Sound Mixed Fixed-Point Quantization of Neural Networks

CS637: Embedded and Cyber-Physical Systems

Project Presentation

Paper Authors:
DEBASMITA LOHAR, MPI-SWS, Saarland Informatics Campus, Germany
CLOTHILDE JEANGOUDOUX, MPI-SWS, Germany
ANASTASIA VOLKOVA, Nantes Université, France
EVA DARULOVA, Uppsala University, Sweden

Conference: EMSOFT 2023

Presented by:
- Abhijit Dalai (220030)
- Abhishek Kumar (220045)
- Aryan Singh (220229)
- Ayush Srivastava (220272)

Understanding the Title

- **Sound**: In computing, "soundness" implies that the approach is guaranteed to meet certain correctness criteria. Here, it means that the quantization approach can be trusted to stay within a specified error margin, ensuring that any rounding errors from reduced precision do not compromise the accuracy or safety of the neural network's outputs. This is especially important for applications in safety-critical systems, like autonomous vehicles or industrial controls.

- **Mixed**: "Mixed" refers to the *mixed precision* strategy used in the paper. Instead of applying the same level of precision (i.e., the number of bits) across all calculations, this approach assigns different precisions to different parts of the neural network. This helps reduce computation and memory requirements while keeping errors within acceptable limits, optimizing performance for embedded systems.

- **Fixed-Point Quantization**: "Fixed-point quantization" is the core technique used in this research. Fixed-point arithmetic represents numbers with a set number of digits before and after the decimal, making it efficient for hardware without specialized floating-point units. Quantization reduces the precision of neural network calculations to fixed-point representations, saving hardware resources but introducing some rounding error. The paper focuses on finding optimal fixed-point representations for neural networks.

- **Neural Networks**: The study applies this approach specifically to neural networks, which are widely used in machine learning. As neural networks often need high precision for training and execution, quantizing them efficiently without significant performance loss is a challenging task, especially for real-time, embedded, or safety-critical applications.

(a)

(c)

Problem Context

Drones often have limited on-board computational resources, especially those that are lightweight and battery-powered. They need fast, efficient processing to evaluate surroundings and make real-time adjustments to avoid obstacles. A neural network controller can serve as the “brain” for making these real-time decisions. However, using high-precision floating-point calculations would require powerful processors, which are typically too costly in terms of weight, power, and processing cycles for a lightweight drone.

Application of Research

This paper's **sound mixed fixed-point quantization** approach would allow a neural network controller to be efficiently implemented on an FPGA or microcontroller without the need for floating-point processing. By reducing the precision of certain operations in the network—while keeping the error within safe bounds—the neural network can operate faster and use less power.

Impact

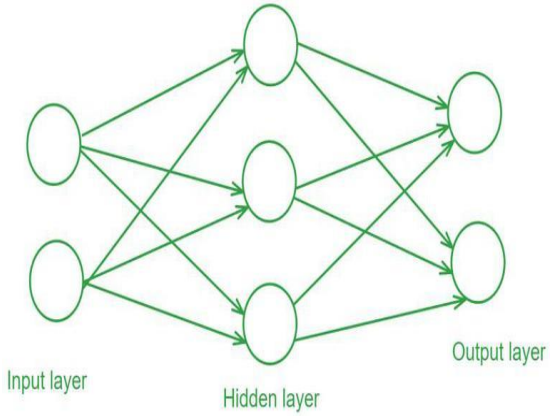
By deploying a quantized neural network with carefully tuned precisions:

- **Speed of Collision Avoidance**: The drone's controller could perform calculations more quickly, reducing response time and enabling safer, more reliable navigation.
- **Battery Efficiency**: Lowering computational intensity would reduce power consumption, increasing flight time.
- **Cost Reduction**: It would allow manufacturers to use less expensive hardware (no floating-point hardware), which could make drone technology more accessible for consumer and industrial applications.

Basics of Neural Networks

- **Definition**: Neural networks (NNs) are computational models inspired by the human brain. They consist of layers of interconnected nodes (neurons) that process input data to produce an output.

- **Structure**: A typical feed-forward neural network includes:
  - **Input Layer**: Receives input data.
  - **Hidden Layers**: Perform computations using weighted connections and activation functions.
  - **Output Layer**: Produces the final output.



- **Activation Functions**:
  - **ReLU (Rectified Linear Unit)**: Commonly used in hidden layers, it outputs the input directly if it is positive; otherwise, it outputs zero. This function helps introduce non-linearity and accelerates convergence during training.
  - **Linear Activation**: Often used in the output layer for regression tasks, it provides continuous output without introducing non-linearity.

Example of Neural Network

- Consider a unicycle model of a car that models the dynamics of a car with 4 parameter variables — the Cartesian coordinates (x, y), the speed, and the steering angle. We use a neural network that was trained as a controller for this model as our initial example. It is a fully connected feed-forward neural network with 1 hidden layer. The network is fed as input a 4-parameter vector, denoted by $\overline{x_0} = [x_0^1\ x_0^2\ x_0^3\ x_0^4]$ where each parameter is constrained by real-valued intervals: $x_0^1 \in [-0.6, 9.55]$, $x_0^2 \in [-4.5, 0.2]$, $x_0^3 \in [-0.6, 2.11]$, and $x_0^4 \in [-0.3, 1.51]$, each specifying valid input values to the network. The inputs are propagated through each layer by successive application of the dot product operations, bias additions and activation functions:

layer 1: $\overline{x_1} = \text{ReLU}\ (W_1 \cdot \overline{x_0} + \overline{b_1})$ output: $\overline{x_2} = \text{Linear}\ (W_2 \cdot \overline{x_1} + \overline{b_2})$

$\overline{x_1}$ is the output of the first layer, which is then fed as input to the next, output layer. The output of the overall network is $\overline{x_2}$. Each layer is parameterized by a weight matrix W_i and a bias vector $\overline{b_i}$:

$$W_1 = \begin{bmatrix} -0.037 & \cdots & 0.129 \\ -0.003 & \cdots & 0.099 \\ -0.128 & \cdots & 0.047 \\ 0.045 & \cdots & -0.166 \end{bmatrix}, \quad \overline{b_1} = [0.028 \quad \cdots \quad 0.342], \quad W_2 = \begin{bmatrix} 0.052 & 0.137 \\ \cdots & \cdots \\ -0.200 & 0.154 \end{bmatrix}, \quad \overline{b_2} = [0.273 \quad 0.304]$$

Example of Neural Network

- While the network has only a single hidden layer, it has 500 neurons (elided above) and thus results in a non-trivial size of the overall network.

- To be deployed in a safety-critical system, such a neural network controller needs to be proven safe before implementation, for instance, one may need to prove that the system reaches a set of safe states within a given time window. Since exact reasoning about finite-precision arithmetic does not scale, existing verification techniques assume real-valued parameters and arithmetic operations for the network, but can typically deal with bounded uncertainties, from the implementation or measurements. We will thus assume that a bound on the output error $\in$ is given.

- As a controller is primarily executed on resource-constrained hardware, using floating-point arithmetic may be overly expensive, as it requires either additional floating-point processor support or slow software emulations. The alternative is to **quantize** the NN controller in **fixed-point arithmetic**.

Fixed-Point Quantization

- **Definition:** Fixed-point quantization involves representing variables and constants using integers with an implicit format $\langle Q, \pi \rangle$.
  - **Total Word Length Q** ($\in \mathbb{N}$): Overall number of bits, including the sign bit.
  - **Fractional Part π** ($\in \mathbb{N}$): The position of the binary point counting from the least significant bit.

- **Bit Allocation:**
  - **Integer Part I :** $I = Q - \pi - 1$, which holds the range $[-2^I, 2^I]$**.**
  - ***Fractional Part* π:** Larger π results in finer precision with a maximum roundoff error of $2^{-\pi}$.

- **Efficient Implementation:**
  - Utilizes integer operations and bit-shifting, suitable for embedded systems, especially FPGAs.
  - Supports efficient, runtime truncation rounding mode.

- **Overflow and Safety:**
  - Ensuring enough integer bits to avoid overflow is critical.
  - Overflow freedom and precision management are necessary to maintain the overall accuracy.

Challenges in Fixed-Point Quantization

- **Precision Trade-offs:**
  - Uniform precision for all operations may result in unnecessary resource usage or inadequate precision.
- **Roundoff Errors:**
  - Errors from fixed-point arithmetic accumulate and can affect the overall output.
- **Manual Precision Assignment:**
  - Manually balancing precision for integer and fractional parts is complex and error-prone.

Proposed Solution

- **MILP-based mixed-precision quantization approach:** We encode fixed-point precision tuning as a mixed integer linear programming (MILP) problem, and perform several over-approximations to generate a linearized problem from the fundamentally nonlinear constraints. Our MILP constraints optimize the number of fractional bits such that overflow-freedom is ensured even in the presence of errors and a cost function is minimized. Our approach is parametric in the cost function to be optimized.

- Advantages of **Mixed Precision**:
  - **Resource Optimization:** It can be more resource - efficient to assign different precisions (word lengths) to different operations to achieve a target error bound, and thus implement the model in mixed precision.

  - **Precision Tuning:** Existing sound techniques applicable to fixed-point arithmetic rely on a heuristic search that repeatedly evaluates roundoff errors for different precision assignments.

- While this technique works well for small programs, the repeated global roundoff error analysis quickly becomes expensive.

MILP-based Mixed-Precision Tuning

- In MILP, some decision variables are constrained to be integers, and other variables can be real-valued. Integers allow us to directly encode the discrete decisions about how many bits to use for operations, and real-valued constraints effectively express error constraints.

- The primary constraint ensures that the total roundoff error stays within the target error bound, while also preventing overflow. Encoding both conditions together introduces non-linearity due to dot product multiplications, which MILP solvers cannot efficiently handle. To address this, we use soundly over-approximations and linearize constraints, making the solution efficient and feasible.

- **Source of Non-Linearity – Range Computation:**
  - **Source:** The computation of ranges for all arithmetic operations to prevent overflow introduces non-linearity.

  - **Solution:** Pre-compute a sound over-approximation of real-valued ranges using interval arithmetic. From these ranges, compute the integer bits needed and encode a linear range constraint in the MILP formulation to ensure no overflow, even with roundoff errors.

MILP-based Mixed-Precision Tuning

- **Source of Non-Linearity – Fractional Bit Optimization:**
  - **Source:** Optimizing fractional bits for all variables and constants to maintain the error bound is non-linear.
  - **Solution:** Treat the **dot product** as a single operation and encode the assignment of fractional bits only for the results of dot products. Use existing techniques to assign precisions to all intermediate variables and constants to ensure roundoff errors in intermediate computations remain within bounds.

- With a linear cost function, the precision tuning problem for ReLU and linear activation functions can be encoded with linear constraints, solvable by MILP solvers. Other activations, like sigmoid, require linearization before encoding. For performance, uniform bit lengths are used within each layer, reducing constraints but not limiting the overall approach.

$$\begin{bmatrix} x_1^1 \\ x_1^2 \end{bmatrix} = ReLU\left(\underbrace{\begin{bmatrix} 0.1 & 0.2 \\ 0.2 & 0.15 \end{bmatrix}}_{W_1} \cdot \begin{bmatrix} x_0^1 \\ x_0^2 \end{bmatrix} + \underbrace{\begin{bmatrix} 1.0 \\ 2.0 \end{bmatrix}}_{\overline{b_1}} \right), \quad x_2 = Linear\left(\underbrace{\begin{bmatrix} 0.1 & 0.2 \end{bmatrix}}_{W_2} \cdot \begin{bmatrix} x_1^1 \\ x_1^2 \end{bmatrix} + \underbrace{\begin{bmatrix} 0.5 \end{bmatrix}}_{\overline{b_2}} \right) \qquad (2)$$

Fig. 1. Running example for MILP modeling.

Dataset Selection

- We used a dataset of weights from a pre-trained neural network model built to simulate a single pendulum(which is tested in the research paper itself). Originally implemented in Scala, this model contained multiple layers with weights in 32-bit floating-point precision.

- This dataset was quantized to reduce bit-widths for each layer based on its effect on accuracy, aiming to maintain model performance while optimizing memory usage.

- Our main objective with this dataset was to achieve efficient memory utilization by assigning an optimal bit-width to each layer. This optimization process would allow us to retain the model's accuracy within an acceptable range while using less memory overall.

MILP formulation with Pyomo

- Our methodology consisted of three primary components: mixed-precision quantization, MILP formulation with Pyomo, and solving the MILP with the CBC solver.

- To implement this, we defined an optimization model using Pyomo, where each layer's bit-width was represented as a variable constrained to integer values ranging from 5 to 32 bits.

- The objective function was set to minimize the total number of fractional bits across all layers to reduce memory usage. In addition, constraints were established to ensure only one bit-width was selected per layer and that quantization error per layer remained within a set tolerance, thus preserving accuracy

MILP formulation with Pyomo

Objective and Model Setup:

- Our aim is to optimize fractional bit widths for neural network weight quantization to minimize storage and maintain accuracy within a specified error limit, ***error_bound***(acceptable maximum quantization error per layer).

- Using the Pyomo library, it sets up an optimization model where each layer's bit width is an integer variable (***frac_bits***), bounded by the variables ***min_frac_bits*** and ***max_frac_bits***(the range of bits available for fractional representation.

MILP formulation with Pyomo

- **Constraints and Optimization Variables:**
- **Bit Choice Constraints**: Binary variables (select_bits) represent each possible fractional bit width for each layer, ensuring only one bit width is active per layer.

- **Quantization Error Constraints**: Quantization error is approximated as ½^(*frac_bits).* The quant_errors variable is constrained to remain within error_bound, linking it to select_bits.

- **Objective**: The objective is to minimize the sum of frac_bits across all layers, reducing memory requirements while meeting the error bound.

Solving MILP with CBC solver

- The model is solved using the cbc solver to find optimal bit widths (***optimized_bits***). Then, each weight matrix is quantized based on its fractional bit width, scaling by 2^{frac_bits}, rounding, and rescaling. The function returns both ***optimized_bits*** and ***quantized_weights***.

- The **CBC (Coin-or branch and cut) solver** is an open-source solver, compatible with Pyomo, for **Mixed-Integer Linear Programming (MILP)** problems

- CBC provides an efficient method for solving large MILP problems by combining branch-and-bound and cutting plane techniques to explore possible solutions and find the optimal result.

Optimization Problem: Fractional Bit Quantization

- **Goal**: Minimize total fractional bit widths while maintaining acceptable quantization error.

Objective Function:

- min Σ ***frac_bits***[i]
- N: Number of layers (weight matrices).

Constraints:

1. Bit Selection:

- Σ ***select_bits***[i, b] = 1
- (Only one bit width selected per layer).

2. Bit Width Linkage:

- ***frac_bits***[i] = Σ b * select_bits[i, b]
- (Assign fractional bit width based on selected bit).

Quantization Error Management

Quantization Error Upper Bound:

$$\textit{quant\_errors}[i] \leq 1 / (2^b), \forall\ b \in \textit{possible\_bits}$$

Quantization Error Enforcement:

$$\textit{quant\_errors}[i] \leq \Sigma\ (1 / (2^b)) * \textit{select\_bits}[i, b]$$

Key Point:

Ensures quantization error for each layer is bounded by error_bound, linked to the selected fractional bit width.

Weight Quantization Process

1. Scaling:

- ***scaled_weight*** = w * 2^***frac_bits***
- (Shift fractional part to integer domain).

2. Rounding:

- ***rounded_weight*** = round(***scaled_weight***)
- (Quantize to nearest integer).

3. Rescaling:

- ***quantized_weight*** = ***rounded_weight*** / 2^***frac_bits***
- (Restore original scale with reduced precision).

- Quantization Error:
- error ≈ 1 / (2^***frac_bits***)

Results

- **Mean Absolute Error (MAE)**: The MAE between the original (Scala) weights and quantized (C++) weights was calculated as **0.0321**, indicating minor deviation.
- **Memory Reduction**:
  - Achieved significant memory savings by using lower bit-widths for certain layers.
  - Example: Layers with low-impact on accuracy were assigned smaller bit-widths (e.g., 8 or 10 bits), while more accuracy-sensitive layers retained higher bit-widths (e.g., 16 bits).
- **Accuracy Impact**:
  - Minimal change in model predictions after quantization, suggesting successful optimization with low impact on accuracy.
  - Quantization allowed the model to retain similar performance to the original floating-point version while using less memory.

Steps in Our Approach

- Overall, our technique consists of **three steps**:
  1. computing the integer bits of program variables using interval arithmetic
  2. optimizing the fractional bits of dot, bias, and activation operations by reducing it to MILP
  3. computing the precision of all constants and intermediate variables in dot products

- Our proposed approach can also be extended to other types of neural networks that have sparse matrices and activation functions that can be piece-wise linearized

Limitations of the MILP based Mixed Fixed-point quantization

- **Real-Time Multimodal Sensor Fusion Controller**
  - This benchmark involves a neural network controller for an autonomous robotic system that integrates data from multiple real-time sensors (e.g., LIDAR, camera, IMU) to navigate complex environments. The system needs to process inputs from sensors with varying update rates and data types (e.g., high-frequency IMU readings vs. high-resolution, lower-frequency camera frames). The fusion algorithm for these multimodal data streams is designed as a large, multi-layer neural network that combines recurrent and convolutional layers, designed to produce high-frequency outputs for real-time navigation and obstacle avoidance.

Limitations of the MILP based Mixed Fixed-point quantization

- **Specific Characteristics to Stress Test the Approach**
  - **Complex Network Architecture with Diverse Layers**:
    - The network structure combines multiple layer types, such as convolutional layers (common in image processing) and recurrent layers (e.g., LSTM, GRU for sequence data from IMU or LIDAR). These layers have complex data dependencies and varying precision needs, especially with accumulating error across time steps in recurrent units.
    - The MILP-based approach in the paper primarily targets feed-forward networks with dot products and bias additions, so it may not support or efficiently handle the mixed layer requirements without significant adjustments.
  - **Large, Dynamic Range of Input Data**:
    - Sensor data may have widely varying numeric ranges; for instance, LIDAR values might be on the scale of meters, while IMU values are on the scale of angular velocity, which could cause the integer bit estimation (using interval arithmetic) to be overly conservative.
    - This broad range could lead to an unmanageably high number of integer bits, increasing total bit width requirements, which could hinder the fixed-point optimization or even make it infeasible for practical deployment.

Limitations of the MILP based Mixed Fixed-point quantization

- **Specific Characteristics to Stress Test the Approach**
  - **Strict Real-Time Constraints**:
    - Since the network output directly affects navigation and collision avoidance, it has stringent timing constraints, requiring low-latency execution. Any excessive resource allocation or computational delay due to MILP optimizations could lead to deadline misses.
    - The MILP approach might struggle to balance precision and computational efficiency in this setting, especially as it aims to reduce bit width without directly minimizing latency.

Limitations of the MILP based Mixed Fixed-point quantization

- **Expected Limitations and Challenges**
  - **MILP Scalability**: The MILP problem becomes increasingly complex with large networks and diverse layer types, potentially exceeding the solver's ability to find feasible solutions within reasonable timeframes, especially for networks with recurrent structures.
  - **Overflow and Error Bound Constraints**: Ensuring error bounds and preventing overflow in the face of wide-ranging data inputs and recurrent data propagation might require overly conservative bit-width assignments, reducing quantization efficiency.
  - **Difficulty with Real-Time Processing Constraints**: The MILP-based quantization may yield solutions with minimal bit-widths but not necessarily the lowest latency, posing a challenge for real-time applications.

Applications in Embedded and CPS

- **Automotive Systems**
  - **Adaptive Cruise Control (ACC)**: In ACC, neural networks help maintain a safe distance between vehicles by controlling acceleration and braking. Optimized quantization of the neural network controller reduces power and resource consumption, allowing these systems to run on embedded microcontrollers without dedicated floating-point units.
  - **Autonomous Driving and Collision Avoidance**: Neural networks are widely used for real-time object detection, lane-keeping, and emergency braking. Quantized networks can process sensor data with minimal delay and at lower power costs, which is crucial for electric vehicles and low-power edge devices in autonomous driving systems.
- **Aerospace Systems**
  - **Aircraft Collision Avoidance Systems (e.g., ACAS X)**: Quantized neural networks can help process radar and sensor data in real-time to avoid collisions, particularly in drone swarms and autonomous aircraft. Fixed-point quantization enables real-time operation on embedded hardware, reducing the need for high-power processing units and making these systems more suitable for compact avionics.
  - **Flight Control Systems**: Quantized neural networks are used in flight controllers for UAVs (unmanned aerial vehicles) and manned aircraft. These controllers must run continuously on constrained hardware, and optimizing their quantization enhances reliability and reduces the computational burden, ensuring smoother and safer flight operations.

Applications in Embedded and CPS

- **Healthcare Devices**
  - **Implantable and Wearable Medical Devices:** Quantized neural networks are increasingly used in wearables and implantables, such as ECG monitors and insulin pumps, to analyze real-time patient data and make decisions based on learned patterns. Lower bit precision and reduced power requirements allow these devices to function for extended periods on small batteries, ensuring patient safety with minimal maintenance.
  - **Autonomous Driving and Collision Avoidance**: Neural networks are widely used for real-time object detection, lane-keeping, and emergency braking. Quantized networks can process sensor data with minimal delay and at lower power costs, which is crucial for electric vehicles and low-power edge devices in autonomous driving systems.
- **Industrial and Robotics Systems**
  - **Robotic Motion Control**: Embedded neural networks control robotic actuators, helping maintain balance, optimize movement, and respond to real-time sensor data. Fixed-point quantization reduces latency and allows these controllers to run on microcontrollers and DSPs, improving reaction times and energy efficiency in robots.
  - **Predictive Maintenance**: Quantized neural networks embedded in industrial sensors can analyse vibration, temperature, and sound data to predict equipment failures. Optimized quantized networks allow processing directly on low-power hardware, reducing the need for constant data streaming and extending sensor battery life.

Applications in Embedded and CPS

- **Energy and Environmental Monitoring**
  - **Smart Grid Control:** Quantized neural networks monitor power usage and optimize energy distribution in smart grids. By reducing computational overhead, quantized networks make it feasible to deploy energy-efficient controllers at each grid node, supporting real-time adjustments to energy distribution.
  - **Environmental Sensors:** Quantized models deployed in sensors monitor environmental factors like air quality, water contamination, and weather changes. These sensors operate on minimal power, extending deployment time in remote or hard-to-reach locations.

Benefits of Optimized Quantized Neural Networks in Embedded and CPS

- **Reduced Power Consumption**: Fixed-point arithmetic requires significantly less power than floating-point, making quantized networks ideal for battery-powered devices in safety-critical CPS. Lower power demands are crucial for extending the operation of devices in remote and energy-limited applications, such as healthcare wearables and environmental sensors.
- **Faster Execution and Reduced Latency**: Quantized networks have a smaller memory footprint and lower computational requirements, leading to faster processing times. This is essential for real-time applications, such as collision avoidance in autonomous vehicles, where rapid data processing directly impacts safety.
- **Lower Memory and Storage Requirements**: By minimizing the bit-length used for computations, quantized networks reduce memory usage, making them suitable for embedded systems with limited RAM and storage. This enables the deployment of complex neural networks in compact devices without the need for additional hardware resources.
- **Enhanced Reliability and Predictability**: In safety-critical applications, precision must be predictable, as uncontrolled rounding errors can compromise safety. The MILP-based quantization approach ensures sound, bounded error propagation, enhancing system reliability and predictability—a vital feature for CPS applications in automotive and aerospace.
- **Cost-Effective Hardware Utilization**: Quantized neural networks can be implemented on affordable hardware (such as microcontrollers and FPGAs) without specialized floating-point units. This reduces the overall cost of deploying intelligent CPS solutions, making them more accessible for a range of industries, including healthcare and energy.

Key contributions and findings

- **First Sound, Automated Mixed-Precision Quantization**: The paper introduces the first fully automated, sound approach to mixed fixed-point quantization for neural networks, specifically designed for feed-forward networks in embedded systems.

- **Optimization Using MILP**: By formulating the precision tuning problem as a **Mixed-Integer Linear Programming (MILP)** problem, the approach leverages MILP's capabilities to optimize bit allocation across network layers systematically, ensuring that specified error bounds are respected while minimizing resource usage.

- **Real-world Applicability**: The approach(Mixed Precision Quantisation) been evaluated on neural network controllers used in safety-critical applications, such as adaptive cruise control and aircraft collision avoidance. These benchmarks underline the practical effectiveness and scalability of the approach, particularly in systems where precise control and limited resources are critical.

Advantages of the Proposed MILP-Based Approach

- **Soundness and Safety Guarantees**: The approach is designed to produce quantized implementations with guaranteed bounds on roundoff errors, a critical requirement in safety-critical applications. This soundness is achieved by formulating constraints in MILP that over-approximate errors conservatively.
- **Resource Efficiency**: By minimizing the number of bits allocated to each operation within the network, the approach reduces memory and computational requirements. This efficiency translates to faster execution on hardware and more compact neural network implementations, suitable for resource-constrained devices.
- **Automated Precision Tuning**: Aster automates the complex task of assigning optimal bit lengths to each layer and operation, reducing the time and manual effort involved in precision tuning. The MILP formulation systematically explores the space of precision assignments, making it more robust and faster than heuristic approaches, especially for larger networks.
- **Scalability and Flexibility**: The approach adapts to different neural network architectures and can be extended to other types of activation functions or cost functions, making it versatile for a wide range of applications in embedded control systems.
- **Hardware Compatibility**: The generated quantized code from Aster can be directly compiled to hardware (e.g., FPGAs using Xilinx's HLS compiler), demonstrating the approach's readiness for deployment on embedded hardware platforms. This compatibility is essential for real-world applications requiring low-power, high-efficiency hardware implementations.

Future Direction

- Extending Support to Additional Neural Network Types
  - **Recurrent Neural Networks (RNNs)**:
    - Adapting quantization for RNNs could expand applications to time-series analysis and control tasks involving sequential data.
    - Challenge: Handling sequential dependencies in the quantization process, with issues around error propagation and state retention.
  - **Convolutional Neural Networks (CNNs)**:
    - Supporting CNNs would benefit image and video processing tasks.
    - Challenge: Adapting MILP-based optimization to handle convolutional layers, maintaining low precision while quantifying and managing errors.
  - **Activation Functions Beyond ReLU and Linear**:
    - Many networks use nonlinear functions like sigmoid and tanh.
    - Challenge: These functions aren't linear, so achieving accurate fixed-point quantization requires linear approximations or new quantization techniques to ensure implementation fidelity.

Future Direction

- Improving Scalability
  - **Advanced MILP Solvers:**
    - The MILP formulation for quantization can be computationally heavy for large networks.
    - Future work could explore efficient MILP solvers or hybrid optimization methods, combining MILP with gradient-based or heuristic techniques to improve handling of larger networks.
  - **Layer-wise or Block-wise Quantization**:
    - Instead of quantizing the entire network at once, a hierarchical or layer-wise approach could break down the quantization into smaller subproblems, reducing optimization complexity.
  - **Approximate or Relaxed Constraints**:
    - Loosening certain constraints, such as error bounds in intermediate layers where precision may be less critical, could speed up the optimization process while still meeting overall accuracy requirements.

Future Direction

- Enhanced Precision Tuning Techniques
  - **Adaptive Precision Adjustment**:
    - Developing dynamic techniques to adjust precision based on real-time requirements or hardware conditions can improve efficiency.
    - This would be valuable in scenarios where high precision is only needed occasionally or for specific inputs.
  - **Cost-sensitive Precision Optimization**:
    - Extending the cost function to include hardware-specific factors like energy use or memory could optimize networks for different devices, such as low-power or edge devices.
  - **Error-propagation Modeling**:
    - Finer-grained error propagation models within networks would improve precision assignment accuracy, especially for complex architectures or networks with many layers.

Future Direction

- Stochastic/Probabilistic Approaches
  - **Probabilistic Bounds on Errors**:
    - Instead of strict error bounds, probabilistic approaches could provide error bounds with confidence levels, reducing precision requirements in some layers for more efficient implementations.
  - **Monte Carlo Methods for Precision Tuning**:
    - Monte Carlo simulations of multiple precision configurations could help refine precision assignments, especially for configurations that may be challenging to optimize traditionally.

Future Direction

- Tooling and Usability Enhancements
  - **Automatic Parameter Selection**:
    - Tuning parameters such as input precision and error bounds manually can be challenging.
    - Automated parameter tuning or recommendations could simplify this process, adapting based on network size, layer types, or specific applications.
  - **Support for Hardware-Specific Optimizations**:
    - Integrating quantization tools with hardware synthesis tools, like FPGA or TPU, would enable optimizations tailored to specific hardware architectures, benefiting platforms like low-power microcontrollers.

Work done beyond the Research paper

- The Research paper implemented the Aster tool for quantization, which, we were unable to run on the system because of unavailability of older versions of softwares. So, we rather solved the MILP problem(needed for mixed precision quantization approach) using a new technique, pyomo and cbc solver(python library) to get similar results(since the main technique,MILP formulation, was same). We referred to articles, python documentation to learn about this.

- Researched about Limitations of Mixed point quantization approach (based on MILP),Applications of Embedded & CPS, Future directions of the research

References

- Github repository: link
- Neural Network Quantization and Verification
  - [21] K. Esmailzadeh et al., *"SeeDot: A Compiler for Stable and Efficient ML Inference"*
- Formal Verification of Neural Networks
  - [27] X. Huang et al., *"Safety Verification of DNNs"*
  - [31] G. Katz et al., *"The Marabou Framework for DNN Verification"*
  - [49] H. Xu et al., *"DeepPoly: Polynomial Approximation of DNNs with ReLU"*

References

- Mixed Integer Linear Programming (MILP) Optimization
  - *https://william-lindsay.medium.com/what-is-mixed-integer-linear-programming-e2470b74750*
  - ***https://medium.com/@chanon.krittapholchai/optimization-with-pyomo-in-python-88d4a9674826***
  - *Pyomo-cbc documentation*
  - **[10]** M. Lam et al., *"Mixed Precision in ML: Optimization and Performance"*
- Fixed-Point Arithmetic and Quantization Techniques
  - **[6]** P. De Dinechin et al., *"Fixed-Point Arithmetic for FPGA Hardware"*
  - **[13]** G. de Dinechin et al., *"Faithful Rounding of Fixed-Point Sums"*

References

- Benchmarking and Applications in Control Systems
  - [34] T. Dreossi et al., *"ARCH Competition on Verification of Neural Network Controllers"*
  - [22] A. Bak et al., *"VNN-LIB: Standard Benchmark Suite"*
- Embedded System Synthesis and Compilation
  - **[52]** Xilinx Inc., *"Vivado High-Level Synthesis User Guide"*