**Accelerate Programme Blog Post**By Mikel Bober-Irizar, MPhil student in Advanced Computer Science

AI today powers everything from your phone's autocorrect to your YouTube feed. However, all these models have one thing in common: they're very good at doing things in their training dataset, and are very bad at things outside it. A particular example of this is self-driving cars, which were "nearly there" 6 years ago [1] and have made surprisingly little progress since. The difficulty is that AI systems underperform when met with unexpected situations outside their training data (like a tree fallen on a road). To build more robust, safe and versatile AI systems, we need to build models that are more intelligent, and can perform reasoning to solve novel situations in the way that me and you effortlessly do every day. But building such intelligence is a daunting task (even defining intelligence is hard!), so where do we start?As part of my Master’s course, I took a course on Unconventional Approaches to Ai with Accelerate Research Fellow Dr Soumya Banerjee and explored new ways machine learning systems can tackle abstraction and reasoning tasks.

**A long journey to intelligence**

In 1967, long before the advent of AI like we know it today, Mikhail Bongard published *Pattern Recognition*, noting how scientists such as Alan Turing have long posited the concept of a thinking machine; but while machines can be built to solve specific tasks (such as solving quadratic equations or playing chess), no progress had been made to imitate or even understand the ability of us humans to adapt to new situations. Bongard suggests that pattern recognition, the ability to decompose situations into objects and concepts, is central to the abilities of human intelligence.

Bongard introduced a set of logic puzzles (known now as the Bongard Problems) which were designed to test whether a (hypothetical future) machine can perform the sort of logical reasoning that we take for granted each day. Each problem has 12 images, split into two groups. The goal of each problem is to answer the question: "What sets apart these two sets of images?" Can you solve the below Bongard problems?[2]

A diagram of a graph

Description automatically generated with medium confidence

Over 50 years later, a machine that can solve these types of problems has yet to surface. It turns out that modern machine learning algorithms are exceptionally good at learning patterns in large training data, but struggle to generalise to unseen situations that never appeared during training.

In 2019, Francois Chollet called this ability *broad generalisation*, and introduced a new more rigorous benchmark, the Abstraction & Reasoning Corpus (ARC), against which computer systems can be tested. Similarly to the Bongard Problems, ARC problems ask people (or computers!) to solve logic puzzles involving colourful grids, where each problem involves learning a transformation from 3-5 examples and applying it to a test input to produce an output grid that must be pixel-perfect:

A screenshot of a game

Description automatically generated

There have been three international competitions on ARC with over $100,000 in prizes, but we've yet to see a machine-learning-based solution that can solve a significant portion of ARC problems: the best solutions (while impressive) instead rely on hand-crafted rules and heuristics. From an ML perspective, we've got as little as 3 training examples and 10300 possible outputs.

**A new approach**

In my Masters project, I looked at two new ways that a machine-learning system could be used to solve ARC tasks.

The first is a system called DreamCoder (originally built by MIT), which uses neural networks to write **computer programs** that represent transformations, which we adapted to work on ARC tasks. By defining a programming language of potentially useful primitives (like rotation, filling in), DreamCoder is able to compose these into complex transformations. A neural network trained on "dreamed" problems looks at a task and suggests which transformations to try next, massively reducing the exponential search space. Our language, PeARL (Perceptual Abstration and Reasoning Language), allows DreamCoder to use higher-order functions and lists in its solutions.

Here's an example of a task solved by DreamCoder. It first constructs a function that says "set the background to orange", and then applies this function individually to each object in the grid, before overlaying them on their original positions. This program gives the correct output for the test example!

A diagram of a computer program

Description automatically generated

You might be wondering at this point... well, what about Large Language Models (LLMs)? LLMs are an interesting exception: by training a model on *all the data you can get your hands on*, every possible task you could give it becomes an interpolation of things that it's seen in its training data. This has led to staggeringly good performance across a variety of tasks. So we also ran comprehensive tests across a text-based translation of ARC, and found that the biggest LLMs today are able to solve a surprising portion of ARC, with GPT-4 just beating our own DreamCoder solution.

**What's next?**

Imparting abstraction and reasoning abilities into our machine learning models remains an unsolved and elusive touchstone problem, and is likely to remain so for some time. However, we see two very different avenues can both make progress: a complex symbolic reasoning system harnessing neural networks (like DreamCoder), as well as relying on the emergent reasoning capabilities of extremely large foundation models. When we crack this problem, it will be fascinating to see on which side we end up.

If you want to learn more about this work, please read our paper: https://arxiv.org/abs/2402.03507

[1] <https://www.theverge.com/2018/7/3/17530232/self-driving-ai-winter-full-autonomy-waymo-tesla-uber>

[2] If you're interested in learning more about Bongard Problems, or the solutions, check out Harry Foundalis' diligent collection: <http://www.foundalis.com/res/bps/bpidx.htm>