Optimal Hardware Generation for Neural Network Acceleration

Rohit T P[1][0000−0002−2064−5020], Vaishnav Vimal[1], and Sasi Gopalan[1]

Cochin University of Science and Technology, Kalamassery, Kerala India
20cs078rohi@ug.cusat.ac.in 20cs104vaih@ug.cusat.ac.in
sasigopalan@cusat.ac.in

1 Abstract

The advancement of neural network acceleration has predominantly hinged on software optimizations tailored for general-purpose hardware. This research introduces an innovative methodology to generate custom hardware modules, specifically optimized for executing neural network models. These modules are designed such that they can be directly synthesised without any modification on hardware devices like FPGAs. The approach involves transforming a trained neural network model into an optimized Verilog code, specifically designed to complement the model's structure and trained weights. The process unfolds in three distinct steps: Complexity Reduction, Parallelization, and Implementation/Code Generation.

In the Complexity Reduction phase, a novel approach for model quantization is introduced. Instead of the standard practice of uniform quantization across the network[1][5], each neuron is quantized independently, preserving only the necessary bits of precision. This method involves a specialized extra epoch after training to determine the range of activation at each neuron, which in turn informs the required bit length. The process also identifies and removes 'dead' neurons[5], and merges neurons with approximately the same activations for all inputs, optimizing the hardware footprint and energy efficiency.

The Parallelization step involves a more intricate analysis of the optimized network. The objective is to identify components of the network that can operate in parallel, this enables the efficient utilization of the parallel nature of FPGA-like devices. This is achieved by constructing the dependency graph of the network. The graph is then analyzed, and edges are selectively pruned where parallel processing can be introduced without markedly affecting the network's output. This process results in a series of disjoint graphs, each representing a segment of the network that can be processed concurrently. The identification of these parallel segments is a crucial step, as it directly influences the efficiency and speed of the resulting hardware implementation. Relevant literature on neural network optimization for FPGA implementation [7][8][9] and efficient neural network inference [2][3] provide context and validation for these techniques.

In the final phase, Implementation/Code Generation, the discrete graph elements are translated into corresponding Verilog units. Nodes with two or fewer connections are realized through combinational logic, while more complex nodes are developed as separate Verilog modules. The network graph is then restructured into a unified graph, incorporating special nodes that signify a set of input and output points. The edges of this graph are weighted according to the cumulative signal time delays of each node. A critical aspect of this phase is ensuring uniformity in the edge weights to guarantee consistent signal propagation. In cases of weight discrepancies, additional dummy nodes are introduced to balance the uneven edges. This graph is then used to directly generate synthesisable Verilog code for a given hardware constraint.

The method of directly producing synthesizable verilog code for arbitrary neural networks aims to significantly simplify the hardware design process that is required to create custom accelerator cards. The method proposed can be used in developing accelerator boards that are highly customised to specific use cases or as a low-cost redundancy for systems that require high availability.

Keywords: Neural Network Acceleration · Hardware Generation · Verilog Synthesis · Hardware Design · Parallel Hardware Architecture

References

1. Markus Nagel et al. "A White Paper on Neural Network Quantization." 2021. https://ar5iv.org/abs/2106.08295
2. "A Survey of Quantization Methods for Efficient Neural Network Inference." 2021. https://arxiv.org/abs/2103.13630
3. Olivia Weng. "Neural Network Quantization for Efficient Inference: A Survey." 2021. https://ar5iv.org/abs/2112.06126
4. Pierre-Emmanuel Novac, Ghouthi Boukli Hacene, Alain Pegatoquet, et al. "Quantization and Deployment of Deep Neural Networks on Microcontrollers." Sensors 2021, 21(9), 2984. https://www.mdpi.com/1424-8220/21/9/2984
5. The fine line between dead neurons and sparsity in binarized spiking neural networks https://arxiv.org/abs/2201.11915
6. Hardware-Aware Neural Network Optimization for Efficient On-Device Deep Learning Applications. IEEE Xplore. https://ieeexplore.ieee.org/document/8587663
7. Optimizing Neural Networks for Efficient FPGA Implementation. Springer. https://link.springer.com/article/10.1007/s11831-021-09530-9
8. An Exploration of State-of-the-Art Automation Frameworks for FPGA-Based DNN Acceleration. IEEE Xplore. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=10017269
9. FPGA-based acceleration of neural network training. IEEE Xplore. https://ieeexplore.ieee.org/document/7561676
10. FPGA Hardware Implementation and Optimization for Neural Network based Chaotic System Designhttps://dl.acm.org/doi/abs/10.1145/3241793.3241812