Dynamics of the Logistic Differential Equation and Transfer Entropy

Cheng En Ho
* The University of New Mexico, Albuquerque, NM 87131, USA
gretor0231@unm.edu

***Abstract*—Natural systems[1] are complex and hard to measure. In this paper, we will use logistic differential equations[2], which used for predicting populations growth rates, to simulate species in the natural world. We will also use Transfer Entropy[3] to define the relationship between two different species. We can easily understand Evolutionary Transitions and Top-Down Causation from this model. Although this model cannot present every individuals in the natural environment, we still can say most of individuals have the same behavior in natural systems and estimate it from oscillates behavior.**

***Index Terms*—Chaos, Logistic Map, Dynamics, Differential Equation, Entropy.**

I. Introduction

The objective of this paper is to demonstrate and reproduce Walker's[4] paper. We use logistic differential equation to simulate species living in the same environment. The logistic differential equation is defined as:

$$P'(t) = rP(t)[1 - P(t)/K]$$

In this equation, K is carrying capacity, r is the growing rate of P(t), and the whole equation can be solved by first order ODE[5]. This logistic model can present most species' growth behavior in the natural world. We expect species have exponentially growing rates[6] when it has enough resources and it also has exponentially decreasing rate when it start reaching the K limit environment condition. We defined the local dynamics of each element. Moreover, we also want to know the species' coupling strength[7] with other species, which can present species relationships in the whole environment. We use the logistic differential equation and add one more parameter ϵ . The species only has local dynamics of each element. We can change this value ϵ to observe species behavior.

$$x_{i,n+1} = (1-\epsilon) f_i(x_{i,n}) + \epsilon\, m_n \quad ; \quad (i = 1, 2, \ldots N) \quad (1)$$

Fig. 1. Logistic Differential Equation

In this paper, we also want to discuss dynamical information shared between two processes x and y. It can be determined by the Transfer Entropy equation.

$$T_{X \to Y} = H\left(Y_t \mid Y_{t-1:t-L}\right) - H\left(Y_t \mid Y_{t-1:t-L}, X_{t-1:t-L}\right)$$

Fig. 2. Transfer Entropy equation

We can use the tool from Java Information Dynamics Toolkit (JIDT) to calculate Transfer Entropy between two time series by this equation. Furthermore, we can calculate both sides which are Top-Down and Bottom-Up, to understand unbalances' information transfer behavior.

Therefore, we use the transfer entropy equation to tell us the relationship between many species. In short, the transfer entropy tells us the deviation from the expected entropy of two completely independent processes.

II. Methods and Results

Figure 3 is a collection of population plots over one hundred and ten time steps, labeled x in figure 3.

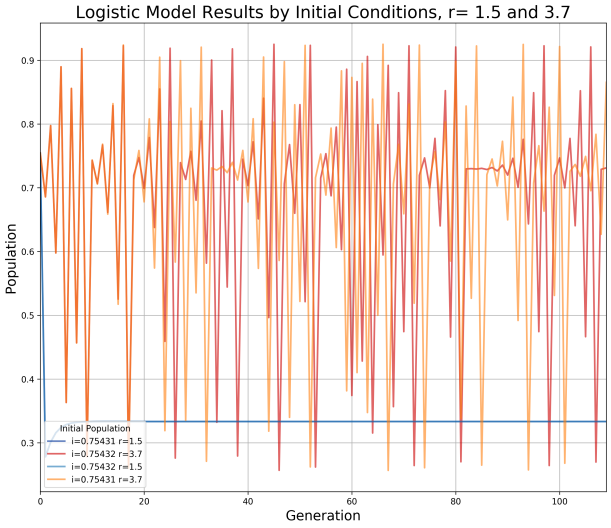
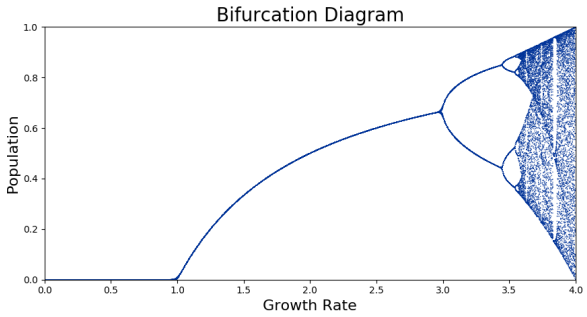


Fig. 3. Logistic-map with different initial-conditions

We see the different values of R and different initial conditions[8] in Fig 3 and we choose R = 3.7 and R = 1.5 for our examples. We expect for R = 3.7 that the system becomes chaotic[9], and we can see it start to diverge[10] after 20 time steps. We can get more information from the frequency domain[11], sometime it is just two single convolution[12] or has long periods cycle. To make sure it is chaotic system, we can gain R value information from the bifurcation diagram[13]. The R value will diverge for all population after R greater than 3.7.

We see when R = 3.7 that the system becomes chaotic as the time steps increases' and if R = 1.5 the system has a stable trajectory after only a few time steps. A nonlinear system which has a Sensitive dependence on initial conditions(SDIC)[14], often talked about as the theorized butterfly effect[15], is a system in which small differences in the initial value can result in large differences in the systems prediction after a



short number of time steps. We only changed initial conditions from 0.75432 to 0.75431, but the R = 3.7 cannot be predicted after 30 time steps[10] due to it being a chaotic.

The two random processes[16] and the amount of data sets is measured using Transfer Entropy[17]. According to the Entropy formula, we can find the relationship between two data sets[18]. We use the tool Java Information Dynamics Toolkit (JIDT)[19] to calculate Transfer Entropy between two of the time series. Transfer entropy is a non-parametric statistic[20] measuring the amount of directed (time-asymmetric[21]) transfer of information between two random processes. We calculate it from Figure 3 and show the result in figure 5 and figure 6.

Input files:

List names	number of elements	number of unique elements
i=0.75431 r=1.5	110	49
i=0.75431 r=3.7	110	110
i=0.75432 r=1.5	110	49
i=0.75432 r=3.7	110	110
Overall number of unique elements		**306**

Fig. 5. Venn Table

Figure 5 and 6 give us the information about Transfer Entropy. Transfer entropy is a non-parametric statistic measuring the amount of directed (time-asymmetric) transfer of information between two random processes. The table and Venn diagram[22] can tell us how random it is. We know it will become chaotic when R greater than 3.7, which has high Entropy. However, if we do not have too many time steps, the result might totally inverse, because it starts diverging after 20 time steps, according to signal compression theory or information transfer theory. If the signal are more independent or chaotic, we can get higher Transfer Entropy.

In the Fig 7, we are trying to reproduce Walker's paper. We use the same logistic differential equation from Walker's paper, but it has random value R between [3.9 4.0], and different steps of value epsilon. Thus, our figure is slightly different from Walker's paper. We use logistic differential equations to simulate species relationships in the same natural environment system. Here is a sample case to describe this equation. We

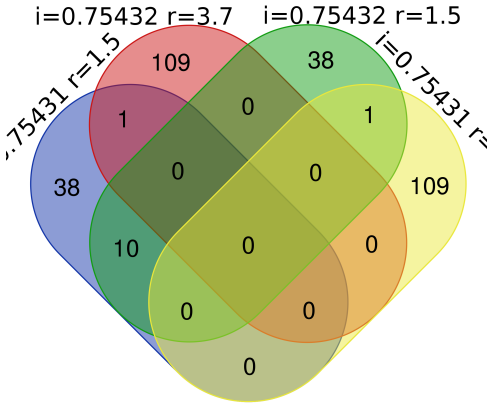


Fig. 6. Venn diagram

imagine a country has high natural resources at the beginning, and each individual has high correlation with the average of the whole world's populations, which is high epsilon. When they have enough natural resources, such as food, water, and land. They can crossover[23] at each time step, and the whole population would have exponential growth rate until they start reaching the natural resource limit K. The growth rate would become an exponentially decreasing rate. We can also imagine another small country at the ocean. They are highly independent and almost have no relationship with the whole world's populations, which is low epsilon.

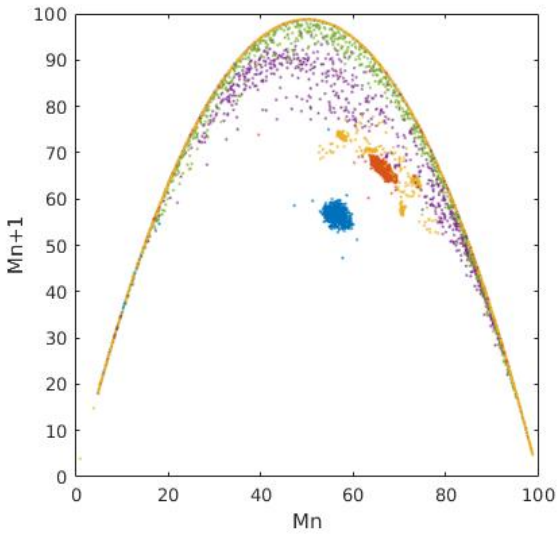


Fig. 7. Multiple values of epsilon

For $\epsilon = 0$, the whole specie is just a fixed value in the cloud. We can see the blue points cover in the same area, which means most of species have the same behavior. Our code increases' the value ϵ 0.1 for every calculation. We can see when ϵ grows, the whole population starts diverging.

In general, when ϵ is high, the population is like the mean of all populations. It is hard to predict their behavior, with each

species complete coupled. On the other hand, when ϵ is low, we can say each of the individuals has very similar behavior, because we see a cloud of fixed value.

We can also imagine a Figure of Transfer entropy from Figure 7. We got the entropy formula from figure 2. The transfer entropy tells us if there is no global behavior that the transfer entropy should be equal zero, because there is no independent species. However, it is not like a linear equation. In general, if we cannot gain any information from figure 7, which shows no converge behavior, the Transfer Entropy should be zero.

In figure 8, we evaluate the claim in Walkers' paper, which is epsilon = 0.075. We can say the individuals have three different clusters in this figure, which have three different behaviors. We can see it start to diverge, but it still has three fixed values.

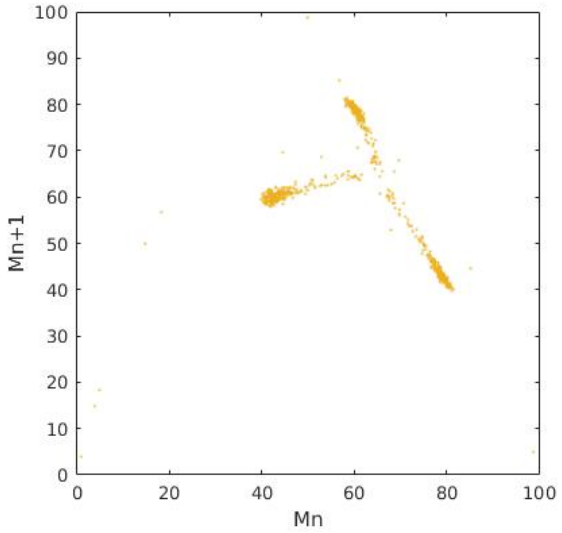


Fig. 8. Epsilon = 0.075

Finally, we try to analyze the claim "We suggest that many major evolutionary transitions might therefore be marked by a transition in causal structure[24]. We use logistic growth as a toy model[25] for demonstrating how such a transition can drive the emergence of collective behavior in replicative systems." and "we have proposed that increasing levels of biological complexity, corresponding to increased depth in the hierarchical organization of living systems, correspond to information gaining causal efficacy over increasingly higher levels of organization."

In the first claim, our paper is all discuss about logistic growth and Transfer Entropy. However, we assume all species have the same growing rate r, and the environment condition K is fixed. In fact, different species should have different growth rates, and the value K might also increase or decrease when species reach some conditions. Second, our information gain is according to the Transfer Entropy formula. Therefore, we have to assume increasing levels of biological complexity would cause individuals to be more independent. We can say human biological complexity is greater than ants, so human society should have higher Transfer Entropy and higher levels of organization, but complexity is hard to measure for any species.

III. Conclusion

We use the Logistic Differential Equation and Transfer Entropy to measure the relationship between a species with whole species in a natural environment. This module can be used in any natural systems which has Logistic Equation behavior. On the other hand, if we can find a system and it has logistic behavior, we can use this Toy model to estimate its Transfer Entropy. In this paper, we use the random value R between 3.9 to 4.0. If we change this growth rate value r, we can say different species should have different growth rates in the same environment. But if species are very similar like different human races, then we can use this Toy model to analyze human behavior.

IV. Contributions Statement

Cheng En Ho
-All the paper.

V. Code turn in

-All code in the attach file.

References

1 Woese, C. R., Kandler, O., and Wheelis, M. L., "Towards a natural system of organisms: proposal for the domains archaea, bacteria, and eucarya." *Proceedings of the National Academy of Sciences*, vol. 87, no. 12, pp. 4576–4579, 1990.

2 Gopalsamy, K. and Zhang, B., "On delay differential equations with impulses," *Journal of Mathematical Analysis and Applications*, vol. 139, no. 1, pp. 110–122, 1989.

3 Schreiber, T., "Measuring information transfer," *Physical review letters*, vol. 85, no. 2, p. 461, 2000.

4 Walker, S. I., "Evolutionary transitions and top-down causation," in *Artificial Life Conference Proceedings 12*. MIT Press, 2012, pp. 283–290.

5 Krstic, M. and Smyshlyaev, A., "Backstepping boundary control for first-order hyperbolic pdes and application to systems with actuator and sensor delays," *Systems & Control Letters*, vol. 57, no. 9, pp. 750–758, 2008.

6 Slatkin, M. and Hudson, R. R., "Pairwise comparisons of mitochondrial dna sequences in stable and exponentially growing populations." *Genetics*, vol. 129, no. 2, pp. 555–562, 1991.

7 Bethke, S., Allison, J., Ambrus, K., Barlow, R. J., Bartel, W., Bowdery, C., Cartwright, S., Chrin, J., Clarke, D., Dieckmann, A. *et al.*, "Experimental investigation of the energy dependence of the strong coupling strength," *Physics Letters B*, vol. 213, no. 2, pp. 235–241, 1988.

8 Blundell, R. and Bond, S., "Initial conditions and moment restrictions in dynamic panel data models," *Journal of econometrics*, vol. 87, no. 1, pp. 115–143, 1998.

9 Pecora, L. M. and Carroll, T. L., "Synchronization in chaotic systems," *Physical review letters*, vol. 64, no. 8, p. 821, 1990.

10 Mitchell, M., *Complexity*. Oxford University Press, 2009.

11 Malliani, A., Pagani, M., Lombardi, F., and Cerutti, S., "Cardiovascular neural regulation explored in the frequency domain." *Circulation*, vol. 84, no. 2, pp. 482–492, 1991.

12 Krizhevsky, A., Sutskever, I., and Hinton, G. E., "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

13 Fujii, H., Mimura, M., and Nishiura, Y., "A picture of the global bifurcation diagram in ecological interacting and diffusing systems," *Physica D: Nonlinear Phenomena*, vol. 5, no. 1, pp. 1–42, 1982.

14 Glasner, E. and Weiss, B., "Sensitive dependence on initial conditions," *Nonlinearity*, vol. 6, no. 6, p. 1067, 1993.
15 Shenker, S. H. and Stanford, D., "Black holes and the butterfly effect," *Journal of High Energy Physics*, vol. 2014, no. 3, p. 67, 2014.
16 Grimmett, G., Stirzaker, D. *et al.*, *Probability and random processes*. Oxford university press, 2001.
17 Lotfi, F. H. and Fallahnejad, R., "Imprecise shannons entropy and multi attribute decision making," *Entropy*, vol. 12, no. 1, pp. 53–62, 2010.
18 Wibral, M., Rahm, B., Rieder, M., Lindner, M., Vicente, R., and Kaiser, J., "Transfer entropy in magnetoencephalographic data: quantifying information flow in cortical and cerebellar networks," *Progress in biophysics and molecular biology*, vol. 105, no. 1-2, pp. 80–97, 2011.
19 Lizier, J. T., "Jidt: An information-theoretic toolkit for studying the dynamics of complex systems," *Frontiers in Robotics and AI*, vol. 1, p. 11, 2014.
20 Pettitt, A., "A non-parametric approach to the change-point problem," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 28, no. 2, pp. 126–135, 1979.
21 Bowen, J. M. and York Jr, J. W., "Time-asymmetric initial data for black holes and black-hole collisions," *Physical Review D*, vol. 21, no. 8, p. 2047, 1980.
22 Chen, H. and Boutros, P. C., "Venndiagram: a package for the generation of highly-customizable venn and euler diagrams in r," *BMC bioinformatics*, vol. 12, no. 1, p. 35, 2011.
23 Srinivas, M. and Patnaik, L. M., "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 4, pp. 656–667, 1994.
24 Markus, M. L. and Robey, D., "Information technology and organizational change: causal structure in theory and research," *Management science*, vol. 34, no. 5, pp. 583–598, 1988.
25 Stillinger, F. H., Head-Gordon, T., and Hirshfeld, C. L., "Toy model for protein folding," *Physical review E*, vol. 48, no. 2, p. 1469, 1993.

Solving Exponentially grow Cells Automata by using Genetic Algorithm

Cheng En Ho
* The University of New Mexico, Albuquerque, NM 87131, USA
gretor0231@unm.edu

***Abstract*—Exponentially Increasing Functions are hard to solve in the real world. In this paper, we will use cells automata to demonstrate multiple dimensions grids, and calculate its next generation offspring. We will show the Cells Automata (CA) formula and try to find its converge pattern. We will also measure the big O complexity of the problems. We can easily understand CA theory, and realize the brute force method takes too much time to solve the problems. Here, we try to improve upon the brute force method by applying a Genetic Algorithm (GA) in a fraction the time of a brute force search. Eventually we use a Genetic Algorithm to find an optimized solution. Although the solution is not the best solution, it is an optimized solution that we can find in the limit time. We need to know Cells Automata, Genetic Algorithm, Neural Network and Basic big O notation to solve exponentially grow problems. We expect this method can solve complicated problems to improve human future life.**

***Index Terms*—Cells Automate, Genetic Algorithm, Neural Network, Big O Notation, Large Number Problems.**

I. Introduction

To begin, we introduce three main concepts of this paper. These concepts are Cells Automata, Genetic Algorithm, and Neural Network. We also need some background of Computation, Machine learning, and Algorithm.

A. Cells Automata

In this paper, we are working on one dimensional binary state cellular automate (CA)[1]. We can see an example rule from the Figure 1. Each cell states has 0s or 1s. In a one dimensional CA, we can see CA have three parameters to effect its total numbers of next generations. The cell string length 'l', the dimension of CA 'd', and radius of neighbors 'r'. The total numbers of states show in formula below.

$$Total = l \times 2^{2^{(2r+1)^d}} \tag{1}$$

We see the equation 1 above and basic type of CA from Figure 1. Each current state is 0 or 1, and it also has two states for next generation. String length 'l' spends polynomial time. However, neighbors 'r' and dimensions 'd' both cost exponential time. We can imagine when we increase the dimensions, it will also increase neighbors. Therefore, our evolution takes exponential time in neighbors and dimensions. A brute force method will spend too much running time. In this paper, we will use another empirical method to solve the problems.

B. Neural Network

We need background of support vector machine (SVM)[2] before knowing Deep Learning Neural Network[3]. We should know making a decision boundary to maximize the accuracy,

η	000	001	010	011	100	101	110	111
s	0	0	0	1	0	1	1	1

Fig. 1. r = 1, d = 1, Cells Automata

and know regression function[4]. Here, we start at two dimensions layer with two neurons. Then we add more logic functions to increase neurons of Neural Network. Finally, we will have a multiple layer Neural Network of Supervised Deep Machine Learning.

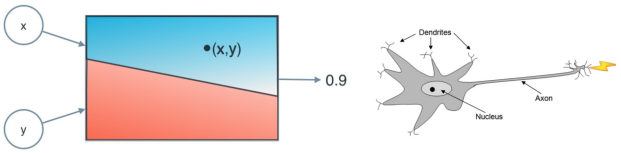


Fig. 2. two dimensions input one dimension output Neural Network

Figure 2 show two dimensions input means which multiplied their weights separately. Its output depend on sum of the values and the threshold function. In this case, it is a stair step function and only has a output result true or false, just like a high frequency pass filter component. We can see equation 2 to present this Neuron's threshold function[5] - Boolean Algebra.

$$f(x1, x2, ...xn) = 1, w1x1 + w2x2 + ...wnxn \geqslant t \tag{2}$$

Once we understand a single Neuron, we can add more and more Neurons and connect their output to Neuron's input from the next layer. We can imagine it should build a multiple dimension Neural Network. We define the first layer as an input layer, the last layer as an output layer, and rest of them as hidden layers. We can observe a single Neuron that we mention before only can classify 'AND', 'OR' and 'NOT' logic type data.

If we encounter 'XOR' logic type data, we cannot classify data by using a single line equation. Therefore, we have to add another line which is another Neuron at hidden layer. We can also add a Neuron at the first layer to increase one dimension. This will become a plane classifing a 3d grid. In general, the complexity[6] of a Neural Network system depend on our data sets, and it has a minimum neurons requirement of the Neural Network. If we do not have enough neurons then it cannot classify our data sets. We can also move the threshold function

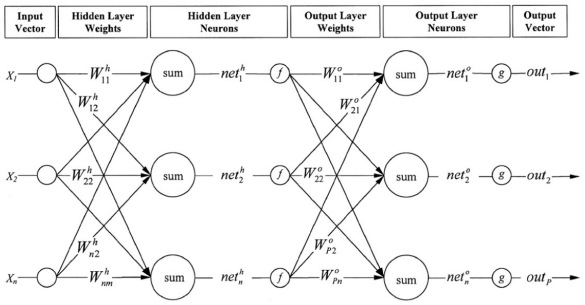


Fig. 3. Neural Network with sum of weight and threshold functions

to the equation's left side. Thus, we only have to consider weights and adjust them.

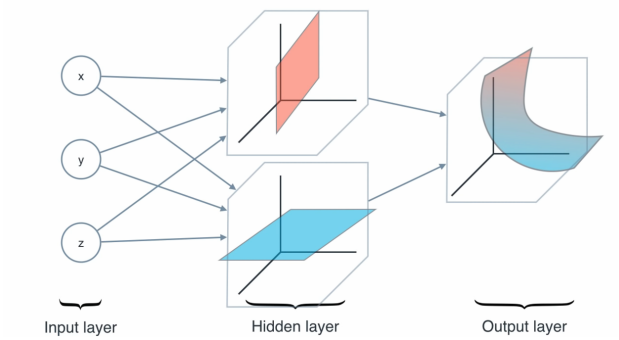


Fig. 4. three dimensions inputs Neural Network

Another issue is that the stair step function is not differentiable at corner side and we are taking gradient descent. Hence, we have to using sigmoid function instead of stair step function. We can easily understand the sigmoid function from solving a logistic differential equation[7] by ODE.

$$P'(t) = rP(t)[1 - P(t)/K]$$

Finally, our accuracy curve[8] might up and down several times like mountains. It depend on our Neural Network. There is a trade off between large steps and small steps. We can see a Deep Learning Neural Network example in Figure 4.

C. Genetic Algorithm

We can see the block diagram from Figure 5. It is There are three process of them, which are selection, crossover, and mutation. In this article, GA[9] will be utilized effectively to determine the CA.

To solve the CA problem, we need to do the following:

1.Create random populations of CAs.

2.Calculate the fitness function.

3.Select the few highest fitness.

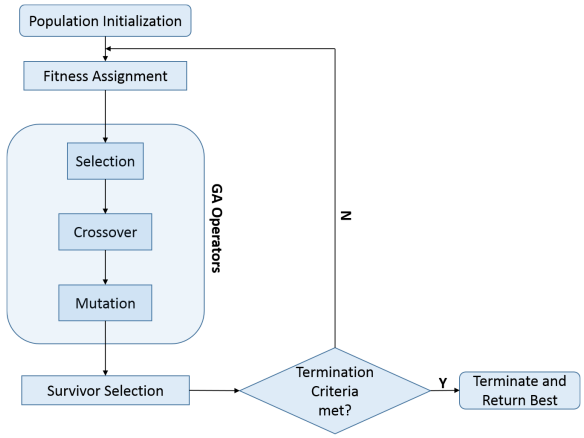


Fig. 5. Genetic Algorithm

4.Crossover those high fitness score strings

6.Randomly Mutate character in the string.

7.repeat GA process new population until meet the condition.

II. Methods and Results

In this article, our goal is solving general problems and creating a general model. We will follow Mitchell's 1994 paper process and replicate it. Our goal is to choose the best GA rule. Therefore, that GA should converge most of string CA in a short time. We follow its k = 2 and r = 3 parameter. Our target string is 0s or 1s and the goal is to produce a target string starting from a random initial publication string of the same length.

A. Initial populations

Figure 5 can be simplified to three steps. First, We have to create initial populations. Second, we train them and calculate the fitness function[10], then we choose high fitness score populations to run GA process. Third, when we reach the condition, we will stop the loop and acquire the result. Here, we roughly choose random initial populations with random rules, which can converge CA to 0s or 1s in 1000 steps. Here, we set string length equal to 99 chars to avoid an even number, which might happen the number of 1s equal 0s. We expect this initial populations can significantly reduce the whole running time, because we calculated strings to avoid the divergent string CA for any rules. For creating initial populations, the run time is $O(Steps \times StringLength \times Units) + O(CheckingLastStep)$.

Our goal is to find the best GA which can convergence by most of publications. Once we have the initial population strings, we can start running. Eventually, we want to know if the strings CA can converge or not. Thus, we start building a Fitness Function. Here we choose r = 3, d = 1, l = 99, CA initial population built by our C++ code.

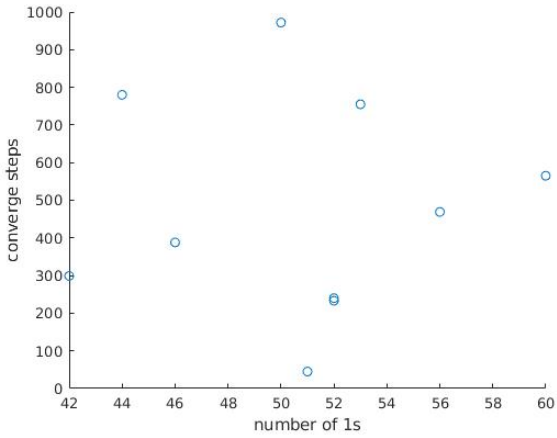


Fig. 6. Initial population's converge rate

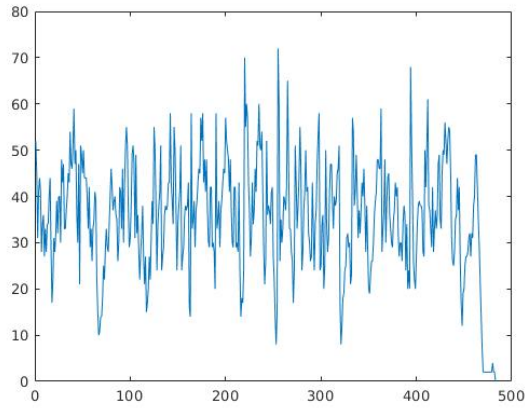


Fig. 7. Initial population's converge rate

Figure 6 shows Initial population's convergence rate. We assume the random distribution should be very hard to converge. However, our experiment shows hypothesis is wrong. This can see the result might depend on the rule. We observed even random distributions might converge very fast. We do not know the relationship between number of 1s and convergence rate.

B. Fitness Function

Our Fitness Function depends on the initial population and speed of convergence rate, which means it can finish the pattern or not. We will find the last step which is totally 1s or 0s, and if we use normal distribution, 1s numbers should almost equal 0s numbers, when we use the random distribution function with a long string. Here, we ignore the special case, which switchs 0s and 1s back and forth. We only looking for the first converge step. We define the fitness score equal its converge step, which means lowest score population will be selected. Therefore, we can see the figure 7 is different than Mitchell's paper. However, the main concept is the same. We see Figure 7 shows the Fitness Functions. Our code shows low fairness should be crossover. We encountered something that wrong in our fitness function or C++ code. Our pattern is significantly different than Mitchell's 1994 paper. We acquired a lot of experience and will improve it in next project. For now, we know the fitness function is important to define.

C. Neural Network

We have not finished this part and built a neutral network. However, we can imagine Wagner's hypothesis. We can very easily imagine if the network become more robustness, than it is hard to mutation. Our mutation should like fitness function score. High score is easy to survive, and it should has less mutations.

III. Conclusion

First, We use the Genetic Algorithm to crossover rules. It is powerful, but the process is totally random, so we do not have any weight and assign a random number. We use the environment condition to filter the result. We could not know the time for running the process.

Second, Evolution might go to a wrong path. Keeping species diversity[11] is very important. Therefore, we can not just crossover and mutate one population. Basically, we have to crossover many populations, to avoid wrong path. We should compare many different fitness function,s and their convergence rate.

Finally, solving an Exponentially Increasing Function is hard. We will do a Genetic Algorithm coupled with a Neural Network to acquire their advantage from both sides. Moreover, we will do some real world problems.

IV. Contributions Statement

-All of the paper - Cheng En Ho .

V. Code turn in

-All of code in the attach file.

References

1 Mitchell, M., Crutchfield, J. P., and Hraber, P. T., "Evolving cellular automata to perform computations: Mechanisms and impediments," *Physica D: Nonlinear Phenomena*, vol. 75, no. 1-3, pp. 361–391, 1994.

2 Chang, C.-C. and Lin, C.-J., "Libsvm: A library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

3 LeCun, Y., Bengio, Y., and Hinton, G., "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

4 Montgomery, D. C., Peck, E. A., and Vining, G. G., *Introduction to linear regression analysis*. John Wiley & Sons, 2012, vol. 821.

5 Festen, J. M. and Plomp, R., "Effects of fluctuating noise and interfering speech on the speech-reception threshold for impaired and normal hearing," *The Journal of the Acoustical Society of America*, vol. 88, no. 4, pp. 1725–1736, 1990.

6 Papadimitriou, C. H., *Computational complexity*. John Wiley and Sons Ltd., 2003.

7 Gopalsamy, K. and Zhang, B., "On delay differential equations with impulses," *Journal of Mathematical Analysis and Applications*, vol. 139, no. 1, pp. 110–122, 1989.

8 Ghosh, S. and Reilly, D. L., "Credit card fraud detection with a neural-network," in *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, vol. 3. IEEE, 1994, pp. 621–630.

9 Whitley, D., "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

10 Gen, M. and Lin, L., "Genetic algorithms," *Wiley Encyclopedia of Computer Science and Engineering*, pp. 1–15, 2007.
11 Chesson, P., "Mechanisms of maintenance of species diversity," *Annual review of Ecology and Systematics*, vol. 31, no. 1, pp. 343–366, 2000.

Improving Neural Network's Hyper-Parameters by using Deep Evolve Algorithms

Cheng En Ho
* The University of New Mexico, Albuquerque, NM 87131, USA
gretor0231@unm.edu

***Abstract*—There are many methods for improving neural networks, one way is collect more data, another way is tuning its Hyper-Parameters. Tuning Neural Network's Hyper-Parameters is a exhausting work. If we have infinite time or infinite computing space resources, then we can use brute-force method to compare all Hyper-Parameters. In general, we eventually will acquire a optimised Neural Network. Nevertheless, We have to trade of all conditions (time, cost, performance, etc...) in the real world. If we want to train a optimal Neural Network in a short time, we can use Deep Evolve Algorithms in dynamic environments. Deep Evolve is a Genetic Algorithm combine with Neural Networks. We are discovering optimal genomes in evolving populations, which are deep Neural Network. It spend much computing space resources. However, timing is more important for large company. After we evolve several generations and select data set from natural environment. We can obtain optimised Hyper-Parameters. In case, Deep Evolve Algorithms are very useful for our training procedure.**

***Index Terms*—Genetic Algorithm, Neural Network, Hyper-Parameters, Deep Evolve Algorithms.**

I. Introduction

Here, We describe a Hyper-Parameter[1][2] tuning strategy. One way is to Babysitting a model, when we do not have too much computation resources, such as GPU and CPU. We training only one model, and adjust Hyper-Parameter in a period. People who use one model and focusing on training it, usually can acquire better performance and a unique learning curve. Another way is we training many models in parallel. So that we can try many different Hyper-Parameters setting, then we compare different models and pick the best.

We can see that Fig 1 shows two different tuning strategies. One way is call Panda strategy, mammals only have few children and parents spent a lot of time to breed them. Another way that call Caviar strategy, we crossover many children and expect one child can acquire good performance that can survive from a strict environment. Our Deep Evolve Algorithms can have advantages from both two strategies[3][4].

In this article, we introduce several main concepts of this paper. First, we will describe Hyper-Parameters. Second, We also introduce how to find Hyper-Parameters by using Deep Evolve Algorithms. Finally, we are discovering the Adamax of optimal optimizer[5].

A. Tuning Hyper-Parameters

Parameters which define the model architecture are referred to as hyper-parameters and thus this process of searching for the ideal model architecture is referred to as hyper-parameter tuning[6]. Hyper-parameters are not model parameters[7] and they cannot be directly trained from the data. It should be define before training. For example:

How many neurons should I have in my neural network layer? How many layers should I have in my neural network? What should I set my learning rate to for gradient descent?

If we do not set up the optimal model architecture which is the wrong hyper-parameters, the training time might increase from few days to few months. The hyper-parameters define how our model is actually structured.

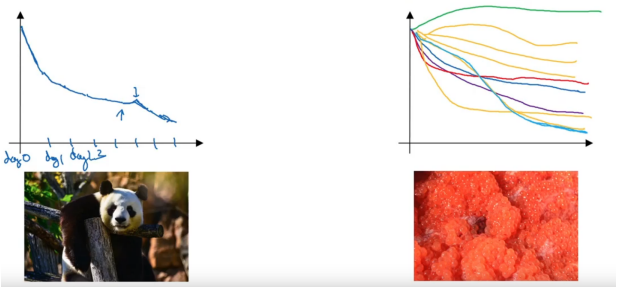


Fig. 1. Different Hyper-Parameter tuning strategies

B. Genetic Algorithm coupled with Neural Network

We are now using Neural Network (NN) and Genetic Algorithms (GA) to help NN to learn better and more efficiently. GA is gradient free Algorithms, and NN needs to training hyper-parameters.

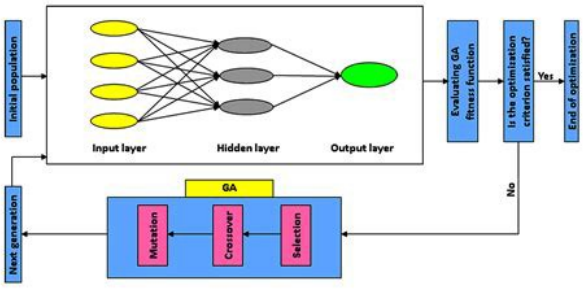


Fig. 2. Genetic Algorithm coupled with Neural Network

We can see the block diagram from Figure 2. There are three processes, which are selection, crossover, and mutation. In this article, GA will be utilized effectively to determine the Neural Network weights and hyper-parameter. To tune the hyper-parameter problem, we need to do the following:

1.Create a population of several Neural Networks.

2.Assign random hyper-parameter to all the NNs.

3.Train all the NNs simultaneously.

4.After training, ealculate their ”fitness” for GA.

5.Find the maximum fitness and Crossover and Mutate them.

6.repeat training new population until meet the condition.

C. The Evolution of Gradient Descent

The idea of gradient descent is find the minimum value and convergence it, so that we can acquire the maximum our accuracy. However, traditional way need to calculate whole data set, it is significant slow and waste memory. Therefore, we can use Stochastic Gradient Descent[8][9][10].

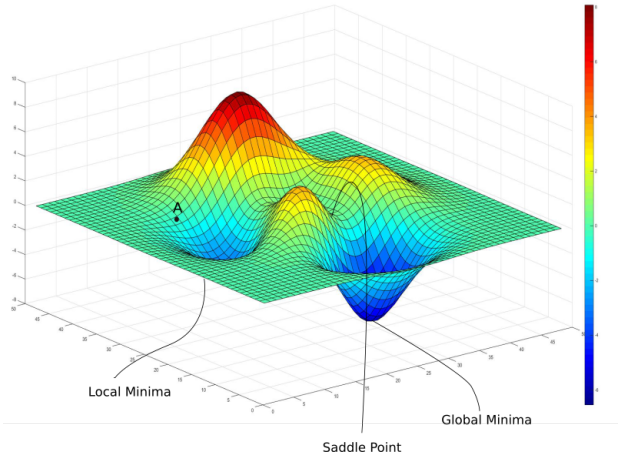


Fig. 3. 3D dimensions of Gradient Descent

We can see the figure 3 above, our target is to look for global minimal, not local. There are several ways could increase convergence rate and reduced oscillations. In general, Adaptive Moment Estimation(Adam) optimizer can quickly find the right direction and converge very fast. Adam is usually overall best optimizer in the real world data set[11][12][13].

II. Methods and Results

Here, we say each training cost time ’t’ to on our network data set. If we have ’x’ parameters with ’y’ possible settings each, then our total training time as below:

$$TotalTime = t \times y^x \quad (1)$$

We can see brute-force cost exponentially increasing time for training. Here, our goal is find an other method that cost polynomial time. We use a genetic algorithm to evolve X generations with a population of Y, each training cost T. If we keep top species and plus few more Z. Every generation only require Y-Z, the total running time as below:

$$TotalTime = T \times Y + (X-1) \times T \times (Y-Z) \quad (2)$$

We can see in equation 2, only the first round cost the whole population Y, the rest of round cost Y-Z, and the total running become polynomial time. However, we might never find the best solution since genetic algorithm do not looking for whole set. This is the trade off of Deep Evolve Algorithms.

Our network architecture is represented as a string of genes. We import the python3 library TensorFlow and data set Cifar10. We can see many tiny pictures from Cifar10. Those architectures/genomes recombine with some frequency, at one randomly selected position along the genome. To increase the rate of discovering optimal Hyper-parameters, we also keep track of all genomes in all previous generations. We can see the figure 4 present the crossover behavior. Finally, we also mutation at random choices to ones that differ from the gene’s current value.

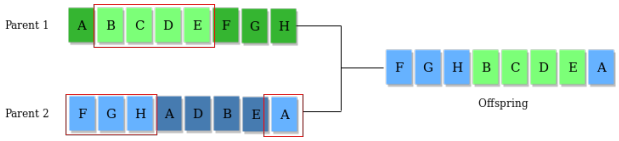


Fig. 4. Crossover at randomly selected position along the genome

We will tune four parameters:
1.Number of layers (or the network depth)
2.Neurons per layer (or the network width)
3.Network optimizer

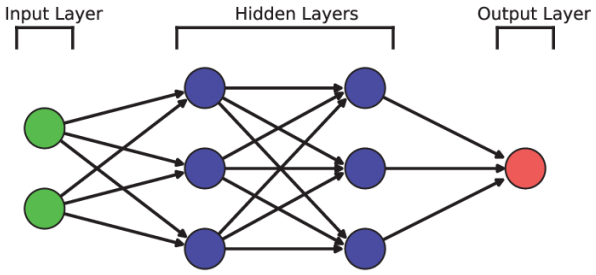


Fig. 5. Neural Network’s diagram

Figure 6 shows our experiment result, we can see 4 epochs from the first curve. We can see its converge at 55 percent accuracy after 4 epochs. Each time cost 9-12 hours, so we only run it twice and did not have enough time to run brute-force method, it might cost more than one week for each run.

III. Conclusion

The Deep Evolve method is very powerful. When we have enough computation resource, we can run it on different computers at the same time. We can expect some genomes will fail to learn, but most of them can reach the accuracy of

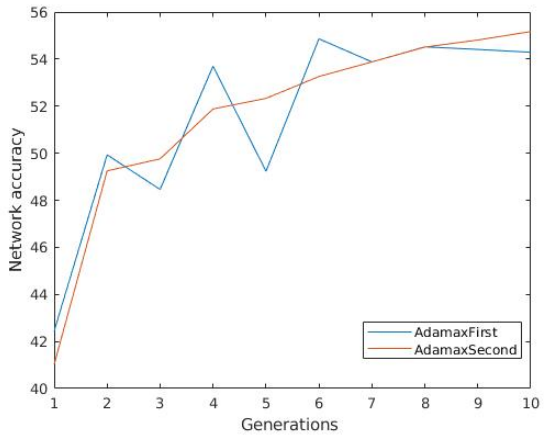


Fig. 6. Genome's percentage accuracy at each generation

brute-force method. We use more space to save time. Although the Genetic Algorithm might never find the best solution, if we have enough samples, we still can pick the top accuracy and it should be very similar the brute-force result.

For further work, we still can tuning the Genetic Algorithm, such as populations and generations. How much resource should I use, etc. We hope this method can speed up all Hyper-Parameters tuning process.

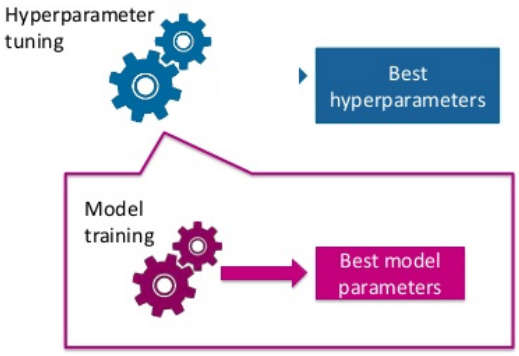


Fig. 7. Hyper-parameter tuning block diagram

IV. Contributions Statement

-All of the paper - Cheng En Ho .

V. Code turn in

-All of the code in the attach file.

References

1 Bergstra, J. and Bengio, Y., "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

2 Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B., "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

3 Nguyen, A., Yosinski, J., and Clune, J., "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 427–436.

4 Arel, I., Rose, D. C., Karnowski, T. P. *et al.*, "Deep machine learning-a new frontier in artificial intelligence research," *IEEE computational intelligence magazine*, vol. 5, no. 4, pp. 13–18, 2010.

5 Kingma, D. P. and Ba, J., "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

6 Young, S. R., Rose, D. C., Karnowski, T. P., Lim, S.-H., and Patton, R. M., "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*. ACM, 2015, p. 4.

7 Cawley, G. C. and Talbot, N. L., "Preventing over-fitting during model selection via bayesian regularisation of the hyper-parameters," *Journal of Machine Learning Research*, vol. 8, no. Apr, pp. 841–861, 2007.

8 Bottou, L., "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

9 Johnson, R. and Zhang, T., "Accelerating stochastic gradient descent using predictive variance reduction," in *Advances in neural information processing systems*, 2013, pp. 315–323.

10 Recht, B., Re, C., Wright, S., and Niu, F., "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in neural information processing systems*, 2011, pp. 693–701.

11 Patacchiola, M. and Cangelosi, A., "Head pose estimation in the wild using convolutional neural networks and adaptive gradient methods," *Pattern Recognition*, vol. 71, pp. 132–143, 2017.

12 Ruder, S., "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

13 Ma, D.-J., Makowski, A. M., and Shwartz, A., "Estimation and optimal control for constrained markov chains," in *1986 25th IEEE Conference on Decision and Control*. IEEE, 1986, pp. 994–999.