Function approximation using Cellular Automata and Genetic algorithms

*University of New Mexico Complex Adaptive Systems. Code available at: https://github.com/AbubakarKasule/CS523-Final-Project

Abubakar Kasule
Computer Science (graduate student)
University of New Mexico
Albuquerque, NM USA
akasule@unm.edu

***Abstract*—In this paper, we attempt to find ways to use cellular automata to perform basic arithmetic. To achieve this end, we employed the concept function approximation as well as genetic algorithms to help us find cellular automata configurations that allowed us to perform a specific arithmetic task within a range of values. We converted cellular automata into functions by limiting their cell states to a range of values determined by the numerical base system selected for the automata (i.e, for $base_3$ we have the set of states {0, 1, 2}). We also used the automata's initial population to represent any given input number in the given numerical base system for this automata. Correspondingly, the automata's final population states represented the 'output' of the automata. The automata class we developed in python for this project was flexible enough to allow for a user to define an automata for any base between 2 and 16, with their choice of what constitutes a neighborhood in this automata. In the end, we found that, at least while limited to a single iteration (rule is only applied once), cellular automata's require a the use of what we call 'perfect memory neighborhoods' to be able to consistently approximate a given function. We would like to note however, that this approach is more analogous to an encoding technique where we are, in essence, simply looking for a rule, base system, data dimension, and neighborhood construction that encodes all the information we need. We remain hopeful about the possibility of pure computational solutions which incorporate more iterations to allow for simpler neighborhoods.**

***Index Terms*—cellular automaton, function approximation, non von Neumann machines, genetic algorithms**

I. Introduction

Cellular Automata as a concept were first imagined in the 1940s by Hungarian Mathematician John von Neuman. Cellular Automata's were initially conceived as a potential alternative to the standard CPU and RAM based model of computation. Since their discovery, many researchers have managed to find ways to make automata display a variety of different behaviors such as cellular automata where a particular pattern of cell states are capable of self-replication.

Stephen Wolfram is one of the most prominent researchers to studied the nature of cellular automata [1]. In his exploration of the space, Wolfram limited himself to the studying the elementary cellular automata and its corresponding 256 rules. Wolfram defined these elementary automata as 1-Dimensional automata with only two states where every cell is only neighbored by the cells directly to its left and right. Wolfram's survey of the Elementary automata rule space led him to create 4 unique classifications for cellular automata rules: (1) evolves to stable state quickly, (2) transitions between chaotic and stable states, (3) nearly complete random behavior, (4) evolves to create complex patterns. Since Wolfram's survey, other researchers, such as Norman H. Packard and Melanie Mitchell, have employed the use of genetic algorithms to find rules with certain defined behaviours. Melanie Mitchell and her team were able to identify an automata configuration that was capable to performing a majority classification task with 100% accuracy.

In this paper, we seek to continue Norman H. Packard and Melanie Mitchell's genetic algorithm based cellular automata research. We use our own custom multidimensional cellular automata module to develop a cellular automata genetic algorithm (CAGA) that could evolve cellular automata rules until a rule was found that could approximate a given target function up to a certain threshold. We primarily focused on an approach that we termed 'perfect neighborhoods'. We define a cellular automata as using the perfect neighborhood approach when the definition of the neighborhood of a given cell contains more cells than there are in the cellular automata's cell population.

II. Methods

Multi-Dimensional Cellular Automata Class

We implemented the Multi-Dimensional Cellular Automata class to be able to represent a multitude of different cellular automata configurations in order to maximize the search space for our genetic algorithm. To maximize our search space, we tried to think about what variables could be altered to increase the number of rules available to a given automata configuration. To achieve this, we went back to wolfram's elementary automata model and derived the following generalized formula for determine the total number of rules for a given configuration:

$$length\ of\ rule = base^{neighborhood\ size}$$
$$total\ number\ of\ rules = base^{length\ of\ rule}$$

Another important observation we made while re-examining the elementary automata was uncovering the logic behind the ordering of neighborhoods in the figure illustrating the transitions between neighborhoods and states as seen in Fig. 1. If we take the leftmost neighborhood that transitions into the most significant bit of rule 30 and we treat it as a binary number, we get '111' or as we know it in $base_{10}$ '7'. 7 also happens to be the largest whole number less than the rule length for elementary automata, which is 8. If we applied this same process to each neighborhood transition, we would get values in the range $[6, 0]$. Using this knowledge, we create a transition dictionary for a given automata configuration by reversing this process while looping through the range (length of rule, 0].

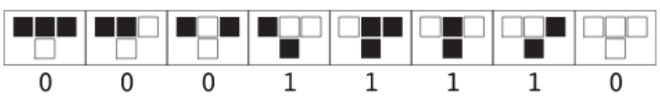

Fig. 1. Rule 30 transition map for elementary cellular automata

Finally, we stored the population of cells in our Multi-Dimensional Cellular Automata (MDCA) class in a regular 1 dimensional array. The manner in which we achieve higher dimensional cellular automata is by allowing the user to define their own neighborhoods. These two features are interconnected as the only time the automata interacts with the data is when it is looking for a cell's neighbors. Both these features were implemented by having users of the automata provide the automata with a custom neighborhood function that takes an index of a cell and returns a list containing all the indices of its neighbors. Users need to consider how they want to represent their higher dimensional data as an array. Accordingly, they should consider how this representation will determine the indices returned by their neighborhood function. The MDCA class also has a generate_media function that produces visual representations of 1 & 2D automata.

Genetic Algorithm Structure

For our genetic algorithm, we sought to select for Cellular Automata rules that approximate a given function well. Due to limitations to the computational power available to us, every individual in one of our genetic algorithms will have the exact same configuration except for their rule. Writing a genetic algorithm that considered more features than just rules would have been computationally expensive. In other words, cellular automata rules function as the 'DNA' of this system with x-its getting swapped in and out when an individual mutates

Term	Definition
Dimensions	Refers to the shape of the grid that contain the automata's cells
Neighborhood	The list of cells considered to be "next to" a given cell
Neighborhood size	The number of cells in a given neighborhood
Transition Dictionary	Dictionary that takes a neighborhood as a key and returns the appropriate transition state.
Base	Base system used to represent our numbers. Also refers to the number of cell states.

TABLE I
DEFINITIONS TABLE.

a particular transition. Crossover was also implemented in the create_children function by first randomly selecting a size for the crossover region that is less than the rule length and secondly selecting an appropriate start index for the crossover region. The crossover regions of two parent configurations are then swapped to create their children. Because each pair of rules creates two new rules, the lower performing half of the previous generation is removed from the population prior to the remaining configurations being randomly paired up. To help us converge on rules with our desired behaviour, automata configuration whose fitness falls below a certain threshold are terminated and replaced with a new randomly generated configuration.

The Perfect Memory Neighborhood

For all of our experiments, we only allowed for one iteration of each rule to be applied. This means that a candidate rule must contain the properties to produce the correct output after only being applied once. This is a pretty challenging task, especially for small neighborhood definitions which do not really provide the automaton with much context. With simple neighborhoods (less than 5 neighbors), we struggled to find a rule that was more than 10% accurate in a reasonably sized testing range. In order to get past this barrier, we devised 'perfect memory neighborhoods'. We decided to name them this because we defined them to be neighborhoods that contain more cells than there are cells in the automata itself (implying repeat entries). The logic behind why perfect neighborhoods were effective at boosting our accuracy is that a 'perfect neighborhood' could store all the information an automata could need to make a decision on what to flip a particular xit in a particular number into in order to correctly approximate the target function. Because of this, running a genetic algorithm along side the use of perfect neighborhoods is analogous to finding the rule with the highest number of correct transitions.

Designing perfect neighborhood comes with its challenges. For every neighbor added, the search space is increased by a considerable amount. For this reason, the genetic algorithms we made ran much slower when working with perfect neighborhoods. Minimizing size is not the only difficult task associated with designing a perfect neighborhood. While there may be other more effective approaches, the perfect neighborhoods we designed were meant to convey the number in question and the index of the cell/bit in question. With those two pieces of information, we just have to find a rule with the correct transition in that location. So far, we have not been able to design a perfect memory neighborhood that could convey those two pieces of information with no conflicts over the entire range of real numbers although we have gotten fairly close at creating some that function within a fixed range of values.

III. Results

$f(x) = 2 \times x$

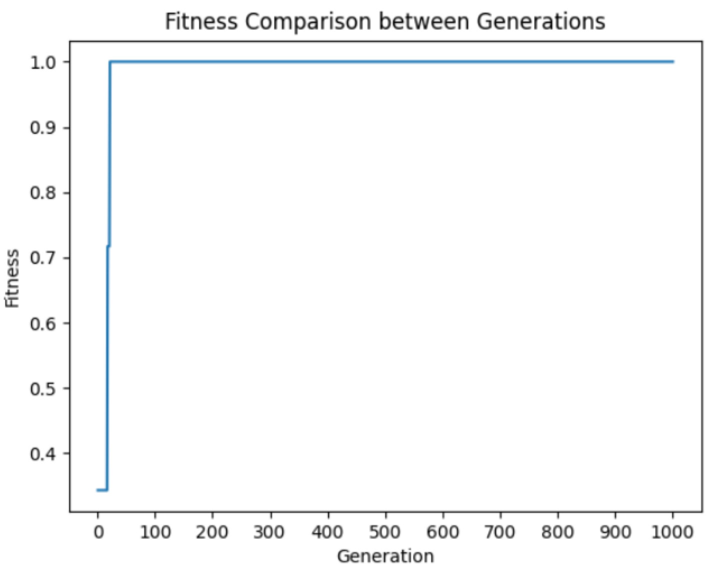

Fig. 2. Max Fitness - Generation graph for the genetic algorithm that produced rule 170

The first target function we chose to train for was a relatively easy one. We were already aware of the fact that shifting a binary number to the left effectively doubles it. After some careful observation of Fig. 1, we noticed that, if we took each cell on the right of each neighborhood, we could construct a rule that would shift any given input to the left effectively doubling its value. The rule that we had constructed was rule 170. Surely enough, our genetic algorithm was very quickly able to find rule 170, which had a 100% accuracy as shown in Fig. 2. Interestingly enough, a subsequent run in an arbitrary dimension space unveiled another rule that effectively had the same behaviour of rule 170. Both these rules are listed at the end of this report.

$f(x) = x + 3$

$f(x) = 2 \times x$ was a fairly trivial task as we were already aware of the existence of an ideal solution within the contained search space of the elementary automata. Despite the ease we encountered while working with $f(x) = 2 \times x$, $f(x) = x + 3$ was surprisingly difficult to find a rule for and is the reason behind our motivation to create perfect

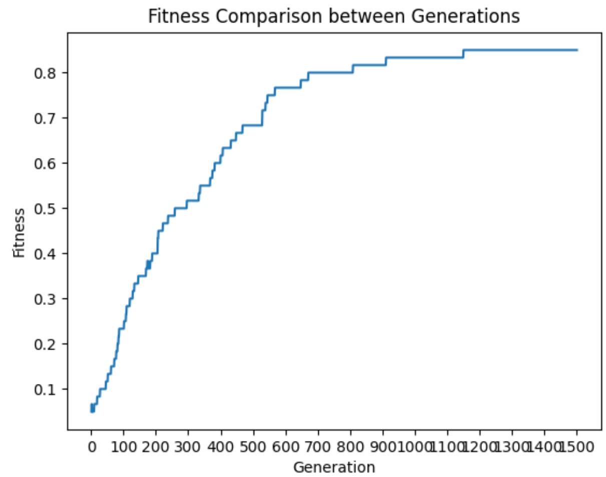

Fig. 3. Max Fitness - Generation graph for the genetic algorithm that produced rule number 3 in the rule table at the end of this report

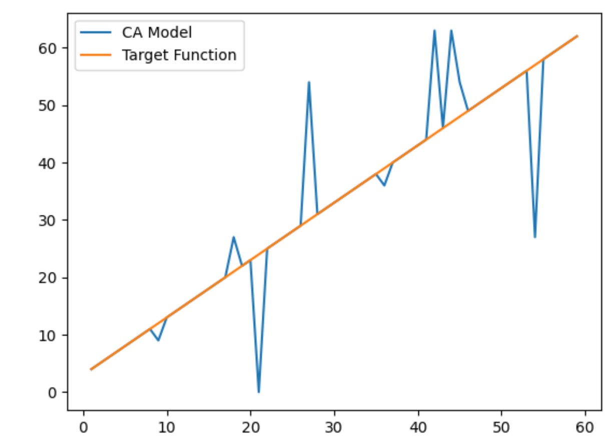

Fig. 4. Approximation graph comparing the Automata's results to the true outcomes to their respective target function $f(x) = x + 3$

memory neighborhoods. After the inclusion of perfect memory neighborhoods, the runtime of our genetic algorithm increased exponentially, but luckily, so did our accuracy. The rules we were finding went from having an accuracy of about 10% to the maximum accuracy we achieved of 85% when testing with numbers in the range between 0 and 60 (due to 63 being the biggest number our automata could represent). Fig. 4 indicates that our approach is not really terribly invested in the nature of the function it is trying to copy. In fact, we believe that the wild fluctuations in error are due to the fact that our approach only cares about finding and storing optimal transitions. If an incorrect transition causing an error happens to fall on a significant bit, the magnitude of that error is increased greatly. This indicates to us that our approach is a bit 'all or nothing' in how it approximates function value.

$$f(x) = x^2$$

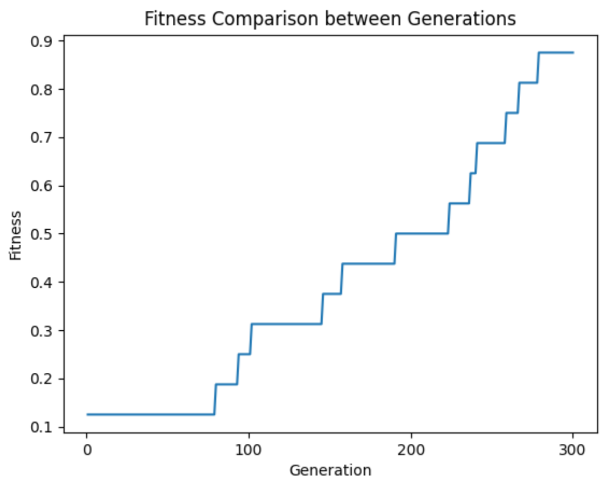

Fig. 5. Max Fitness - Generation graph for the genetic algorithm that produced rule number 4 in the rule table at the end of this report

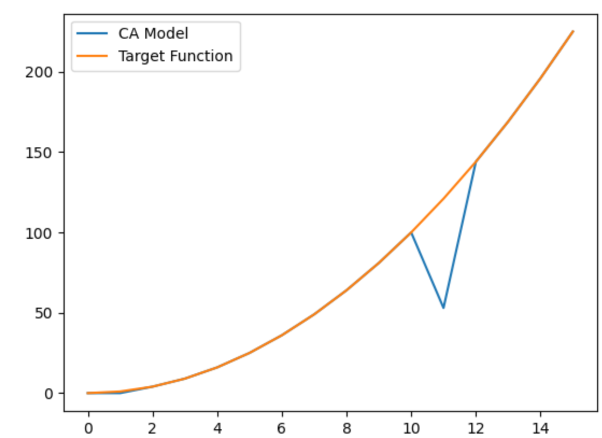

Fig. 6. Approximation graph comparing the Automata's results to the true outcomes to their respective target function $f(x) = x^2$

$f(x) = x^2$ was the most complex function we studied. However, due to computation limitations, we could only search for rules that functioned in the fairly limited range of [0, 15]. After several hours of processing, we were able to identify a rule that was around 87% accurate in the given range as shown in figures 5 & 6. We also utilized perfect neighborhoods for this target function which further increased runtime.

IV. Discussion & Conclusion

While we enjoyed searching this expanded space of cellular automata rules, we did learn a lot about their limitations and trade-offs. Our first major observation is that computation with cellular automata will require the incorporation of multiple iterations. In our project, we only ever gave each rule one iteration to solve a problem. We observed that, without using perfect neighborhoods to essentially memorize the correct answers, finding rules that can perform even basic arithmetic over a wide range of numbers in one iteration is difficult if not just simply impossible. Fig. 7 is a testament to this fact. The rule that we had found for $f(x) = x^2$ completely fell apart when tested on numbers outside the range of values it was trained on. This is further evidence that the perfect neighborhood approach looks more like encoding the answers to a given target problem than actual computation.

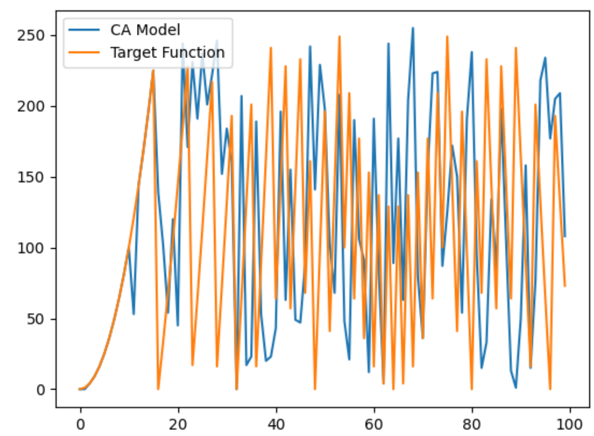

Fig. 7. Approximation graph identical to Fig. 6 but for a much larger range. y values have been modded by 256 as that is the integer overflow limit of the automata used here.

V. Reference & Contribution

Abubakar Kasule: Wrote the code for the genetic algorithm and authored this report.

VI. Rules

Mapping

Function	Rules
$f(x) = 2 \times x$	1 & 2
$f(x) = x + 3$	3
$f(x) = x^2$	4

TABLE II
Rules Table.

1

170

Neighborhood: elementary neighborhood
Neighborhood size: 3
Dimensions: (8,)
Base: 2

2

173794200776016493578319819358345308345793242
847256373025703935056742348913715725963142844
663479739863037044054492961090414250445728265
3928380633937662372

Neighborhood: elementary neighborhood
Neighborhood size: 3
Dimensions: (2, 10)
Base: 2

3

848637738250156742683518739735687334552097540
160551227103550133341631541255058378719611970
699960006487221740649979507286165159777196762
954567221094675667417960943932809090159213803
606880226172337122645658179860360141423498134
465485736412021259871139801428859978822431452
1701135117677788149181532934622882608805936711
821400674449777368313080103818521802489993542
165205115614323644813577654343171344132703454
490922809560164052704307234031318211362316148
928057947711583683138409909849428530345187967
220182631431112335498940154457734991151186688
757595393995862298983313626756593852932613330
080513044030429499811301299713850502044332661
965353775329082183167915549958580839065489893
286184014966913137205872269430117973693002755
146127957791057774077387106999310188904310378
486894472234204663242037454599633426563879055
216106999206767424985113117755148223697817468
403363516120126692753261721832615696179626096
248124916577982147108455528959078426878742243
454891001785175148659974836048474415908166099
016342479798247805936017497614305993038879433
525553440265930764295991584185477039537412105
333293330787856981583219342209000208409175324
719410622380601144124441575034677205613400890
318723136163198312022684867340489577728557204
242701346479943576

Neighborhood: perfect neighborhood
Neighborhood size: 12
Dimensions: (6,)
Base: 2

4

228517284526081819071897842917071291636737363
082469312429637153310726064791312223050375030
244975413356115058363621181802486479508766962
020329532496486857660388713340779555017078821
623996144721392483411722634154301726046703514
794891726142485012425619633091521032342370253
449369930336372864535769519604875875090448530
057882078336708816467270861610584894470234183
181737393194966910742610495272217632768756454
591753363655749528471960419748563725807960941
925150201760129738479822274748819611788623123
742679133325687905017733299118199368424071033
249025015318699739191266905588421594620454244
512453574517781093495137932458187587711163411
515358865408632922231596165776863115820346098
171198984750873668273095645710365807790203380
854650018499737868256111129494463539608014932
770411798218905881229819380999125426037061633
422046333798330465544202074066670565703104169
935946021348459521365074640579143791332724244
555370172222123271145617595112371830218742942
650981466473686454573239083802670964522243453
620107191856291489250295351240989750684818769
742671069029477561903028217398590693258766204
967758452777137587273295240979272576649039182
522837906982656309287387495263516607954482988
012052755351479845308295944306960565497441981
122927342914244708853024270929849663733041245
533002657774279204495230527330830047953247743
509569762140535791255694687385023364009354451
817776578321042760703549273789543204843481820
073196151976704447599070880583514830799417719
328284810746002900518122158922892690031766815
304738947476551547012634954049437574100794854
148728982994914641805411308778722139979192961
289524806109778269007458492028593017112249140
563744367611066264344793637744649062266152509
952356356392762124010839669611746750060631155
903421375200690423196398742546686246289222538
452105967187211251532876317908995136274541574
005435889018237425374929707957218610677814004
730514881622338487077896307722528778782611163
561725919701927230711490936062673810285952990
216688740333185570446261883514018481632194425
366958002511399662779979873112658434222605607
412453389423101259289111148698794866302099948
656238151537435103605706484025869320291539264
961280179280299744980232104105230026365589874
556443524090875885355395939040066589476197623
490085042720210615694128692517124035413637655
471417839126216486948398195745229147789564861
740575166885404814119451302230688610182944346
372239179927563671738764531237564802972746791
542859651177570555065881658338641168196469300
595966639055339456063750786137241208329538049
046722491413439227882240546583869186200917410
292524615607656851917186542503345970020176296
776969894212047429845907008440090753219829949
041301148289458803524802089177175796874330282
720474015538134902449520245760320273916202767
664610202......not even half of this number

Neighborhood: perfect neighborhood
Neighborhood size: 12
Dimensions: (8,)
Base: 2

References

[1] Mitchell, Melanie. Complexity: A guided tour. Oxford University Press, 2009.

[2] Melanie Mitchell, James P. Crutchfield, and Peter T. Hraber. 1994. Evolving cellular automata to perform computations: mechanisms and impediments. Phys. D 75, 1–3 (Aug. 1, 1994), 361–391. https://doi.org/10.1016/0167-2789(94)90293-3

[3] P. Kiely, "Introduction to genetic algorithms," FloydHub Blog, 25-May-2020. [Online]. Available: https://blog.floydhub.com/introduction-to-genetic-algorithms/. [Accessed: 20-Mar-2022].

[4] Packard, N.H., Wolfram, S. Two-dimensional cellular automata. J Stat Phys 38, 901–946 (1985). https://doi.org/10.1007/BF01010423