MACHINE LEARNING AND PURE MATHEMATICS: EXPERIMENTS AND SPECULATIONS

Jordan Ellenberg, University of Wisconsin-Madison: ANTS XVI

In Memoriam, Nigel Boston, 1961-2024

Article | Open access | Published: 14 December 2023

Mathematical discoveries from program search with large language models

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli & Alhussein Fawzi

Nature **625**, 468–475 (2024) | Cite this article

Capsets

A *capset* in $(\mathbf{Z}/3\mathbf{Z})^n$ is a set S of vectors such that no three distinct elements of s satisfy

s + t + u = 0

Capsets

A *capset* in $(\mathbf{Z}/3\mathbf{Z})^n$ is a set S of vectors such that no three distinct elements of s satisfy

s + t + u = 0

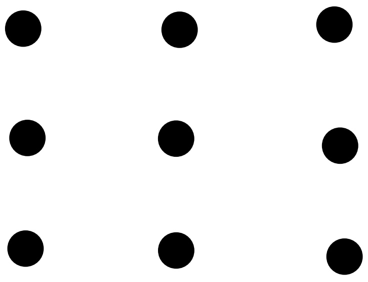
t = -(s+u) = (1/2)(s+u)

Capsets

f(n) = size of the largest capset in $(\boldsymbol{Z}/3\boldsymbol{Z})^{\wedge}n$

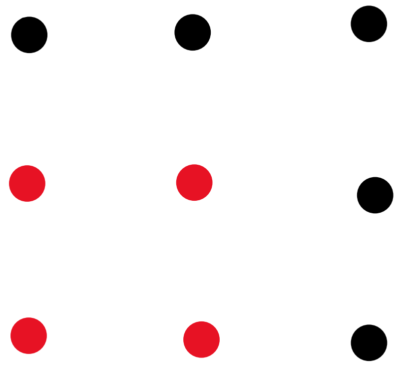
Perhaps my favourite open question is the problem on the maximal size of a *cap set* – a subset of $\mathbb{F}_3^n$ ($\mathbb{F}_3$ being the finite field of three elements) which contains no lines, or equivalently no non-trivial arithmetic progressions of length three. As

f(1) = 2, f(2) = ?



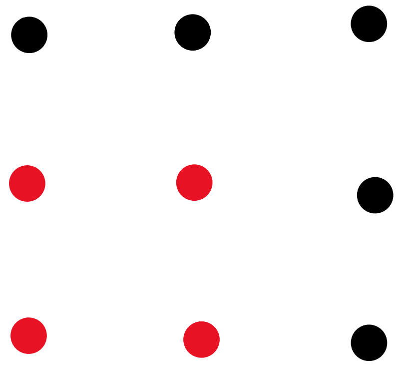
Capsets

$f(2) = 4$



Capsets

f(2) = 4

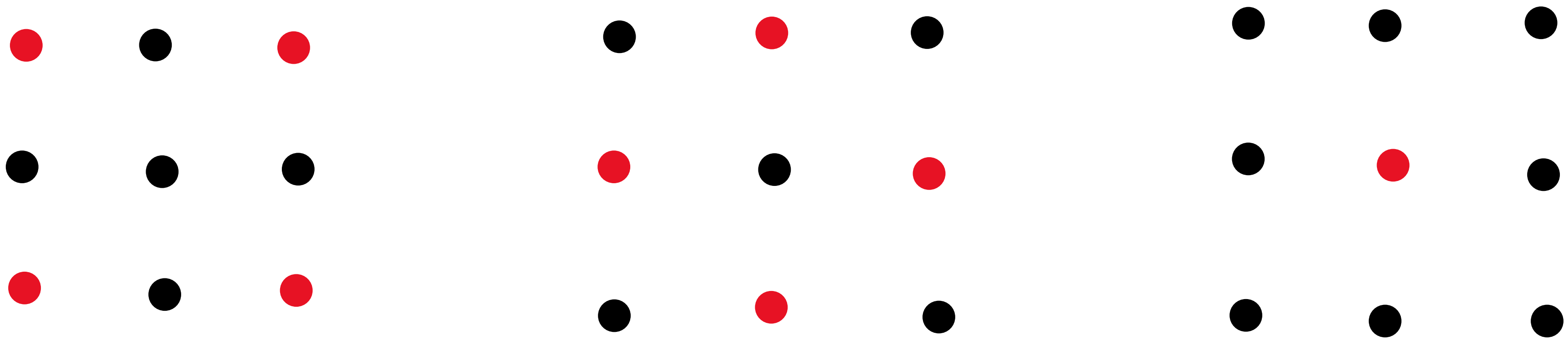


S = {0,1}^n is a capset, so $f(n) \geq 2^n$

In fact, $f(m+n) \geq f(m)f(n)$

Capsets

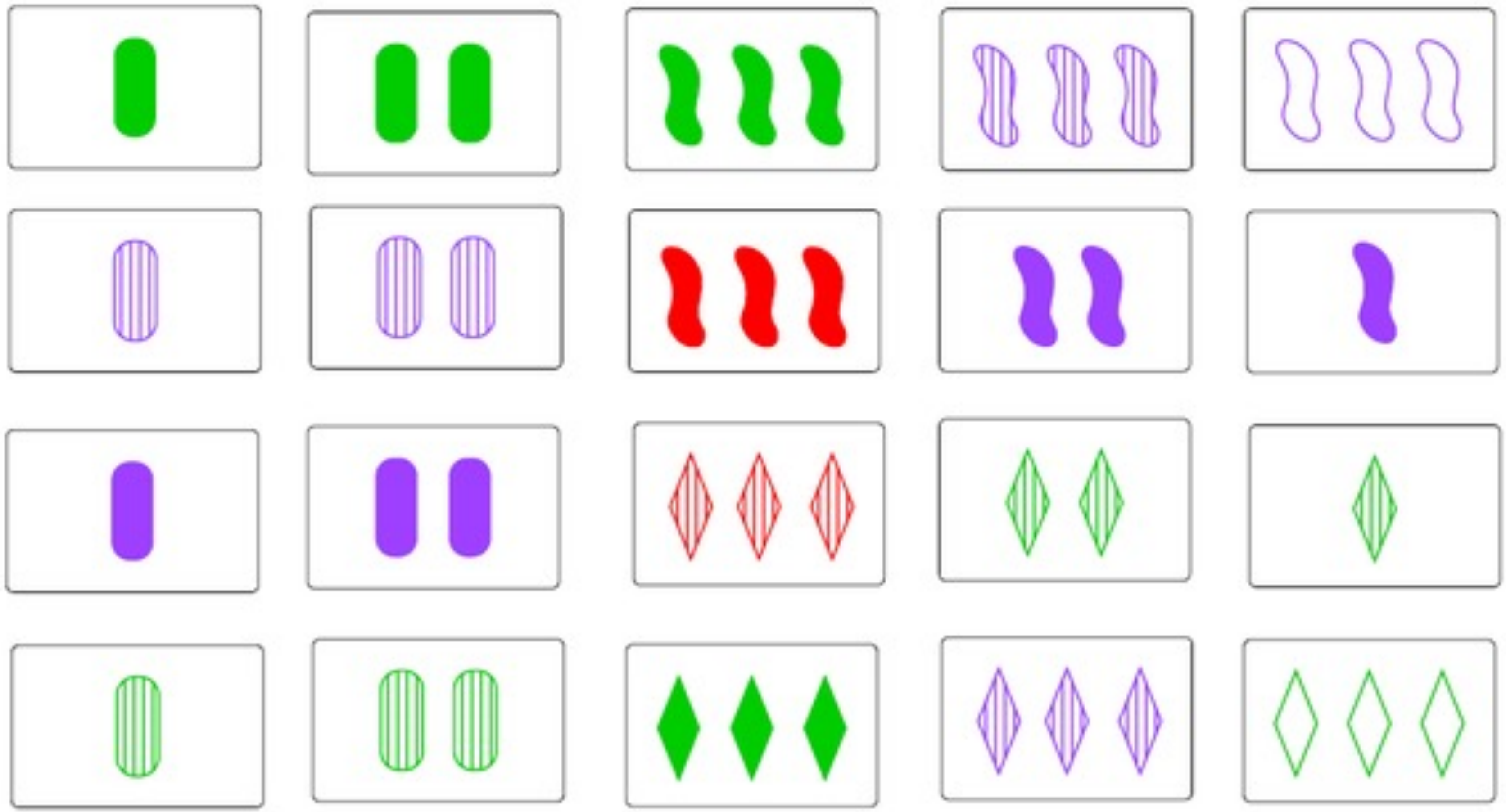
But:



f(3) = 9!

Capsets

- f(4)=20



Capsets

What is lim $f(n)^{1/n}$?

At least $(20)^{1/4} = 2.11$ ...
At most 3.
E_-Giswijt, 2017: at most 2.756.
Best known lower bound: Tyrell, 2022: at least 2.218.

Can a machine generate a large capset?

(even a single large example can narrow the gap!)

Can a machine generate a large capset?

Standard approach: somehow optimize over functions $F\colon \boldsymbol{R}^n \to \boldsymbol{R}$, rewarding those such that

$$\{v \in \{0,1,2\}^n : F(v) > 0\}$$

is a large capset.

[Submitted on 29 Apr 2021]

Constructions in combinatorics via neural networks

Adam Zsolt Wagner

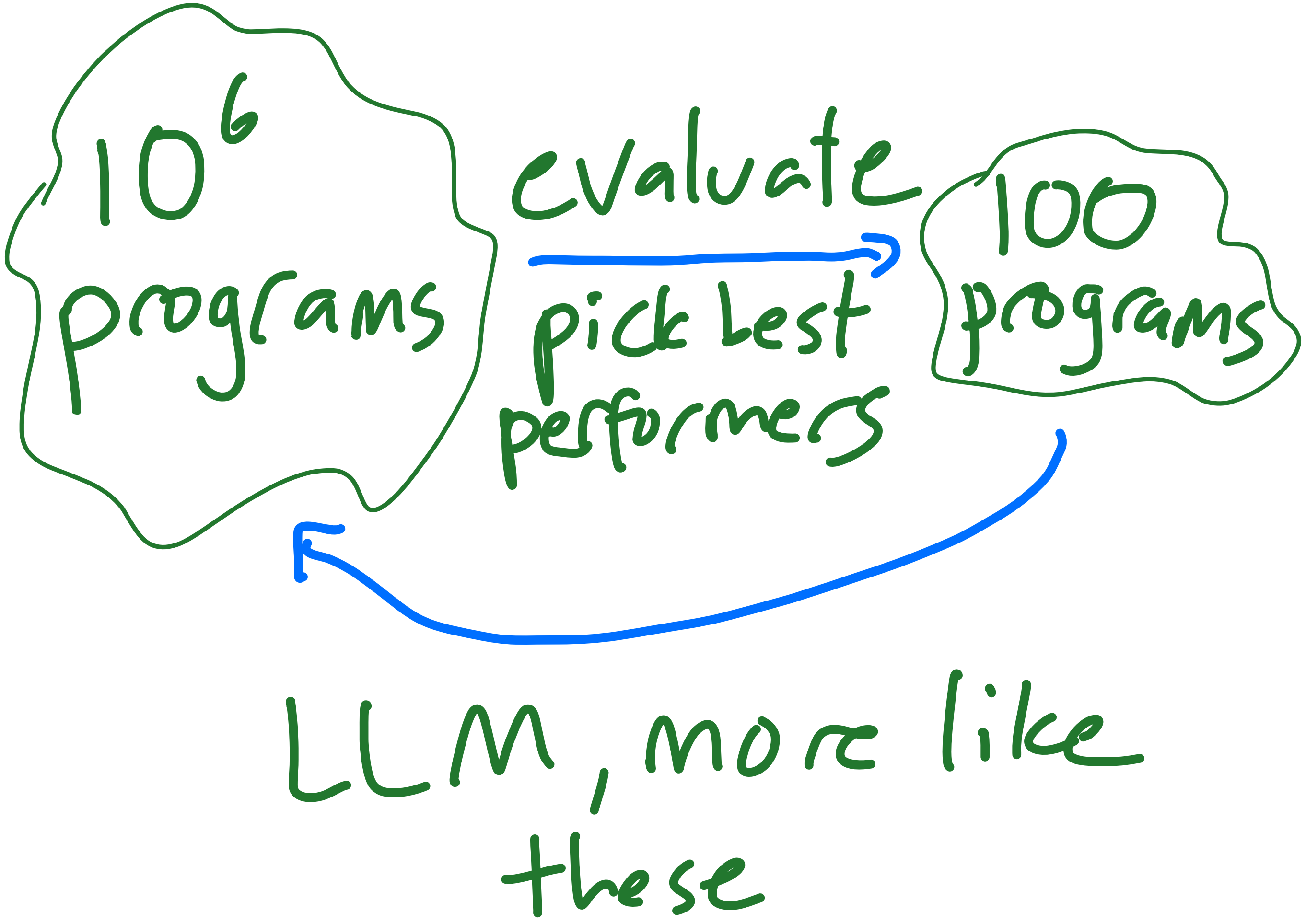
We demonstrate how by using a reinforcement learning algorithm, the deep cross-entropy method, one can find explicit constructions and counterexamples to several open conjectures in extremal combinatorics and graph theory. Amongst the conjectures we refute are a question of Brualdi and Cao about maximizing permanents of pattern avoiding matrices, and several problems related to the adjacency and distance eigenvalues of graphs.

OUTPUT: Some huge mess of a function.

FUNSEARCH

Instead: search the space of short Python programs, rewarding those whose output is a large n-dimensional capset.

FUNSEARCH: A CARTOON



```python
def priority(el: tuple[int,...],
↪   n: int) -> float:
  score = n
  in_el = 0
  el_count = el.count(0)

  if el_count == 0:
    score += n**2
    if el[1] == el[-1]:
      score *= 1.5
    if el[2] == el[-2]:
      score *= 1.5
    if el[3] == el[-3]:
      score *= 1.5
  else:
    if el[1] == el[-1]:
      score *= 0.5
    if el[2] == el[-2]:
      score *= 0.5

  for e in el:
    if e == 0:
      if in_el == 0:
        score *= n * 0.5
      elif in_el == el_count - 1:
        score *= 0.5
      else:
        score *= n * 0.5 ** in_el
      in_el += 1
    else:
      score += 1

  if el[1] == el[-1]:
    score *= 1.5
  if el[2] == el[-2]:
    score *= 1.5

  return score
```

Observations on Funsearch

It actually works! Matches best known lower bound on f(n) (i.e. matches largest known n-dimensional capsets) for n = 1,2,..7, and improves f(8) from 496 to 512.

Observations on Funsearch

It actually works! Matches best known lower bound on f(n) (i.e. matches largest known n-dimensional capsets) for n = 1,2,..7, and improves f(8) from 496 to 512.

(and improves the lower bound in general, though Naslund recently beat this…)

Observations on Funsearch

We needed to search a very restrictive class of functions: *priority functions* that assign a score to each element of $(\boldsymbol{Z}/3\boldsymbol{Z})^{\boldsymbol{n}}$ *in advance*, then add vectors to the capset in order of score, skipping any that violate the capset rule.

!!!!

```python
def priority(el: tuple[int,...],
↪   n: int) -> float:
  score = n
  in_el = 0
  el_count = el.count(0)

  if el_count == 0:
    score += n**2
    if el[1] == el[-1]:
      score *= 1.5
    if el[2] == el[-2]:
      score *= 1.5
    if el[3] == el[-3]:
      score *= 1.5
  else:
    if el[1] == el[-1]:
      score *= 0.5
    if el[2] == el[-2]:
      score *= 0.5

  for e in el:
    if e == 0:
      if in_el == 0:
        score *= n * 0.5
      elif in_el == el_count - 1:
        score *= 0.5
      else:
        score *= n * 0.5 ** in_el
      in_el += 1
    else:
      score += 1

  if el[1] == el[-1]:
    score *= 1.5
  if el[2] == el[-2]:
    score *= 1.5

  return score
```

Observations on Funsearch

No hallucination problem because assessment doesn't involve the LLM!

Observations on Funsearch

Results are interpretable; we can read the code ourselves and try to figure out what it's doing.

Observations of Funsearch

My attempts to seed Funsearch with “well-chosen” functions did not improve performance!

Observations on Funsearch:

Funsearch does not seem to learn that $(Z/3Z)^n$ has symmetry by GL_n(F_3), but does learn it has symmetry by S_n.

```
if el_count == 0:
  score += n**2
  if el[1] == el[-1]:
    score *= 1.5
  if el[2] == el[-2]:
    score *= 1.5
  if el[3] == el[-3]:
    score *= 1.5
```

Observations on Funsearch

Funsearch does not seem to learn how to combine an m-dimensional capset and an n-dimensional capset into an (m+n)-dimensional capset.

Funsearch can:

- Learn to produce a program that gives good (better than previous human-generated) solutions for a particular n

Funsearch can:

- Learn to produce a program that gives good (better than previous human-generated) solutions for a particular n
- ~~Learn to produce a program that gives good solutions for general n~~

Funsearch can:

- Learn to produce a program that gives good (better than previous human-generated) solutions for a particular n
- Learn to produce a program that gives a human reader a good idea for creating good solutions for general n (interpretability!)
- ~~Learn to produce a program that gives good solutions for general n~~

Funsearch can:

- Learn to produce a program that gives good (better than previous human-generated) solutions for a particular n
- Learn to produce a program that gives a human reader a good idea for creating good solutions for general n
- ~~Learn to produce a program that gives good solutions for general n~~
- ~~Enslave/eat/otherwise annihilate all humans~~
- ~~Render research mathematics obsolete~~

Forward from here

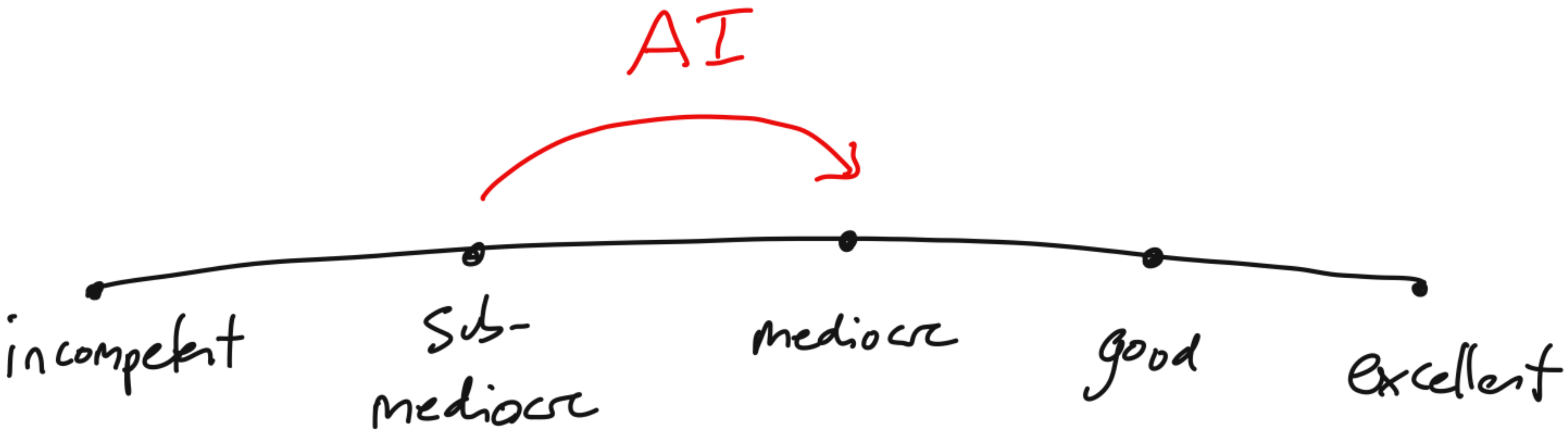
What kind of problems are most suitable for Funsearch-style attack? Can we get outside combinatorics?

Forward from here

DeepMind wants to work on hard problems; I want to work on easy problems

Forward from here

DeepMind wants to work on hard problems; I want to work on easy problems




Get up and running right!...
with the Heathkit H8 8080 Personal Computing System!


apple

Forward from here

“we have empirically observed that the results obtained in this paper are not too sensitive to the exact choice of LLM, as long as it has been trained on a large enough corpus of code.”

Can we make Funsearch (or Funsearch-likes) a tool for working mathematicians?

Learning the Möbius function

$$\mu(n) =$$

- 1 if n is the product of an even number of distinct primes
- -1 if n is the product of an odd number of distinct primes
- 0 if n has a nontrivial square factor

Can a neural net learn the Möbius function?

First try: learning the Möbius function

Neural nets are very good at:

INPUT:

cat faucet cat

OUTPUT: A function F from space of pixels (R^N) to [0,1] sending pictures of cats to 0 and pictures of faucets to 1

First try: learning the Möbius function

INPUT:

(17,1)

(15, -1)

(105, -1)

(12, 0), ….

First try: learning the Möbius function

INPUT:
(17,1)
(15, -1)
(105, -1)
(12, 0), ….

OUTPUT:
A function that matches Möbius much better than chance!

First try: learning the Möbius function

$\mu(n)$=

- 1 if n is the product of an even number of distinct primes
- -1 if n is the product of an odd number of distinct primes
- 0 if n has a nontrivial square factor

Our function: 0 if n is a multiple of 4, random ± 1 otherwise.

Philosophical question:

If the machine, given f(p) data points, can learn to return 0 when n is a multiple of p^2, should we say the machine has learned to detect squarefreeness?

Philosophical question:

If the machine, given f(p) data points, can learn to return 0 when n is a multiple of p^2, should we say the machine has learned to detect squarefreeness?

Accuracy of answers near 100%, proportion of concept learned is 0%.