2023 IEEE International Conference on Smart Computing (SMARTCOMP) | 979-8-3503-2281-1/23/$31.00 ©2023 IEEE | DOI: 10.1109/SMARTCOMP58114.2023.00037

A Novel Weight Dropout Approach to Accelerate the Neural Network Controller Embedded Implementation on FPGA for a Solar Inverter

Jordan Sturtz
Department of Computer Science
North Carolina A&T State University
Greensboro, NC, USA
jasturtz@aggies.ncat.edu

Xingang Fu
Department of Electrical Engineering and Computer Science
Texas A&M University-Kingsville
Kingsville, USA
xingang.fu@tamuk.edu

Chanakya Dinesh Hingu
Department of Electrical Engineering and Computer Science
Texas A&M University-Kingsville
Kingsville, USA
chanakya.hingu@students.tamuk.edu

Letu Qingge
Department of Computer Science
North Carolina A&T State University
Greensboro, NC, USA
lqingge@ncat.edu

***Abstract*—This paper introduces a novel weight-dropout approach to train a neural network controller in real-time closed-loop control and to accelerate the embedded implementation for a solar inverter. The essence of the approach is to drop small-magnitude weights of neural network controllers during training with the goal of minimizing the required numbers of connections and guaranteeing the convergence of the neural network controllers. In order not to affect the convergence of neural network controllers, only non-diagonal elements of the neural network weight matrices were dropped. The dropout approach was incorporated into Levenberg-Marquardt and Forward Accumulation Through Time algorithms to train the neural network controller for trajectory tracking more efficiently. The Field Programmable Gate Array (FPGA) implementation on the Intel Cyclone V board shows significant improvement in terms of computation and resource requirements using the sparse weight matrices after dropout, which makes the neural network controller more suitable in an embedded environment.**

***Index Terms*—weight dropout, neural network controller, Levenberg-Marquardt, Forward Accumulation Through Time, Field Programmable Gate Array(FPGA)**

I. Introduction

As renewable energy sources become increasingly important in modern electrical systems, there has been increased attention on optimizing energy obtained from renewable energy sources such as solar and wind. A key component in generating energy from renewable energy sources is the three-phase grid-connected converter (GCC), which is responsible for converting direct current (DC) from solar panels into alternating current (AC) that can be connected to an electrical grid. GCCs are traditionally controlled using a conventional d-q vector control mechanism [1], which has been shown to have many limitations [2]–[4] due to a need for complex algorithms with higher-order computability.

Recent research has demonstrated the efficacy of recurrent neural networks (RNN) as a replacement for the standard vector control in optimizing grid-connected rectifiers [5], [6]. To maximize the effectiveness of these trained neural network (NN) controllers in embedded systems, e.g. Digital Signal Processors(DSPs), it is crucial to reduce overall computations and resource requirements of the trained model as the process of training NN Controller is time-consuming with limited processing power in embedded system.

Dropout is typically employed in neural networks as an efficient method to reduce overfitting in a wide range of applications, such as convolutional neural networks (CNN) and RNNs [7]–[9]. The idea behind the dropout technique is that for each training instance, neurons have a probability, p, of being dropped. In other words, neurons in the network are temporarily removed or deactivated by ignoring all the incoming and outgoing connections to the neuron. At test time, all the neurons in the neural network are reactivated and their outputs are multiplied by p to scale down the actual output to approximate the output during training. However, this kind of dropout policy will not benefit the implementation of the NN Controller in an embedded environment because those neurons are still present during the inference process, which will not reduce the computational requirements.

To accelerate the NN Controller in an embedded environment, e.g. FPGA, which is more suitable for computing the NN Controllers due to its parallel nature, this paper proposes a novel weight-dropout approach to refining the NN Controller and reducing the computation and resource requirements. The specific contributions of this paper include the following 1) an approach to refine the NN Controller through dropping non-diagonal and small-magnitude weights; 2) a weight-dropout approach incorporated into Levenberg-Marquardt (LM) and

2693-8340/23/$31.00 ©2023 IEEE
DOI 10.1109/SMARTCOMP58114.2023.00037

Forward Accumulation Through Time (FATT) algorithms to train the neural network controller for trajectory tracking; and 3) FPGA validation of the proposed dropout approach in an embedded environment.

The rest of the paper is organized as follows. Section II introduces the NN Controllers in the closed-loop control system for a solar inverter. The weight-dropout approach for training the NN Controllers is detailed in Section III. The dropout implementation in c++ is presented in Section IV. Section V provides detailed FPGA implementation to validate the new NN Controller after dropout. Finally, the paper concludes with a summary of the main points in Section VI.

II. NN Controller for a Solar Inverter

A. A Solar Microinverter

Typically, solar inverters consist of two components: the DC-DC converter and the DC-AC inverter, as illustrated in the case of the Texas Instruments (TI) Microinverter in Fig. 1 [10]. The PhotoVoltaic (PV) solar panels attach to the DC-DC converter, while the DC-AC inverter maintains the voltage of the DC Bus at its rated value while feeding controlled AC current to the main power grid.

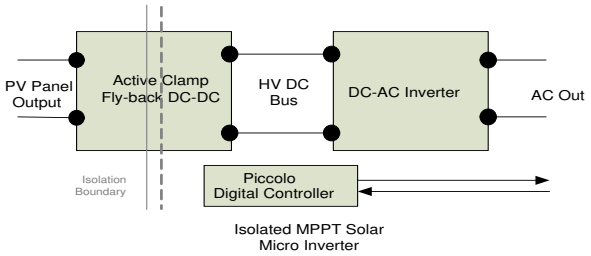


Fig. 1: TI Microinverter Block Diagram [10].

B. The NN Controller

A NN will be implemented in the Piccolo real-time digital controller in Fig. 1 to regulate the currents to follow the reference trajectories in a closed-loop control system.

The structure of the proposed NN Controller is shown in Fig. 2. The NN has two hidden layers, each with six neurons, and one two-neuron layer that controls the outputs.

The input block of the NN takes the tracking error input signals $\overrightarrow{e_{dq}}$ and their special error integral values $\overrightarrow{s_{dq}}$. To avoid input saturation, $\overrightarrow{e_{dq}}$ and $\overrightarrow{s_{dq}}$ are divided by constant gain values, $Gain$ and $Gain2$, respectively, and normalized by the hyperbolic tangent function, whose values are limited in the range [-1, 1]. The special error integral terms $\overrightarrow{s_{dq}}$ will guarantee that there is no steady-state error for step references [12].

Further, the NN Controller can be represented explicitly by equation (1), where $\overrightarrow{w_1}$, $\overrightarrow{w_2}$, and $\overrightarrow{w_3}$ stand for the weights of the input layer to the first hidden layer, second hidden layer, and output layer, respectively. The bias for each layer is incorporated into weights $\overrightarrow{w_1}$, $\overrightarrow{w_2}$, and $\overrightarrow{w_3}$.

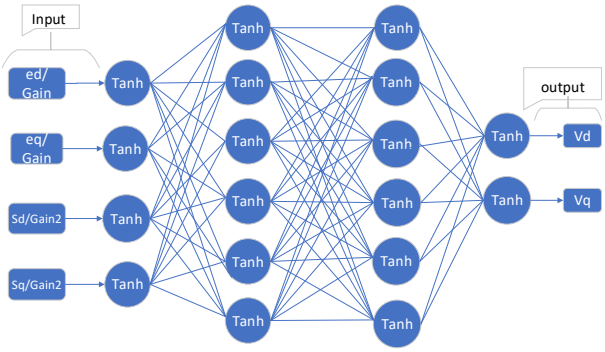


Fig. 2: The NN Controller with special tracking error integrals [11].

$$R\left(\overrightarrow{e_{dq}}, \overrightarrow{s_{dq}}, \overrightarrow{w_1}, \overrightarrow{w_2}, \overrightarrow{w_3}\right) = \tanh\left\{\overrightarrow{w_3}\left[\tanh\left\{\overrightarrow{w_2}\left[\tanh\left\{\overrightarrow{w_1}\left[\tanh\left[\begin{matrix}\frac{e_d}{Gain} \\ \frac{e_q}{Gain} \\ \frac{s_d}{Gain2} \\ \frac{s_q}{Gain2}\end{matrix}\right]\right\} \\ -1 \\ -1\end{matrix}\right]\right\} \\ -1\end{matrix}\right]\right\}\right\} \quad (1)$$

III. Training the NN Controller with Dropout

The network architecture in the proposed NN Controller Fig. 2 is assumed to be as nearly optimal as possible in terms of the layers and neurons in each layer. However, there is still a room for optimization, in particular by deactivating weights in the neural network. Small weights should theoretically contribute very little to the final output of the NN Controller. Each weight in the neural network represents blocks of computation that significantly affect the speed and resource requirement of the FPGA implementation. Thus, by deactivating small largely inconsequential weights in the neural network during training, the NN Controller can be implemented in integrated circuits using fewer resource requirements and computations.

A. Levenberg-Marquardt in a Vector Controlled System

The LM optimizer is a compromise between the speed of the Gauss-Newton method and the guaranteed convergence of gradient descent in solving nonlinear least squares problems [13]. If $\overrightarrow{w}$ is a parameter vector of a model and $f(x_i, \overrightarrow{w})$ is the loss function for the ith sample, then the sum of squared errors, $S(\overrightarrow{w})$ is the following:

$$S(\overrightarrow{w}) = \sum_{i=1}^{m}[y_i - f(x_i, \overrightarrow{w})]^2 \quad (2)$$

The parameter vector $\overrightarrow{w}$ is iteratively improved by replacing it with a new estimate, $\overrightarrow{w} + \triangle\overrightarrow{w}$, and to find this estimate,

the following first-order approximation of $f(x_i, \overrightarrow{w} + \triangle\overrightarrow{w})$ is substituted into $S(\overrightarrow{w})$:

$$f(x_i, \overrightarrow{w} + \triangle\overrightarrow{w}) \approx f(x_i, \overrightarrow{w}) + J_i \triangle\overrightarrow{w} \tag{3}$$

where J_i is the row-vector gradient of f with respect to $\overrightarrow{w}$.

Taking the derivative of this estimate for $S(\overrightarrow{w} + \triangle\overrightarrow{w})$ and setting the result to zero produces a system of linear equations that can be solved for $\triangle\overrightarrow{w}$:

$$(J^T J)\triangle\overrightarrow{w} = J^T[y - f(\overrightarrow{w})] \tag{4}$$

Levenburg modified this system of linear equations by adding a dampening factor, λ to the equation:

$$(J^T J + I\lambda)\triangle\overrightarrow{w} = J^T[y - f(\overrightarrow{w})] \tag{5}$$

The dampening factor λ can be adjusted during training. The larger the value for λ, the closer the adjustments to $\overrightarrow{w}$ approximate the gradient-descent method, and the smaller the value for λ the closer the adjustments to $\overrightarrow{w}$ are to the Gauss-Newton method for minimizing non-linear least squares [13]. Thus, in the LM algorithm, if a particular value for λ leads to a reduction in the cost function, λ is increased by a factor, λ_{inc}; if not, λ is decreased by the factor, λ_{dec}. This approach is repeated until some stopping criteria are met. The stopping criteria may vary across implementations of the LM algorithm. In this implementation, the stopping criteria occur when λ reaches some max λ_{max} or until $maxEpochs$ is reached.

To use LM as an optimizer for a model, the model's cost function must be expressed as a sum of squared errors. The ultimate goal of training the NN Controller is to minimize the cost function associated with a vector-controlled system defined as follows:

$$C(\overrightarrow{i_{dq}}) = \sum_{k=j}^{\infty} \gamma^{k-j} U(\overrightarrow{e_{dq}}(k)), j > 0, 0 < \gamma \leq 1 \tag{6}$$

where k refers to the kth time step and γ is some discount factor. U is defined as such:

$$\begin{aligned} U(\overrightarrow{e_{dq}}(k)) &= [{e_d}^2 + {e_q}^2]^{\alpha} \\ &= \left\{[i_d(k) - i_{d\_ref}(k)]^2 + [i_q(k) - i_{q\_ref}(k)]^2\right\}^{\alpha} \end{aligned} \tag{7}$$

where $\alpha > 0$ [6].

In other words, the discount factor γ ensures that the kth time step is considered more weighty in the cost function than prior steps. To convert this cost function to the appropriate sum of squared errors, it is sufficient to note that in the simple case where $\gamma = 1$ and $j = 1$ and $V(k)$ is defined as $\sqrt{U(\overrightarrow{e_{dq}}(k))}$, then

$$\begin{aligned} C &= \sum_{k=1}^{N} U(\overrightarrow{edq}(k)) \\ &= \sum_{k=1}^{N} (V(k))^2 \end{aligned} \tag{8}$$

More technical details about LM can be found in [6].

B. Forward Accumulation Through Time

The FATT algorithm was selected to speed training the NN Controller [14]. The FATT algorithm efficiently computes the cost of the NN Controller, $C_1 = S(\overrightarrow{w})$, the Jacobian matrix, J, and the training pattern, P which is the difference between the target trajectories, y, and the output of the NN Controller, $f(\overrightarrow{w})$ at all time steps. The Jacobian J and the training pattern, P, are used to solve for $\triangle\overrightarrow{w}$ to compute the cost of the new estimate, $C_2 = S(\overrightarrow{w} + \triangle\overrightarrow{w})$. Those costs, C_1 and C_2 can then be compared to determine how to adjust λ in accordance with the LM algorithm.

C. Adding Dropout

The notion of dropout employed in this paper is different from the traditional dropout policy for several reasons. First, the goal is not to reduce overfitting in a model but instead to minimize the total connections necessary for the neural network during inference. Second, rather than dropping all the incoming and outgoing connections to a neuron, the approach here drops only one connection at a time after the initial round of training, followed by another pass of training for each dropped weight. Once training is complete, the ignored connections can be omitted in embedded implementations to reduce computations at test time. This can be a particularly useful approach for small networks that have already been optimized in terms of their layers and neurons.

To represent a deactivated weight, the dropout algorithm merely sets a weight to zero so it has no effect on the output of the neural network. However, because the LM algorithm involves solving a system of linear equations to produce a new vector of weight updates, $\triangle\overrightarrow{w}$, and also $\overrightarrow{w} + \triangle\overrightarrow{w}$ becomes the new estimate for $\overrightarrow{w}$ each iteration, it is important when a weight has been selected for deactivation that will not be modified during any iteration. For this reason, the algorithm tracks the indices of dropped weights in a set, $w_{dropped}$, during each iteration to set deactivated weights in $\overrightarrow{w}$ back to zero. The dropout algorithm also ignores any diagonal elements in each weight matrix. The stability and convergence have connections with the system eigenvalues, which are determined by the system equations and the weight matrices of the NN Controllers [12]. For a matrix, the diagonal elements normally have a large impact on the eigenvalues, e.g. the trace of a square matrix, which is defined to be the sum of elements on the main diagonal (from the upper left to the lower right), equals the summation of all the eigenvalues [15]. Thus, in order not to change system eigenvalues too much, the proposed dropout algorithm will not drop any diagonal elements. The algorithm maintains a set of candidate weights, $w_{candidate}$, which is initialized to all nondiagonal indices and then kept up to date as new weights are dropped during training.

Algorithm 1 shows the pseudocode for the full dropout algorithm, and Fig. 3 shows a flowchart of the logic of the proposed algorithm. The essence of the training algorithm is as follows: train with LM+FATT until $\lambda == \lambda_{max}$. At that point, the dropout algorithm drops the smallest weight in $\overrightarrow{w}$ from $w_{candidate}$, and then $w_{candidate}$ and $w_{dropped}$

Algorithm 1 LM+FATT With Dropout

$w_{candidate} \leftarrow$ { indices in $\overrightarrow{w}$ except diagonal weights }
$dropped \leftarrow 0$; $w_{dropped} \leftarrow \{\}$
for $k = 1$ to $maxEpochs$ **do**
  J, C_1, $P \leftarrow FATT(X, \overrightarrow{w})$
  while $\lambda < \lambda_{max}$ **do**
    $\triangle\overrightarrow{w} = solve(J^T J + \lambda I, P)$
    $\overrightarrow{w_{temp}} \leftarrow \overrightarrow{w} + \triangle\overrightarrow{w}$
    for i in $w_{dropped}$ **do**
      $\overrightarrow{w_{temp}}(i) \leftarrow 0$
    end for
    $C_2 \leftarrow FATT(X, \overrightarrow{w_{temp}})$
    if $C_2 < C_1$ **then**
      $\overrightarrow{w} \leftarrow \overrightarrow{w_{temp}}$
      $\lambda \leftarrow min(\lambda \times \lambda_{dec}, \lambda_{min})$
      break
    else
      $\lambda \leftarrow \lambda \times \lambda_{inc}$
    end if
  end while
  if $\lambda == \lambda_{max}$ and $dropped < maxDrop$ **then**
    $i_{drop} \leftarrow$ smallest weight in $\overrightarrow{w}$ from $w_{candidate}$
    $\overrightarrow{w}(i_{drop}) \leftarrow 0$
    $w_{candidate} \leftarrow w_{candidate} \setminus \{i_{drop}\}$
    $w_{dropped} \leftarrow w_{dropped} \cup \{i_{drop}\}$
    $dropped$++
    $\lambda \leftarrow \lambda_{start}$
  else if $\lambda == \lambda_{max}$ **then**
    break
  end if
end for

are adjusted accordingly. Since removing a weight potentially opens the possibility of finding a new local minimum, the algorithm resets λ back to λ_{start} to retrain the weights after each new weight is dropped. This process repeats until either $maxEpochs$ or $maxDropped$ is reached.

IV. Training Results and Trajectory Convergence

A. Training Implementation

The training algorithm was implemented in c++ using Armadillo [16], [17] for the linear algebra operations. The training program was run on a Windows 10 machine with an Intel(R) Core(TM) i3-6300 CPU and NVIDIA GeForce RTX 3070 GPU. The initial weights in the network were initialized to random small values between 0 and 0.1 The starting λ was initialized to 1.0, the maximum λ was initialized to 10^{10}, and the amount to increment / decrement λ was set to 10.

The reference currents were initialized to realistic values given the physical constraints of a practical inverter system. To generate these starting reference currents, first, the d-axis and q-axis currents were selected randomly from a uniform distribution from -500A to 500A. These randomly generated

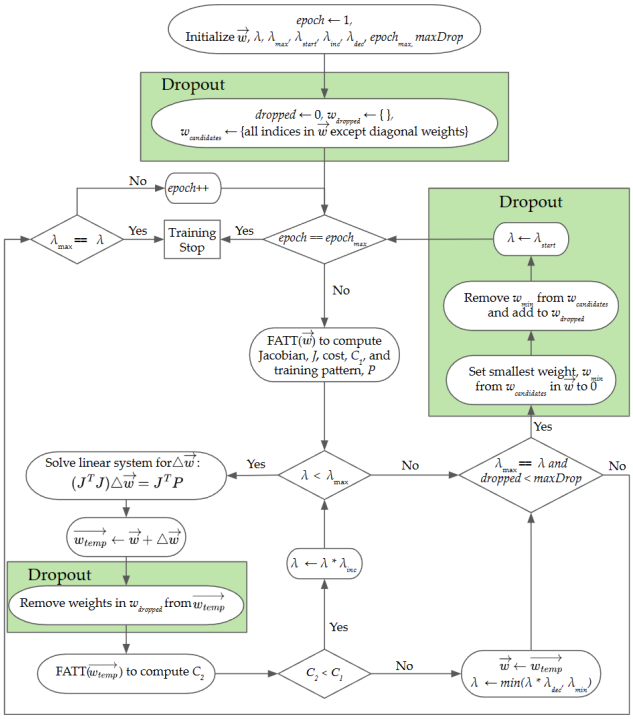


Fig. 3: Flowchart for dropout algorithm

values were then restricted such that the resultant magnitude did not exceed the inverter system current limit, which refers to the maximum positive or negative voltage that the action network can output. For each reference current, there are 1000 timesteps with a time step interval of 1 ms.

B. Trajectory Convergence Validation

Since the goal of this paper is to minimize the number of required connections in the NN Controller, the dropout training algorithm was run by setting an arbitrary number for maximum weights to drop. To test convergence, the trained network was tested on a new trajectory to plot the convergence of the NN Controller with the expected output.

Prior to applying dropout, the training algorithm produced weights that demonstrated clear convergence to the target trajectories. Fig. 5a shows that ID_REF and IQ_REF follow fairly closely to the reference currents, ID and IQ respectively.

As the number of weights dropped, a trend emerged: the cost of the test data set tended to decrease. Though reducing overfitting was not the goal of our implementation of dropout, this is an unsurprising result, insofar as dropping weights may reduce the chance the model overfits the training data. Fig. 4 shows the test costs per trajectory after each weight is dropped in the training algorithm.

Dropping the 12th weight caused the test cost to jump significantly. This corresponds to the convergence graphs obtained for the test set when the 11th and 12th weights are dropped. Fig. 5 shows the convergence for the NN Controller

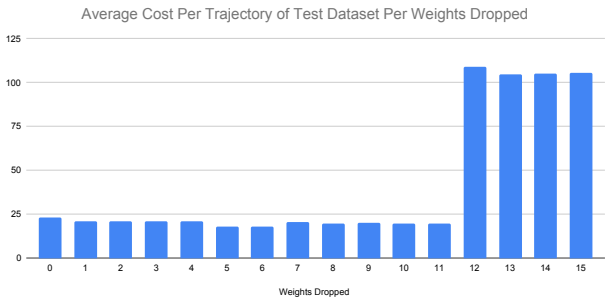


Fig. 4: Test cost per trajectory across weights

prior to dropping weights and after dropping the 10th, 11th, and 12th weights. The convergence appears best when 11 weights are dropped, which corresponds to the smallest test cost. The graph for dropping 12 weights shows that the NN Controller fails to follow the reference trajectories at this point.

V. FPGA Validation

This section focuses on comparing major resource requirements of the design with all weights and with reduced weights after dropout.

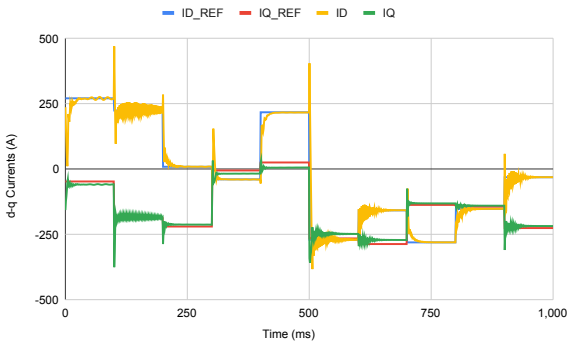
Hardware simulation was carried out on Intel Cyclone Family FPGAs. Altera Floating-Point Arithmetic (ALTFP) blocks were used to design and implement the NN schematic on Intel Quartus Prime software [18]. The NN Controller has two hidden layers and the weights in the schematic are represented using the LPM_CONS block available in IP Library. The weights are wired using the orthogonal bus tool in a matrix multiplication fashion. Every individual layer is connected together and linked by a common clock signal running throughout the schematic to simulate an Intel Cyclone 5CGX FPGA board.

A. Resource Requirements with All Weights

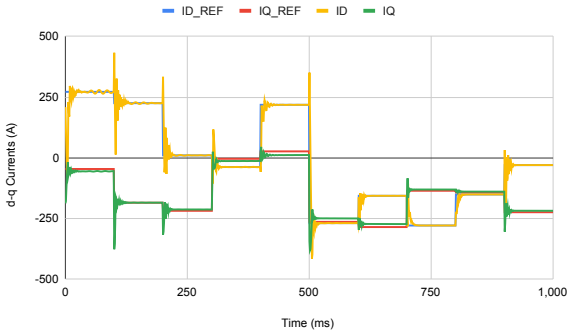
Fig. 6 shows a snapshot of the resource requirement for compiling the NN Controller design with all the weights included. The three layers in the NN Controller are wired together using a common clock signal. FPGAs come with predefined resources, so the goal is to implement the NN Controller within the given resource requirement. The Adaptive Logic Module (ALM) in Cyclone V FPGA determines the total area occupied by the NN on the FPGA board. Every block and ALM used in the design occupies memory on the chip. These circuits enable adjacent logic functions to share logic, which provides higher performance, greater efficiency, and reduced clock latency.

B. Resource Requirement with Reduced Weights

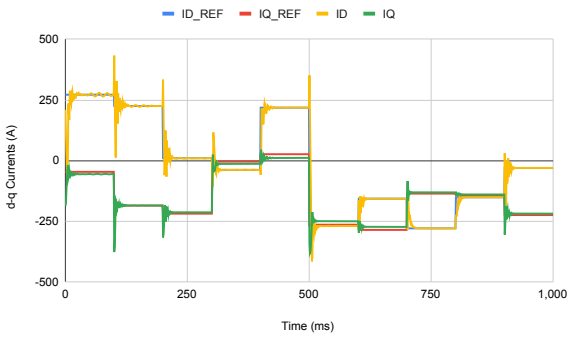
The schematic design was implemented by removing the 11 weights that were dropped during training. The resource requirements of the design are depicted in Fig. 7. Since every block in the schematic occupies memory on the board and is linked with a common clock signal, reducing the number



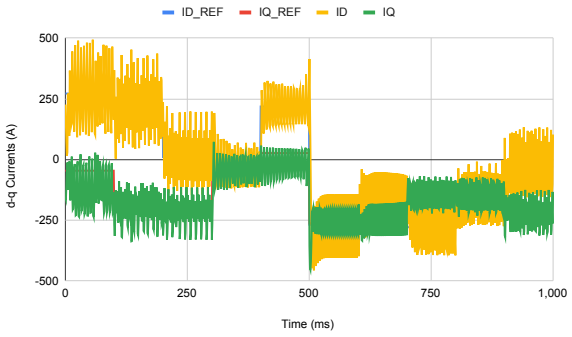
(a) Convergence without dropping any weights



(b) Convergence when 10 weights are dropped

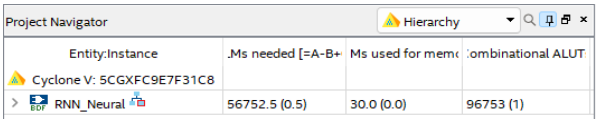


(c) Convergence when 11 weights are dropped

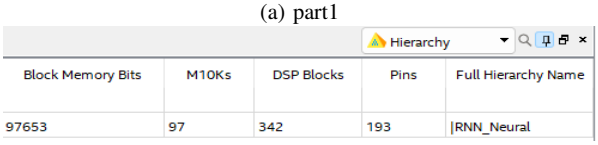


(d) Convergence when 12 weights are dropped

Fig. 5: Comparing convergence for different numbers of weights dropped



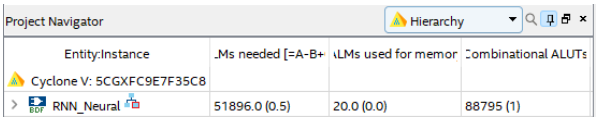
(a) part1



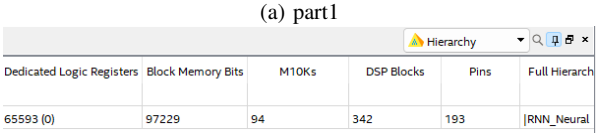
(b) part2

Fig. 6: NN resource requirement with all weights

of weights produced will result in smaller wire length during the placement and routing processes of compilation leading to optimized resource requirements in comparison to the full NN Controller without dropout. Hence the new design incurred less memory and provided enhanced performance with reduced compilation time when simulated and synthesized.



(a) part1



(b) part2

Fig. 7: NN resource requirement with 11 weights dropped

C. Resource Requirement Comparison

Table I shows the comparison between the resource requirement with all the weights connected and when weights are removed. It is clear that by reducing 11 weights from the design, the total ALM blocks needed to synthesize and simulate are reduced by 8.5% which reduced the memory occupancy by 33%. Table I shows that block memory bits have also been reduced by 0.5% on the board. This confirms that the weight-dropout approach has significantly reduced the computations of the NN Controller, improved the area, and enhanced efficiency.

TABLE I: Comparison of Resource Requirements

Resource	All Weights	Dropping 11 Weights	Improvement
ALM needed	56752	51896	8.5%
ALM used for Memory	30	20	33.33%
Combinational LUT	96753	88795	8.22%
Dedicated logic Registers	68887	65593	4.78%
Block Memory Bits	97653	97061	0.5%
MK10s	97	93	4.1%

CONCLUSION

This paper shows how to integrate a new dropout algorithm into the LM-FATT training algorithm to reduce the overall weights necessary to implement the NN Controller in embedded systems for a solar inverter. By eliminating 11 weights from the network, the NN Controller improved on tracking its reference currents. At the same time, when implemented in an FPGA, the model with fewer weights incurred substantially less memory occupancy and computations. Our proposed dropout technique will significantly improve the performance of the NN Controller by reducing overall weights in the embedded system.

ACKNOWLEDGMENT

This work is partially supported by the National Science Foundation of the United States under Award CISE-MSI 2131214 and 2131175. We thank anonymous reviewers for their insightful comments.

REFERENCES

[1] S. Li, T. A. Haskew, Y.-K. Hong, and L. Xu, "Direct-current vector control of three-phase grid-connected rectifier–inverter," *Electric Power Systems Research*, vol. 81, no. 2, pp. 357–366, 2011.

[2] J. Dannehl, C. Wessels, and F. W. Fuchs, "Limitations of voltage-oriented pi current control of grid-connected pwm rectifiers with *lcl* filters," *IEEE transactions on industrial electronics*, vol. 56, no. 2, pp. 380–388, 2008.

[3] S. Li and T. A. Haskew, "Analysis of decoupled dq vector control in dfig back-to-back pwm converter," in *2007 IEEE power engineering society general meeting*. IEEE, 2007, pp. 1–7.

[4] L. Shuhui and T. A. Haskew, "Transient and steady-state simulation study of decoupled dq vector control in pwm converter of variable speed wind turbines," in *IECON 2007-33rd Annual Conference of the IEEE Industrial Electronics Society*. IEEE, 2007, pp. 2079–2086.

[5] S. Li, D. C. Wunsch, M. Fairbank, and E. Alonso, "Vector control of a grid-connected rectifier/inverter using an artificial neural network," in *The 2012 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2012, pp. 1–7.

[6] X. Fu, S. Li, M. Fairbank, D. C. Wunsch, and E. Alonso, "Training recurrent neural networks with the levenberg–marquardt algorithm for optimal control of a grid-connected converter," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 9, pp. 1900–1912, 2014.

[7] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[8] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[9] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout improves recurrent neural networks for handwriting recognition," in *2014 14th international conference on frontiers in handwriting recognition*. IEEE, 2014, pp. 285–290.

[10] M. Bhardwaj and S. Choudhury, "Digitally controlled solar micro inverter design using c2000 piccolo microcontroller user's guide," Technical report, Tech. Rep., 2017.

[11] W. Waithaka, X. Fu, A. Al Hadi, R. Challoo, and S. Li, "Dsp implementation of a novel recurrent neural network controller into a ti solar microinverter," in *2021 IEEE Power & Energy Society General Meeting (PESGM)*. IEEE, 2021, pp. 1–5.

[12] X. Fu, S. Li, D. C. Wunsch, and E. Alonso, "Local stability and convergence analysis of neural network controllers with error integral inputs," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[13] J. Pujol, "The solution of nonlinear inverse problems and the levenberg-marquardt method," *Geophysics*, vol. 72, no. 4, pp. W1–W16, 2007.

[14] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
[15] G. Strang, *Introduction to Linear Algebra 4th Edition*. Wellesley, MA, USA: Wellesley-Cambridge Press, 2009.
[16] C. Sanderson and R. Curtin, "Armadillo: a template-based c++ library for linear algebra," *Journal of Open Source Software*, vol. 1, no. 2, p. 26, 2016.
[17] S. Conrad and C. Ryan, "A user-friendly hybrid sparse matrix class in c++," in *Mathematical Software–ICMS 2018: 6th International Conference, South Bend, IN, USA, July 24-27, 2018, Proceedings 6*. Springer, 2018, pp. 422–430.
[18] C. Hingu, X. Fu, S. Smith, T. Saliyu, and L. Qingge, "Fpga acceleration of a real-time neural network controller for solar inverter," in *2022 IEEE 13th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE, 2022, pp. 0413–0420.