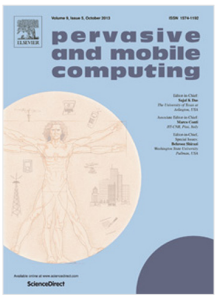
Journal Pre-proof



Accelerating the neural network controller embedded implementation on FPGA with novel dropout techniques for a solar inverter

Jordan Sturtz, Kushal Kalyan Devalampeta Surendranath, Maxwell Sam, Xingang Fu, Chanakya Dinesh Hingu, Rajab Challoo, Letu Qingge

PII: S1574-1192(24)00100-7
DOI: https://doi.org/10.1016/j.pmcj.2024.101975
Reference: PMCJ 101975

To appear in: *Pervasive and Mobile Computing*

Received date : 31 December 2023
Revised date : 13 August 2024
Accepted date : 14 August 2024

Please cite this article as: J. Sturtz, K.K.D. Surendranath, M. Sam et al., Accelerating the neural network controller embedded implementation on FPGA with novel dropout techniques for a solar inverter, *Pervasive and Mobile Computing* (2024), doi: https://doi.org/10.1016/j.pmcj.2024.101975.

Response to Reviewers

Available online at www.sciencedirect.com
ScienceDirect
Pervasive and Mobile Computing 00 (2024) 1–19
PMC

Accelerating the Neural Network Controller Embedded Implementation on FPGA with Novel Dropout Techniques for a Solar Inverter

Jordan Sturtz[a], Kushal Kalyan Devalampeta Surendranath[a], Maxwell Sam[a], Xingang Fu[b], Chanakya Dinesh Hingu[b], Rajab Challoo[c], Letu Qingge[0a]

[a]*Department of Computer Science, North Carolina A&T State University, Greensboro, NC, USA*
jasturtz@aggies.ncat.edu, ksurendranath@aggies.ncat.edu, msam@aggies.ncat.edu, lqingge@ncat.edu
[b]*Department of Electrical and Biomedical Engineering, University of Nevada, Reno, Reno, NV, USA*
xfu@unr.edu, chingu@nevada.unr.edu
[c]*Department of Electrical Engineering and Computer Science, Texas A&M University-Kingsville, Kingsville, TX, USA*
rajab.challoo@tamuk.edu

Abstract

Accelerating neural network (NN) controllers is important for improving the performance, efficiency, scalability, and reliability of real-time systems, particularly in resource-constrained embedded systems. This paper introduces a novel weight-dropout method for training neural network controllers in real-time closed-loop systems, aimed at accelerating the embedded implementation for solar inverters. The core idea is to eliminate small-magnitude weights during training, thereby reducing the number of necessary connections while ensuring the network's convergence. To maintain convergence, only non-diagonal elements of the weight matrices were dropped. This dropout technique was integrated into the Levenberg-Marquardt and Forward Accumulation Through Time algorithms, resulting in more efficient training for trajectory tracking. We executed the proposed training algorithm with dropout on the AWS cloud, observing a performance increase of approximately four times compared to local execution. Furthermore, implementing the neural network controller on the Intel Cyclone V Field Programmable Gate Array (FPGA) demonstrates significant improvements in computational and resource efficiency due to the proposed dropout technique leading to sparse weight matrices. This optimization enhances the suitability of the neural network controller for embedded environments. In comparison to Sturtz et al. (2023), which dropped 11 weights, our approach eliminated 18 weights, significantly boosting resource efficiency. This resulted in a 16.40% reduction in Adaptive Logic Modules (ALMs), decreasing the count to 47,426.5. Combinational Look-Up Tables (LUTs) and dedicated logic registers saw reductions of 17.80% and 15.55%, respectively. However, the impact on block memory bits is minimal, showing only a 1% improvement, indicating that memory resources are less affected by weight dropout. In contrast, the usage of Memory 10 Kilobits (MK10s) dropped from 97 to 87, marking a 10% improvement. We also propose an adaptive dropout technique to further improve the previous results.

Keywords:
weight dropout technique, neural network controller, Levenberg-Marquardt algorithm, Forward Accumulation Through Time, Field Programmable Gate Array (FPGA), cloudbank

[0]This work is supported by the U.S. National Science Foundation under Grant CISE-MSI 2131175 and 2131214. Corresponding author: Letu Qingge. Email: lqingge@ncat.edu.

1. Introduction

In today's energy landscape, the pursuit of sustainable power systems has gained significant momentum, with solar inverters at the forefront of this revolution. Solar inverters have great potential to increase energy conversion rates and maintain grid resilience, making them a crucial component in the shift to renewable energy sources. These devices are not only essential for harnessing solar energy but also ensure a steady and uninterrupted energy supply in various settings due to their efficient and reliable operations.

The field of neural network (NN) controllers has seen significant advancements, especially with the integration of NN techniques into embedded systems. These advancements have enhanced the performance, adaptability, and efficiency of NN controllers. For example, Hingu *et al.* (2023) [1] used a 32-bit fixed-point design instead of a 32-bit floating-point representation to implement a neural network controller for the real-time control of a solar inverter by utilizing a Field Programmable Gate Array (FPGA) for faster calculation in real-time environments. Sturtz *et al.* (2023) [2] removed redundant connections in the neural network to reduce its size and improve computational efficiency. Hingu *et al.* (2022, 2023) [1, 3] executed NN operations on Field-Programmable Gate Arrays (FPGAs), allowing for parallel processing and lower latency. Fu *et al.* (2024) presented a novel parallel trajectory mechanism that combines Levenberg-Marquardt and Forward Accumulation Through Time algorithms to train a recurrent neural network controller in a closed-loop control system. Recurrent neural networks (RNNs) are more successful than traditional vector control in the optimization of grid-connected rectifiers in recent studies, underscoring their potential in embedded systems such as digital signal processors (DSP) [4]. Crucial to these systems is the NN controller, which learns from changing circumstances to maximize performance.

In [5], the RNN implements a dynamic programming algorithm and is trained with Backpropagation Through Time (BPTT), which is combined with Resilient Propagation (RPROP) to accelerate the training speed. However, training an RNN with BPTT combined with RPROP poses some problems, including slow convergence speed and oscillation problems that may cause training to diverge. In [6], a Real-Time Recurrent Learning (RTRL) is proposed to train an RNN. But, the high computational cost of RTRL makes it only appropriate for online training of a small recurrent neural network [6], [7], and [8]. Alternatively, the Extended Kalman Filter (EKF) is useful in training RNN controllers for linear and nonlinear dynamical systems [9], [10], and [11]. Nevertheless, EKF is also computationally expensive, because it needs lots of matrix calculations at each estimation. Besides, the eventual success and quality of EKF training depend very much on professional experience, including an appropriate selection of the network architecture, learning rates, presentation of network inputs, etc. [8]. In real-time systems, such as solar inverters, NN controllers must process inputs and produce outputs within strict time constraints. Faster processing ensures the system can respond promptly to changes in the environment, maintaining stability and efficiency. Faster computations can allow for more frequent updates and finer control granularity. This improvement can lead to more accurate and reliable system performance, which is critical in applications where precision is vital. Furthermore, embedded systems often have limited computational resources and power availability. Due to limited memory and calculating plus the complexity of an NN, some major concerns from professionals in this field are: how an NN controller can be implemented while using regular digital signal processor (DSP) chips that have limited memory and computing capability; what the detailed hardware configuration should be for the NN controller; how to implement the DSP-based NN control algorithm, how the DSP-based NN controller behaves in hardware experimental conditions when compared to existing methods and so on. These problems challenge the RNN controller to be used in a real-time embedded environment. Accelerating NN controllers allows them to run efficiently on these constrained platforms, reducing the need for more powerful, costly, or power-hungry hardware. Training the NN controller is vital for the performance of the controller, which includes two aspects: training algorithm and training speed.

To overcome these challenges, this paper presents a novel weight dropout approach for training the NN controller, which produces faster convergence and accurate prediction with regard to given trajectories. We incorporated the weight dropout approach into Levenberg-Marquardt (LM) and Forward Accumulation Through Time (FATT) algorithms to accelerate the training of the NN controller. LM algorithm is essential for optimizing NN Controllers by striking a balance between robustness and fast convergence [12]. Additionally, to improve training efficacy, the Forward Accumulation Through Time (FATT) approach is applied for handling sequential data in RNNs. The weight dropout technique is excellent at reducing overfitting, to further strengthen the robustness of the methodology and improve the neural network's capacity to generalize in a variety of scenarios [13]. This research strives to push forward

the boundaries in energy-efficient solar inverter operations by integrating these approaches in a way that maximizes resource efficiency and improves the proposed algorithm's performance.

The following are specific contributions made by this paper. 1) The development of a weight-dropout technique within the Levenberg-Marquardt (LM) and Forward Accumulation Through Time (FATT) algorithms, to enhance the training efficiency of the neural network controller for precise trajectory tracking; 2) the introduction of weight dropout technique with random weights initialization strategy for NN controllers to improve neural network training process robustness and diversity; 3) running our methods on cloud and GPU platforms, leveraging their computational power to achieve faster training and optimization results; and 4) The FPGA validation with reduced weight NN structure of our proposed dropout approach within an embedded environment; 5) design the adaptive dropout approach to further improve the weight dropout technique applied in NN controller.

The rest of the paper is organized as follows. Section 2 introduces the NN controllers in the closed-loop control system for a solar inverter. The weight-dropout approach for training the NN controllers is designed in Section 3. The dropout implementation in C++ on both cloud and local machines is presented in Section 4. Section 5 provides a detailed experimental analysis. Section 6 discusses FPGA implementation to validate the new NN controller after dropout. Lastly, the paper concludes with a summary of the main points in Section 7.

2. Related Work

Besides the conventional standard vector control [14], other inverter control technologies include direct current control, direct power control, predictive current control, proportional resonant control, H2/$H\infty$ based optimal control, etc. The idea behind the Direct Power Control approach, proposed by Noguchi [15], is the direct control of active and reactive power. This approach does not require inner current control loops or a PWM modulator because the converter switching states are selected by a switching table based on the instantaneous errors between the commanded and estimated values of the active and reactive power. The Direct Power Control boasts a high dynamic response to active and reactive power demands and simplicity of implementation [16, 17]. However, its primary disadvantages include high harmonic distortion, unbalanced system current, variable switching frequency under different operating conditions, and the requirement of a high switching frequency converter [17]. Predictive Current Control [18, 19] utilizes a current prediction equation to estimate the grid current at the next sampling interval and a control equation to determine the next GCC control voltage. It has a fast current tracking response [19, 20] but becomes unstable when the programmed filter inductance differs from its actual value. Also, if the resistive part of the filtering inductor is not measured and programmed accurately, the predictive control presents a steady-state error. Since filter parameters vary along with the inverter operation, achieving an adequate static and dynamic performance is difficult [20]. Proportional-integral-resonant (PI-PR) control is similar to conventional standard vector control, except that it combines PI control with several PR (proportional-resonant) control paths to enhance tracking of the current reference that may contain a lot of AC disturbance components [21, 22]. The PI-PR approach requires appropriately tuned parameters of different resonant terms, and its performance can be adversely affected when system parameters change, or when disturbance harmonics are different from those used to tune the resonant terms. The direct-current control (DCC) methodology, as detailed in [23, 24], effectively addresses the deficiencies inherent in the conventional standard vector control technique. Unlike the standard vector control approach that generates the d-axis or q-axis voltage, DCC outputs a current signal using the d-axis or q-axis current-loop controller, in which the output of the controller is the d- or q-axis tuning current id or iq while the input error signal tells the controller how much to adjust the tuning current during the dynamic control process. However, a major challenge of the DCC is that no well-established, systematic approach exists for tuning the controller PI gains, so an optimal DCC controller is difficult to obtain. H2/$H\infty$ control has become a favorite optimal control technique [25, 26]. Although it is developed based on a complete GCC dynamic equation, this approach requires a reasonably accurate system model [27]. Also, it does not handle non-linear constraints very well [28]. In [29], it was found that applying a mixed H2/$H\infty$ optimal controller in an experimental condition is much more challenging and different than in the simulated environment of the same system. For such situations, a neural network (NN) controller as the universal function approximator [30], developed based on approximate dynamic programming (ADP) optimal control principles [31, 32] and using the complete system dynamic equation including all the terms, would be the most suitable for obtaining optimal control actions contributed considering multiple factors through training [5, 33]. Furthermore, a deep neural network has demonstrated a strong ability to capture complicated features and has found numerous industrial applications such as image processing [34], video processing [35], natural language

processing [35], etc. Therefore, a deep neural network controller is expected to overcome the challenges faced by the conventional controllers, to better capture and respond to the nonlinear switching behavior in controlling a power electronics device, and to adapt to disturbance and noise from the system and environment [5, 33]. In the research, [4] examines optimal control of a Grid-Connected Converter (GCC) by using a recurrent neural network (RNN). A Forward Accumulation Through Time (FATT) algorithm is proposed in the paper to calculate the Jacobian matrix required by LMBP efficiently. Results show that the combination of LMBP and FATT algorithms is very efficient and produces superior performance for GCC control using neural networks. The approximate dynamic programming (ADP) based NN controlling of the LCL filter-based three-phase [36] and single-phase [33] GCC systems was also demonstrated to be able to yield an excellent performance compared to the conventional PI controller-based control methods. In [36], it has been demonstrated that Recurrent Neural Network (RNN) vector control does not require damping for a three-phase GCC with an LCL filter and has a wider stability region for the system parameter change than Active Damping (AD) or Passive Damping (PD) vector control for LCL-based grid-connected converter systems. [37] demonstrates the effectiveness and efficient outcomes of the proposed neural network controller for grid-tied multilevel inverters. The advantages of the proposed neural control include a faster response speed and fewer oscillations compared with the conventional Proportional Integral (PI) controller-based vector control strategy. In particular, the neural network control technique provides better harmonics reduction ability. In [6], the well-trained RNN controller for LCL-based inverters was validated through a Texas Instruments (TI) LCL filter-based solar microinverter kit that contains a C2000 TI microcontroller.

The FPGAs bring great benefits to power electronics, especially when it comes to reliability, efficiency, and real-time processing capacity. The combination of FPGA hardware and neural network technology creates a wealth of opportunities for sophisticated, intelligent control systems in renewable energy applications, leading to more reliable and efficient power electronics solutions. Ahsan et al. [38] investigated the conceptual framework of NGSG, incorporating intelligent control, agent-based energy conversion, edge computing for energy management, IoT-enabled inverters, and agent-oriented demand-side management. They discussed the development of data-driven NGSG, highlighting the use of emerging data-driven techniques (DDTs) for sustainable operation. Spiking neural networks (SNNs), particularly Spike-by-Spike (SbS) networks, offer noise robustness and reduced complexity but face memory and computational challenges for embedded applications [39]. Their study addressed these challenges by designing a vector dot-product hardware unit using approximate computing with custom floating-point and logarithmic representations, implemented on a Xilinx SoC-FPGA. The design achieves a 20.5× reduction in computational latency and 8× reduction in memory footprint with minimal accuracy loss less than (0.5%) in handwritten digit recognition. Wu et al. [40] explored FPGA-based acceleration of neural networks, analyzing architectures and FPGA characteristics. Their analysis outlined five acceleration strategies which included computing complexity, computing parallelism, data reuse, pruning, and quantization. Jiang et al. [41] investigated a customized adaptive activation function (AAF) that matches DNN accuracy. An FPGA implementation of a customized segmented spline curve neural network (SSCNN) using AAF replaces traditional fixed activation functions. Their findings compared to DNNs, SSCNN implementation which used 40% fewer hardware resources and no block RAMs, validated for digital predistortion of RF power amplifiers on the AMD/Xilinx RFSoC ZCU111, using less than 3% of available resources and enabling increased clock frequency for wide bandwidth signal transmission.

3. NN controller for a Solar Inverter

3.1. A Solar Microinverter

A solar microinverter is a device used in photovoltaic (PV) systems to convert direct current (DC) generated by solar panels into alternating current (AC) that can be fed into the utility grid or used by local electrical devices. 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 Figure 1 [42]. 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. DC-DC converter converts the variable DC output from the solar panel to a stable DC voltage. DC-AC inverter converts the stable DC voltage to AC voltage. The Maximum Power Point Tracking (MPPT) continuously monitors the voltage and current from the solar panel and adjusts the load impedance to maximize power extraction from the panel under varying sunlight conditions.

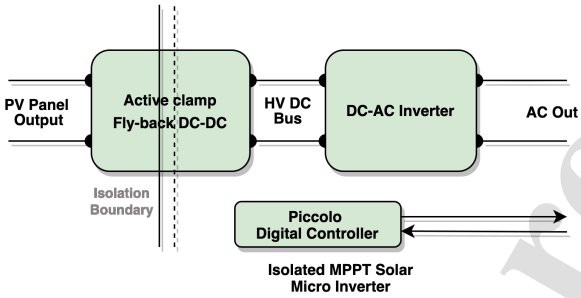


Figure 1: Microinverter Block Diagram [43]

3.2. NN controller

Figure 2 shows the NN controller in a closed-loop control system. The system reference signal trajectories serve as the feedback connections for the NN controller as seen in Figure 2. Moreover, the calculation of the error integration terms $\overrightarrow{s_{dq}}(k)$ has to accumulate all past error terms $\overrightarrow{e_{dq}}(j)$ from $j = 0$ to $j = k$ and each past error term $\overrightarrow{e_{dq}}(j)$ computation will involve the outputs of the NN controller in the corresponding past step j. Thus, the proposed NN is a recurrent NN and will be denoted as RNN thereafter.

A control technique is adapted using a fully connected Neural Network (NN). This technique will be implemented within a Piccolo digital real-time controller to regulate the currents to follow the reference trajectories in a closed-loop control system, as depicted in Figure 1. The NN's architecture as outlined in Figure 2, consists of two hidden layers with six neurons each, an input layer with four neurons and an output layer with two neurons that control outputs.

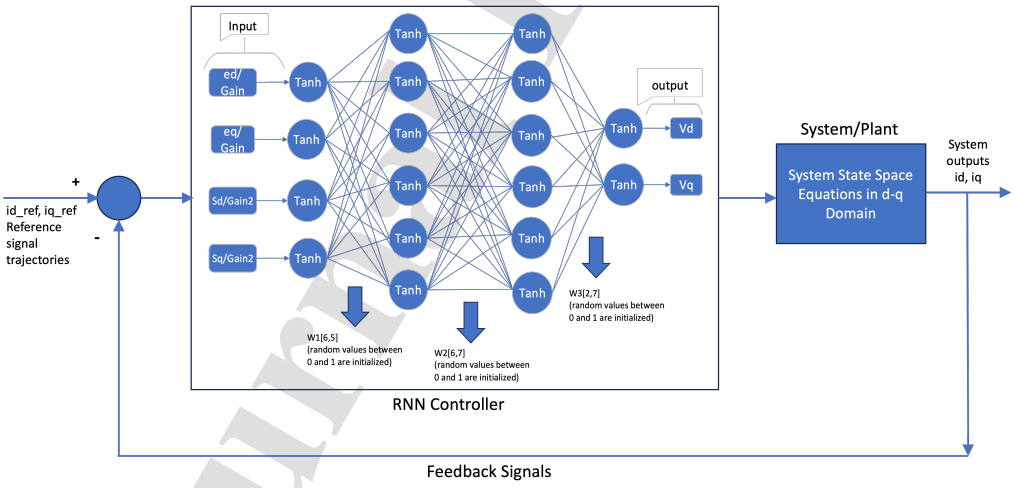


Figure 2: The NN controller with special tracking error integrals and random weight initialization [2] in a closed-loop control system. The system equations serve as the feedback connections for the NN controller.

Though Figure 2 appears to be a feedforward NN, since we use the NN in the closed control loop, it is an RNN controller with the system equations acting as a feedback connection for the NN as shown in Figure 2. The tracking error signals represented as $\overrightarrow{e_dq}$, and the associated integral error values, represented as $\overrightarrow{s_dq}$, are fed into the input

segment of the NN controller. The hyperbolic tangent function is used to normalize these signals within the [-1, 1] range after they have been scaled down by fixed gain constants, called Gain and Gain2 [43]. This prevents the problem of input saturation. Error integral terms $\overrightarrow{sdq}$ are included to guarantee that steady-state errors arising from step input references are eliminated.

NN controller can be further represented by equation 1 where $\overrightarrow{w1}$, $\overrightarrow{w2}$, and $\overrightarrow{w3}$ denote the synaptic weight matrices with random values connecting the input to the first hidden layer, the first hidden layer to the second, and the second hidden layer to the output layer, respectively. To make the structure simpler, the bias terms for each layer have been integrated within the weight matrices $\overrightarrow{w1}$, $\overrightarrow{w2}$, and $\overrightarrow{w3}$.

$$R\left(\overrightarrow{e_{dq}}, \overrightarrow{s_{dq}}, \overrightarrow{w_1}, \overrightarrow{w_2}, \overrightarrow{w_3}\right) = \tanh\left\{\overrightarrow{w_3}\left[\tanh\left\{\overrightarrow{w_2}\left[\tanh\left\{\overrightarrow{w_1}\left[\tanh\left[\begin{array}{c}\frac{e_d}{Gain}\\ \frac{e_q}{Gain}\\ \frac{s_d}{Gain2}\\ \frac{s_q}{Gain2}\\ -1\end{array}\right]\right]\\ -1\end{array}\right]\right\}\\ -1\end{array}\right]\right\}\right\} \quad [2] \tag{1}$$

Equation (1) represents the transformation function R used in the neural network controller for grid-connected converters. This equation models the relationship between the input vectors $\vec{e}_{dq}$ and $\vec{s}_{dq}$ (representing error and state vectors in the d-q axis) and the control outputs through a series of weighted transformations and nonlinear activations. The input vectors are first processed through multiple layers of the neural network, each characterized by a weight matrix W_1, W_2, or W_3. At each layer, the hyperbolic tangent (tanh) function is applied as an activation function to introduce nonlinearity, mapping the input to a range between -1 and 1. This nonlinearity is crucial for the network to learn complex relationships within the data. After passing through the layers, the final output undergoes scaling by gain factors Gain, Gain_2, and is adjusted by specific constants to ensure the output matches the required control signal characteristics. The subtraction of 1 from certain tanh outputs helps to re-center the signal for appropriate control dynamics. Overall, this equation encapsulates the neural network's forward pass, transforming the error and state information into precise control signals for optimizing the performance of the grid-connected converter.

4. Training the NN controller with Dropout

The network architecture we adopted [4] is assumed to be as nearly optimal as possible in terms of the layers and neurons in each layer. However, there is still 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.

4.1. Application of Levenberg-Marquardt in Vector Controlled Systems

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 [12]. 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:

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

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 [12]. 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 $\alpha > 0$ [4].

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 [4].

4.2. Forward Accumulation Through Time

The FATT algorithm was selected to speed training the NN controller [44]. 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.

4.3. Adding Dropout with Random Initial Weights

The dropout method used in this section is mainly concentrated on the deliberate dropping of weights to minimize the neural network's connections during inference rather than reduce model over-fitting. However, the current study also presents the idea of adding the dropout technique with random initial weights. By using traditional dropout techniques, all of a neuron's incoming and outgoing neural connections are eliminated. The approach is different, though, as it only drops one connection after the first training round and then adds a training round for each weight dropped. After training, these unnecessary connections are removed during embedded implementation, which allows for a decrease in computation when testing. This approach is found to be beneficial for the neural networks that have been pre-optimized concerning their layers and neurons [2].

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, $\triangle\vec{w}$, and also $\vec{w} + \triangle\vec{w}$ becomes the new estimate for $\vec{w}$ 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 $\vec{w}$ 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 [45]. For a matrix, the diagonal elements normally have a large impact on the eigenvalues, since the trace of a square matrix on the main diagonal (from the upper left to the lower right) equals the summation of all the eigenvalues [46]. Thus, in order to avoid changing 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 with random initialization to dropout more weights, and Figure 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 $\vec{w}$ from $w_{candidate}$, and then $w_{candidate}$ and $w_{dropped}$ 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.

The computational complexity of algorithm1, which integrates Levenberg-Marquardt (LM) optimization with Forward Accumulation Through Time (FATT), is determined by the operations executed within each component and their cumulative execution over time.

The Forward Accumulation Through Time (FATT) algorithm involves recursive calculations over time, mainly matrix multiplications associated with state transitions and weight updates. Given matrices of dimension $m \times M$, where m represents the size of the RNN output layer and M represents its weights, the primary computational demand arises from matrix multiplications across N time steps, which results in a complexity of $O(m^2NM)$. In this component, the response to a trajectory is calculated over a length of N. The Levenberg-Marquardt (LM) component optimizes the parameter updates by solving the system $J^TJ + \lambda I$, where J is the Jacobian matrix of derivatives for the weights. In general, solving this equation by matrix multiplication and inversion takes $O(M^3)$ per epoch, assuming J is of dimension $M \times M$. For each epoch of the LM+FATT algorithm, the FATT method for forward accumulation must be sequentially executed, and the LM method for optimization. As a result, the computational complexity per epoch is $O(m^2NM + M^3)$. By using this formula, we can summarize the primary computational efforts within each epoch, highlighting the key matrix operations.

4.4. Adaptive dropout in NN controller

Adaptive dropout in neural network controllers is a regularization technique that strategically deactivates neurons during training to prevent overfitting and enhance generalization. Unlike standard dropout, which uses a fixed dropout rate, adaptive dropout adjusts the rate based on the current state of the network, specifically the variability and importance of neuron activations.

The dropout probability in the adaptive dropout approach is dynamically adjusted based on three important factors which are base dropout rate, importance factor, and Training progress. Where, the base dropout rate is a predetermined value set to 0.5, which can be specified as the initial rate of dropout. The importance factor is calculated from the standard deviation of the neuron activations and this scales the dropout rate. The more important the neurons (i.e., higher variability in activations), the less likely they are to be dropped. As the training progresses, this factor adjusts to potentially reduce the rate of dropout, reflecting the increasing stability and performance of the model. The dropout probability for each neuron in a layer during training is calculated as,

$$\text{Dropout Probability} = \text{base\_dropout\_rate} \times (1.0 - \text{global\_importance\_factor}) \times \text{training\_progress}$$

The actual number of neurons dropped in each layer during a training iteration is determined by sampling from a Bernoulli distribution with the computed dropout probability. Each neuron has a chance equal to the dropout probability to be turned off (i.e., its output is set to zero). This process is repeated for each neuron in the layer, and the count of neurons that are set to zero is tracked.

In this adaptive dropout mechanism, the dropout rate is not static but varies throughout the training process. It adjusts according to the significance of neuron activations and the stage of training. This method aims to retain more information from neurons that are crucial for the network's performance, particularly as the model learns and stabilizes. By scaling the dropout rate with training progress, the model gradually relies more on its learned weights, reducing the randomness introduced by dropout as it converges towards optimal performance. The exact number of neurons dropped is probabilistically determined based on this adaptive rate, making it dependent on both the model's evolving state and inherent data characteristics.

5. Implementation Details

5.1. Training Implementation

This research takes advantage of the complementary effects of a large-scale cloud platform and a stable local environment to train a neural network (NN) controller. This divided strategy preserves user control and data preprocessing in the Local Environment while utilizing the processing power of cloud-based GPUs. The architecture serves as an example of a distributed machine learning strategy, utilizing cloud and local resources to effectively train neural networks (as in Figure 4). Because it combines the advantages of local data access with the scalable computational resources of cloud environments, this distributed paradigm is beneficial for handling increasingly complex and data-intensive models.

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 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.

We also use Amazon Web Services (AWS) as the cloud platform for training the NN controller. C++ is used to develop the training algorithm using Armadillo [47, 48] for the linear algebra operations along with other necessary libraries. The training program was run on AWS with a c5a.24xlarge instance powered by 2nd Generation AMD EPYC 7002 series CPUs which are designed for compute-intensive tasks.

Algorithm 1 LM+FATT with random initial weights and Dropout [2]

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

5.2. Cloud Computing Platform:

Amazon Web Services (AWS) is used as the cloud platform as shown in Figure 4, reliable and expandable cloud services, which are perfect for computationally demanding jobs like neural network training and massive data processing. Our research is global in scope, so AWS was selected because of its high availability, wide geographic coverage, and flexible instance kinds.

5.3. Instance Configuration:

- **Instance Type**: On AWS, c5a.24xlarge instance type is used in the research. These instances are powered by 2nd Generation AMD EPYC 7002 series CPUs and are designed for compute-intensive tasks. This instance offers high-performance computing environments a balance of memory, computing, and networking resources. With 96 virtual CPUs, 48 default CPU cores, and up to 192 GB of memory, these types of instances are very capable of handling demanding computing tasks.
- **Storage**: Amazon Elastic Block Store (EBS) is employed for high-speed storage, utilizing SSD-backed volumes with a volume size of 35GB to ensure quick data read/write operations, which is crucial for machine learning workflows.

- **Networking**: Enhanced networking with Amazon VPC was configured to ensure secure and fast data transfer between instances.

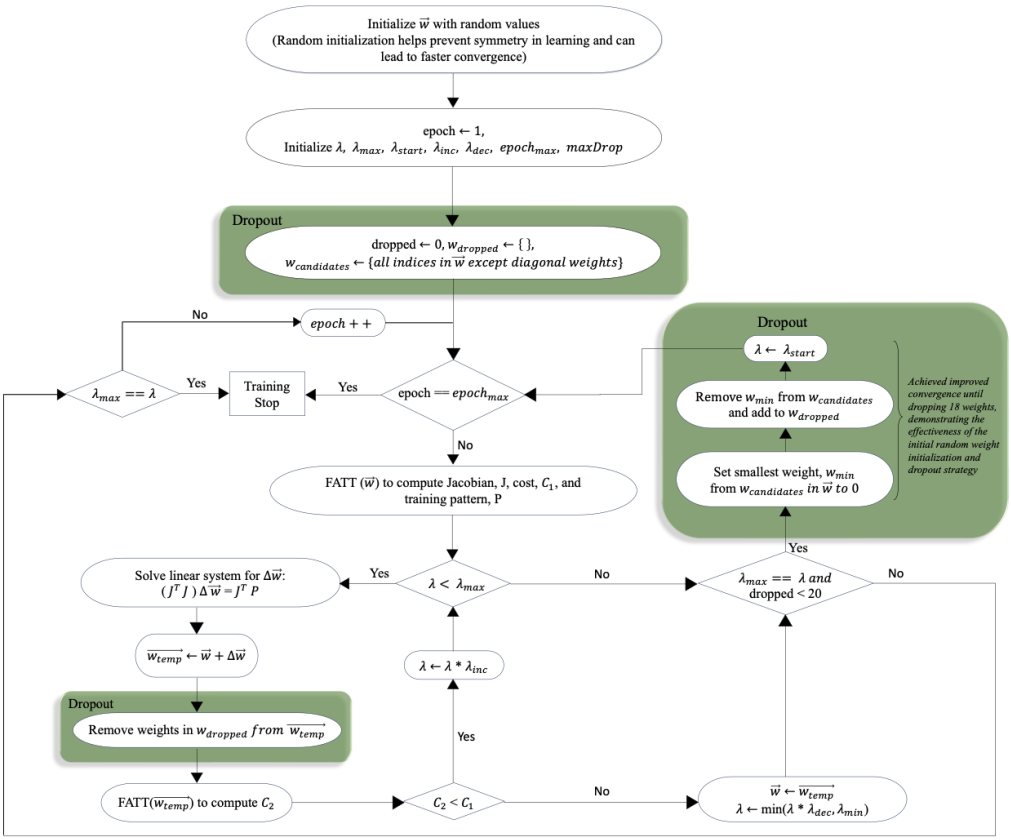


Figure 3: Flowchart for dropout algorithm with random initial weights

5.4. Software and Development Tools

- **Programming and Libraries**: The RNN model was implemented in C++, and libraries such as Eigen for effective matrix operations, Armadillo, and OpenMPI for distributed computing are set up on the instance. These libraries are essential to neural network computations.
- **Development and Version Control:**: GitHub was used to manage the codebase, which offered a stable environment for version control and development.
- **Data Transfer and Management**: Secure Shell (SSH) is used for secure file transfers to and from the cloud environment, ensuring the integrity and confidentiality of data.

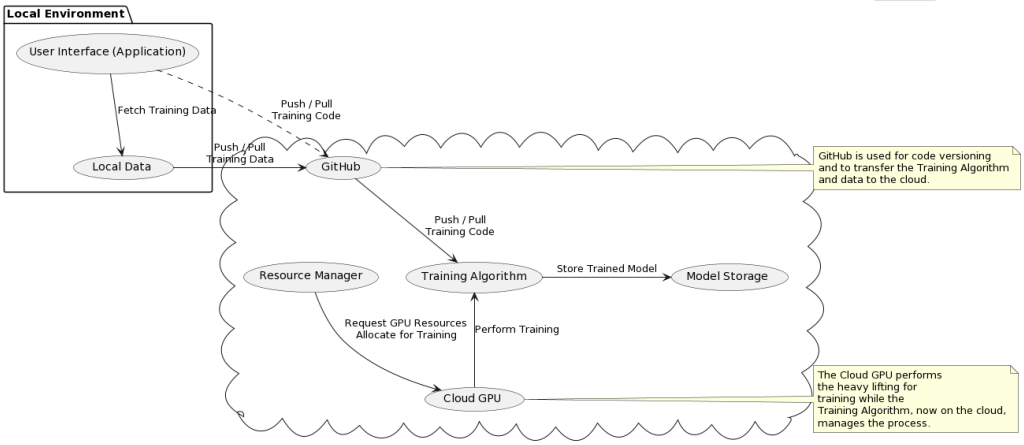


Figure 4: Training NN controller on cloud

6. Training Results and Trajectory Convergence

6.1. Trajectory Convergence Validation

The results of the current research point to a more reliable and strong NN controller. Even with the removal of 18 weights, a clear convergence was still possible with the improved training strategy. The results have significantly improved, which highlights the training methodology's progress and the NN controller's ability to produce desired outputs with fewer weights. The convergence results were significantly improved by using the dropout training algorithm with random initial weights. By increasing the number of weights dropped during training, and further explored the capabilities of the NN controller. The NN controller demonstrated optimal convergence in [2] until 11 weights were dropped. Convergence declined after the 11th weight was eliminated, particularly after the removal of the 12th weight. This phenomenon became apparent when the test cost increased and the NN controller began to stray from the reference trajectories.

Essentially, the current research has pushed that boundary further, achieving clear convergence up to the dropping of 18 weights by randomly initializing the weights in the training algorithm. As illustrated in the Algorithm 1, instead of using predefined weights in [2], we start with the random inputs and choose the converged results with the maximum number of weights dropped. This improvement not only showed that the new training method works, but also opened up new ways to use less computing power without lowering the accuracy and performance of the NN controller. Figure 5 represents our final trained RNN, where the red lines indicate the 18 dropped weights and biases that have been set to zero during the training phase. The only visible trend is that most dropped weights occur between the hidden layers and on the biases of the final hidden layer. Figure 6 shows the test costs per trajectory after each weight is dropped in the training algorithm.

Convergence plots without dropping weights and with dropping 18 weights are shown in the below figures. The training algorithm has produced the weights that have convergence with the target trajectories. Figure 6a shows that ID REF and IQ REF follow the reference currents, ID and IQ, respectively. The NN controller clearly shows convergence before dropping the weights and after dropping the 15th (Figure 6b), 16th (Figure 6c), 17th (Figure 6d), and 18th (Figure 6e) weights, which have the smallest test cost. The graph for dropping 19 weights shows that the NN controller fails to follow the reference trajectories at this point, as shown in Figure 6f. The convergence plot in Figure 6e shows that the predicted trajectories (ID_REF and IQ_REF) closely follow the reference trajectories (ID and IQ), with minor deviations. This reinforces the quantitative metrics, showcasing the model's robustness in trajectory prediction despite the reduction in weights.

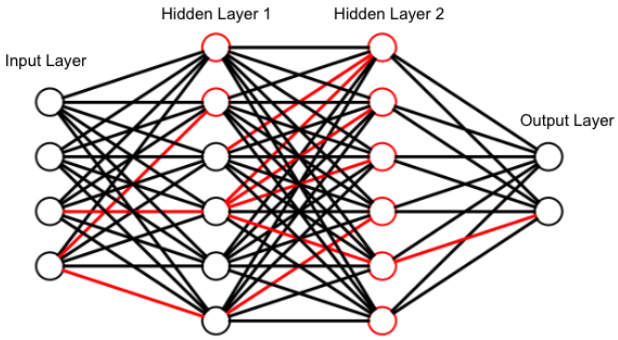


Figure 5: NN Architecture with 18 weights dropped including bias

6.2. Cloud vs Local Environment

When running the code on the local machine which is a MacBook Pro with a 3.1 GHz Dual-Core Intel Core i7 processor, and 16GB memory, and on cloud settings (refer sub-section 5.3), some consistent and promising results were found in our evaluation of our code's performance on the cloud. This section seeks to evaluate these results and provide arguments in favor of using cloud computing in particular situations. It was also observed that the cloud settings, which are devoid of local machine limits like other apps in use or hardware issues, typically offer more constant performance.

In the comparison Table 1, cloud performance is better when compared to local in some instances. However, it is unrelated to the cloud's processing capacity. Cloud computing is more cost-efficient than maintaining the local hardware, particularly in situations where large resources are occasionally required.

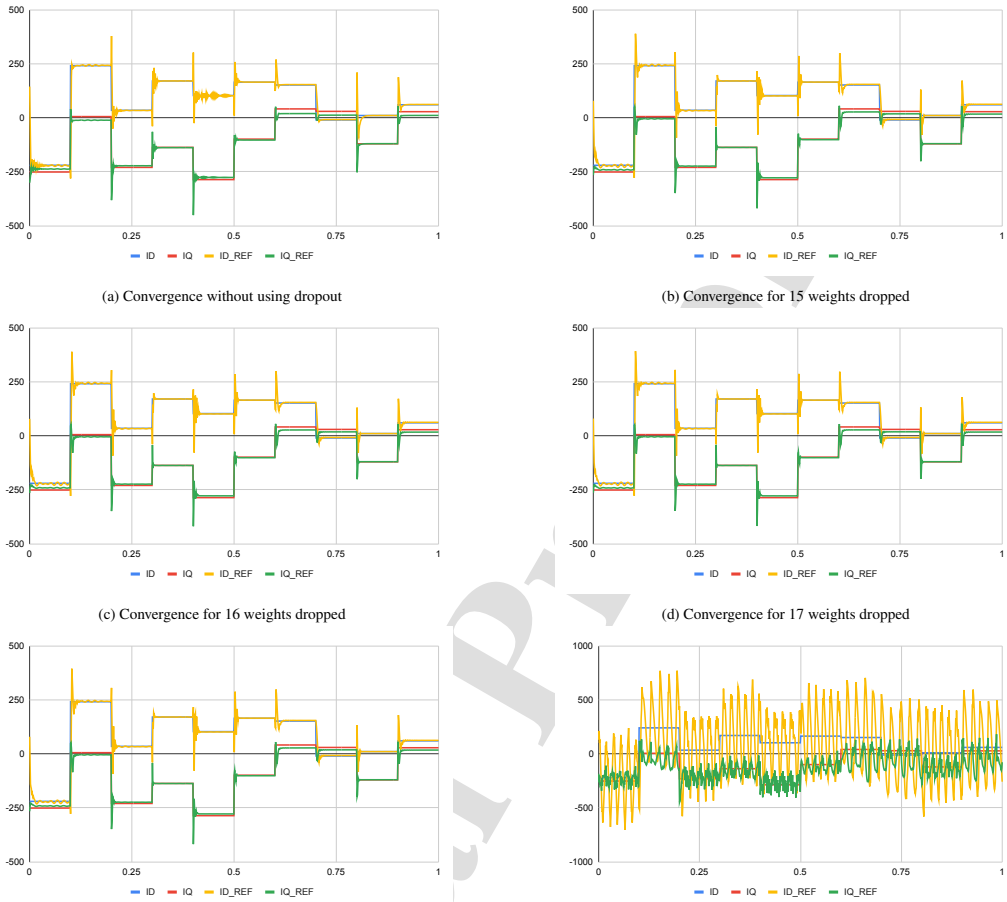
Because cloud data centers have specialized architecture and constant infrastructure updates, which gives access to the newest technology cloud processing is naturally more advantageous for some operations than others. This ensures that the work remains relevant over time. The cloud provides significant benefits in terms of scalability, flexibility, and availability of cutting-edge technologies, making it a good choice for our research.

Table 1: Comparison of Runtime on Cloud with Local

Local (code runtime in sec)	**Cloud (code runtime in sec)**
457.41	155.01
445.53	153.13
457.30	152.34
461.39	153.47
444.23	154.50

6.3. Result Comparison with Adaptive Neuron Dropout Technique

The adaptive neuron dropout technique is implemented to compare the runtime results with the proposed weight dropout technique. Adaptive Dropout is a sophisticated variation of the traditional dropout technique used to prevent overfitting in neural networks. Unlike standard dropout, which drops neurons with a fixed probability, adaptive dropout dynamically adjusts the dropout rates during training based on the importance of each neuron. This adaptability allows the network to retain more critical neurons while reducing less important ones, potentially enhancing model performance and robustness. To compare dynamic adaptability with computational efficiency, adaptive dropout



(a) Convergence without using dropout

(b) Convergence for 15 weights dropped

(c) Convergence for 16 weights dropped

(d) Convergence for 17 weights dropped

(e) Convergence for 18 weights dropped

(f) Convergence for 19 weights dropped

Figure 6: Comparison of convergence for different weights

was chosen to train the proposed our NN controller only on hidden layers. By comparing these two advanced dropout techniques, we can gain insights into their performance across different computational environments.

According to Table 2, Weight Dropout works better in cloud environments, where the resources are optimized. Because of its dynamic nature, adaptive dropout incurs additional computational overhead, leading to longer runtimes in cloud environments. This analysis underscores the importance of selecting the dropout technique based on the specific computational environment to achieve optimal performance. Weight Dropout has a simpler mechanism that allows for faster execution, particularly in cloud setup, making it a preferable choice for resource-intensive applications. Also, with the adaptive dropout, there are one or two neurons are dropped during training, which is consistent with the proposed weight dropout technique. Because each neuron in the first hidden layer of the adopted NN controller has 10 connections with input layer neurons and second hidden layer neurons. Each neuron in the second hidden layer of the adopted NN controller has 8 connections with output layer neurons and first hidden layer neurons. Thus, dropping one or two neurons with different probability and different training in the adaptive dropout technique is similar to dropout

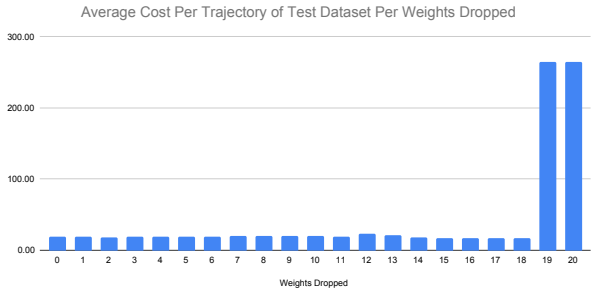


Figure 7: Average test cost per trajectory across weights

Table 2: Comparison of Weight Dropout and Adaptive Neuron Dropout Techniques

Weight Dropout			**Adaptive Neuron Dropout**		
Local (runtime in sec)	**Cloud (runtime in sec)**	**GPU (runtime in sec)**	**Local (runtime in sec)**	**Cloud (runtime in sec)**	**GPU (runtime in sec)**
457.41	155.01	396.75	370.61	248.43	295.85
445.53	153.13	395.64	313.11	247.33	292.08
457.3	152.34	396.41	331.19	247.9	291.62
461.39	153.47	392.55	362.75	250	291.5
444.23	154.5	396.87	347.28	250.44	292.15

11 weights [2] and the improved 18 weights dropout in section 6.1.

7. FPGA Implementation on Reduced Weights NN Structure

FPGAs are semiconductor devices built around configurable logic blocks (CLBs) connected via programmable interconnects. These devices are primarily driven by their adaptability, hardware-timed speed, reliability, and inherent parallel processing capabilities. Unlike conventional processors, FPGAs possess features such as parallelism and pipelining, ensuring that distinct processing tasks can operate without contention for shared resources. Each independent processing task is assigned to a specific chip section, functioning autonomously from other logic blocks. Consequently, adding extra processing tasks does not negatively impact the performance of any individual component within the application.

The hardware analysis detailed in this section extends the work described in [4]. This simulation enhancement was performed on the Intel Cyclone 5CGX board, employing a 32-bit floating-point format. Initialization of the dropped weights was carried out using the floating-point **LPM_CONS**, a component of the IP library in the Quartus Prime EDA tool. The NN with four inputs and two outputs was partitioned into three layers for a streamlined implementation. Each layer's weight matrix was interconnected using the orthogonal bus tool, following a traditional matrix multiplication approach [3] as depicted in Figure 8. With the clock signal integrated through all layers, this setup renders the design more transparent and comprehensible.

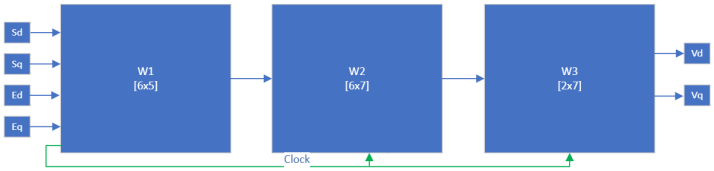


Figure 8: Reduced weight NN FPGA block structure

7.1. Hardware Resource Utilization

The FPGA comes with predefined resources that are available to implement the design; the **A**daptive **L**ogic **M**odule (ALM) in Intel FPGA determines the total area available on the board. Another important block memory resource suitable for wide memory storage applications is the **M**emory 10 **K**ilobits (M10k). These are re-configurable blocks and provide fast memory access within the FPGA fabric. Clock latency is another important aspect influenced by the wire used for making connections. Excessive clock latency necessitates additional on-chip logic or pipeline stages and disturbs the timing requirement by adding additional slack. To avoid this kind of problem during simulation, the design is constructed by not using longer wires, reducing the total clock latency of the circuit, which further reduces the total resource utilization on the chip.

Figure 9 showcases the resources required to implement 18 weight-dropped NN structures on the Cyclone V FPGA (model 5CGXFC9E7F35C8). The design calls for 47,425.5 ALMs. The architecture also demands the usage of 79,511 Adaptive Look-Up Tables (ALUTs), with the number 109 in brackets signifying additional ALUTs employed for design optimization purposes.

The Dedicated Logic Registers are crucial for ensuring synchronized operations within the FPGA, with a count of 58,173, indicating a foundational requirement for the NN's stateful computations. The memory allocation is complemented by 87 M10K blocks, which serve the larger storage demands, such as maintaining the weights and biases vital for the NN's operation. Moreover, the design incorporates 342 DSP blocks, which are fundamental for executing high-speed arithmetic tasks, a common requirement in neural network processing for tasks like matrix multiplication and accumulation.

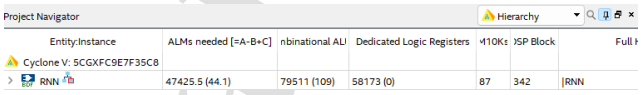


Figure 9: Resource requirement with 18 weights dropped

7.2. Resource Requirement Comparison

The resource utilization data in Table 3 illustrated the efficiency gain achieved by reducing the number of weights in an NN implemented on the Intel FPGA board. Initially, the full-weight neural network configuration necessitated a substantial allocation of hardware resources, consuming 56,752 ALMs, 96,753 combinational Look-Up Tables (LUTs), and a sizable count of 68,887 dedicated logic registers. An approach to selectively drop 11 weights from the network was undertaken to optimize this demand, resulting in a notable decrease in resource consumption. The required ALMs were reduced by approximately 8.5% to 51,896, and the combinational LUTs observed an even more substantial reduction, shrinking by about 8.3% to 88,795. The strategy's effectiveness was further underscored when evaluating the dedicated logic registers, which saw a decrement of approximately 4.8%, lowering the count to 65,593.

Pushing the boundaries of optimization, the decision to drop 18 weights presented an even more striking improvement in resource efficiency. The ALMs required for the network fell sharply to 47,426.5, indicating a significant 16.40% improvement from the original configuration. Combinational LUTs followed suit with a remarkable reduction, now registering at 79,511, thus marking a 17.80% improvement. The dedicated logic registers also reflected this positive trend, decreasing to 58,170, amounting to a 15.55% overall improvement. Interestingly, the reduction

Table 3: Refinement in Resource Requirements

Resource	All weights	Dropping 11 Weights	Dropping 18 Weights	Total Improvement
ALM needed	56752	51896	47426.5	16.40%
Combinational LUT	96753	88795	79511	17.80%
Dedicated logic Registers	68887	65593	58170	15.55%
Block Memory Bits	97653	97061	96310	1%
MK10s	97	93	87	10%

strategy had a relatively muted effect on block memory bits, with a minimal 1% improvement, suggesting that memory resources are less affected by weight pruning. However, the optimization was more pronounced in the usage of MK10s, which decreased to 87 from the initial 97, translating to a 10% improvement.

This validates the efficiency of weight reduction regarding resource utilization but also illustrates a scalable approach to NN design on FPGAs. By adopting such strategies, the design can achieve a more resource-conservative model, potentially enabling the deployment of more complex NN within the same FPGA, thereby enhancing the hardware's computational capacity and application scope.

8. 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. The random initialization is applied to the neural network controller to dropout more weights and ensure the training algorithm is converged. By eliminating 18 weights from the network, the NN controller improved on tracking its reference currents. The proposed training algorithm with the dropout technique is also tested on both AWS cloud and local machines. AWS cloud computing platform provides more speedup compared with running the algorithm locally. At the same time, when implemented in an FPGA, the model with fewer weights incurred substantially less memory occupancy and computations. We also propose an adaptive dropout technique to improve the results from the weight dropout approach. Our proposed dropout techniques will significantly improve the performance of the NN controller by reducing overall weights in the embedded system.

Acknowledgement

This work is supported by the National Science Foundation of the United States under Award CISE-MSI 2131175 and 2131214, and CloudBank [49] which provided essential cloud computing resources for this research. We thank anonymous reviewers for their insightful comments.

References

[1] C. Hingu, X. Fu, R. Challoo, J. Lu, X. Yang, L. Qingge, Accelerating fpga implementation of neural network controllers via 32-bit fixed-point design for real-time control, in: 2023 IEEE 14th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON), IEEE, 2023, pp. 0445–0450.

[2] J. Sturtz, X. Fu, C. D. Hingu, L. Qingge, A novel weight dropout approach to accelerate the neural network controller embedded implementation on fpga for a solar inverter, in: 2023 IEEE International Conference on Smart Computing (SMARTCOMP), 2023, pp. 157–163. doi:10.1109/SMARTCOMP58114.2023.00037.

[3] C. Hingu, X. Fu, S. Smith, T. Saliyu, 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.

[4] X. Fu, S. Li, M. Fairbank, D. C. Wunsch, 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 26 (9) (2014) 1900–1912.

[5] S. Li, D. C. Wunsch, M. Fairbank, 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), 2012, pp. 1–7. doi:10.1109/IJCNN.2012.6252614.

[6] W. Waithaka, X. Fu, A. A. Hadi, R. Challoo, 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), 2021, pp. 1–5. doi:10.1109/PESGM46819.2021.9637840.

[7] R. J. Williams, D. Zipser, A learning algorithm for continually running fully recurrent neural networks, Neural Computation 1 (2) (1989) 270–280. doi:10.1162/neco.1989.1.2.270.

[8] R. J. Williams, D. Zipser, Gradient-based learning algorithms for recurrent networks and their computational complexity, L. Erlbaum Associates Inc., USA, 1995, p. 433–486.

[9] H. Jaeger, A tutorial on training recurrent neural networks , covering bppt , rtrl , ekf and the " echo state network " approach - semantic scholar, in: Proceedings of an unspecified conference, 2005.

[10] S. Li, D. C. Wunsch, E. O'Hair, M. G. Giesselmann, Extended kalman filter training of neural networks on a simd parallel machine, J. Parallel Distributed Comput. 62 (2002) 544–562.

[11] G. Puskorius, L. Feldkamp, Neurocontrol of nonlinear dynamical systems with kalman filter trained recurrent networks, IEEE Transactions on Neural Networks 5 (2) (1994) 279–297. doi:10.1109/72.279191.

[12] J. Pujol, The solution of nonlinear inverse problems and the levenberg-marquardt method, Geophysics 72 (4) (2007) W1–W16.

[13] R. Moradi, R. Berangi, B. Minaei, A survey of regularization strategies for deep models, Artificial Intelligence Review 53 (2020) 3947–3986.

[14] S. Mir, M. Elbuluk, D. Zinger, Pi and fuzzy estimators for tuning the stator resistance in direct torque control of induction machines, IEEE Transactions on Power Electronics 13 (2) (1998) 279–287. doi:10.1109/63.662841.

[15] T. Noguchi, H. Tomiki, S. Kondo, I. Takahashi, Direct power control of pwm converter without power-source voltage sensors, IEEE Transactions on Industry Applications 34 (3) (1998) 473–479. doi:10.1109/28.673716.

[16] D. Zhi, L. Xu, B. W. Williams, Improved direct power control of grid-connected dc/ac converters, IEEE Transactions on Power Electronics 24 (5) (2009) 1280–1292. doi:10.1109/TPEL.2009.2012497.

[17] J. A. Restrepo, J. M. Aller, J. C. Viola, A. Bueno, T. G. Habetler, Optimum space vector computation technique for direct power control, IEEE Transactions on Power Electronics 24 (6) (2009) 1637–1645. doi:10.1109/TPEL.2009.2014953.

[18] J. C. Moreno, J. M. Espi Huerta, R. G. Gil, S. A. Gonzalez, A robust predictive current control for three-phase grid-connected inverters, IEEE Transactions on Industrial Electronics 56 (6) (2009) 1993–2004. doi:10.1109/TIE.2009.2016513.

[19] J. M. Espi Huerta, J. Castello, J. R. Fischer, R. Garcia-Gil, A synchronous reference frame robust predictive current control for three-phase grid-connected inverters, IEEE Transactions on Industrial Electronics 57 (3) (2010) 954–962. doi:10.1109/TIE.2009.2028815.

[20] J. M. Espi, J. Castello, R. García-Gil, G. Garcera, E. Figueres, An adaptive robust predictive current control for three-phase grid-connected inverters, IEEE Transactions on Industrial Electronics 58 (8) (2011) 3537–3546. doi:10.1109/TIE.2010.2089945.

[21] M. Castilla, J. Miret, J. Matas, L. Garcia de Vicuna, J. M. Guerrero, Linear current control scheme with series resonant harmonic compensator for single-phase grid-connected photovoltaic inverters, IEEE Transactions on Industrial Electronics 55 (7) (2008) 2724–2733. doi:10.1109/TIE.2008.920585.

[22] A. G. Yepes, F. D. Freijedo, J. Doval-Gandoy, O. Lopez, J. Malvar, P. Fernandez-Comesaña, On the discrete-time implementation of resonant controllers for active power filters, in: 2009 35th Annual Conference of IEEE Industrial Electronics, 2009, pp. 3686–3691. doi:10.1109/IECON.2009.5415136.

[23] S. Li, T. Haskew, Y.-K. Hong, L. Xu, Direct-current vector control of three-phase grid-connected rectifier–inverter, Electric Power Systems Research 81 (2011) 357–366. doi:10.1016/j.epsr.2010.09.011.

[24] S. Li, X. Fu, M. Ramezani, Y. Sun, H. Won, A novel direct-current vector control technique for single-phase inverter with l, lc and lcl filters, Electric Power Systems Research 125 (2015) 235–244.

[25] D. Banjerdpongchai, J. How, Lmi synthesis of parametric robust hscr; controllers, Vol. 1, 1997, pp. 493 – 498 vol.1. doi:10.1109/ACC.1997.611848.

[26] S. Das, I. Pan, On the mixed H_2/H_∞ loop-shaping tradeoffs in fractional-order control of the avr system, IEEE Transactions on Industrial Informatics 10 (4) (2014) 1982–1991. doi:10.1109/TII.2014.2322812.

[27] L. A. Maccari, J. R. Massing, L. Schuch, C. Rech, H. Pinheiro, V. F. Montagner, R. C. L. F. Oliveira, Robust control for grid connected pwm inverters with lcl filters, in: 2012 10th IEEE/IAS International Conference on Industry Applications, 2012, pp. 1–6. doi:10.1109/INDUSCON.2012.6451389.

[28] B. Chen, Y.-C. Chang, Nonlinear mixed h 2/h control for robust tracking design of robotic systems, International Journal of Control 67 (1997) 837–857. doi:10.1080/002071797223811.

[29] Z. Li, C. Zang, P. Zeng, H. Yu, S. Li, X. Fu, H2/h control for grid feeding converter considering system uncertainty, International Journal of Electronics 104. doi:10.1080/00207217.2016.1244864.

[30] M. T. Hagan, H. B. Demuth, M. Beale, Neural network design, PWS Publishing Co., USA, 1997.

[31] A. Al-Tamimi, M. Abu-Khalaf, F. L. Lewis, Adaptive critic designs for discrete-time zero-sum games with application to h_∞ control, Trans. Sys. Man Cyber. Part B 37 (1) (2007) 240–247.

[32] S. N. Balakrishnan, J. Ding, F. L. Lewis, Issues on stability of adp feedback controllers for dynamical systems, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 38 (4) (2008) 913–917. doi:10.1109/TSMCB.2008.926599.

[33] S. Li, M. Fairbank, C. Johnson, D. C. Wunsch, E. Alonso, J. L. Proao, Artificial neural networks for control of a grid-connected rectifier/inverter under disturbance, dynamic and power converter switching conditions, IEEE Transactions on Neural Networks and Learning Systems 25 (4) (2014) 738–750. doi:10.1109/TNNLS.2013.2280906.

[34] X. Fu, S. Li, Control of single-phase grid-connected converters with lcl filters using recurrent neural network and conventional control methods, IEEE Transactions on Power Electronics 31 (7) (2016) 5354–5364. doi:10.1109/TPEL.2015.2490200.

[35] G. Litjens, T. Kooi, B. Ehteshami Bejnordi, A. Setio, F. Ciompi, M. Ghafoorian, J. van der Laak, B. Ginneken, C. Sánchez, A survey on deep learning in medical image analysis, Medical Image Analysis 42. doi:10.1016/j.media.2017.07.005.

[36] S. Pouyanfar, S. Sadiq, Y. Yan, H. Tian, Y. Tao, M. E. Presa-Reyes, M.-L. Shyu, S. Chen, S. S. Iyengar, A survey on deep learning, ACM Computing Surveys (CSUR) 51 (2018) 1 – 36.

[37] X. Fu, S. Li, I. Jaithwa, Implement optimal vector control for lcl-filter-based grid-connected converters by using recurrent neural networks, IEEE Transactions on Industrial Electronics 62 (7) (2015) 4443–4454. doi:10.1109/TIE.2015.2390140.

[38] F. Ahsan, N. H. Dana, S. K. Sarker, L. Li, S. M. Muyeen, M. F. Ali, Z. Tasneem, M. M. Hasan, S. H. Abhi, M. R. Islam, M. H. Ahamed, M. M. Islam, S. K. Das, M. F. R. Badal, P. Das, Data-driven next-generation smart grid towards sustainable energy evolution: techniques and

technology review, Protection and Control of Modern Power Systems 8 (3) (2023) 1–42.
[39] Y. Nevarez, D. Rotermund, K. R. Pawelzik, A. Garcia-Ortiz, Accelerating spike-by-spike neural networks on fpga with hybrid custom floating-point and logarithmic dot-product approximation, IEEE Access 9 (2021) 80603–80620.
[40] R. Wu, X. Guo, J. Du, J. Li, Accelerating neural network inference on fpga-based platforms—a survey, Electronics 10 (9) (2021) 1025.
[41] Y. Jiang, A. Vaicaitis, J. Dooley, M. Leeser, Efficient neural networks on the edge with fpgas by optimizing an adaptive activation function, Sensors 24 (6).
[42] M. Bhardwaj, S. Choudhury, Digitally controlled solar micro inverter design using c2000 piccolo microcontroller user's guide, Tech. rep., Technical report (2017).
[43] M. Bhardwaj, S. Choudury, Digitally controlled solar micro inverter design using c2000 piccolo microcontroller, Texas Instrument, TX, USA, TIDU405B.
[44] P. J. Werbos, Backpropagation through time: what it does and how to do it, Proceedings of the IEEE 78 (10) (1990) 1550–1560.
[45] X. Fu, S. Li, D. C. Wunsch, E. Alonso, Local stability and convergence analysis of neural network controllers with error integral inputs, IEEE Transactions on Neural Networks and Learning Systems.
[46] G. Strang, Introduction to Linear Algebra, 4th Edition, Wellesley-Cambridge Press, Wellesley, MA, 2009.
[47] C. Sanderson, R. Curtin, Armadillo: a template-based c++ library for linear algebra, Journal of Open Source Software 1 (2) (2016) 26.
[48] S. Conrad, 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.
[49] M. Norman, V. Kellen, S. Smallen, B. DeMeulle, S. Strande, E. Lazowska, N. Alterman, R. Fatland, S. Stone, A. Tan, K. Yelick, E. Van Dusen, J. Mitchell, Cloudbank: Managed services to simplify cloud access for computer science research and education, in: Practice and Experience in Advanced Research Computing, PEARC '21, Association for Computing Machinery, New York, NY, USA, 2021. doi:10.1145/3437359.3465586.

Declaration of interests

☐ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☒ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Letu Qingge reports financial support was provided by National Science Foundation. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.