Evolved binary neural networks through harnessing FPGA capabilities

Raul Valencia
School of Computer Science
University of Auckland, NZ

Dr. Chiu Wing Sham
School of Computer Science
University of Auckland, NZ

Dr. Oliver Sinnen
Department of Electrical,
Computer and Software Engineering
University of Auckland, NZ

***Abstract*—The exponential progress of semiconductor technologies has enabled the proliferation of deep learning as a prominent area of research, where neural networks have demonstrated its effectiveness to solve very hard multi dimensional problems. This paper focuses on one in particular, Binary Neural Networks (BNN), which use fixed length bits in its connections and logic functions to perform excitation operations. Exploiting those characteristics, hardware accelerators that integrate field-programmable gate arrays (FPGAs) have been adopted to hasten inference of deep learning networks, given its proficiency to maximize parallelism and energy efficiency. This work will show how the algorithm Binary Spectrum-diverse Unified Neuroevolution Architecture (BiSUNA) can perform training and inference on FPGA without the need of gradient descent. Source code can be found in github.com/rval735/bisunaocl**

***Index Terms*—Neuroevolution, binary neural networks, BiSUNA, FPGA.**

I. Introduction

The most well known approach taken by researchers to train deep neural networks (DNN) to date is the utilization of gradient descent[**?**], where the original intent of the algorithm was designed to deal with continuous numbers via derivatives and floating point values, not discrete spaces. Despite the fact that the possibility of precision reduction (Float 32 $\succ$ Float 16 $\succ$ Int16) can train DNN with no significant loss in accuracy [**?**].

When precision reduction reach the minimum representable set of values (0,1), then DNN that use those elements are classified as Binary Neural Networks (BNN); for this special type, research has mostly been focused on techniques to discretize gradient descent in order to use binary numbers. However, the original problem persist, given that this algorithm was originally designed for continuous contexts, of which, when it is utilized with a different space set, it creates instabilities when training is performed, specially if a high learning rate is adopted [**?**].

In order to overcome those instabilities, it is imperative to take a different approach, with a completely new perspective on how BNNs should be trained; the likes of which was proposed by the authors of this papers and published as code in [**?**]; there was exposed a novel methodology to train neural networks that use fixed-length binary set in weights, activations, operations and is completely gradient free. As a natural next step in the research of this novel method, the authors have taken the original CPU implementation to then be accelerated by a FPGA using OpenCL, time at which the following details were discovered:

- A pipeline design deployed to a Cyclone V FPGA
- Kernel reaches only 47% FPGA utilization (DSP free)
- FPGA's power consumption is 3.2W vs CPU's $\sim$85W

Considering that background, this paper is organized as follows: a brief introduction to alternative algorithms based on neuroevolution is summarized, to then show BiSUNA's learning generalization characteristics. Followed by the main contribution of this paper, explain how BiSUNA is accelerated by taking advantage of OpenCL single task kernels deployed onto an FPGA, which is tested on a well known reinforcement learning environment: "Mountain Car". Finally, a conclusion is outlined summarizing the benefits this fresh procedure offers.

II. Neuroevolution

Generally speaking, Evolutionary Algorithms (EAs) are optimization techniques based on population metaheuristics, where possible solutions to a problem are characterized as individuals in a group, for which subsequent iterated modifications of inhabitants commonly leads to multiple results. Individual's qualifications to solve an environment is graded by design, at which a system of commonly known evolutive functions is applied to each agent to then form a new set in subsequent trials of that assemblage.

When EA work together with DNNs, they form a current area of research that combines strengths on generalization of artificial NN with evolution's search capabilities, coined as Neuroevolution. One among the many advantages of bio-inspired algorithms is their ability to find global optima in complex multidimensional spaces, even more when they are used to train DNNs, as shown in the book [**?**].

One more algorithm that deserves further explanation is BiSUNA, which stands for Binary Spectrum-diverse Unified Neuroevolution Architecture [**?**], the latest addition to the neuroevolutionary family of algorithms that push towards DNN topology creation using mutation operators; it starts with a basic structure, then selecting only the fittest individuals to the problem at hand reaches a solution.

III. BiSUNA

In its original publication [**?**], SUNA demonstrated it was capable of solving reinforcement learning problems with the employment of floating point values as inputs/weights/outputs;

nonetheless the architecture could be easily repurposed to binary variables that can transform its domain from continuous to discrete space. Using an activation function described in **??**, neurons can calculate its trigger point using fixed binary set, packing them together and perform simple logic operations which are then propagated to the next neuron.

$$\upsilon = popcount\left(\bigvee_{i=1}^{n} w_i \wedge x_i\right) \tag{1}$$

$$y_t = y_{t-1} \oplus \alpha \wedge (\upsilon \odot y_{t-1}) \tag{2}$$

Explaining variables in **??** & **??**, υ forms a bitset with a fixed range, which is the result of all binary weights connected to that particular neuron. The $popcount$ function counts all "on" bits in a binary word, for example 100101 would return 3 (111). Because there are no floating point multiplications from typical DNN, BiSUNA replaces them with logic AND ($\wedge$); summation is reinstated with logic OR ($\vee$). y_{t-1} is as well another discrete variable with the same properties as υ, that is XNOR'ed ($\odot$) with α. This is a variable bitset that neurons have embedded with different firing rates and used as "inhibitor" of values in parts of such bitset. In other words, every neuron has discrete "membranes", the thicker it is, the more "on" bits it needs to propagate "excitation" to connected neurons. The last section applies logic XOR ($\oplus$) to the result of previous operations against y_{t-1}.

Deployment of these operators is not arbitrary, they were structured based on previous research around BNN, its functions and how they were designed to solve the problem at hand ([**?**], [**?**], [**?**]), all with the goal to successfully transform continuous functions that achieve a similar goal although operate in a discrete environment. Simple replacements like floating point summations with binary OR, multiplications with AND, whereas the presence of XOR helps transform a logic summation between previous values of the neuron state and new inputs presented. XNOR function serves as a moderator of new inputs that should match previous circumstances given its native properties as an equivalence gate.

Referring to Fig **??**, it shows graphically how a single binary neuron is connected to other elements, with also having different α values corresponding to its "thickness". Multiple other elements are displayed, x_n are other neurons in the arrangement, each with their own state; w_n are the connection's weight that link x_n to y_t; weight values are bitsets with numbers established after an agent is evolved. Not to forget, y_t neuron has two important properties: y_{t-1} and α, the first represents previous state stored and the second is the neuron firing rate, which has the purpose of creating neurons that work at different scales, here 1, 7 and 49, each increasing value means that the neuron will require more inputs to trigger its action forward.

Using functions described in **??** with bitset variables, allows BiSUNA's connections to completely look upon information in binary format, replacing all floating point operations, which translates as well to supersede the need of Digital Signal Processing (DSP) blocks with only Adaptive Look-Up Tables (ALUT). Another improvement is the reduction of connections length from traditional 32 bits of IEEE Float to unsigned 16 bitset; this subtle but important change signals the difference between running BiSUNA on an entry level FPGA (ex. Cyclone V) or not, as discussed in the experiments section **??**. Likewise, there is a paper that have shown how power consumption can be reduced 31x simply by switching ALU with logic functions [**?**].

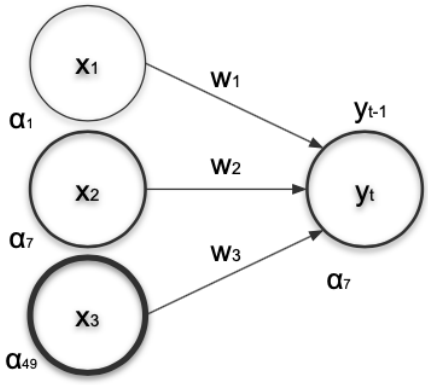

Fig. 1: Binary neuron representation with neuromodulation represented as the circle thickness.

Regarding the evolutionary procedures BiSUNA employs: spectrum diversity, reproduction and mutation, all of them follow the same pattern as established in the original publication [**?**]. Those functions only require comparators, memory assignation and counters, all of them are basic instruments of any processing unit.

Also worth mentioning that BiSUNA capitalizes of a special data structure named Novelty Map: a simple lookup table that sections individuals given a "novelty" metric, a number derived out the spectra of each individual; in other words, it adds up how many different neurons type it has. In order to keep everything within the realm of discrete values, BiSUNA calculates the hamming distance to discern between spectra amounts ($y = popcount(x_1 \oplus x_2)$), which of course keeps the utilization of only logic operations at the spot.

The work shown in this paper provides an alternative that, to the authors' knowledge, there are no BNN being initiated using neuroevolutionary as its main training algorithm, replacing the classical gradient descent used by most DNN architectures. Where other researchers have focused on making marginal improvements to traditional discrete gradient descent implementations, this work takes on a completely newfangled approach.

A. OpenCL kernel accelerated by FPGA

With the brief sketch of how BiSUNA works, this section will dive deeper on the reasons behind accelerating the algorithm using reconfigurable silicon, why that is important and a high level overview on how that is achieved. First, with the "all binary" philosophy, the authors were able to transfer

the original C/C++ code from [**?**], improve on aspects like dynamic memory management, code organization and speed, to later translate the heavy duty of processing each agent into a OpenCL kernel [**?**].

Generally speaking, OpenCL has two modes of execution: NDRange and Single Task. The first refers to the dexterity of offloading functions to multiple computing units at the same time. The second option performs the same operations, though focused on a single entity. Depending on the characteristics of the work to be performed, the target device, memory available and many other factors, software architects need to define which execution mode suits their needs. In this particular case, it was decided to wield a pipelined Single Task model, given the characteristics of the FPGA it was used for the experiments shown in section **??**.

Even though it is mentioned the kernel runs a single task, the FPGA offers the advantage of data pipeline when correctly programmed, which means that every clock cycle, the values of a new agent pass through a different process without stalling others progress. A graphic representation can be reviewed in Fig **??**, which avails one reason this type of architecture was selected. Another critical justification for the chosen model were the resources available on the target platform, Cyclone V - OpenVINO Starter Kit by Terasic, which only allow us to compile in a pipelined single task mode.

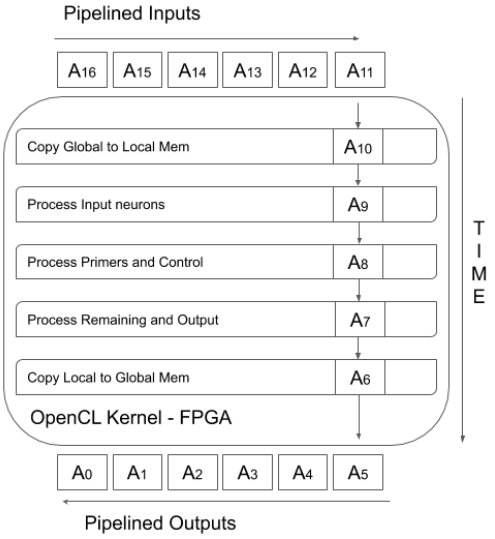

Fig. 2: Example of processing multiple agents in a pipeline architecture.

With optimized kernel code and correct initialization interval loops, it is possible to push at every clock cycle a new agent ($A_0 \succ A_n$) to the next stage. For example, Fig **??** shows how agents 0 through 5 have already finished their execution; 6 to 10 are being calculated in the OpenCL kernel, while 11 - 16 are queued to be calculated next. At each stage, some operations are performed: copy from global to local memory, process neurons; to then deliver the information to the output buffer.

While Fig **??** shows us a high level detail of how the OpenCL kernel works, Fig **??** displays the communication process it is used to prepare, enqueue and retrieve data that is processed in the FPGA. There are two main components, the CPU, which is in charge of preparing population data and control the flow of information with the reinforcement learning environment. On the other side, the FPGA is in charge of chaining agents in a population, to then process every neuron/connection in each iteration, with all data transfers exchanged via a PCI-E Gen1-4X.

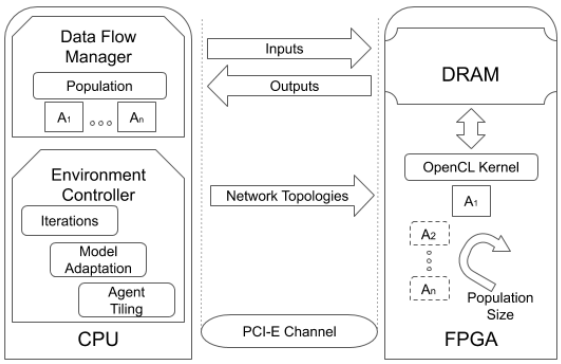

Fig. 3: OpenCL CPU-FPGA communication pattern

IV. Experiments

One drastic distinction in the utilization of BiSUNA networks are their aptness to solve reinforcement learning (RL) problems, all other BNN publications reviewed to date apply their models to solve supervised learning riddles. Given space limitations, here it is shown the OpenCL (Bi)SUNA implementation. The task solved is a standard RL experiment known as *Mountain Car* [**?**]. The objective is to move a "car" located at the bottom of a hill, then agents should learn to push it backwards/forward with enough potential energy to be able to displace it up to high ground, solving the problem when flag at the right section is reached

As standard procedure, same meta-parameters are utilized as reported in SUNA's paper ([**?**] table III). Because the mountain car RL problem operates with floating point numbers, BiSUNA connects a discretization function to transform values to a 16 bitset form. CPU tests ran on a Nectar virtual machine [**?**], similar to an AWS m2.small single core with 4GB ram and Ubuntu 18.04 LTS. At this moment, no GPU comparison was shown given the very poor performance a pipelined kernel has with this architecture (an average of 7ms on Process neurons), joined with multiple conditional branches needed to evaluate recurrent non-linear connections. Raw data related to environment execution is provided in repository [**?**], with more details about the runtime and device information.

OpenVINO Starter Kit by Terasic was the FPGA hardware employed, lended as part of the InnovateFPGA competition. It has a Cyclone V with 301K ALUT, 64MB SDRAM, 1GB DDR3 and communicates to a OpenCL runtime via PCI-E 4X Gen 1. Thanks to its OpenCL 1.0 runtime, it was possible

TABLE I: Results of running SUNA, BiSUNA and OpenCL-BiSUNA on multiple architectures.

Architecture	CPU	CPU	FPGA
Variable	Float	Binary	Binary
Score	-107	-122	-119
Process neurons	239	317.1	3572.5
Output read	0.896	0.834	109
Execution time	27s	35s	346s
Power	∼85W	∼85W	3.2W
Frequency	∼3GHz	∼3GHz	118.8MHz
Process / Power	2.8	3.7	1116.4

to transition from C/C++ to OpenCL's C domain language smoothly. Table **??** shows a results summary obtained by running the current implementation with two executable states, one with a compiled flag that facilitates floating point code when calculating operations and connections weight between neurons. The other had the flag disabled, assembling a program where all variables are bitsets. That explains the row *Variable* in table **??**.

The mountain car environment was repeated 10 times, each ran for 100 iterations with a population size of 100 agents. All executions solve the problem within the number of iterations (score greater than 130), here the best outcome obtained from all trials is typified. *Process neurons* is the time it takes to calculate all neuron interactions from input until a result is obtained; *Output read* is process of retrieving the result back from OpenCL, even when those calculations are performed in the CPU; totals are measured in μsec. *Execution time* is reported using seconds, telling us how long it took to finish a trial.

Regarding *Power*, CPU was an estimation consumption given the processor type and TDP reported by the manufacturer; in the FPGA case, it was possible to obtain an estimation using the *Power Analyzer* tool, using the *Total Thermal Power Dissipation* from that report. Same scenario happens with the *Frequency* field. About line *Process / Power*, it was tallied as the process time divided by the power consumption of each architecture.

One last consideration, with the resources available in the Cyclone V and how the kernel is programmed, it was not possible to generate a bitstream with the floating point flag enabled, provided that the estimated resource usage reported by the Altera Offline Compiler (*aoc*) exceeded the FPGA capabilities, which is summarized in table **??**. On the other hand, using BiSUNA, it acceded to compile and execute without problems, where maximum gains were reported as DSP usage, 0%: none were needed to calculate all operations in the BNN, as expected. Raw data with compilation reports can be found in [**?**].

V. Conclusion

This work provides a novel approach the DNN community could consider when training BNN using evolutionary principles, which is an alternative that avoids completely the use of gradient descent. Thanks to BiSUNA's mastery to conditionally switch between discrete/floating point at compile time, this work can be generalized further, setting precedence on the application of BNN to RL. It was shown as well that only BNN were able to compile to the Cyclone V in its current form whereas the continuous case did not. Future research will analyze more architectures and extend the RL environments to use the OpenAI gym to test how BiSUNA solves those problems.

TABLE II: AOC Resource utilization on Cyclone V.

Variable	Float	Binary	Improvement
Logic Utilization	174%	47%	3.7x
ALUTs	127%	31%	4.0x
Dedicated logic registers	61%	19%	3.2x
Memory Blocks	69%	33%	2.0x
DSP Blocks	58%	0%	58x
Compiles	No	Yes	Binary

As shown in Section **??**, simple changes exposed here form the foundation in research of discrete values to the application of RL problems, typical domain of NN algorithms like QLearning, which rely on floating components to reach a solution. Simplifying hardware requirements to train BNN contributes towards the creation of more efficient networks and circuits, facilitating entry level FPGA to not only work at time of inference, but also train at the edge, which will enable IoT devices to expand their capabilities, increasing competence of such technology to the next level, Intelligent Internet of Things (IIOT).

References

[1] S. Linnainmaa, "Taylor expansion of the accumulated rounding error," *BIT Numerical Mathematics*, vol. 16, no. 2, pp. 146–160, Jun 1976.

[2] M. Blott, T. B. Preusser, N. J. Fraser, G. Gambardella, K. O'brien, Y. Umuroglu, M. Leeser, and K. Vissers, "Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 16:1–16:23, Dec. 2018.

[3] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?" in *AAAI*, 2017.

[4] G. I. Sher, *Handbook of Neuroevolution Through Erlang*. New York, NY: Springer New York, 2013.

[5] R. Valencia, C. W. Sham, and O. Sinnen, "Using neuroevolved binary neural networks to solve reinforcement learning environments," p. 4, Nov 2019.

[6] J. M. Danilo Vargas, "Spectrum-diverse neuroevolution with unified neural models," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 8, pp. 1759–1773, Aug 2017.

[7] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "Xonn: Xnor-based oblivious deep neural network inference," in *USENIX Security*. USENIX 2019.

[8] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 525–542.

[9] S. Han, X. Liu, H. Mao, et al, "Eie: Efficient inference engine on compressed deep neural network," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. IEEE Press, 2016, pp. 243–254.

[10] R. Valencia. BNN-FPT source code. github.com/rval735/bisunaocl

[11] A. Moore, "Efficient memory-based learning for robot control," Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, March 1991.

[12] N. R. I. for Australia. (2017, Nov) New zealand nectar cloud. https://nectar.org.au/new-zealand-researchers-join-nectar-cloud/