Cellular automata

CA are a modelling technique, consisting of cells that interact and initialize and grow and shrink and propagate. All their interactions are local, yet global behavior emerges, almost as if we are looking at a scan of the brain. Very close, actually, very very close. So close that an interesting hypothesis imposes itself.

Cellular automata can be trained to recognize patterns, make accurate predictions, and potentially serve as a form of artificial intelligence (AI). A hypothesis I researched in debt in my thesis and will explain in the following 8 minutes.

So, first things first: CA, more specifically NUECA. CA means a grid with local interactions that evolves over timesteps following specific rules. ECA means that the grid has but one dimension, the local interactions are limited to the direct neighbors and the states of each cell are limited to 0 and 1. NUECA means that not every cell follows the same rule.

So, what would such a CA look like? First define a fixed amount of cells, say 25. Then define a fixed amount of timesteps, say 25 as well. Now we need to choose an input-output pair, for example a number as input and half of the number as output. The last step is to define input cells at timestep 0 and output cells at timestep 25. What primarily defines this CA is the ruleset, the set of rules followed by the different cells. Now we can run. The initial seed is blank, except for the input cells, which follow the input. Then we evolve this for 25 timesteps, resulting in a final configuration. From this final configuration we learn the output. Now we can compare. The expected output was 0011 and the actual output was 1100, thus it is bad. As bad as possible, to be precise. But of course, we didn't define the CA to connect one input to one output. Instead we want to find patterns in input-output datasets. This we run this same CA against multiple pairs and take the average of the reported fitness's.

But let's try again, with a different ruleset and this time the original output is better, but it could be even better, and the average is better as well. By continuously iterating we hope to find better and better fitnesses, resulting in fitness-growth. Fitness growth is defined as follows, where we take the final fitness minus the original fitness. Very important is that the original fitness depends on the distance metric that is used to compare calculated output and actual output. Some distance metrics leave little growth room, some leave a lot. To cancel out this effect, we divide the actual growth by the growth potential, how much could the fitness have grown?

The process of trial and error lands us squarely in the field of Genetic Algorithms.

GA are a class of optimization algorithms that use calculations based in evolution to learn optimal solutions. A schematic representation looks like this. Now, the most important thing in this graph is that every single step has multiple options. For example

mutation can happen through bitflip mutation or gaussian mutation. Very important is the step where we define the initial ruleset. This could be populated with a random set of rules, but we could also limit the choices. To figure out a classification for the initial population of the ruleset, a thorough study on MNUECA was performed.

Which of these choices matter, and which don't, which are better and which are worse, which to include and which to evade? It is impossible to answer this question as the behavior of LCA and GA is so unpredictable and thus I implemented all of them. For every implementation I measured the fitness improvement of the CA. This resulted in different versions of the GA leading to different fitness growths. To separate out the different effects an ANOVA analysis was performed and a linear model was fitted resulting in an optimal GA for teaching LCA.

An optimal solution that worked, a solution that showed actual growth. But to adequately quantify this growth we took a page out of the pharmaceutical playbook. For a medicine to be approved, it must not show that it works, but it must work better than the current state of the art. The state of the art in pattern recognition: NN. This it is time to pit the LCA and the NN against each other.

Both start at more or less the same location but quickly evolve over the different generations until they flatten out and finish at comparable results.

What does this mean? We have found an alternative way of doing pattern recognition that performs comparably with Neural Networks. However, this simple observation hides some truths. First of all, both the LCA and NN are plotted on generations, but generation doesn't mean the same for both and by the time the LCA has performed the first 10 generations, the NN is almost halfway through. This means that our technique that performs about equally takes magnitudes of extra time and computing power. However, the NN had multiple hidden layers and nodes and connections, each defined by a weight and bias, resulting in numerous parameters, each picked form a continuous distribution between 0 and 1, meaning that it is heavy on memory. Our CA, that does a comparable pattern recognition is fully defined by a mere 25 numbers, and each is an integer between 0 and 255 including, resulting in a magnitude less of memory requirement.

So, in the end, can CA perform pattern recognition and be used as a type of AI? Yes*.