ISSN 2395-1621

SELF LEARNING NEURAL NETWORK

#1NOEL NADVI, #2ZULKARNAYAN MALIK, #3PROF. KAMBLE J.R.

1noelnadvi@gmail.com
2zulkarmalik@gmail.com
3kamble.jayshree3@gmail.com

#123DEPARTMENT OF COMPUTER ENGINEERING

Al-Ameen College of Engineering, Koregoan Bhima, Pune.

ABSTRACT

The goal of this project is to make a program that can teach itself to do a specific task on its own which can also be called as "Self Learning AI". Our attempt here would be to understand the basic AI that can be implemented in a program and understanding the benefits and the drawbacks of the same. We would design a program that would be able to teach itself to play a 8 bit game on its own without human intervention and without coding the path to play that game. Here the program will not have any prior knowledge of how to play the game or what should be done to play it correctly. This project will only know that it has to reach the end of the game. The program will have to teach itself on what decisions to take in the game in order to complete a stage. Every decision that the program will take will be marked as a neuron and it will be tested constantly to find out if that decision will help it survive. Decisions will be changed from time to time. Only the decisions that are Optimum will be kept and other decisions will be discarded. The process will be similar to how a species will evolve with passing years in the wild, where the habits that help the species to survive longer are identified and passed on to the future generations through genes. The commercial aspect of this topic is yet to be explored as this area is not very common and simple to implement. The code will be independent and would receive input from a physical or virtual machine that will be used to run the copy of the game. All the inputs will be provided by the algorithm and no human intervention will be done. Every time a set of decision is finalized by the algorithm we will call that group of decision as "Generation". To depict how a species evolve over 100's of years.

ARTICLE INFO

Article History

Received: 31st May 2018

Received in revised form : 31st May 2018

Accepted: 3rd June 2018

Published online : 4th June 2018

I. INTRODUCTION

Neural networks, more accurately called Artificial Neural Networks (ANNs), are computational models that consist of a number of simple processing units that communicate by sending signals to one another over a large number of weighted connections. They were originally developed from the inspiration of human brains. In human brains, a biological neuron collects signals from other neurons through a host of fine structures called dendrites. The neuron sends out spikes of electrical activity through a long, thin stand known as an axon, which splits into thousands of branches. At the end of each branch, a structure called a synapse converts the activity from the axon into electrical effects that inhibit or excite activity in the connected neurons.

When a neuron receives excitatory input that is sufficiently large compared with its inhibitory input, it sends a spike of electrical activity down its axon. Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes. Like human brains, neural networks also consist of processing units (artificial neurons) and connections (weights) between them. The processing units transport incoming information on their outgoing connections to other units. The "electrical" information is simulated with specific values stored in those weights that make these networks have the capacity to learn, memorize, and create relationships amongst data. A very important feature of these networks is their adaptive nature where "learning by example" replaces "programming" in solving problems. This feature makes such computational models very appealing in application domains where one has little or incomplete understanding of the problem to be solved but where training data is readily available. These networks are ―neural‖ in the sense that they may have been inspired by neuroscience but not necessarily because they are faithful models of biological neural or cognitive phenomena.

ANNs have powerful pattern classification and pattern recognition capabilities through learning and generalize from experience. ANNs are non-linear data driven self adaptive approach as opposed to the traditional model based methods. They are powerful tools for modeling, especially when the underlying data relationship is unknown. ANNs can identify and learn correlated patterns between input data sets and corresponding target values. After training, ANNs can be used to predict the outcome of new independent input data. ANNs imitate the learning process of the human brain and can process problems involving non-linear and complex data even if the data are imprecise and noisy. These techniques are being successfully applied across an extraordinary range of problem domains, in areas as diverse as finance, medicine, engineering, geology, physics, biology and agriculture. There are many different types of neural networks. Some of the most traditional applications include classification, noise reduction and prediction.

We are designing a program that can teach itself to achieve a goal determined by user without any prior knowledge of it. This program would have the ability to evolve from one state to the other without any human intervention. Similar to the real evolution cycle. Every decision that is taken by the program will be registered and evaluated on how long the decision helps it survive. Similar to this there would be a set of decision registered too where it will evaluate on how closer can it get to the assigned goal using the set. Decisions that are best for the program will be Saved or ―Breed and passed on to the next iteration ―Generation. This process will continue until the goal is met or reached.

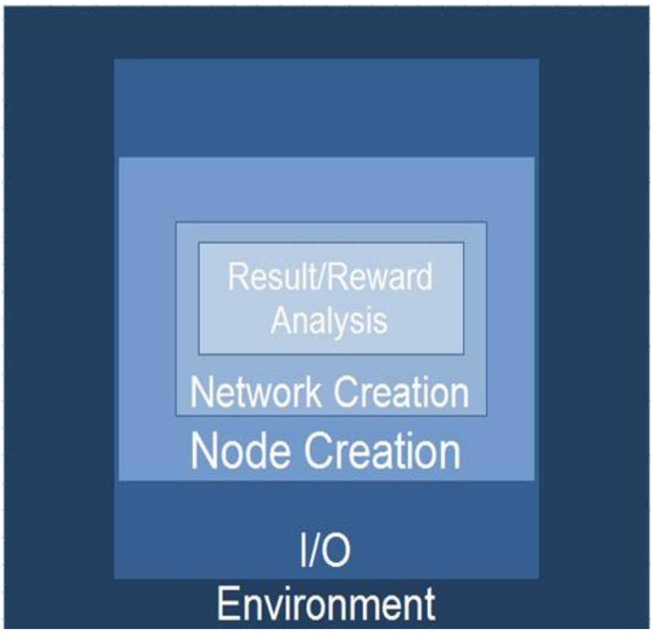
II. LITERATURE SURVEY

[1] We studied neuroevolution and how to gain an advantage from evolving neural network topologies along with weights. We present a method, NeuroEvolution of Augmenting Topologies (NEAT), which outperforms the best fixed-topology method on a challenging benchmark reinforcement learning task. We claim that the increased efficiency is due to (1) employing a principled method of crossover of different topologies, (2) protecting structural innovation using speciation, and (3) incrementally growing from minimal structure.

[2] We learn about Genetic Algorithms, the mutation operator is used to maintain genetic diversity in the population throughout the evolutionary process. Various kinds of mutation may occur ove time, typically depending on a fixed probability value called mutation rate. In this work we make use of a novel data science approach in order to adaptively generate mutation rates for each locus to the Neuroevolution of Augmenting Topologies (NEAT) algorithm. The trail of high quality candidate solutions obtained during the search process is represented as a third order tensor; factorization of such a tensor reveals the latent relationship between solutions, determining the mutation probability which is likely to yield improvement at each locus. The single pole balancing problem is used as case study to analyze the effectiveness of the proposed approach. Results show that the tensor approach improves the performance of the standard NEAT algorithm for the case study.

[3] This paper introduces the real-time Neuroevolution of Augmenting Topologies (rtNEAT) method for evolving increasingly complex artificial neural networks in real time, as a game is being played. The rtNEAT method allows agents to change and improve during the game. In fact, rtNEAT makes possible an entirely new genre of video games in which the player trains a team of agents through a series of customized exercises. To demonstrate this concept, the Neuroevolving Robotic Operatives (NERO) game was built based on rtNEAT. In NERO, the player trains a team of virtual robots for combat against other players' teams. This paper describes results from this novel application of machine learning, and demonstrates that rtNEAT makes possible video games like NERO where agents evolve and adapt in real time. In the future, rtNEAT may allow new kinds of educational and training applications through interactive and adapting games.

III. SYSTEM ARCHITECTURE



Result/Reward Analysis

This is the center of the entire system where the Computation and comparisons of the rewards yielded by using activity are monitored. This block ensures that the activities yielding the most reward points and positive results are kept in the program and saved for improving.
If it receives a negative mutation input from the network creation block; then it starts computing if the recent action

has generated this negative result or the entire network is negative.
For positive mutation nodes it would calculate the reward and then the entire process will be repeated until end of the stage is reached.

Network Creation
Once this block receives control from the node creation block, a link will be created to the fitness of the network. This is done to monitor the effects of the current input on the survival of the character. If the survival fitness is good then that link is approved and the network of that node is created as a good input for the AI, and the control is passed on to the Reward analysis with a positive mutation node. If incase the result generated by the particular input is not good or preexisting the fitness calculation is done and the control is passed on to the Reward/Result analysis with a negative mutation.

Node Creation

Once this block receives inputs from the Input Generator, this block will create an effective node with the keystroke received; then it will check if the same input node has be generated previously to verify the survivability of the action. Once this is cleared it will then relinquish the control to the Network creation block.

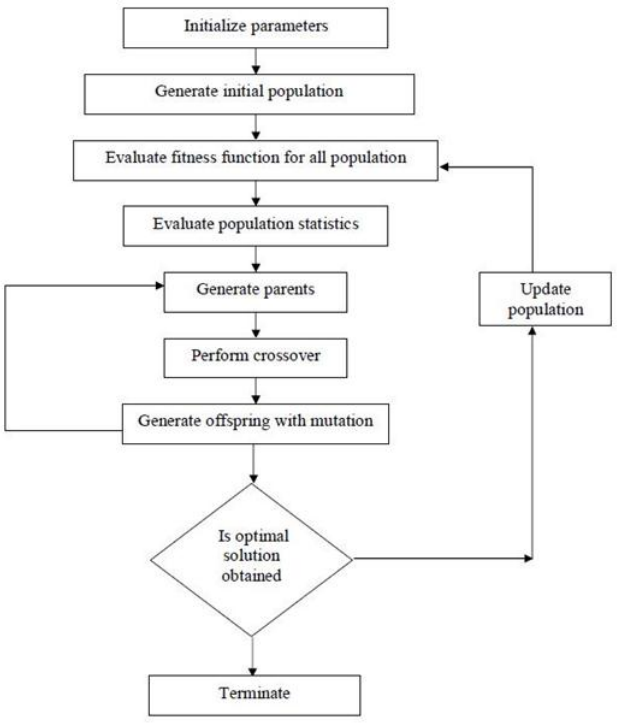
I/O(Input / Output)

This block generates keystroke when locations are received from the environment block.
All the inputs that are generated are monitored by this block, also all the possible combinations of the inputs are tried to test the conditions of the environment and the character.
Initially in the program this block plays a vital role as this block will ensure that the program starts working as soon as it is started.
Outputs from this block are then forwarded to Node Creation.

Environment

In this block environment refers to the positions and locations of the character on the screen.
These need to be monitored to ensure that progress is being made by the algorithm which can be then monitored by AI or human if necessary.
This block will basically deal with creating position of the game character with respect to objects that are stationary in the game, while doing so it will also monitor the locations and paths of the object that are harmful for the game character.
Output generated by this block are forwarded to the input generation block where the input as keystroke from the gamepad are taken.

IV. PROPOSED FLOW


Initialize parameters
Generate initial population
Evaluate fitness function for all population
Evaluate population statistics
Generate parents
Perform crossover
Generate offspring with mutation
Is optimal solution obtained
Update population
Terminate

V. RESULT

We were able to create a program which had the ability to implement evolution theorized in the paper and learn to play the game on its own, the game we used here is 8 bit Mario. The initial run for the program to gather data for making decisions is about 3 hours when it reaches a set number of generations. This is variable as each run is dynamically different that the older run.
The program only has knowledge of possible inputs that it can take i.e. Up, Down, Right, Left, A,B,X and Y. All these are standard inputs that can be given using a joystick and are common all over the world. The program starts with taking random inputs for the first few generations and later progresses to improving on the existing inputs or discarding the old once and using better input in each iteration. The working data is stored in RAM of the computer and used for comparison purposes of the program. The program will improve generation wise and reach the end of the stage without human help.
Once the program clears the stage on its own and moves to the next stage of the game and starts working on the next stage. Here, you can choose whether to move to next stage or not. On the stage-selection screen.
There is a stage where the performance of the program is saturated and it cannot be improved any further, if the program is run after the pool is saturated then improvements in the network are negligible in nature. This is because the game is designed in a way where the user needs a specific time to reach the stage end. The current record for fastest human player to reach the end of the stage is time 4:56.245 (mm.ss) held by username: somewes dated: 25 May 2018 on Nintendo platform.
Regardless of the saturation the breeding process continues, however the time required hence forth for encountering a game-changing connection is higher in linear fashion.

This program is just to portray that implementing AI on a smaller scale is possible and can be programmed to do simple tasks on its own. This can be further improved by implementing newer ideas in the respective fields.

REFERENCES

[1] Evolving Neural Networks through Augmenting Topologies Kenneth O. Stanley kstanley@cs.utexas.edu Department of Computer Sciences, The Uni- versity of Texas at Austin, Austin, TX 78712, USA Risto Miikkulainen risto@cs.utexas.edu Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712,
USA

[2] A Tensor-based Mutation Operator for Neuroevolution of Augmenting Topologies Aldo Marzullo, Department of Mathematics and Computer Science University of Calabria, Italy Claudio Stamileyz, Department of Mathematics and Computer Science University of Cal-abria, Italy Giorgio Terracina, Department of Mathematics and Computer Science University of Cal- abria, Italy

[3] Human-level AIs killer application: Interactive computer games, J. E. Laird and M. van Lent, in Proc. 17th Nat. Conf. Artif. Intell., and 12th Ann. Conf. Innov. Appl. Artif. Intell., 2000, pp. 11711178.

[4] Active guidance for a _nless rocket through neuroevolution F. J. Gomez and R. Miikkulainen, in Proc. Genetic Evol. Comput. Conf., 2003, pp. 20842095.

[5] Machines that Learn to Play Games M. Gardner, Sci. Amer., vol. 206, no. 3, pp. 138144, 1962. J. Frnkranz, Machine learning in games: A survey, , pp. 1159, 2001.

[6] Using a genetic algorithm to tune _rst-person shooter bots N. Cole, S. Louis, and C. Miles, in Proc. Congr. Evol. Comput., vol. 1, 2004, pp. 139145.

[7] Learning to play Pac-Man: An evolutionary,rule-based approach M. Gallagher and A. Ryan, in Proc. Congr. Evol. Comput., vol. 4, 2003, pp. 24622469.

[8] Generating war game strategies using a genetic algorithm T. Revello and R. Mc-Cartney, in Proc. Congr. Evol. Comput., vol. 2, 2002, pp. 10861091.