators
- Operators are the 4 basic operation (+-*%), and elementary functions (Liouville): exp, log, sqrt, pow, sin, cos, tan, sinh, cosh, tanh and their inverses
- Coefficients are integers between -5 and 5

	Forward	Backward	Integration by parts
Training set size	20M	40M	20M
Input length	18.9±6.9	70.2±47.8	17.5±9.1
Output length	49.6±48.3	21.3±8.3	26.4±11.3
Length ratio	2.7	0.4	2.0
Input max length	69	450	226
Output max length	508	75	206

Training models

- 6-layer encoder-decoder transformers with 256 dimensions and 8 attention heads
- The model is trained on generated data
  - Supervised learning, minimizing cross-entropy
  - A pure language task: the model has no understanding of maths
- Tested on held-out data (i.e. not seen during training)
- Solutions are verified with an external tool (SymPy)
  - Using problem-related mathematical metrics

In-domain results

- Performance on held-out test sets with the same distribution as training
- Almost 100% no matter the generation procedure
- Outperforms best computer algebras

	Integration (FWD)	Integration (BWD)	Integration (IBP)
Beam size 1	93.6	98.4	96.8
Beam size 10	95.6	99.4	99.2
Beam size 50	96.2	99.7	99.5

	Integration (BWD)
Mathematica (30s)	84.0
Matlab	65.2
Maple	67.4

Limitations : distribution woes

- Generated data: training and test examples come from the same generator
- What if they don't?

Training data	Forward (FWD)			Backward (BWD)		
	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
BWD	18.9	24.6	27.5	98.4	99.4	99.7

Generating data

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

Generating data

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

Distribution woes

Training data	Forward (FWD) Beam 1	Beam 10	Beam 50	Backward (BWD) Beam 1	Beam 10	Beam 50	Integration by parts (IBP) 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

- IPB stands “in-between” FWD and BWD: better generalization
- Training distribution matters
- Out-of-distribution generalization is possible so long test distribution is not ‘too far’

Take aways

- Symbolic mathematics can be learned from examples only
- In-domain, we achieve comparable performance with computer algebras (Mathematica)

- Out of distribution generalization is hard
- Training distribution matters

Transformers for scattering amplitudes (2023)

(Cai, Merz, Nolte, Wilhelm, Cranmer, Dixon, Charton)

- Scattering amplitudes: complex functions predicting the outcome of particle interactions
- Computed by summing Feynman diagrams of increasing complexity
  - loops: virtual particles created and destroyed in the process
- A hard problem: each loop introduces two latent variables, their integration give rise to generalized polylogarithms
  - For the standard model the best computational techniques only reach loop 3

Amplitude bootstrap (Dixon, Wilhelm)

- Polylogarithms have many algebraic properties
  - Leverage them to predict the structure of the solution, up to some coefficients
  - Compute the coefficients from symmetry consideration, known limit values, etc.
- In Planar N=4 supersymmetric Yang-Mills, solutions are “simple”
  - Calculated from symbols: homogeneous polynomials, degree 2L (L=loop), with integer coefficients

The three-gluon form factor

- Three gluons and a massless Higgs
- Loop symbols are homogeneous polynomials of degree 2L
  - in six (non commutative) variables: a,b,c,d,e,f
  - with integer coefficients, most of them zero
  - 16 aabddd + 48 aabbff - 12 abcece + ...
- Symmetries and asymptotic properties translate into constraints:
  - An gigantic integer programming problem
  - Lots of regularities in the symbol
- Can transformers help?

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.

Experiment 1 : Predicting zeroes

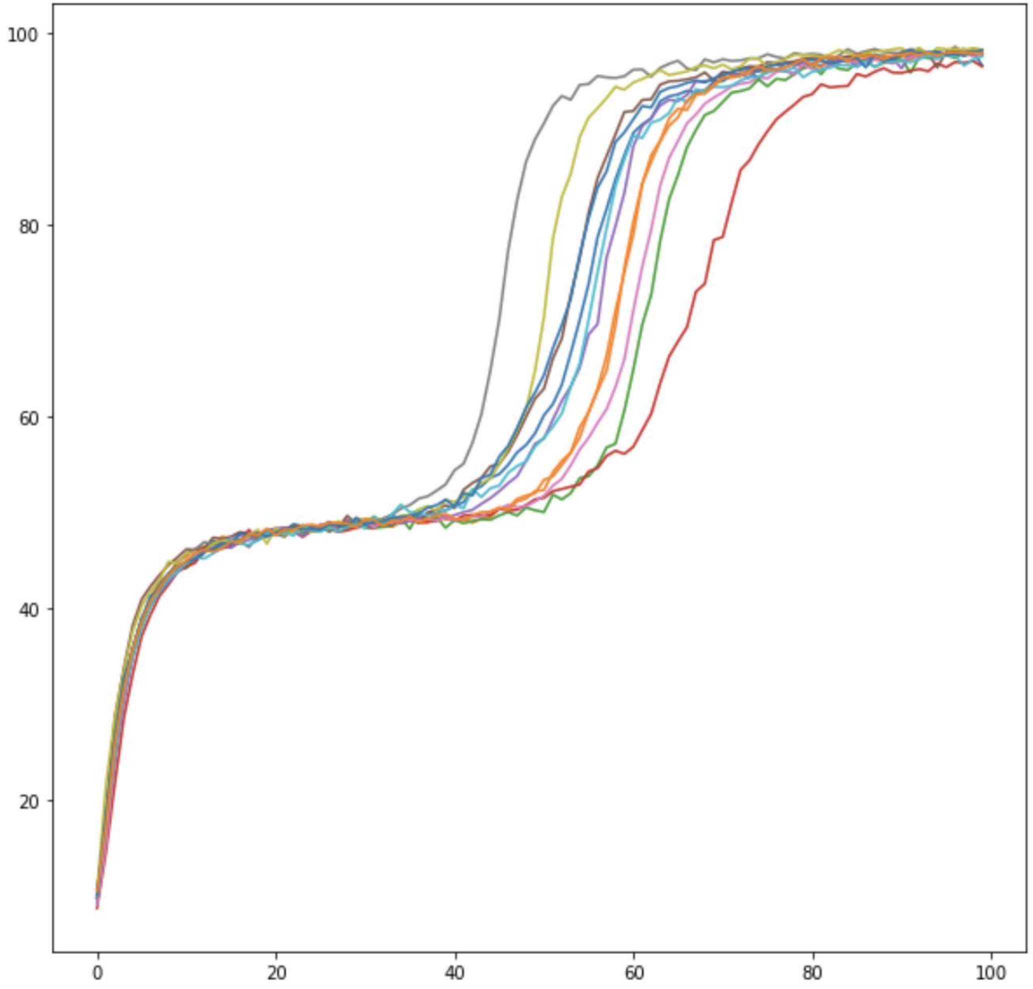
- For Loop 5 and 6, predict whether a term is zero or nonzero
  - afdcfdadfe is zero
  - aaaeeceaaf is not
- Build a 50/50 training sample of zero/non zero terms
- Reserve 10k terms for test, they will not be seen at training
- Train the model, and measure performance on the test set (% of correct prediction)
  - For input a,f,d,c,f,d,a,d,f,e predict 0
  - For input a,a,a,e,e,c,e,a,a,f predict 1

Experiment 1 : Predicting zeroes

- Loop 5 : after training on 300,000 examples (57% of the symbol), the model predict 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

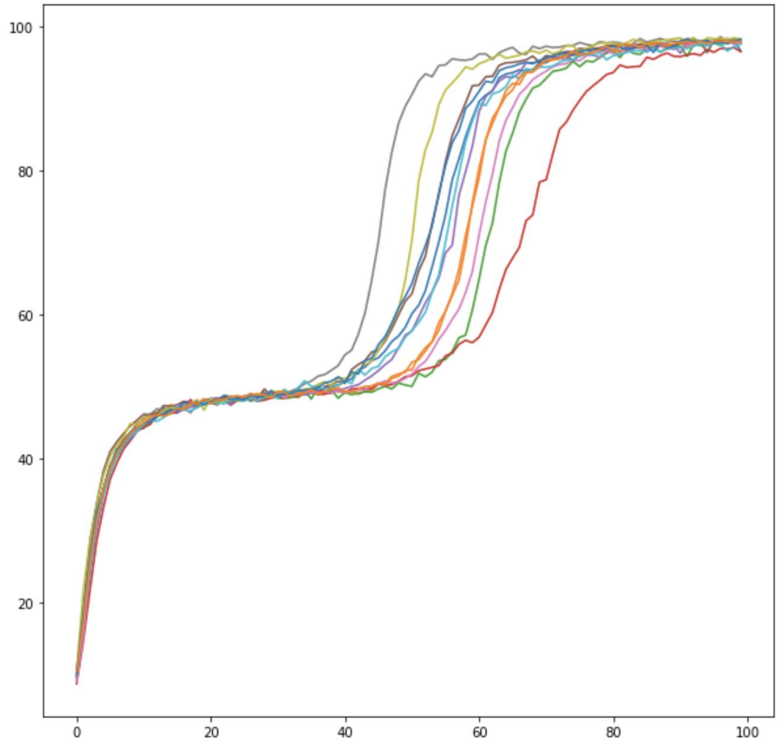
Experiment 2 : Predicting non-zeroes

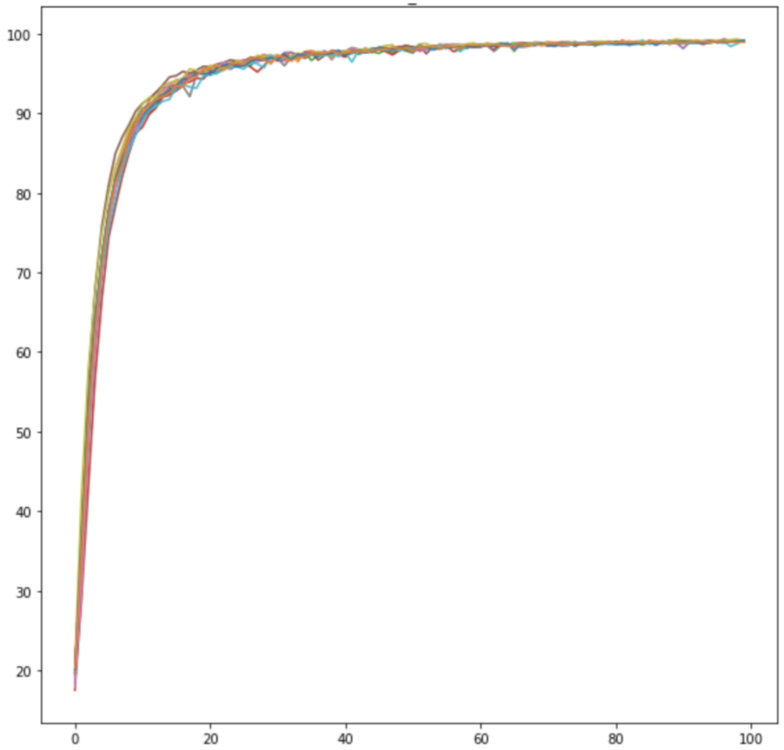
- From keys, sequences of 2L letters, predict coefficients, integers encoded in base 1000
- For loop 5, models trained on 164k examples (62% of the symbol), tested on 100k
  - 99.9% accuracy after 58 epochs of 300k examples
- For loop 6, models trained on 1M examples (20% of the symbol), tested on 100k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

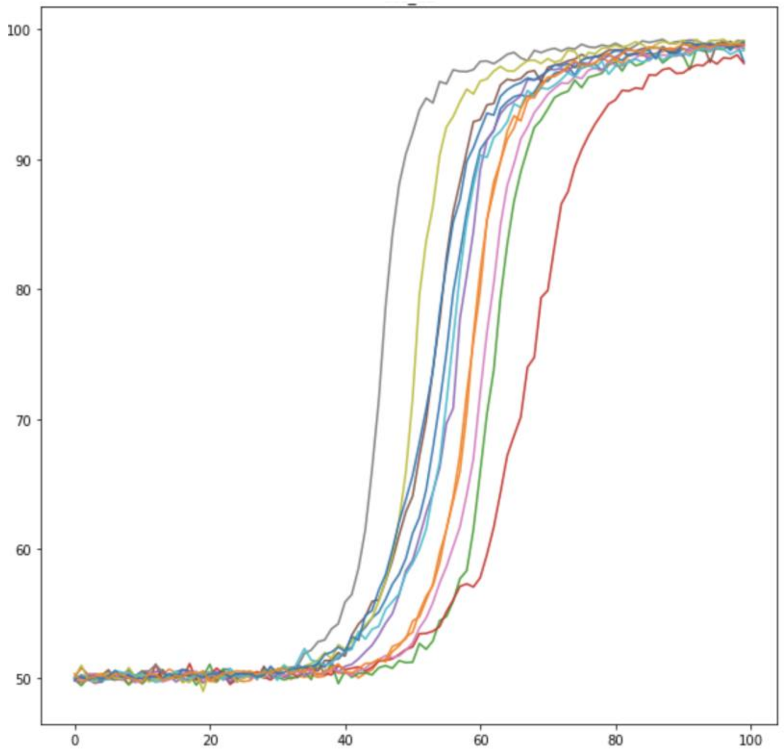


Experiment 2 : Predicting non-zeroes

- full prediction, magnitude and sign





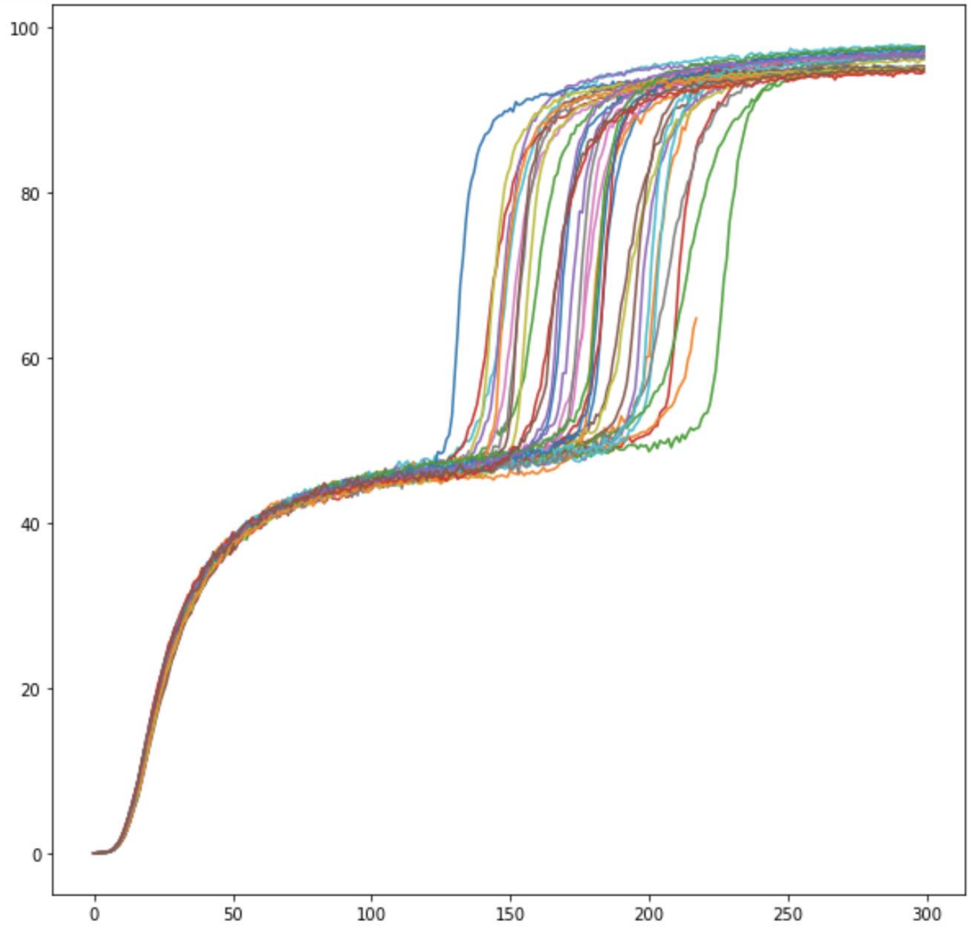


Experiment 3 : Learning with less symmetries

- Non zero coefficients
  - Must begin with a,b,c and end with d,e,f
  - Are invariant by dihedral symmetry
  - Cannot have a next to d (b next to e, c next to f)
  - Cannot have d next to e or f (e next to d or f)
- Only a few endings are possible:
  - 8 “quads” (4 letter endings, up to cyclic symmetry (a,b,c), (d,e,f))
  - 93 octuples

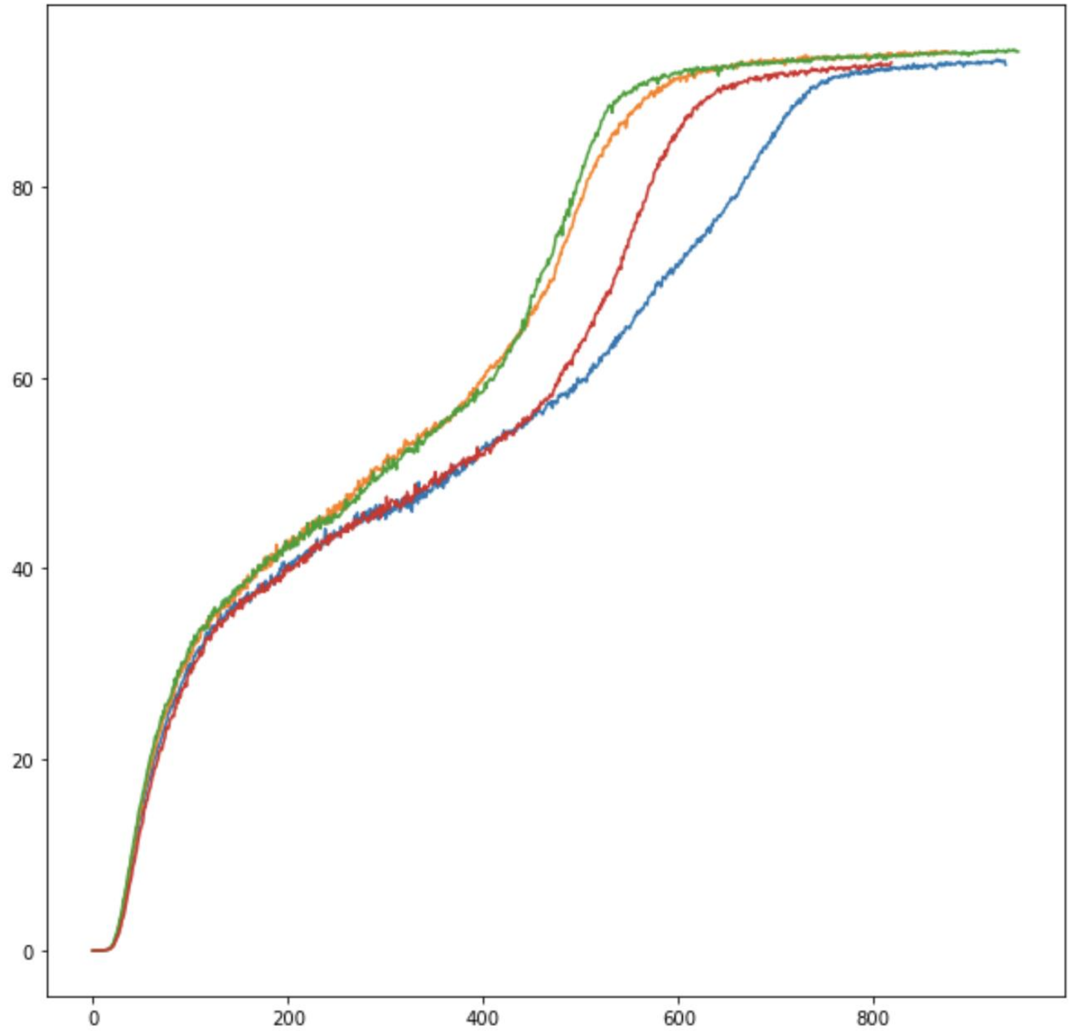
Experiment 3 : Learning loop 7 quads

- 7.3 million elements in the symbol (vs 93 millions in full representation)
- Models learn to predict with 98% accuracy
- Same “two step” shape



Experiment 3 : Learning loop 8 octuples

- 5.6 million elements in the symbol (vs 1.7 billions in full representation)
- Models learn to predict with 94% accuracy
- Attenuated “two step” shape
- Slower learning (600 epochs, vs 200 for quads, and 70 for full representation)



Take aways from experiments 1-3

- We can use transformers to complete partially calculated loops
- Coefficients are learned with high accuracy
  - Even when only a small part of the symbol is available
- A few unintuitive observations happen:
  - hardness of learning the sign
  - might shed new light on the underlying phenomenon

Experiment 4: predicting the next loop

- A loop L element E is a sequence of 2L letters
- Strike out 2 of the 2L letters
  - From aabd make bd, ad, ab...
  - There are L(2L-1) parents, call them P(E)
- Try to find a recurrence relation, that predicts the coefficient of E from its parents: E = f(P(E))
  - A generalized Pascal triangle/pyramid (in 6 non-commutative variables)

- 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 we have no idea what it is)

Experiment 4: understanding the recurrence

- To collect information on f, the unknown recurrence, we could
  - Remove information about the parents
  - See if the model still learns
- Can we use less parents?
  - Only strike letters at most k tokens apart; e.g. k=1 only consecutive tokens
  - k=2: 21 parents, k=1: 11 parents

	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 4: understanding the recurrence

- Shuffling/sorting the parents do not prevent learning
- Coupling between parent/children 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

Next steps

- Better understanding the recurrence relation
  - Try building loop 9, or loops for related problems
- Discovering local properties/symmetries in the symbol
  - Symbols were calculated by exploiting known symmetries in nature
  - If we discover new regularities in the symbols, what does is tell us about nature?
  - Antipodal symmetries

Questions for transformers

- Do they learn really learn maths?
  - Or are they learning shortcuts, i.e. parroting statistical patterns
- Are the failures predictable and principled?
  - Or do models confabulate, and fail at random
- Can their predictions be explained?
  - Or are they black boxes?
- Training data are generated, what is the impact their distribution?

Linear algebra with transformers
(Charton 2021)

- Basic linear algebra is learned, with small models
  - Transposition: 100% accuracy, up to 30x30 matrices, with 1-layer transformers
  - Addition: 99% accuracy, up to 20x20 matrices, 2-layer transformers
  - Matrix-vector product: 100% accuracy, up to 10x10 matrices, 2-layer transformers
  - Multiplication: 100% accuracy, 5x5 matrices, 1 / 4 layer transformers

- Advanced tasks can also be learned
  - Eigenvalues: 100% accuracy for 5x5 to 20x20 matrices
  - Eigen decomposition: 97% for 5x5, 82% for 6x6 matrices
  - SVD decomposition: 99% accuracy for 4x4 matrices
  - Matrix inversion: 90% for 5x5 matrices

Learning to diagonalize

- Given a symmetric matrix M
- Find a vector D and a matrix H such that

  $HMH^T = HMH^{-1} = \mathrm{diag}(D)$
- From examples only, i.e. triplets (M, D, H)

- We know from theory (spectral theorem):
  - That D are the eigenvalues (unique up to a permutation)
  - That H is unitary, and its rows and columns
    - are orthogonal
    - have unit norm

Learning to diagonalize

- Train a model to 92% accuracy, test it on 100 000 matrices
  - 92 000 correct predictions, 8 000 errors

- In all test cases but 6 (99.99%), the eigenvalues are predicted with less than 1% error
- In 98.9% of test cases, all rows and columns of H have unit norm

- The two properties of diagonalization are respected even when the model fails
- Some maths have been learned

Learning the spectral theorem

- These results hold early in training:
  - With a half-trained model, with 70% accuracy
  - eigenvalues are correct in 99.6% of test cases,
  - rows and columns of H have unit norms in 96.7%.
- For harder cases, on 6x6 matrices, a model only achieves 43% accuracy, yet
  - eigenvalues are correct in 99.6% of test cases
  - rows and columns of H have unit norms in 93.1%

- No hallucination: the model always remains “roughly right”

Understanding model failures

- Almost all failures are due to rows and columns of H not being quite orthogonal
- Eigenvalues, and the norm of eigenvectors are always learned
- The model does not output absurd solutions (aka hallucinations)

- Error can be predicted from the condition number of H (ratio of its extreme singular values), which should be 1.
  - $c(H) > 1.045$ predicts 99.3% of model outcomes (99.9% of successes, and 96.7% of failures)

Understanding model failures - Matrix inversion

- Given a matrix M, predict P such that MP≈Id
  - M is invertible, so M^{-1} always exists
- Models struggle to achieve more than 90% accuracy
  - They predict $P \approx M^{-1}$, but we don't have MP≈Id
  - Ill-conditioned matrices: a typical difficulty with this task
- The condition number of M (c(M)>66) predicts 98% of model outcomes (only the input is needed, no need to run the model)
- Our models fail for good mathematical reasons
- Failures are principled and predictable

Computing eigenvalues – out-of-distribution results

- Models are trained on symmetric matrices with independent coefficients
- Wigner Matrices: eigenvalues are distributed as a semi-circle
  - Symmetric around 0
  - Variance depends on coefficient varianc and matrix dimension
  - Bounded support

- Can we generalize to non-Wigner matrices?

Eigenvalues – out-of-distribution generalization

	Semi-circle	Uniform	Gaussian	Laplace	abs-sc	abs-Lapl	Marchenko
Semi-circle	100	34	36	39	1	5	0
Uniform	93	100	76	70	92	70	2
Gaussian	100	100	100	100	100	100	99
Laplace	100	100	100	100	100	100	100
Abs-semicircle	0	5	4	4	100	78	20
Abs-Laplace	0	4	5	5	100	100	100
Marchenko-Pastur	0	4	4	4	100	76	100

Table 1: **Out-of-distribution generalization. Eigenvalues of 5x5 matrices**. Rows are the training distributions, columns the test distributions.

- Gauss and Laplace generalize to Wigner (but not the other way around)
- Can generalize far away from training distribution: to positive definite matrices

Eigenvalues – out-of-distribution generalization

- Robust distributions learn faster

	Semi-circle	Uniform	Gaussian	Laplace	abs-sc	abs-Lapl	Marchenko
8x8 matrices							
Semicircle	0	0	0	0	0	0	0
Uniform	91	100	65	57	89	55	0
Gaussian	100	100	100	99	100	99	41
Laplace	100	100	100	100	100	100	97
Abs-semicircle	0	1	1	0	100	53	0
Abs-Laplace	0	1	1	1	100	100	98
Marchenko-Pastur	0	0	0	0	1	1	20
10x10 matrices							
Gaussian (12/1 layers)	100	100	100	98	100	97	3
Laplace (8/1 layers)	100	100	100	100	100	100	74

Table 2: **Out-of-distribution generalization. Eigenvalues of 8x8 and 10x10 matrices, accuracy after 36 million examples.** Rows are the training distributions, columns the test distributions.

Take aways

- The underlying mathematics are sometimes learned
  - You need to investigate failures
- Out-of-distribution generalization is possible
- Special ”robust” distributions exist
  - Allow for faster learning
  - Seem problem independent

Can transformers learn greatest common divisor?
(Charton 2023)

- Train a model on sequences of 4 integers, a,b,c,d
  - It can learn to predict if $a/b < c/d$ with 100% accuracy, after just a few examples
  - It will never learn to compute $a/b+c/d$, or ac/bd
  - It cannot even learn to simplify a/b
- Can a transformer learn to compute GCD?

Learning the greatest common divisor

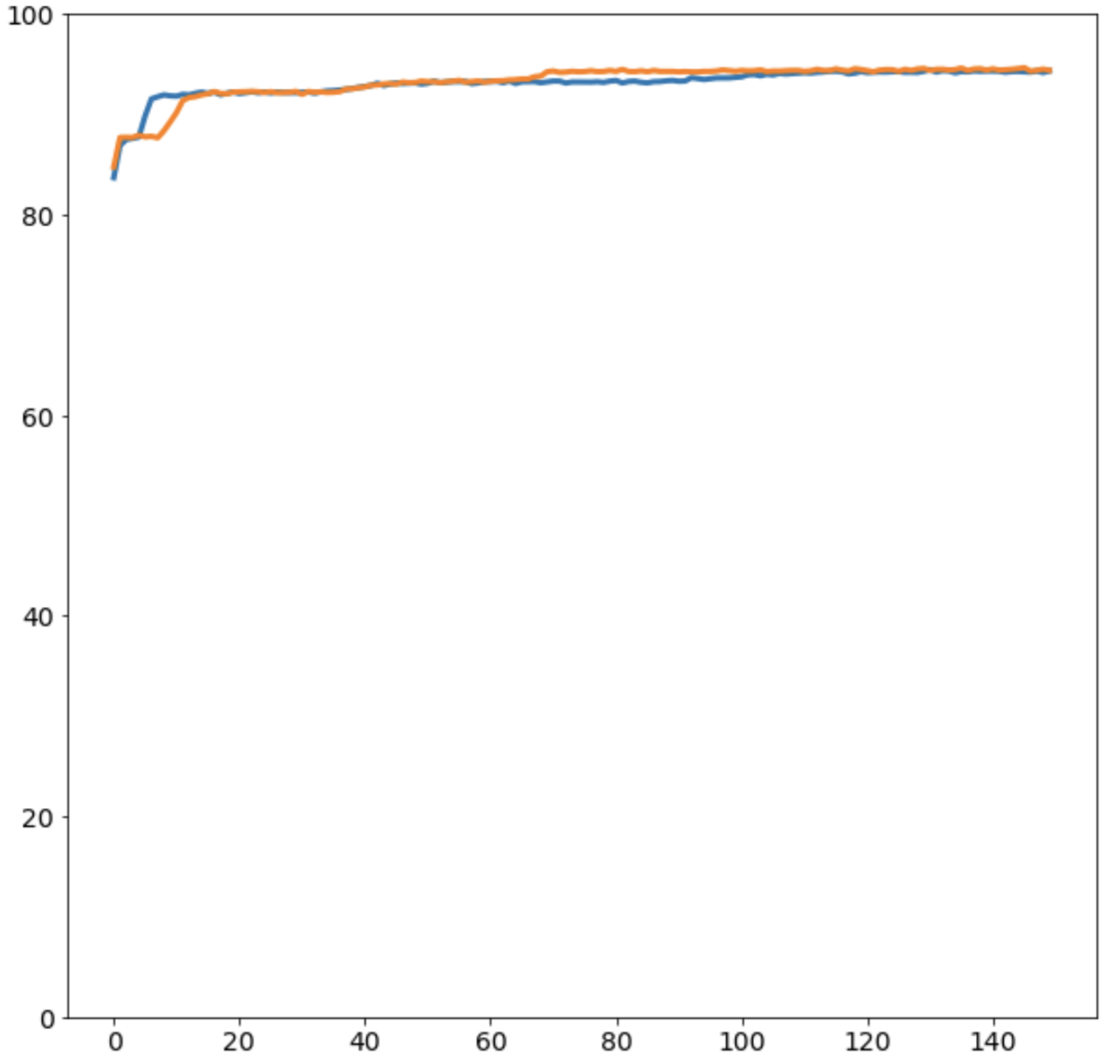
- Generate random pairs of integers between 1 and 1,000,000
- Compute their gcd, train a model to predict it
- Test on a held-out dataset (100k examples)

- Problem space size: 10^{12} , no chance that the model memorizes all the cases
- Uniform inputs, no training distribution specificity to exploit

Learning the greatest common divisor

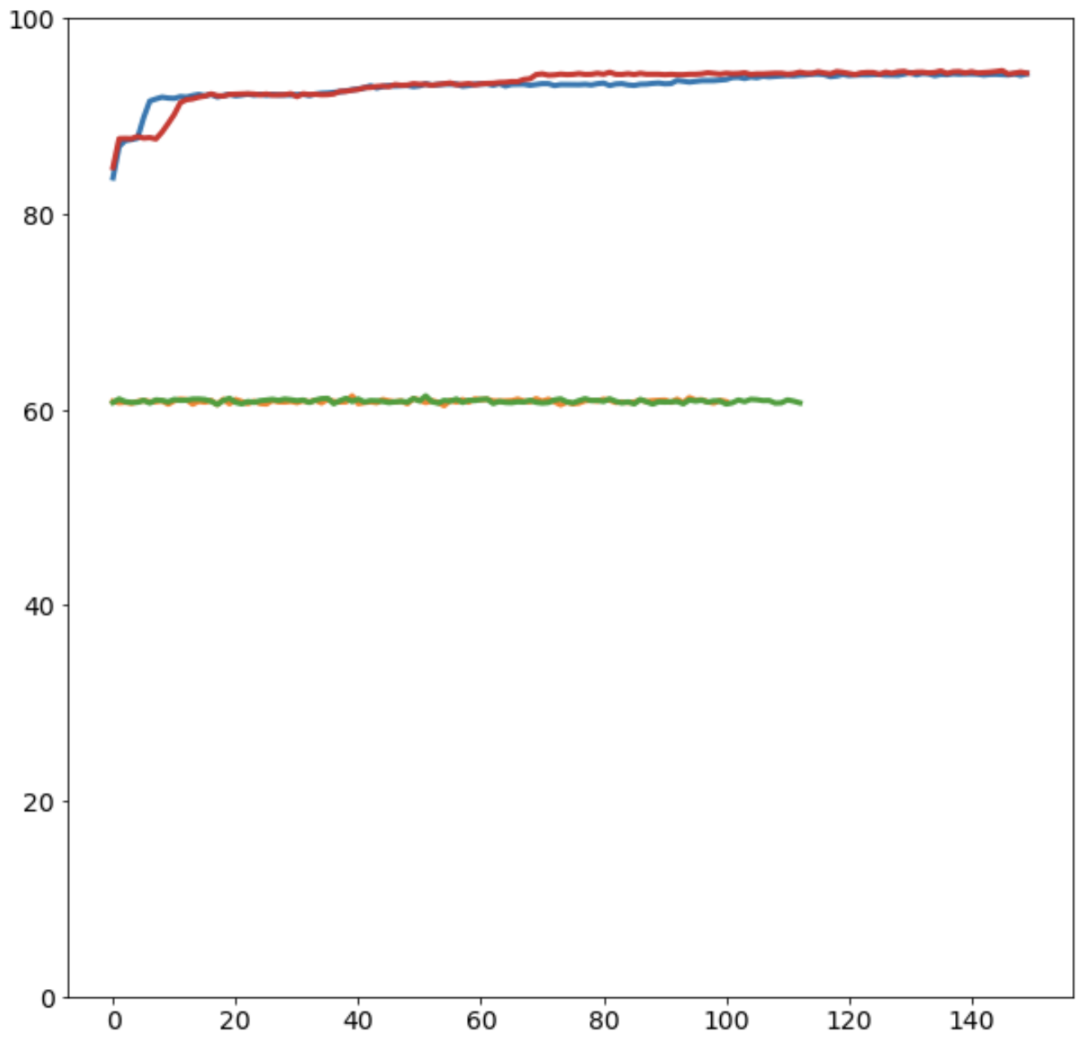
- Encoding input/output in base 30
- 1-layer transformers, 64 dimensions
- 85% accuracy after one epoch (300k examples)
- 94.6% accuracy after 150 epochs (45M examples)

- Surely, the maths are learned



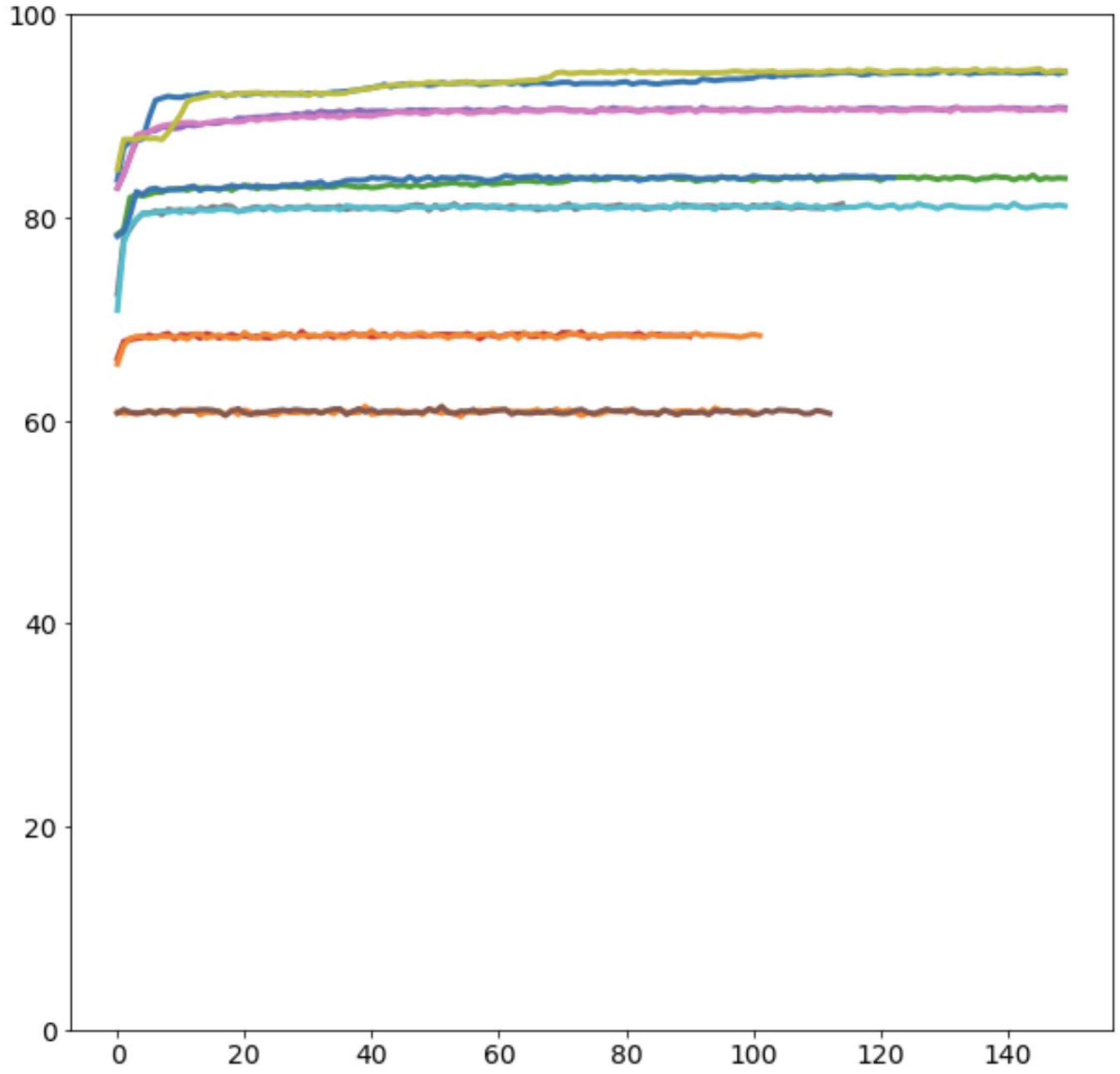
Learning the greatest common divisor?

- Encoding input/output in base 31
- Accuracy plateaus around 61%
- Accuracy seems base-dependent



Learning the greatest common divisor???

- Top to bottom, bases 30, 6, 10, 2, 3, 31…
- The gcd should not be base-dependent
- Are we really learning the maths?



Looking at model predictions

Table 3: Model predictions and their frequencies, for GCD 1 to 36. Correct predictions in bold face.

	Base 2		Base 10			Base 2		Base 10			Base 2		Base 10	
GCD	Pred	%	Pred	%	GCD	Pred	%	Pred	%	GCD	Pred	%	Pred	%
1	**1**	100	**1**	100	13	1	100	1	100	25	1	100	**25**	100
2	**2**	100	**2**	100	14	2	100	2	100	26	2	100	2	100
3	1	100	1	100	15	1	100	5	100	27	1	100	1	100
4	**4**	100	**4**	100	16	**16**	100	**16**	99.7	28	4	100	4	100
5	1	100	**5**	100	17	1	100	1	100	29	1	100	1	100
6	2	100	2	100	18	2	100	2	100	30	2	100	10	100
7	1	100	1	100	19	1	100	1	100	31	1	100	1	100
8	**8**	100	**8**	100	20	4	100	**20**	100	32	**32**	99.9	16	99.9
9	1	100	1	100	21	1	100	1	100	33	1	100	1	100
10	2	100	**10**	100	22	2	100	2	100	34	2	100	2	100
11	1	100	1	100	23	1	100	1	100	35	1	100	5	100
12	4	100	4	100	24	8	100	8	100	36	4	100	4	100

Learning the greatest common divisor???

- In base 2, gcd 1,2,4,8, 16… are correctly predicted
  - The model counts the rightmost zeroes
    - 11100 (28) and 1110 (14) have gcd 2
    - 111100 (60) and 111000 (56) have gcd 4

The three rules

(R1) **Predictions are deterministic.** The model predicts a unique value $f(k)$ for almost all (99.9%) pairs of integers with GCD k. Predictions are correct when $f(k) = k$.

(R2) **Correct predictions are products of primes dividing B.** For base 2, they are 1, 2, 4, 8, 16, 32 and 64. For base 31, 1 and 31. For base 10, all products of elements from $\{1, 2, 4, 8, 16\}$ and $\{1, 5, 25\}$. For base 30, all products of $\{1, 2, 4, 8\}$, $\{1, 3, 9, 27\}$. and $\{1, 5, 25\}$.

(R3) **f(k) is the largest correct prediction that divides k.** For instance, $f(8) = 8$, and $f(7) = 1$, for base 2 and 10, but $f(15) = 5$ for base 10 and $f(15) = 1$ for base 2.

So far disappointing

Table 2: Number of correct GCD under 100 and accuracy. Best of 6 experiments.

Base	2	3	4	5	6	7	10	11	12	15
Correct GCD	7	5	7	3	19	3	13	2	19	9
Accuracy	81.6	68.9	81.4	64.0	91.5	62.5	84.7	61.8	91.5	71.7
Base	30	31	60	100	210	211	**420**	997	1000	1024
Correct GCD	27	2	28	13	32	1	**38**	1	14	7
Accuracy	94.7	61.3	95.0	84.7	95.5	61.3	**96.8**	61.3	84.7	81.5

Large bases and grokking

- Base 2023 = 7.17.17
- After 10 epochs: 1,7, and 17 are learned, accuracy 63%, 3 GCD
- At epoch 101, 3 is learned, together with 21 (3.7) and 51 (3.17)
- At epoch 200, 2 is learned (and 6, 14, 34, 42): 11 GCD
- At epoch 600, 4 is learned: 16 GCD, 93% accuracy

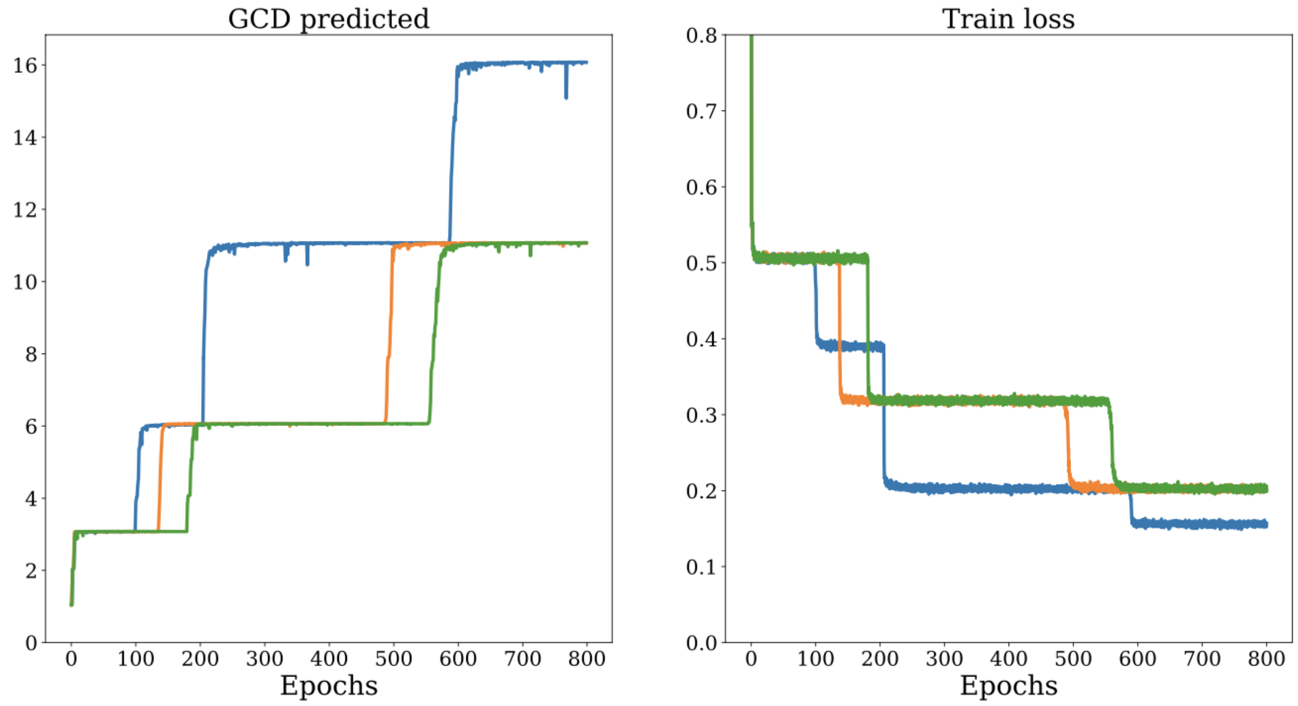


Figure 5: **Learning curves for base B=2023.** 3 different model initializations.

Large bases and grokking

This phenomenon is related to grokking, first described by Power. [22] for modular arithmetic. Table 5 presents model predictions for base 1000, which continue to respect rules R1 and R3. In fact, we can update the three rules into **the three rules with grokking**:

- (G1) **Prediction is deterministic.** All pairs with the same GCD are predicted the same, as $f(k)$.
- (G2) **Correct predictions are products of primes divisors of B, and small primes.** Small primes are learned roughly in order, as grokking sets in.
- (G3) **f(k) is the largest correct prediction that divides k.**

Large bases and grokking

Base	GCD predicted	Divisors predicted	Non-divisors (epoch learned)
$625 = 5^4$	6	{1,5,25}	2 (634)
2017	4	{1}	2 (142), 3 (392)
$2021 = 43.47$	10	{1,43}, {1,47}	2 (125), 3 (228)
$2023 = 7.17^2$	16	{1,7}, {1,17}	3 (101), 2 (205), 4 (599)
$2025 = 3^4.5^2$	28	{1,3, 9, 27, 81}, {1,5,25}	2 (217), 4 (493), 8 (832)
$2187 = 3^7$	20	{1,3,9,27,81}	2 (86), 4 (315) , 5 (650)
$2197 = 13^3$	11	{1,13}	2 (62), 3 (170), 4 (799)
$2209 = 47^2$	8	{1,47}	2 (111), 3 (260), 9 (937)
$2401 = 7^4$	10	{1,7,49}	2 (39), 3 (346)
$2401 = 7^4$	14	{1,7,49}	3 (117), 2 (399), 4 (642)
$2744 = 2^3.7^3$	30	{1,2,4,8,16,32}, {1,7,49}	3 (543), 5 (1315)
$3125 = 5^5$	16	{1,5,25}	2 (46), 3 (130), 4 (556)
$3375 = 3^3.5^3$	23	{1,3,9,27}, {1,5,25}	2 (236), 4 (319)
$4000 = 2^5.5^3$	24	{1,2, 4,8,16,32}, {1, 5, 25 }	3 (599)
$4913 = 17^3$	17	{1,17}	2 (54), 3 (138), 4 (648), 5 (873)
$5000 = 2^3.5^4$	28	{1,2,4,8,16,32}, {1,5,25}	3 (205), 9 (886)
$10000 = 2^4.5^4$	22	{1,2,4,8,16}, {1,5,25}	3 (211)

Table 6: **Predicted gcd, divisors and non-divisors of B.** Best model of 3. For non-divisors, the epoch learned is the first epoch where model achieves 90% accuracy for this gcd.

Engineering the training distribution

- Training sets have uniformly distributed operands
  - 90% of them are over 100 000
  - Small GCD, e.g. gcd(6,9) are never seen
- This is not how we are taught / teach arithmetic
  - From easy cases that we sometimes learn by rote
  - Generalizing to harder cases once easy cases are mastered
- Curriculum learning has draw backs: the distribution changes over time
  - Learn the easy cases, but then forget them

Engineering the training distribution

- Log-uniform operands
  - k appears with probability 1/k
  - As many 1-digit numbers as 6-digit
- No impact on the outcome distribution ($1/k^2$)
- No impact on the test sets

- Learning is noisier, but more GCD are learned

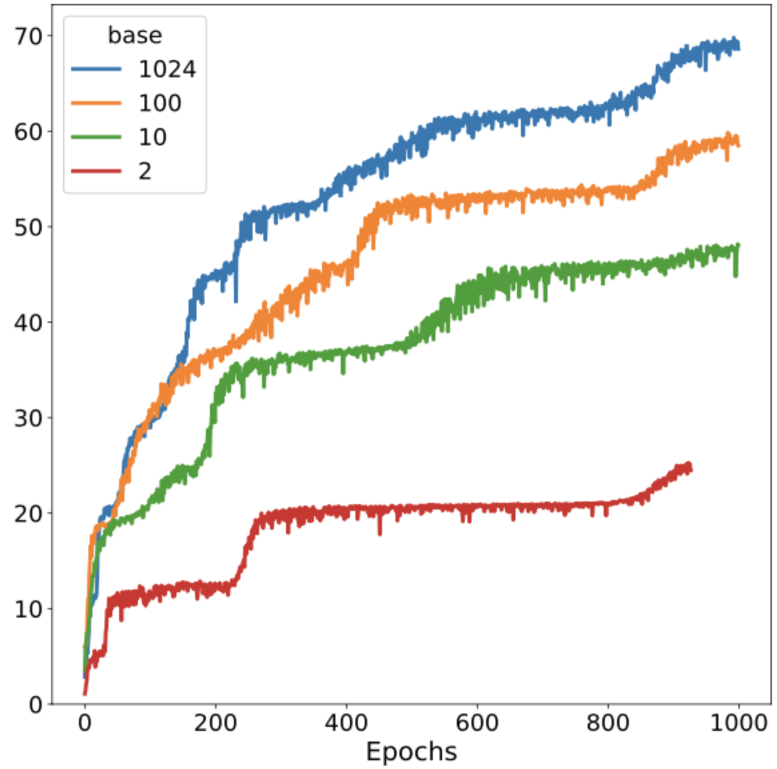


Figure 3: **Learning curves, Log-uniform training set.**

Engineering the training distribution

- Log-uniform operands, fast grokking
- All primes up to 23

Table 6: **Accuracy and correct GCD (up to 100), log-uniform operands.** Best of three models, trained for 1000 epochs (300M examples). All models are tested on 100,000 pairs, uniformly distributed between 1 and 10^6.

Base	Accuracy	Correct GCD	Base	Accuracy	GCD	Base	Accuracy	GCD
2	94.4	25	60	98.4	60	2025	99.0	70
3	96.5	36	100	98.4	60	2187	98.7	66
4	98.4	58	210	98.5	60	2197	98.8	68
5	97.0	42	211	96.9	41	2209	98.6	65
6	96.9	39	420	98.1	59	**2401**	**99.1**	**73**
7	96.8	40	625	98.2	57	2744	98.9	72
10	97.6	48	997	98.3	64	3125	98.6	65
11	97.4	43	1000	99.1	71	3375	98.8	67
12	98.2	55	1024	99.0	71	4000	98.7	66
15	97.8	52	2017	98.6	63	4913	98.2	57
30	98.2	56	2021	98.6	66	5000	98.6	64
31	97.2	44	2023	98.7	65	10000	98.0	56

Learning large primes, the outcome distribution

- GCD are distributed in $1/k^2$, very few examples with large primes
- A log-uniform distribution of operands and outcomes
  - All primes up to 53

Base	Accuracy	Correct GCD	Base	Accuracy	GCD	Base	Accuracy	GCD
2	16.5	17	60	96.4	75	2025	97.9	91
3	93.7	51	100	97.1	78	2187	97.8	91
4	91.3	47	210	96.2	80	2197	97.6	90
5	92.2	58	211	95.3	67	2209	97.6	87
6	95.2	56	420	96.4	88	2401	97.8	89
7	93.0	63	625	96.0	80	2744	97.6	91
10	94.3	65	997	97.6	83	3125	97.7	91
11	94.5	57	1000	97.9	91	3375	97.6	91
12	95.0	70	1024	98.1	90	4000	97.3	90
15	95.4	62	2017	97.6	88	4913	97.1	88
30	95.8	72	2021	98.1	89	5000	97.1	89
31	94.4	64	2023	97.5	88	10000	95.2	88

Table 9: **Accuracy and correct GCD, log-uniform operands and outcomes.** Best model of 3.

Take aways

- Predictions can be deterministic and explainable
- The model learns a sieve:
  - It classifies input pairs (a,b) into clusters with common divisors
  - And predicts the smallest common divisor in the class (when outcomes are not uniformly distributed)
- Training distribution impact accuracy, no matter the test distribution

Conclusions

- Transformers can learn mathematics
  - A new field for research
  - With applications to science
- Mathematical tasks help understand deep learning and transformers