Learning advanced mathematical computations from examples

Anonymous authors
Paper under double-blind review

Abstract

Using transformers over large generated datasets, we train models to learn mathematical properties of differential systems, such as local stability, behavior at infinity and controllability. We achieve near perfect prediction of qualitative characteristics, and good approximations of numerical features of the system. This demonstrates that neural networks can learn to perform complex computations, grounded in advanced theory, from examples, without built-in mathematical knowledge.

1 Introduction

Scientists solve problems of mathematics by applying rules and computational methods to the data at hand. These rules are derived from theory, they are taught in schools or implemented in software libraries, and guarantee that a correct solution will be found. Over time, mathematicians have developed a rich set of computational tools that can be applied to many problems, and have been said to be "unreasonably effective" (Wigner, 1960).

Deep learning, on the other hand, learns from examples and solves problems by improving a random initial solution, without relying on domain-related theory and computational rules. Deep networks have proven to be extremely efficient for a large number of tasks, but struggle on relatively simple, rule-driven arithmetic problems (Saxton et al., 2019; Trask et al., 2018; Zaremba and Sutskever, 2014).

Yet, recent studies show that deep learning models can learn complex rules from examples. In natural language processing, models learn to output grammatically correct sentences without prior knowledge of grammar and syntax (Radford et al., 2019), or to automatically map one language into another (Bahdanau et al., 2014; Sutskever et al., 2014). In mathematics, deep learning models have been trained to perform logical inference (Evans et al., 2018), SAT solving (Selsam et al., 2018) or basic arithmetic (Kaiser and Sutskever, 2015). Lample and Charton (2020) showed that transformers can be trained from generated data to perform symbol manipulation tasks, such as function integration and finding formal solutions of ordinary differential equations.

In this paper, we investigate the use of deep learning models for complex mathematical tasks involving both symbolic and numerical computations. We show that models can predict the qualitative and quantitative properties of mathematical objects, without built-in mathematical knowledge. We consider three advanced problems of mathematics: the local stability and controllability of differential systems, and the existence and behavior at infinity of solutions of partial differential equations. All three problems have been widely researched and have many applications outside of pure mathematics. They have known solutions that rely on advanced symbolic and computational techniques, from formal differentiation, Fourier transform, algebraic full-rank conditions, to function evaluation, matrix inversion, and computation of complex eigenvalues. We find that neural networks can solve these problems with a very high accuracy, by simply looking at instances of problems and their solutions, while being totally unaware of the underlying theory. In one of the quantitative problems where several solutions are possible (predicting control feedback matrix), neural networks are even able to predict different solutions that those generated with the mathematical algorithms we used for training.

After reviewing prior applications of deep learning to related areas we introduce the three problems we consider, describe how we generate datasets, and detail how we train our models. Finally, we present our experiments and discuss their results.

2 Related work

Applications of neural networks to differential equations have mainly focused on two themes: numerical approximation and formal resolution. Whereas most differential systems and partial differential equations cannot be solved explicitly, their solutions can be approximated numerically, and neural networks have been used for this purpose (Lagaris et al., 1998; 2000; Lee and Kang, 1990; Rudd, 2013; Sirignano and Spiliopoulos, 2018). This approach relies on the universal approximation theorem, that states that any continuous function can be approximated by a neural network with one hidden layer over a wide range of activation functions (Cybenko, 1989; Hornik et al., 1990; Hornik, 1991; Petersen and Voigtlaender, 2018; Pinkus, 1999). This has proven to be especially efficient for high dimensional problems.

For formal resolution, Lample and Charton (2020) proposed several approaches to generate arbitrarily large datasets of functions with their integrals, and ordinary differential equations with their solutions. They found that a transformer model (Vaswani et al., 2017) trained on millions of examples could outperform state-of-the-art symbolic frameworks such as Mathematica or MATLAB (Wolfram-Research, 2019; MathWorks, 2019) on a particular subset of equations. Their model was used to guess solutions, while verification (arguably a simpler task) was left to a symbolic framework (Meurer et al., 2017). Arabshahi et al. (2018a;b) proposed to use neural networks to verify the solutions of differential equations, and found that Tree-LSTMs (Tai et al., 2015) were better than sequential LSTMs (Hochreiter and Schmidhuber, 1997) at generalizing beyond the training distribution.

Other approaches investigated the capacity of neural networks to perform arithmetic operations (Kaiser and Sutskever, 2015; Saxton et al., 2019; Trask et al., 2018) or to run short computer programs (Zaremba and Sutskever, 2014). More recently, Saxton et al. (2019) found that neural networks were good at solving arithmetic problems or at performing operations such as differentiation or polynomial expansion, but struggled on tasks like prime number decomposition or on primality tests that require a significant number of steps to compute. Unlike the questions considered here, most of those problems can be solved by simple algorithmic computations.

3 Differential systems and their stability

A differential system of degree n is a system of n equations of n variables $x_1(t), ..., x_n(t)$,

$$\frac{dx_i(t)}{dt} = f_i\big(x_1(t), x_2(t), ..., x_n(t)\big), \qquad \text{for} \quad i = 1...n$$

or, in vector form, with $x \in \mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}^n$,

$$\frac{dx(t)}{dt} = f\big(x(t)\big)$$

Many problems can be set as differential systems. Special cases include n-th order ordinary differential equations (letting $x_1 = y$, $x_2 = y'$, ... $x_n = y^{(n-1)}$), systems of coupled differential equations, and some particular partial differential equations (separable equations or equations with characteristics). Differential systems are one of the most studied areas of mathematical sciences. They are found in physics, mechanics, chemistry, biology, and economics as well as in pure mathematics. Most differential systems have no explicit solution. Therefore, mathematicians have studied the properties of their solutions, and first and foremost their stability, a notion of paramount importance in many engineering applications.

3.1 Local stability

Let $x_e \in \mathbb{R}^n$ be an equilibrium point, that is, $f(x_e) = 0$. If all solutions $x(t)$ converge to x_e when their initial positions $x(0)$ at $t = 0$ are close enough, the equilibrium is said to be

locally stable (see Appendix B for a proper mathematical definition). This problem is well known, if f is differentiable in x_e, an answer is provided by the Spectral Mapping Theorem (SMT) (Coron, 2007, Theorem 10.10):

Theorem 3.1. *Let $J(f)(x_e)$ be the Jacobian matrix of f in x_e (the matrix of its partial derivatives relative to its variables). Let λ be the largest real part of its eigenvalues. If λ is positive, x_e is an unstable equilibrium. If λ is negative, then x_e is a locally stable equilibrium.*

Predicting the stability of a given system at a point x_e is our first problem. We will also predict λ, which represents the speed of convergence when negative, in a second experiment. Therefore, to apply the SMT, we need to:

1. differentiate each function with respect to each variable, obtain the formal Jacobian $J(x)$

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

2. evaluate $J(x_e)$, the Jacobian in x_e (a real or complex matrix)

$$x_e = (0.1, ...0.1) \in \mathbb{R}^n, \quad J(x_e) = \begin{pmatrix} -\cos(0.1) & -\sin(0.1) \\ 0.2 & -(2\sqrt{1+0.1})^{-1} \end{pmatrix},$$

3. calculate the eigenvalues $\lambda_i, i = 1...n$ of $J(x_e)$

   $\lambda_1 = -1.031, \quad \lambda_2 = -0.441$

4. compute $\lambda = -\max(\text{Real}(\lambda_i))$ and return the stability (resp. λ the speed of convergence)

   $\lambda = 0.441 > 0 \rightarrow$ locally stable with decay rate 0.441

3.2 Control theory

One of the lessons of the spectral mapping theorem is that instability is very common. In fact, unstable systems are plenty in nature (Lagrange points, epidemics, satellite orbits, etc.), and the idea of trying to control them through external variables comes naturally. This is the controllability problem. It has a lot of practical applications, including space launch and the landing on the moon, the US Navy automated pilot, or recently autonomous cars (Bernhard et al., 2017; Minorsky, 1930; Funke et al., 2016). Formally, we are given a system

$$\frac{dx}{dt} = f\big(x(t), u(t)\big), \tag{1}$$

where $x \in \mathbb{R}^n$ is the state of the system. We want to find a function $u(t) \in \mathbb{R}^p$, the control action, such that, beginning from a position x_0 at $t = 0$, we can reach a position x_1 at $t = T$ (see Appendix B). The first rigorous mathematical analysis of this problem was given by Maxwell (1868), but a turning point was reached in 1963, when Kalman gave a precise condition for a linear system (Kalman et al., 1963), later adapted to nonlinear system:

Theorem 3.2 (Kalman condition)**.** *Let $A = \partial_x f(x_e, u_e)$ and $B = \partial_u f(x_e, u_e)$, if*

$$Span\{A^i B u : u \in \mathbb{R}^m, i \in \{0, ..., n-1\}\} = \mathbb{R}^n, \tag{2}$$

then the system is locally controllable around $x = x_e, u = u_e$.

When this condition holds, a solution to the control problem that makes the system locally stable in x_e is $u(t) = u_e + K(x(t) - x_e)$ (c.f. Coron (2007); Kleinman (1970); Lukes (1968) and appendix D for key steps of the proof), where K is the $m \times n$ control feedback matrix:

$$K = -B^{tr} \left(e^{-AT} \left[\int_0^T e^{-At} B B^{tr} e^{-A^{tr}t} dt \right] e^{-A^{tr}T} \right)^{-1}. \tag{3}$$

In the non-autonomous case, where $f = f(x, u, t)$ (and A and B) depends on t, (2) can be replaced by:

$$\text{Span}\{D_i u : u \in \mathbb{R}^m, i \in \{0, ..., 2n-1\} = \mathbb{R}^n\}, \tag{4}$$

where $D_0(t) = B(t)$ and $D_{i+1}(t) = D_i'(t) - A(t)D_i(t)$. All these theorems make use of advanced mathematical results, such as the Cayley-Hamilton theorem, or LaSalle invariance principle. Learning them by predicting controllability and computing the control feedback matrix K is our second problem. To measure whether the system is controllable at a point x_e, we need to:

1. differentiate the system with respect to its internal variables, obtain $A(x, u)$
2. differentiate the system with respect to its control variables, obtain $B(x, u)$
3. evaluate A and B in (x_e, u_e)
4. calculate the controllability matrix C with (2) (resp. (4) if non-autonomous)
5. calculate the rank d of C, if $d = n$, the system is controllable
6. (optionally) if $d = n$, compute the control feedback matrix K with (3)

In: $f(x, u) = \begin{pmatrix} \sin(x_1^2) + \log(1 + x2) + \frac{\text{atan}(ux_1)}{1+x_2} \\ x_2 - e^{x_1 x_2} \end{pmatrix}, \begin{matrix} x_e = [0.1] \\ u_e = 1 \end{matrix}$, Out: $\begin{cases} n - d = 0 \\ \text{System is controllable} \\ K = (-22.8 \quad 44.0) \end{cases}$

A step by step derivation of this example is given in Section A of the appendix.

3.3 Stability of partial differential equations using Fourier Transform

Partial Differential Equations (PDEs) naturally appear when studying continuous phenomena (e.g. sound, electromagnetism, gravitation). Over such problems, ordinary differential systems are not sufficient. Like differential systems, PDEs seldom have explicit solutions, and studying their stability has many practical applications. It is also a much more difficult subject, where few general theorems exist. We consider linear PDEs of the form

$$\partial_t u(t, x) + \sum_{|\alpha| \leq k} a_\alpha \partial_x^\alpha u(t, x) = 0, \tag{5}$$

where t, $x \in \mathbb{R}^n$, and $u(t, x)$ are time, position, and state. $\alpha = (\alpha_1, ..., \alpha_n) \in \mathbb{R}^n$ is a multi-index and a_α are constants. Famous examples of such problems include the heat equation, transport equations or Schrodinger equation (Evans, 2010). We want to determine whether a solution $u(t, x)$ of (5) exists for a given an initial condition $u(0, x) = u_0$, and if it tends to zero as $t \to +\infty$. This is mathematically answered (see appendix D and Evans (2010); Bahouri et al. (2011) for similar arguments) by:

Proposition 3.1. *Given $u_0 \in \mathcal{S}'(\mathbb{R}^n)$, the space of tempered distribution, there exists a solution $u \in \mathcal{S}'(\mathbb{R}^n)$ if there exists a constant C such that*

$$\forall \xi \in \mathbb{R}^n \ , \ \widetilde{u}_0(\xi) = 0 \ \ or \ \ \text{Real}(f(\xi)) > C, \tag{6}$$

where $\widetilde{u}_0$ is the Fourier transform of u_0 and $f(\xi)$ is the Fourier polynomial associated with the differential operator $D_x = \sum_{|\alpha| \leq k} a_\alpha \partial_x^\alpha$. In addition, if $C > 0$, this solution $u(t, x)$ goes to zero when $t \to +\infty$.

Learning this proposition and predicting, given an input D_x and u_0, whether a solution u exists, if so, whether it vanishes at infinite time, will be our third and last problem.

To predict whether our PDE has a solution under given initial conditions, and determine its behavior at infinity, we need to: find the Fourier polynomial $f(\xi)$ associated to D_x; find the Fourier transform $\tilde{u}_0(\xi)$ of u_0; minimize $f(\xi)$ on $\mathcal{F}$; output (0,0) if this minimum is infinite, (1,0) is finite and negative, (1,1) if finite and positive. Optionally, output $\mathcal{F}$. A step by step example is given in Appendix A.

$$\text{In: } D_x = 2\partial_{x_0}^2 + 0.5\partial_{x_1}^2 + \partial_{x_2}^4 - 7\partial_{x_0,x_1}^2 - 1.5\partial_{x_1}\partial_{x_2}^2,$$
$$\text{Out:}(1, 0) \ \to \ \text{there exists a solution } u \text{ ; it does not vanish at } t \to +\infty$$

4 Datasets and models

To generate datasets, we randomly sample problems and compute their solutions with mathematical software (Virtanen et al., 2020; Meurer et al., 2017) using the techniques described in Section 3. For stability and controllability, we generate differential systems with n equations and $n+q$ variables (i.e. n random functions, $q > 0$ for controllability).

Following Lample and Charton (2020), we generate random functions by sampling unary-binary trees, and randomly selecting operators, variables and integers for their internal nodes and leaves. We use $+, -, \times, /, \exp, \log, \text{sqrt}, \sin, \cos, \tan, \sin^{-1}, \cos^{-1}, \tan^{-1}$ as operators, and integers between -10 and 10 as leaves. When generating functions with $n+q$ variables, we build trees with up to $2(n+q+1)$ operators.

Generated trees are enumerated in prefix order (normal Polish notation) and converted into sequences of tokens compatible with our models. Integers and floating point reals are also represented as sequences: 142 as [INT+, 1, 4, 2], and 0.314 as [FLOAT+, 3, DOT, 1, 4, e, INT-, 1]. A derivation of the size of the problem space is provided in appendix E.

Local stability Datasets for local stability include systems with 2 to 6 equations (in equal proportion). Functions that are not differentiable at the equilibrium x_e and degenerate systems are discarded. Since many of the operators we use are undefined at zero, setting $x_e = 0$ would result in biasing the dataset by reducing the frequency of operators like division, square root, or logarithms. Instead, we select x_e with all coordinates equal to 0.01 (denoted as $x_e = [0.01]$). This is, of course, strictly equivalent mathematically to sampling systems with equilibrium at the origin or at any other point.

When predicting overall stability, since stable systems become exponentially rare as dimension increases, we use rejection sampling to build a balanced dataset with 50% stable systems. When predicting convergence speed, we work from a uniform (i.e. unbalanced) sample. The value of λ at x_e is expressed as a floating point decimal rounded to 4 significant digits. For this problem, we generate two datasets with over 50 million systems each.

Control theory Datasets for automonous control include systems with 3 to 6 equations, and 4 to 9 variables (1 to 3 control variables). In the non-autonomous case, we generate systems with 2 or 3 equations. As above, we discard undefined or degenerate systems. We also skip functions with complex Jacobians in x_e (since the Jacobian represents local acceleration, one expects its coordinates to be real). We have $x_e = [0.5]$ or $[0.9]$.

In the autonomous case, more than 95% of the systems are controllable. When predicting controllability, we use rejection sampling to create a balanced dataset. In the non-autonomous case, we use a uniform sample with 83% controllable cases. Finally, to predict feedback matrices, we restrict generation to controllable systems and express the matrix as a sequence of floating point decimals. All 3 datasets have more than 50 million examples each.

Stability of partial differential equations using Fourier Transform We generate a differential operator (a polynomial in ∂_{x_i}) and an initial condition u_0. u_0 is the product of n functions $f(a_j x_j)$ with known Fourier transforms, and d operators $\exp(ib_k x_k)$, with $0 \leq d \leq 2n$ and $a_j, b_k \in \{-100, \ldots, 100\}$. We calculate the existence of solutions, their behavior when $t \to +\infty$, and the set of frequencies, and express these three values as a sequence of 2 Booleans and floating point decimals. Our dataset is over 50 million examples.

Models and evaluation In all experiments, we use a transformer architecture with 8 attention heads. We vary the dimension from 64 to 1024, and the number of layers from 1 to 8. We train our models with the Adam optimizer (Kingma and Ba, 2014), a learning rate of 10^{-4} and the learning rate scheduler in Vaswani et al. (2017), over mini-batches of 1024 examples. Additional information can be found in appendix F. Training is performed on 8 V100 GPUs with float16 operations. Our qualitative models (predicting stability, controllability and existence of solutions) were trained for about 12 hours, but accuracies close to the optimal values were reached after about 6 hours. Learning curves for this problem

can be found in appendix H. On quantitative models, more training time and examples were needed: 76 hours for convergence speed, 73 hours for control matrices.

Evaluation is performed on held-out validation and test sets of 10000 examples. We ensure that validation and test examples are never seen during training (given the size of the problem space, this never happens in practice). Model output is evaluated either by comparing it with the reference solution or using a problem-specific metric.

5 Experiments

5.1 Predicting qualitative properties of differential systems

In these experiments, the model is given n functions $f : \mathbb{R}^{n+p} \to \mathbb{R}$ ($n \in \{2, \ldots, 6\}$, $p = 0$ for stability, $p > 0$ for controllability) and is trained to predict whether the corresponding system is stable, resp. controllable, at a given point x_e. This is a classification problem.

To provide a baseline for our results, we use fastText (Joulin et al., 2016), a state-of-the-art text classification tool, which estimates, using a bag of words model, the probability of a qualitative feature (stability) conditional to the distribution of tokens and of small fixed sequences (N-grams of up to five tokens). Such a model can detect simple correlations between inputs and outputs, such as the impact on stability of the presence of a given operator, or the number of equations in the system. It would also find out obvious solutions, due to the specifics of one problem or glitches in the data generator. FastText was trained over 2 million examples from our dataset (training on a larger set do not improve accuracy).

A 6-layer transformer with 512 dimensions correctly predicts the system stability in 96.4% of the cases. Since the dataset is balanced, random guessing would achieve 50%. FastText achieves 60.6%, demonstrating that whereas some easy cases can be learnt by simple text classifiers, no trivial general solution exists for this dataset. Prediction accuracy decreases with the degree, but remains high even for large systems (Table 1).

Table 1: **Accuracy of predictions of stability (chance level: 50%)**

	Degree 2	Degree 3	Degree 4	Degree 5	Overall	FastText
Accuracy	98.2	97.3	95.9	94.1	96.4	60.6

For autonomous controllability over a balanced dataset, a 6-layer transformer with 512 dimensions correctly predicts 97.4% of the cases. The FastText baseline is 70.5%, above the 50% chance level. Whereas accuracy increases with model size (dimension and number of layers), even very small models (dimension 64 and only 1 or 2 layers) achieve performance over 80%, above the FastText baseline (Table 2).

Table 2: **Accuracy of autonomous control task over a balanced sample of systems with 3 to 6 equations.**

	Dimension 64	Dimension 128	Dimension 256	Dimension 512	FastText
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**	**70.5**

For non-autonomous systems, our dataset features systems of degree 2 and 3, 83% controllable. FastText achieves 85.3%, barely above the chance level of 83%. This shows that text classifiers have difficulty handling difficult problems like this one, even in low dimensions. Our model achieves 99.7% accuracy. Again, small models, that would be unsuitable for natural language processing, achieve near perfect accuracy (Table 3).

Table 3: **Accuracy for non-autonomous control over systems with 2 to 3 equations.**

	Dimension 64	Dimension 128	Dimension 256	Dimension 512	FastText
1 layer	97.9	98.3	98.5	98.9	-
2 layers	98.4	98.9	99.3	99.5	-
4 layers	98.6	99.1	99.4	99.6	-
6 layers	98.7	99.1	99.5	**99.7**	**85.3**

5.2 Predicting numerical properties for differential systems

Speed of convergence In these experiments, the model is trained to predict λ, the convergence speed to the equilibrium, up to a certain precision. Here, we consider predictions to be correct when they fall within 10% of the ground truth. Further experiments with different levels of precision (2, 3 or 4 decimal digits) are provided in Appendix C.

A model with 8 layers and a dimension of 1024 predicts convergence speed with an accuracy of 86.6% overall. While reasonably good results can be achieved with smaller models, the accuracy decrease quickly when model size falls under a certain value, unlike when qualitative properties were predicted. Table 4 summarizes the results.

Table 4: **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**

Control feedback matrices In these experiments, we train the model (6 layers, 512 dimensions) to predict a feedback matrix ensuring stability of an autonomous system. We use two metrics to evaluate accuracy:

1) prediction within 10% of all coefficients in the target matrix K given by (3) and provided in the training set,

2) verifying that the model outputs a correct feedback matrix K_1, i.e. that all eigenvalues in $A + BK_1$ have negative real parts. This makes more mathematical sense, as it verifies that the model provides an actual solution to the control problem (like a differential equation, a feedback control problem can have many different solutions).

Using the first metric, 15.8% of target matrices K are predicted with less than 10% error. Accuracy is 50.0% for systems with 3 equations, but drops fast as systems becomes larger. These results are very low, although well above chance level ($<$0.0001%). With the second metric (i.e. the one that actually matters mathematically), we achieve 66.5% accuracy, a much better result. Accuracy decreases with system size, but even degree 6 systems, with 1×6 to 3×6 feedback matrices, are correctly predicted 41.5% of the time. Therefore, while the model fails to approximate K to a satisfactory level, it does learn to predict correct solutions to the control problem in 66.5% of the cases. This result is very surprising, as it suggests that a mathematical property characterizing feedback matrices might have been learned.

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

5.3 Predicting qualitative properties of PDEs

In this setting, the model is given a differential operator D_x and an initial condition u_0. It is trained to predict if a solution to $\partial_t u + D_x u = 0$ exists and, if so, whether it converges to 0 when $t \to +\infty$. The space dimension (i.e. dimension of x) is between 2 and 6.

In a first series of experiments the model is only trained to predict the existence and convergence of solutions. Overall accuracy is 98.4%. In a second series, we introduce an auxiliary task by adding to the output the frequency bounds $\mathcal{F}$ of u_0. We observe it significantly contributes to the stability of the model with respect to hyper-parameters. In particular, without the auxiliary task, the model is very sensitive to the learning rate scheduling and often fails to converge to something better than random guessing. However, in case of convergence, the model reaches the same overall accuracy, with and without auxiliary task. Table 6 details the results.

Table 6: **Accuracy on the existence and behavior of solutions at infinity.**

Space dimension for x	Dim 2	Dim 3	Dim 4	Dim 5	Dim 6	Overall
Accuracy	99.4	98.9	98.7	98.0	96.9	98.4

6 Discussion

We studied five problems of advanced mathematics from widely researched areas of mathematical analysis. In three of them, we predict qualitative and theoretical features of differential systems. In two, we perform numerical computations. According to mathematical theory, solving these problems requires a combination of advanced techniques, symbolic and numerical, that seem unlikely to be learnable from examples. Yet, our model achieves more than 95% accuracy on all qualitative tasks, and between 65 and 85% on numerical computations.

When working from synthetic data, a question naturally arises about the impact of data generation on the results of experiments. In particular, one might wonder whether the model is exploiting a defect in the generator or a trivial property of the problems that allows for easier solutions. We believe this is very unlikely. First, because our results are consistent over different problems, using datasets generated with different techniques. Second, because a trivial solution would be found by the bag of words model we use as a baseline. And finally, because we build our datasets by direcly sampling problems from a distribution that includes all possible functions (up to the basis operators and the random number generator). This eliminates the biases that can result from sampling special instances or solutions (Yehuda et al., 2020). It also means that the training set is an extremely tiny sample of the whole problem space (over the 50 million examples generated, we did not get a single duplicate).

Learning from very large samples often raises questions about overfitting and generalisation out of the training distribution. Due to the size of the problem space, it is very unlikely that the model could memorize a large number of cases and interpolate between them. Note that because the space of functions from $\mathbb{R}^n$ to $\mathbb{R}^n$ has infinite dimension, the universal approximation theorem does not apply here. Note also that for some of our problems (e.g. local stability), mathematical theory states that solutions cannot be obtained by simple interpolation. To investigate out-of-distribution generalization, we modified our data generator to produce 10 new test sets for stability prediction. We changed the distribution of operators and variables, and experimented with systems with longer expressions and more equations than in the training set (see Appendix I). Changes in the distribution of operators and variables have very little impact on accuracy, demonstrating that the model generalizes out of the training distribution. Extending the problem space to systems with longer expressions also results in good performance, which is interesting because we are then generalizing to a larger problem space than the one used for training. Finally, the model manages a relatively high accuracy (78%) on systems of 6 equations, whereas it was only trained on systems with 2 to 5 equations. This is very surprising, and tends to confirm that some underlying mathematical properties of differential systems are learned.

However, it seems unlikely that the model follows the same mathematical procedure as human solvers. For instance, problems involving more computational steps, such as non-autonomous controllability, do not result in lower accuracy. Also, providing at train time intermediate results that would help a human calculator (frequencies for PDE, or Jacobians for stability) does not improve performance. Understanding how the model finds solutions would be very interesting, as no simpler solutions than the classical mathematical steps are known.

To this effect, we tried to analyze model behavior by looking at the attention heads and the tokens the models focus on when it predicts a specific sequence (following Clark et al. (2019)). Unfortunately, we were not able to extract specific patterns, and found that each head in the model, from the first layer onwards, attends many more tokens than in usual natural language tasks (i.e. attention weights tend to be uniformly distributed). This makes interpretation very difficult. To improve visualization, we tried to reduce the number of hidden states a head can attend by using a top-k on the attention weights, but this deteriorated the performance, and we did not investigate more in this direction. We also ran a sequence-to-sequence model without attention, so that each input equation is mapped to a fixed-sized representation. We then fed a set of input equations into the model, and used a t-SNE visualization to see whether we can observe clusters of equations. What we observed is mainly that equations with nearby representations have similar length / tokens. However, even embeddings in similar locations can lead to different decoded sequences. The relevance of the representations built in the encoder depends on how the computation is split between the encoder and the decoder. If the decoder does the majority of the work, encoder representations become less meaningful.

These results open many perspectives for transformers in fields that need both symbolic and numerical computations. There is even hope that our models could help solve mathematical problems that are still open. On a more practical level, they sometimes provide fast alternatives to classical solvers. The algorithmic complexity of transformer inference and classical algorithms for the problems we consider here is discussed in appendix G. However, for the problems in our dataset, the simpler and parallelizable computations used by transformers allow for shorter evaluation times. Table 7 compares the average time needed to solve one problem, for a trained transformer running on a GPU, and a Python implementation of the algorithms, running on a MacBook Pro.

Table 7: **Speed comparison between trained transformers and mathematical libraries.** Average time to solve one system, in seconds.

Task	Mathematical libraries	Trained transformers
Stability end to end	0.02	0.0008
Stability largest eigenvalue	0.02	0.002
Controllability (autonomous)	0.05	0.001
Predicting a feedback matrix	0.4	0.002

7 Conclusion

In this paper, we show that by training transformers over generated datasets of mathematical problems, advanced and complex computations can be learned, and qualitative and numerical tasks performed with high accuracy. Our models have no built-in mathematical knowledge, and learn from examples only. However, solving problems with high accuracy does not mean that our models have learned the techniques we use to compute their solutions. Problems such as non-autonomous control involve long and complex chains of computations, which some of the smaller models we used could certainly not handle.

Most probably, our models learn shortcuts that allow them to solve specific problems, without having to learn or understand their theoretical background. Such a situation is common in everyday life. Most of us learn and use language without understanding its rules. On many practical subjects, we have tacit knowledge and know more than we can tell (Polanyi and Sen (2009)). This may be the way neural networks learn advanced mathematics. Understanding what these shortcuts are, how neural networks discover them, and how they can impact mathematical practice, is a subject for future research.

References

Forough Arabshahi, Sameer Singh, and Animashree Anandkumar. Combining symbolic expressions and black-box function evaluations for training neural programs. In *International Conference on Learning Representations*, 2018a.

Forough Arabshahi, Sameer Singh, and Animashree Anandkumar. Towards solving differential equations through neural programming. 2018b.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Hajer Bahouri, Jean-Yves Chemin, and Raphaël Danchin. *Fourier analysis and nonlinear partial differential equations*, volume 343. Springer Science & Business Media, 2011.

Pierre Bernhard, Marc Deschamps, et al. Kalman on dynamics and contro, linear system theory, optimal control, and filter. Technical report, 2017.

Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

Jean-Michel Coron. *Control and nonlinearity*, volume 136 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence, RI, 2007. ISBN 978-0-8218-3668-2; 0-8218-3668-4.

George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

Lawrence C Evans. *Partial differential equations*, volume 19. American Mathematical Soc., 2010.

Richard Evans, David Saxton, David Amos, Pushmeet Kohli, and Edward Grefenstette. Can neural networks understand logical entailment? *arXiv preprint arXiv:1802.08535*, 2018.

Joseph Funke, Matthew Brown, Stephen M Erlien, and J Christian Gerdes. Collision avoidance and stabilization for autonomous vehicles in emergency scenarios. *IEEE Transactions on Control Systems Technology*, 25(4):1204–1216, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5):551–560, 1990.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

Lukasz Kaiser and Ilya Sutskever. Neural gpus learn algorithms. *CoRR*, abs/1511.08228, 2015.

Rudolf E. Kalman, Yu-Chi Ho, and Kumpati S. Narendra. Controllability of linear dynamical systems. *Contributions to Differential Equations*, 1:189–213, 1963. ISSN 0589-5839.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

David Kleinman. An easy way to stabilize a linear constant system. *IEEE Transactions on Automatic Control*, 15(6):692–692, 1970.

Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

Isaac E Lagaris, Aristidis C Likas, and Dimitris G Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, 2000.

Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1eZYeHFDS.

Hyuk Lee and In Seok Kang. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91(1):110–131, 1990.

Dahlard L Lukes. Stabilizability and optimal control. *Funkcial. Ekvac*, 11:39–50, 1968.

MathWorks. Matlab optimization toolbox (r2019a), 2019. The MathWorks, Natick, MA, USA.

James Clerk Maxwell. I. on governors. *Proceedings of the Royal Society of London*, pages 270–283, 1868.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL https://doi.org/10.7717/peerj-cs.103.

Nicolas Minorsky. Automatic steering tests. *Journal of the American Society for Naval Engineers*, 42(2):285–310, 1930.

Philipp Petersen and Felix Voigtlaender. Optimal approximation of piecewise smooth functions using deep relu neural networks. *Neural Networks*, 108:296–330, 2018.

Allan Pinkus. Approximation theory of the mlp model in neural networks. *Acta numerica*, 8: 143–195, 1999.

Michael Polanyi and Amartya Sen. *The Tacit Dimension*. University of Chicago Press, 2009. ISBN 9780226672984.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Keith Rudd. *Solving partial differential equations using artificial neural networks*. PhD thesis, Duke University Durham, NC, 2013.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*, 2019.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.

Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

Andrew Trask, Felix Hill, Scott E Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. In *Advances in Neural Information Processing Systems*, pages 8035–8044, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: https://doi.org/10.1038/s41592-019-0686-2.

Eugene P Wigner. The unreasonable effectiveness of mathematics in the natural sciences. *communications on pure and applied mathematics*, 12:1–14, 1960.

Wolfram-Research. Mathematica, version 12.0, 2019. Champaign, IL, 2019.

Gal Yehuda, Moshe Gabel, and Assaf Schuster. It's not what machines can learn, it's what we cannot teach. *arXiv preprint arXiv:2002.09398*, 2020.

Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.