Using Language Models for Mathematics

Zhangir Azerbayev

Yale University

2023

Introduction

How can we design a computer program that does mathematics better than any human?

- Solve open problems
- Pose conjectures and create theories
- Convince mathematical community its proofs are correct

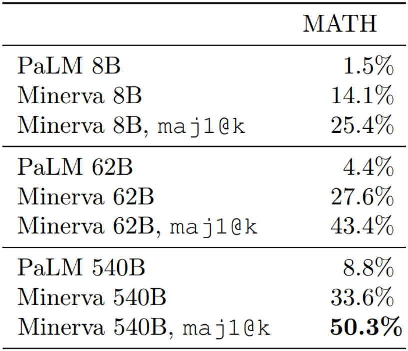
Questions I won't be talking about

- How can we build tools for mathematicians?
- Does it matter whether we understand mathematics done by a machine?

Approach 1: ask GPT

The most obvious thing to try is to ask the language model to do math. How well does this work?

Minerva (Lewkowycz et al. 2022): 540B parameter transformer trained on 39B tokens of math. Based on PaLM, a general domain language model.



	MATH
PaLM 8B	1.5%
Minerva 8B	14.1%
Minerva 8B, `maj1@k`	25.4%
PaLM 62B	4.4%
Minerva 62B	27.6%
Minerva 62B, `maj1@k`	43.4%
PaLM 540B	8.8%
Minerva 540B	33.6%
Minerva 540B, `maj1@k`	**50.3%**

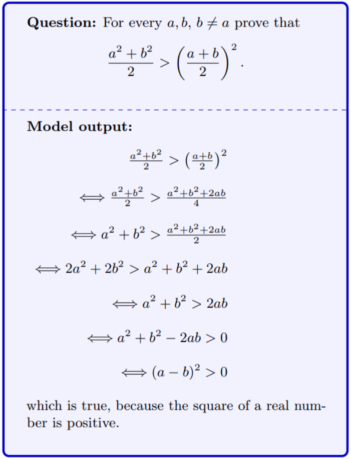
MATH: a benchmark of high school and olympiad math problems

Approach 1: ask GPT

GPT-4:

AMC 10[3]	30 / 150 (6th - 12th)
AMC 12[3]	60 / 150 (45th - 66th)

Approach 1: ask GPT



Question: For every a, b, $b \neq a$ prove that

$$\frac{a^2+b^2}{2} > \left(\frac{a+b}{2}\right)^2.$$

Model output:

$$\begin{aligned} \tfrac{a^2+b^2}{2} &> \left(\tfrac{a+b}{2}\right)^2 \\ \iff \tfrac{a^2+b^2}{2} &> \tfrac{a^2+b^2+2ab}{4} \\ \iff a^2+b^2 &> \tfrac{a^2+b^2+2ab}{2} \\ \iff 2a^2+2b^2 &> a^2+b^2+2ab \\ \iff a^2+b^2 &> 2ab \\ \iff a^2+b^2-2ab &> 0 \\ \iff (a-b)^2 &> 0 \end{aligned}$$

which is true, because the square of a real number is positive.

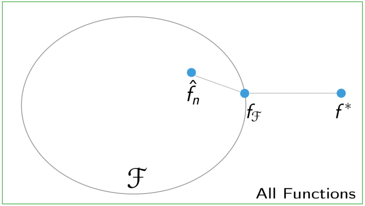
Background: scaling laws

What happens when we keep feeding in more parameters and more data?

Loss function: $L(f) = -\sum_{x \in X} \log P(x|\theta)$

We're approximating the data distribution with $\mathcal{F}$, the set of N-parameter transformers. Let f^* be the Bayes optimal classifier and $f_{\mathcal{F}} = \mathrm{argmin}_{f \in \mathcal{F}} L(f)$

Background: scaling laws

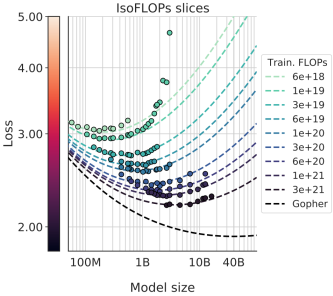


$$\text{Excess risk}(\hat{f}) = \underbrace{(L(f^*) - L(f_{\mathcal{F}}))}_{\text{approximation error}} + \underbrace{\Big(L(f_{\mathcal{F}}) - L(\hat{f})\Big)}_{\text{estimation error}}$$

$$= \frac{A}{N^{\alpha}} + \frac{B}{D^{\beta}}$$

Background: scaling laws

Hoffmann et al. 2022 (Chinchilla): most up to date empirical work on scaling laws.



$$L(\hat{f}) = 1.69 + \frac{406.4}{N^{0.34}} + \frac{410.7}{D^{0.28}}$$

Approach 1: ask GPT

Our model gets arbitrarily good with more parameters and more data.

So let's just ask our model to do mathematics in natural language.

We might be concerned about mistakes, but soon we will train a bigger model with a lower loss and we can ask that model to check the proof. Eventually, all errors will be caught because the loss of our models approaches the entropy of the text.

Data limitations

High quality mathematical text is extremely scarce. There is perhaps only ~ 100 billion tokens of it. This is tiny by language modelling standards.

As we increase the size of our dataset, the proportion of high-quality mathematics in it will vanish.

If we have infinite general domain data but finite mathematics data, can we still approach the entropy of the text?

Scaling laws for transfer

Suppose we have data from a distribution $P_1(\cdot)$, but we wish to model $P_2(\cdot)$. Then the best we can do is

$$\text{Excess risk}_2(\hat{f}) = (L_2(g^*) - L_1(f^*)) - (L_1(f^*) - L_1(f_{\mathcal{F}}) + \left(L_1(f_{\mathcal{F}}) - L_1(\hat{f})\right)$$

Thus for zero-shot transfer, the scaling law get is

$$L_2(N, D_1) = \underbrace{S}_{\text{entropy}} + \underbrace{P}_{\text{penalty}} + \frac{A}{N^\alpha} + \frac{B}{D_1^\beta}$$

Scaling laws for transfer

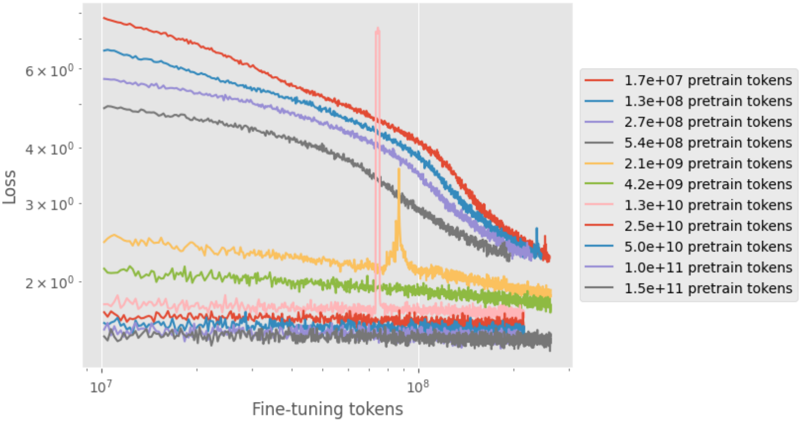
In practice, we have a little bit of data D_2 from the target distribution (mathematics) that we can fine-tune on.

The penalty is some non-zero constant when $D_2 = 0$ and vanishes as $D_2 \to \infty$. So it seems reasonable to posit that our scaling law is

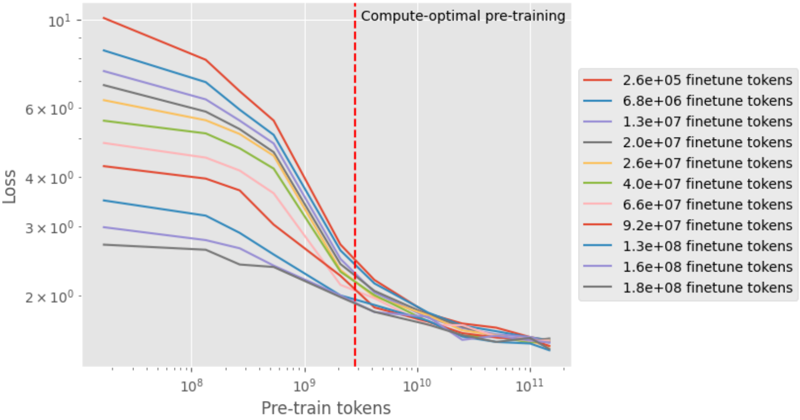
$$L_2(N, D_1, D_2) = S + \frac{A}{N^\alpha} + \frac{B}{D_1^\beta} + \frac{C}{D_2^\gamma}$$

Empirical scaling laws for transfer

Experiments with Pythia-1.4b model and math fine-tuning dataset (arXiv, stack exchange, formal math, numerical computing, etc.)



Empirical scaling laws for transfer



Discussion

It appears that there are limits to how much we can reduce loss on arXiv *with the data we have available to us*.

Although this analysis is not conclusive. We haven't tried varying model parameters yet (maybe there is an interaction term between N and D_2), and we haven't varied data over enough orders of magnitude.

Our only option is to have the model generate its own data, i.e *reinforcement learning*

Interactive Theorem Proving

Interactive theorem provers are the ideal reinforcement learning "environment" for mathematics.

Proof checker catches incorrect reasoning and can be used to implement the reward model.

pySagredo: a basic tool for using language models to prove theorems in Lean 4.

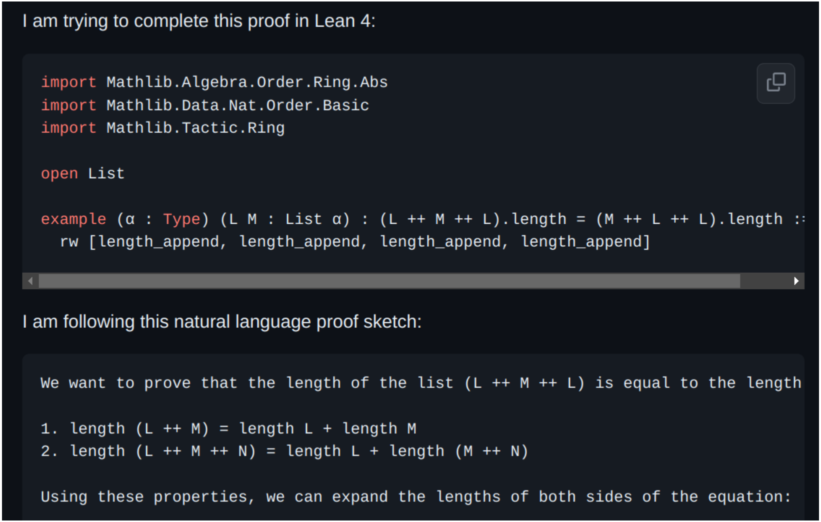
pySagredo

PySagredo is a probabilistic program that samples from language model primitives.

Suppose you have some Lean code T_0 that is missing a proof. First, we sample a natural language proof sketch $S \sim P(S|T_0)$. The interactive theorem prover is able to deterministically give us its state: $L_n = f(T_n)$.

Proving proceeds by iteratively applying a probabilistic kernel $P(T_{n+1}|T_n, L_n, S)$. The kernel either corrects an error message (if there are any raised by L_n) or adds a new tactic step to the proof. When L_n shows no goals or errors, the proof is finished.

Probabilistic kernel: prompt



I am trying to complete this proof in Lean 4:

```
import Mathlib.Algebra.Order.Ring.Abs
import Mathlib.Data.Nat.Order.Basic
import Mathlib.Tactic.Ring

open List

example (α : Type) (L M : List α) : (L ++ M ++ L).length = (M ++ L ++ L).length :
  rw [length_append, length_append, length_append, length_append]
```

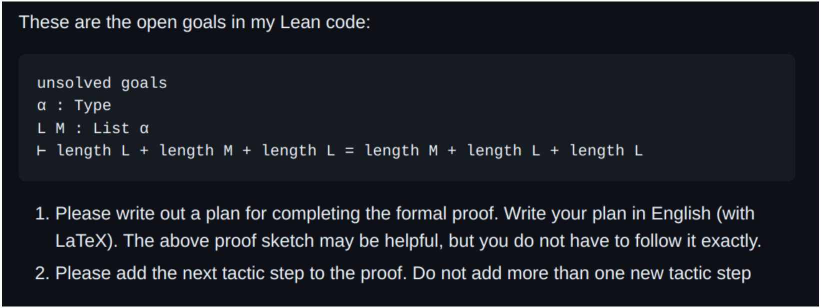
I am following this natural language proof sketch:

```
We want to prove that the length of the list (L ++ M ++ L) is equal to the length

1. length (L ++ M) = length L + length M
2. length (L ++ M ++ N) = length L + length (M ++ N)

Using these properties, we can expand the lengths of both sides of the equation:
```

Probabilistic kernel: prompt

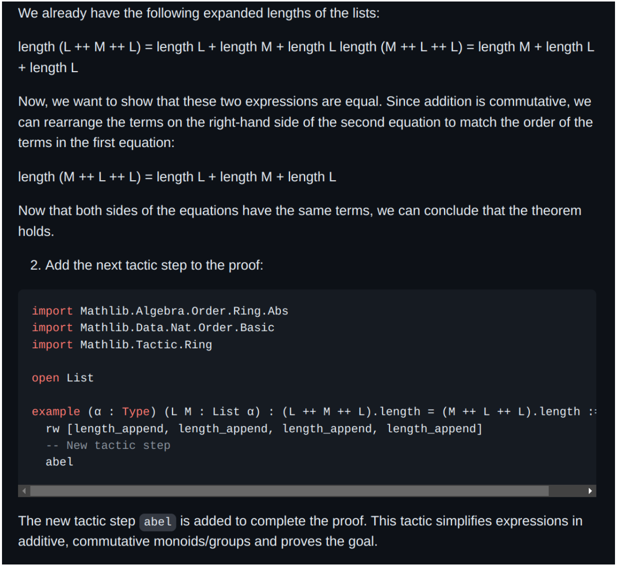


These are the open goals in my Lean code:

```
unsolved goals
α : Type
L M : List α
⊢ length L + length M + length L = length M + length L + length L
```

1. Please write out a plan for completing the formal proof. Write your plan in English (with LaTeX). The above proof sketch may be helpful, but you do not have to follow it exactly.
2. Please add the next tactic step to the proof. Do not add more than one new tactic step

Probabilistic kernel: response



We already have the following expanded lengths of the lists:

length (L ++ M ++ L) = length L + length M + length L length (M ++ L ++ L) = length M + length L + length L

Now, we want to show that these two expressions are equal. Since addition is commutative, we can rearrange the terms on the right-hand side of the second equation to match the order of the terms in the first equation:

length (M ++ L ++ L) = length L + length M + length L

Now that both sides of the equations have the same terms, we can conclude that the theorem holds.

2. Add the next tactic step to the proof:

```
import Mathlib.Algebra.Order.Ring.Abs
import Mathlib.Data.Nat.Order.Basic
import Mathlib.Tactic.Ring

open List

example (α : Type) (L M : List α) : (L ++ M ++ L).length = (M ++ L ++ L).length :
  rw [length_append, length_append, length_append, length_append]
  -- New tactic step
  abel
```

The new tactic step `abel` is added to complete the proof. This tactic simplifies expressions in additive, commutative monoids/groups and proves the goal.

Closing the loop

We've seen language models can

- Learn mathematics, to some modest level, from behavior cloning.
- Interact with an ITP environment.

Can we design an algorithm that allows language models to *learn from their interaction with the ITP environment*.

Prior efforts to use reinforcement learning for language and reasoning don't inspire optimism:

- Tree search + expert iteration saturates quickly. Algorithm doesn't cope with complex action spaces well.
- RL from Human Feedback doesn't help much with capabilities.

Closing the loop

RL consists of two steps

1. Try new things (exploration)
2. Keep doing the things that gave high reward. (exploitation)

We don't have any ideas for (1) except "do something random".

Maybe productive exploration is an emergent property of a sufficiently good supervised policy?