Design of a neural network and particle swarm optimization in FPGA

Dang Tuan Linh\textsuperscript{1}, Hoshino Yukinobu\textsuperscript{1}

\textsuperscript{1}Kochi University of Technology
188001d@gs.kochi-tech.ac.jp, hoshino.yukinobu@kochi-tech.ac.jp

Abstract In this paper, a field programmable gate array (FPGA) was used to create a feed forward neural network trained by particle swarm optimization with a control of velocity (NN-PSOCV). The paper consist of two parts. The first part describes the development of a hardware implementation of NN-PSOCV coded using SystemVerilog programming language. Experimental results demonstrate not only that this NN-PSOCV can be implemented successfully but also that such a hardware implementation of NN-PSOCV achieves better performance than a hardware implementation of the neural network trained by original particle swarm optimization (NN-PSO). However, the hardware implementation of the full program requires a lot of resources such as the programmable logic elements, memory blocks and consequently could be not fitted inside a FPGA chip. To address this problem, the second part of this paper proposes a design for an architecture that combines hardware and software. In this architecture, several parts of the training process are performed by software (NIOS II processor) while all parts of the testing process are implemented in hardware only. Normally, the neural network is used in both training process and testing process while the particle swarm optimization (PSO) is only used in the training process to train the neural network. Therefore, in the proposed design, the neural network is implemented in hardware and the PSO is implemented in software. Details of each module in the proposed architecture are presented. This paper also shows results of the implementation when tested on a real FPGA device called DE1-SoC board.

1. Background

Today, an artificial neural network (NN) has become the interest of many researchers. The concept of NN is based on a human brain in two aspects. Firstly, a learning step is used to acquire the knowledge. Secondly, this knowledge is represented by the form of the weights of each node [1]. A typical NN is divided into three types of layers called input, hidden, and output layer, respectively. The output of a node in one layer becomes the input of nodes in the next layer through an activation function. Sigmoid function is one of the most common choices of the activation function (1).

\begin{equation}
A(x) = \frac{1}{1 + e^{-x}}
\end{equation}

Several researchers have used the particle swarm optimization (PSO) algorithm to train NN (NN-PSO). The concept of PSO is based on the social behaviors of animals, for example, bird flocking. At any given moment, a bird uses knowledge of population and knowledge of itself to find an efficient direction. The movement of each individual is estimated by a fitness function [2]. In this algorithm, each individual (particle) has position \( x^t_p \) which represents the potential solution of the optimization problem at time \( t \). Every particle also has its current velocity \( v^t_p \), the best personal position \( x_{Pbest}^t \), and \( Gbest^t \) are fitness values of these positions. The algorithm for calculate new velocities and positions is shown in (3), (4), (5), and (6):

\begin{align}
v^{t+1}_p &= w v^t_p + c_1 r_1 (x_{Pbest}^t - x^t_p) + c_2 r_2 (x_{Gbest}^t - x^t_p) \\
x^{t+1}_p &= x^t_p + v^{t+1}_p \\
P_{best}^{t+1} &= \begin{cases} f(x_p^{t+1}) & \text{if } f(x_p^{t+1}) < P_{best}^t \\ P_{best}^t & \text{if } f(x_p^{t+1}) \geq P_{best}^t \end{cases} \\
G_{best}^{t+1} &= \arg\min_{x_{Pbest}^{t+1}} P_{best}^{t+1}
\end{align}

where \( w \) is an inertia weight, \( f() \) is a fitness function, \( r_1, r_2 \) are two random numbers that are uniform distributed between 0 and 1. Two
variable \( c_1, c_2 \) are acceleration coefficients.

The basic idea of the NN-PSO is try to obtain a solution that minimizes learning error. The solution in this case is a set of weights of all nodes in NN, or in other words, the positions of particles in PSO.

In PSO algorithm, the control of velocity is a crucial problem. If the velocity is too low, the algorithm is too slow. On the other hand, if the velocity is too high, the algorithm is too unstable. Therefore, authors of PSO-CV [3] introduced the new mechanisms for control the velocity called PSOd-CV (7) and PSOe-CV (8).

\[
v_{p}^{t+1} = w v_p^t + c_1 r_1 (x_{Pbest_p}^t - x_p^t) \\
\quad + c_2 r_2 (x_{Gbest_p}^t - x_p^t) \\
\quad + \frac{c_3 r}{(v_p^t)^2}
\]

\[
v_{p}^{t+1} = w v_p^t + c_1 r_1 (x_{Pbest_p}^t - x_p^t) \\
\quad + c_2 r_2 (x_{Gbest_p}^t - x_p^t) \\
\quad + \frac{c_3 r}{e(v_p^t)^2}
\]

where \( c_3 \) is a new coefficient. The PSO-CV comprises three different phases (Fig. 2):

a) Swimming phase: a particle moves with a normal speed (\( v_p \gg 0 \)), the \( c_3 \) part is almost zero and the velocity control depends only on \( c_1 \) and \( c_2 \) without random factor. In this phase, the particle tends to move to the middle point of \( G_{best} \) and \( P_{best} \).

b) Stopping phase: when the particle is near the middle point, the velocity is slow down and begins to be affected by \( c_3 \) part.

c) Jumping phase: in PSOd-CV, if the velocity changes to a very small speed, the new calculated velocity jumps up quickly. Hence, the particle moves to other searching area and starts a new swimming phase. However, the big dimension problem may have many local solutions and PSO-CV algorithm needs to search these local areas. In this situation, PSOe-CV was introduced to keep the small jump and search local area (the jump size depends on \( c_3 \)).

Nowadays, the hardware implementation of NN and PSO has become an interesting target of research because the field programmable gate arrays (FPGA) may take advantage of the parallelism to obtain a higher operation speed [4] [5]. Authors also argued that FPGA devices are low cost, high performance, and flexibility [6]. The separate implementation of PSO [5] or NN [4] in FGPA has been seen in several publications. A hardware architecture for the training of NN with PSO was also presented [7] [8]. However, previous work have normally used only fixed point numbers, have conducted experiments in simulation with the ModelSim program, or have employed a standard PSO algorithm. From these issues, this paper tries to implement a NN trained by PSO with a velocity control (PSO-CV) and floating point numbers in a real FPGA device called DE-SoC. The PSO-CV is an improved version of PSO that was presented and software implemented [3].

2. Hardware implementation

In our hardware implementation, the PSO-CV algorithm is used to train three layers neural network (one input layer, one hidden layer, and one output layer). The number of neurons in
each layer are \( N_I, N_H, \) and \( N_O \), respectively. To increase the accuracy, bias numbers are also added. The size of an array that contains all weights of this NN is shown in (9).

\[
D = (N_I + 1) \times N_H + (N_H + 1) \times N_O
\]

(9)

\((N_I + 1) \times N_H\) indicates \( N_I \times N_H \) weights and \( N_H \) biases in the hidden layer. In a similar way, \((N_H + 1) \times N_O\) indicates \( N_H \times N_O \) weights and \( N_O \) biases in the output layer. In training step, PSO-CV is used to update these \( D \) weights of the NN in each iteration in order to find a best set of \( D \) weights. Therefore, \( D \) is also the dimension of each particle in PSO-CV.

This research uses SystemVerilog as a programming language and DE1-SoC development kit as a FPGA-targeted hardware. Fig. 2 show the implementation of our system. A global finite state machine (FSM) is used to control all modules. Two modules \( G_{best} \) and \( P_{best} \) are also processed in parallel to take advantage of FPGA. The main components of the implementation are:

a) Floating point calculation module (FPC): implements the floating point IP cores from Altera. Every floating point computation in other modules connects to this module.

b) Random number generator (RNG): uses linear feedback shift register.

c) Neural network (NN): the input of this module is a two-dimensional array of \( D \) weights \( [31:0]W[0:((N_I+1)\times N_H)+(N_H+1)\times N_O)-1] \) where \([31:0]\) demonstrates the floating point number (32 bits). The activation function of the NN is the sigmoid function. The FSM inside the NN has two states called \( idle \) state and \( running \) state, respectively. Normally, it is in the \( idle \) state. When a computational request comes, the FSM changes to the \( running \) state. After finishing the processing, the NN sends results via output port and the FSM returns to the \( idle \) state.

d) Training sample: processes the raw \( T \) training data.

e) Target training: contains \( T \) labeled training data (target) for learning stage of the NN.

f) Particle: each component in this module is one particle which contains \( D \) weights of the NN. The values of these weights are calculated from weight update module.

g) Output particle: outputs of the NN.

h) Fitness function: employs the mean squared error implemented in hardware by FSM.

i) \( P_{best}, G_{best} \): discovers the best fitness value in each particle and best global fitness value in a particular iteration.

j) Weight update: updates the new velocities and the new positions of the weights and the biases based on the PSO-CV algorithm.

k) Check stopping condition: checks whether program satisfies a condition.

For compare three algorithms, an experiment
was conducted. The initial values for \( G_{best}, P_{best} \) were 100.0. Number of iterations, particles, input, hidden, and output nodes were 1000, 6, 2, 2, and 2, respectively. Training pattern were four different patterns (00, 01, 10, 11). Results were captured by using SignalTap embedded logic analyzer. \( G_{best} \) in PSOd-CV was 0.66 (0x3F260000 in floating point IEEE 754 representation) (Fig. 3), in PSOe-CV was 0.67 (0x3F2B0000) (Fig. 4), and in original PSO was 0.76 (0x3F413CAE) (Fig. 5). Therefore, NN-PSOCV which used PSOd-CV and PSOe-CV was more efficient in hardware implementation than NN-PSO (measured by \( G_{best} \), the global minimum value of learning error). Comparing between two update equations of NN-PSOCV, PSOd-CV gave a better result (\( G_{best} = 0.66 \)) than PSOe-CV (\( G_{best} = 0.67 \)).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3F360000h</td>
<td>3F4CA600h</td>
<td>3F4C0000h</td>
<td>3F260000h</td>
</tr>
</tbody>
</table>

Fig. 3. \( G_{best} \) and \( P_{best} \) in PSOd-CV

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3F2B0000h</td>
<td>3F2BC163h</td>
<td>3F2B0000h</td>
<td>3F416348h</td>
</tr>
</tbody>
</table>

Fig. 4. \( G_{best} \) and \( P_{best} \) in PSOe-CV

3. Hardware/Software Co-design

The connection between hardware and software is based on the Avalon Memory Mapped (Avalon MM) slave interface. The NN module (coded by SystemVerilog) is wrapper inside a slave interface called new_component 0 (Fig. 7). This proposed component has two different ports. One is reserved for the data signals and the other one is used for the control signal. The control signal is the \textit{ready signal}. When the processor (NIOS II in this case) needs output data from the NN, it will check this signal. Our design does not need a \textit{start signal} because when the NN receives a \textit{write request}, it starts automatically. The data signals have two resources were excessive. For example, Fig. 6 illustrates the compilation of a small system with 1000 iterations, 30 particles, 2 input nodes, 4 hidden nodes, and 2 outputs nodes. The logic utilization in this situation was 114% and the compilation failed. Therefore, a bigger neural network cannot be fitted in the device. For solve this problem, the combination between hardware and software is presented.

In this architecture, training process in PSO are performed by software while all parts of the testing process in NN are implemented in hardware only. Therefore, the testing speed is maintained and a complex NN can be fitted in the FPGA device.

The connection between hardware and software is based on the Avalon Memory Mapped (Avalon MM) slave interface. The NN module (coded by SystemVerilog) is wrapper inside a slave interface called \textit{new_component 0} (Fig. 7). This proposed component has two different ports. One is reserved for the data signals and the other one is used for the control signal. The control signal is the \textit{ready signal}. When the processor (NIOS II in this case) needs output data from the NN, it will check this signal. Our design does not need a \textit{start signal} because when the NN receives a \textit{write request}, it starts automatically. The data signals have two

![Fig. 6. Compilation Report for the hardware design](image-url)
different data streams for data sent from the NIOS II processor to the NN module and vice versa.

The Avalon slave interface contains three different modules (Fig. 8):
1) The NN module: implements the Neural Network.
2) The Mapping module: translates the address sent from master into the slave’s address space.
3) The floating point calculation module: implements Altera floating-point IP cores.

In the experiment described in previous section of this paper, PSO\textsuperscript{d}-CV obtained better performance than other algorithms. Therefore, the PSO\textsuperscript{d}-CV algorithm was used in this experiment.

In this experiment, the hardware/software co-design was used. Therefore, it was enough resources for a bigger NN (2 input nodes, 6 hidden nodes, 4 output nodes). A big number of the particles (80 particles in this case) was also implemented. Thus, the final $G_{best}$ was decreased.

The training data for the inputs was a set of 11, 10, 01, and 00. Four outputs of NN had the values as can be seen in Table 1.

Table 1. The training data

<table>
<thead>
<tr>
<th>Input pattern</th>
<th>Output1</th>
<th>Output2</th>
<th>Output3</th>
<th>Output4</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

The initial values of $G_{best}$ and $P_{best}$ were 1000000.0. Fig. 9 shows the $G_{best}$ value decreased from the highest value (or initial value) to the minimum value 0.0.

Fig. 9. Reduction of $G_{best}$

Fig. 10 shows the outputs of NN with four different inputs in floating point format. For example, in the first line, if the inputs were 1.0 and 1.0, the output4 was 1.0 and other outputs
were 0.0.

Fig. 10. Output of NN

Fig. 9 and 10 show the accuracy of the NN and the training process of the NN with PSO-DCV.

4. Discussion

This paper has presented an implementation methods of NN-PSOCV in FPGA. The first design is the hardware only and the second design is the co-design between hardware and software. The first design obtains a good speed in both the training and testing processes. However, only a very simple neural network can be fitted inside a FPGA chip because this architecture requires a lot of resources. Thus, the second design is proposed. In this design, the PSO training is performed by software. With this architecture, a bigger neural network can be used. In addition, the speed of the testing process is maintained because the Neural Network is still coded in SystemVerilog.

The testing results on FPGA board suggest that the proposed architecture could be the solution for the hardware implementation of NN-PSOCV. However, only functional testing was conducted in the research. Thus, a useful direction for future research would be a timing testing. We intend to compare the speed of three different versions of NN-PSOCV: a hardware only implementation, a co-design of hardware and software, and a software only implementation. Moreover, for the purpose of testing, the conducted experiments were only simple classifications. Therefore, future research should investigate the efficiency of the NN-PSOCV algorithm with more complex training data and bigger neural networks. In addition, the problem of limited programmable logic elements is also significant. Hence, the next version of our architecture will reduce the load of FPGA chip by storing the node weights in SDRAM. Another possible avenue for future research is to investigate the use of the PSO-CV algorithm in practical applications.

5. Acknowledgement

This work was supported by JSPS KAKENHI Grant Number 25330240.

6. References