Neural Network Quantization Bit Optimization via Multi-Objective Evolutionary Algorithms

Gabriele Santini, gabriele.santini@studenti.unitn.it

***Abstract*—Advancements in neural network technology are often driven by increasing model size, enhancing capabilities but also escalating computational demands. Neural network quantization, particularly reducing the precision of weights and activations, is a crucial technique to mitigate these demands. While effective in reducing memory footprint and computational load, quantization introduces noise, potentially degrading accuracy. This study explores optimizing bit-width assignment across neural networks using multi-objective evolutionary algorithms, evaluating both layer-wise and neuron-wise approaches. The Non-dominated Sorting Genetic Algorithm II (NSGA-II) was employed to balance model size and accuracy trade-offs, demonstrating significant efficiency in handling complex optimization landscapes.**

I. Introduction

Advancements in neural network technology often involve a significant increase in model parameters. This trend shows no signs of stopping, especially since increased model size seems to unlock enhanced capabilities. While larger models offer advantages, their training, deployment, and use require significant resources. Reducing the computational cost of using these models is crucial for integrating them into devices with limited power and processing capabilities.

Neural network quantization involves reducing the precision of weights and activations from floating-point to lower-bit representations, like 8-bit integers. It's one of the most effective ways to decrease the model's memory footprint and computational requirements. However, while it achieves these savings, the additional noise introduced can lead to a degradation in accuracy.

Assigning different bit widths to different layers or neurons in a neural network can be advantageous because different parts of the network carry different types of information. High precision can be used for critical areas where accuracy is sensitive to changes, while less important areas can use lower precision, saving memory and processing power without significantly affecting overall accuracy.

This project employs multi-objective evolutionary algorithms to explore bit-width optimization for neural network quantization, comparing the effects at both neuron and layer levels.

II. Problem Statement

A. Quantization

Quantization is a process that reduces the precision of the parameters in a machine learning model, aiming to decrease model size and computational requirements while trying maintaining acceptable performance. Uniform affine quantization is the most commonly used quantization scheme because it permits efficient implementation of fixed-point arithmetic, consisting in a linear mapping from floating-point values to a limited range of integers. This process involves two key parameters for the conversion: scale and zero point.

The formulas for quantization and dequantization for a uniform affine scheme are as follows:

$$\mathbf{x}_{\text{int}} = \text{clamp}\left(\left\lfloor \frac{\mathbf{x}}{s} \right\rceil + z; 0, 2^b - 1\right)$$

and

$$\hat{\mathbf{x}} \approx \mathbf{x}_b = s\left(\mathbf{x}_{\text{int}} - z\right)$$

where $\mathbf{x}$ represents the original floating-point value, $\mathbf{x}_{\text{int}}$ is the quantized integer value, s is the scaling factor, and z is the integer value that maps to zero in the quantized space.

Choosing the quantization grid limits (q_{min} and q_{max}) affects two types of errors, the clipping error and the rounding error. The grid limits are defined as:

$$q_{\text{min}} = -sz \text{ and } q_{\text{max}} = s(2^b - 1 - z)$$

Any x values outside this range will be clipped to these limits, causing a clipping error. To reduce this clipping error, the quantization range can be increased by raising the scale factor. However, this also increases the rounding error, which will lie between $-\frac{1}{2}s$ and $\frac{1}{2}s$. Scale and zero point value need to be choosen for each weight to minimize the sum of these two errors.

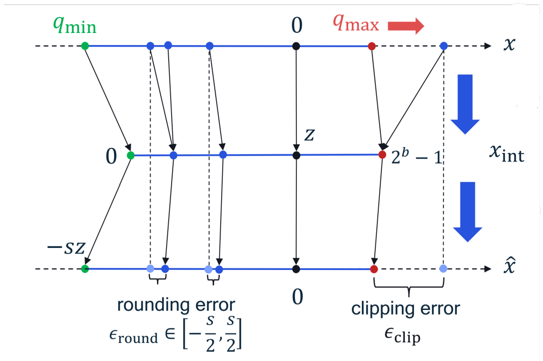

Fig. 1. Quantization errors

In post-training quantization, the quantization approach used in this project, the scale (s) and zero point (z) are calculated using the following steps:

1) Determine the range of the floating-point values: Identify the minimum (x_{min}) and maximum (x_{max}) values in

the tensor that needs to be quantized. This is done by feeding data through the network and observing the values that each tensor takes.

2) Compute the scale: The scale is calculated as the ratio of the range of the floating-point values to the range of the quantized values. For a given bit-width b, the quantized integer values range from 0 to $2^b - 1$. The scale is given by:

$$\text{scale} = s = \frac{x_{\text{max}} - x_{\text{min}}}{2^b - 1}$$

3) Compute the zero point: The zero point aligns the floating-point zero with an integer value within the quantized range. It is calculated as:

$$\text{zero\_point} = z = \text{clamp}\left(\text{round}\left(-\frac{x_{\text{min}}}{\text{scale}}\right); 0, 2^{b-1}\right)$$

To test the performances of the quantized network on a general purpose hardware the most efficient procedure is to do quantization simulation. The network weights are quantized and dequantized, introducing in this way the quantization errors. The computation is then performed using floating-point operations. This simulation makes the quantization operation easier to implement and allows testing the network on GPU.

B. Evolutionary Optimization

In this project, I implemented an evolutionary algorithm to perform multi-objective optimization to optimize the number of quantization bits allocated per layer and per neuron of a simple neural network. Evolutionary algorithms are particularly suited for this task because they can effectively navigate complex multi-objective optimization landscapes. Each bit configuration was evaluated based on two objectives: the total model size and the test error rate. The goal was to find a set of Pareto-optimal solutions, representing the best possible trade-offs between the two competing objectives. The idea behind this optimization process is to identify the parts of the network that contribute less to accuracy and quantize them more than the rest.

III. Methodologies

A. Neural network

I implemented a neural network using a multi-layer perceptron (MLP). This MLP was trained for image classification task on the MNIST digits dataset. For simplicity, the network architecture consisted of two layers: the first layer had 100 neurons, and the second layer had 10 neurons, corresponding to the 10 classes in the MNIST dataset.

Quantizing a neural network from float32 to int8 can often maintain performance levels. However, aggressive quantization, where the bit depth is reduced further, can significantly degrade accuracy. As illustrated in 2, this simple network exhibits robustness to quantization down to 4 bits, but lower bit depths result in a substantial performance drop.

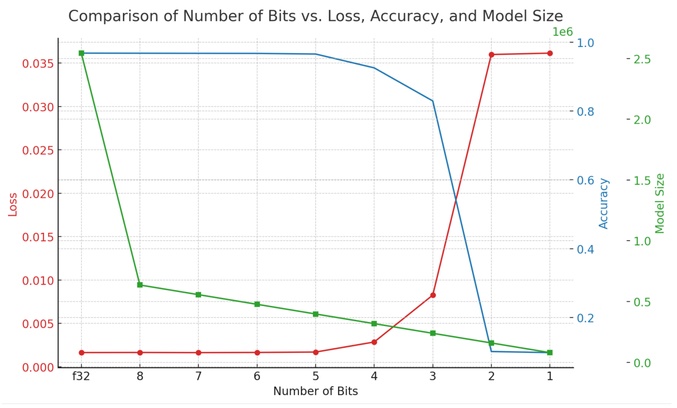

Fig. 2. Number of bits over model size, average test loss and test accuracy

B. NSGA-II

The Non-dominated Sorting Genetic Algorithm II (NSGA-II) is a widely used evolutionary algorithm for multi-objective optimization, aimed at finding a diverse set of Pareto-optimal solutions that balance conflicting objectives. To implement NSGA-II for bit optimization i followed these steps:

1) Initialization: A random initial population P_0 of size N is generated. The variable N represents either the number of layers or the number of neurons.

2) Evaluation: Each individual is evaluated by testing the neural network, which has been quantized with the corresponding number of bits, on the test set. Model size and error rate are computed.

3) Fast Non-dominated Sorting (Appendix A): The population is sorted into non-dominated fronts. The first front contains individuals not dominated by any others, the second front consists of individuals dominated only by those in the first front, and so on. This sorting identifies Pareto dominance relationships among individuals.

4) Crowding Distance Calculation (Appendix B): A crowding distance is assigned to each individual within each front. This distance measures how close an individual is to its neighbors, preserving diversity by favoring those in less crowded regions. It is computed by summing normalized distances between an individual and its neighbors for both objectives.

5) Selection: A binary tournament selection creates a mating pool. Pairs of individuals are chosen, and the one with the better rank (or higher crowding distance if tied) is selected. The chosen tournament size is 5.

6) Crossover and Mutation: To produce an offspring population Q_t of size N crossover and mutation are applied to the selected mating pool. I used blend crossover and Gaussian mutation with a mutation rate of 0.3.

7) Combine and Sort: The parent population P_t and offspring population Q_t are combined to form R_t of size $2N$. Fast non-dominated sorting is applied to R_t, classifying individuals into fronts.

8) Selection for the Next Generation: The next generation population P_{t+1} is formed by selecting individuals from the sorted fronts until the population size N is reached. If the last included front exceeds the required population size, individuals from this front are selected based on their crowding distance, favoring those with higher distances.

9) Termination: - Steps 3 to 8 are repeated for a predefined number of generations or until convergence criteria are met. I run the algorithm for 20 generations.

NSGA-II's strength lies in its ability to balance convergence to the true Pareto front with maintaining diversity among solutions, making it effective for complex multi-objective optimization problems. For detailed algorithms on Fast Non-dominated Sorting and Crowding Distance Calculation, refer to Appendix A and Appendix B respectively.

Algorithm 1 NSGA-II

```
1:  $R_t = P_t \cup Q_t$
2:  $\mathcal{F} =$ fast-non-dominated-sort($R_t$)
3:  $P_{t+1} = \emptyset$ and $i = 1$
4:  while $|P_{t+1}| + |\mathcal{F}_i| \leq N$ do
5:      crowding-distance-assignment($\mathcal{F}_i$)
6:      $P_{t+1} = P_{t+1} \cup \mathcal{F}_i$
7:      $i = i + 1$
8:  end while
9:  Sort($\mathcal{F}_i, \prec_n$)
10: $P_{t+1} = P_{t+1} \cup \mathcal{F}_i[1 : (N - |P_{t+1}|)]$
11: $Q_{t+1} =$ make-new-pop($P_{t+1}$)
12: $t = t + 1$
```

IV. Experimental Results

To evaluate the results I plotted the Pareto fronts of both approaches' initial and final populations. The Pareto fronts, representing optimal trade-offs between objectives, revealed how well the algorithms converged towards solutions with superior trade-offs.

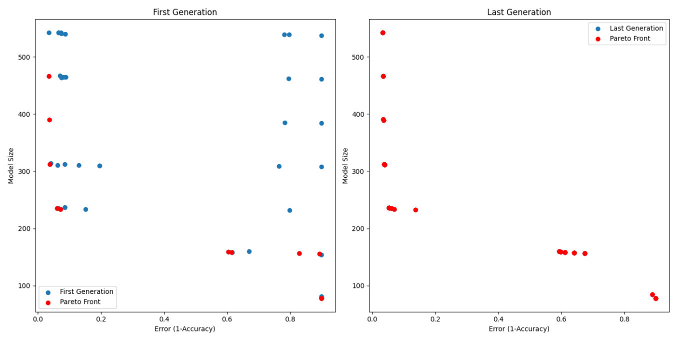

Fig. 3. Pareto Front Layer

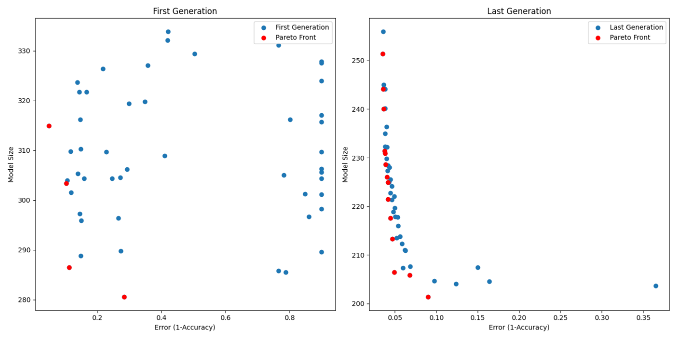

Fig. 4. Pareto Front Neuron

The layerwise approach demonstrates that the NSGA algorithm can find the optimal Pareto front. However, this approach is overkill because the search space is so small (composed of only 512 possible individuals) that it is more efficient to try all possible combinations and plot the pareto front. This hypothesis is confirmed by the fact that the optimal solutions are already present in the initial population of 20 random individuals.

Regarding the neuronwise approach, which involves many more parameters, it appears to be more suitable for optimization with NSGA. This approach shows excellent performance, but when compared to the layerwise method, it becomes evident that the algorithm has explored a much smaller space. However, it effectively explores this space, as indicated by its convergence and the noncrowded distribution of the final Pareto front. To further improve performance, it would be beneficial to use a larger population, increase the mutation rate, and raise the number of generations. Unfortunately, increasing the population size and the number of generations significantly raises the execution time of the algorithm. In Appendix C, with the image 8, I show that even by generating more random individuals, the population always remains in the same search space.

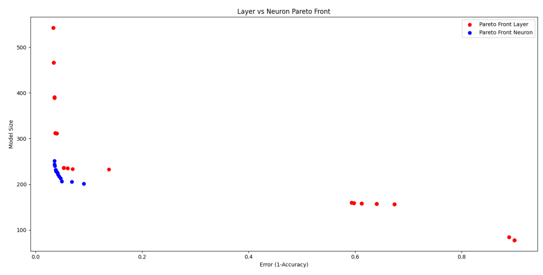

Fig. 5. Neuron vs Layer final Pareto front

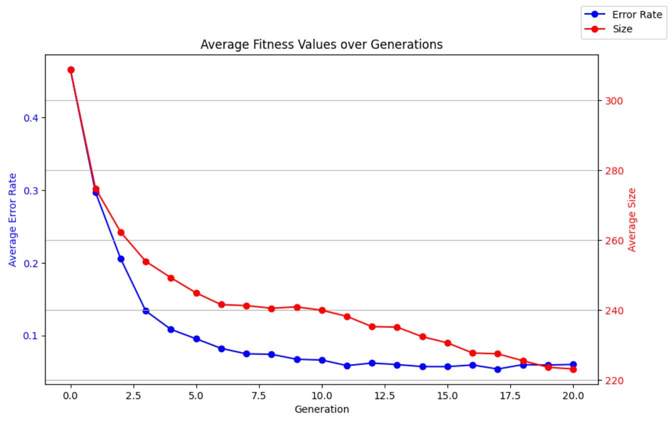

Fig. 6. Neuron fitness improvement over generations

For the neuron-wise approach, I plotted the distribution of the initial and final number of bits per layer. Contrary to my expectations, the optimization algorithm favored solutions with a large number of bits in the input. I thought that a few bits would suffice for the input since the MNIST dataset is easily binarizable while maintaining information. The first layer, which has the greatest impact on the model's weight since has 10 times the parameters of the second layer, exhibits a relatively low bit-per-neuron distribution.

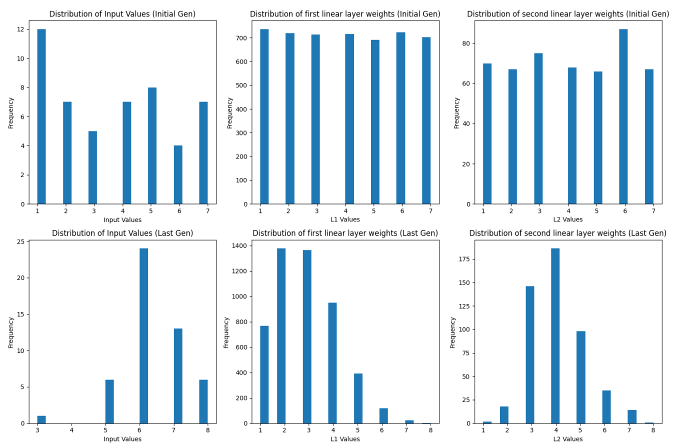

Fig. 7. Num of bits distribution in initial and final neuron population

V. Conclusion

During the development of the project, I had the opportunity to apply various skills and techniques. Even though this project was only a proof of concept, there are promising results that warrant further exploration in the field.

It is particularly interesting to experiment with larger and more complex networks. Therefore, I focused on writing the code in a way that ensures it is easily scalable to other models. By doing so, future expansions and adaptations can be more efficiently implemented, paving the way for broader and more in-depth research in this area.

Appendix A

Algorithm 2 fast-non-dominated-sort(P)

```
1:  for each $p \in P$ do
2:      $S_p = \emptyset$
3:      $n_p = 0$
4:      for each $q \in P$ do
5:          if $p \prec q$ then
6:              $S_p = S_p \cup \{q\}$
7:          else if $q \prec p$ then
8:              $n_p = n_p + 1$
9:          end if
10:     end for
11:     if $n_p = 0$ then
12:         $p_{\text{rank}} = 1$
13:         $\mathcal{F}_1 = \mathcal{F}_1 \cup \{p\}$
14:     end if
15: end for
16: $i = 1$
17: while $\mathcal{F}_i \neq \emptyset$ do
18:     $Q = \emptyset$
19:     for each $p \in \mathcal{F}_i$ do
20:         for each $q \in S_p$ do
21:             $n_q = n_q - 1$
22:             if $n_q = 0$ then
23:                 $q_{\text{rank}} = i + 1$
24:                 $Q = Q \cup \{q\}$
25:             end if
26:         end for
27:     end for
28:     $i = i + 1$
29:     $\mathcal{F}_i = Q$
30: end while
```

Appendix B

Algorithm 3 crowding-distance-assignment(I)

```
1:  $l = |I|$
2:  for each $i$, set $I[i]_{\text{distance}} = 0$ do
3:      for each objective $m$ do
4:          $I = \text{sort}(I, m)$
5:          $I[1]_{\text{distance}} = I[l]_{\text{distance}} = \infty$
6:          for $i = 2$ to $(l - 1)$ do
7:              $I[i]_{\text{distance}} = I[i]_{\text{distance}} + \frac{I[i+1].m - I[i-1].m}{f_m^{\max} - f_m^{\min}}$
8:          end for
9:      end for
10: end for
```

Appendix C

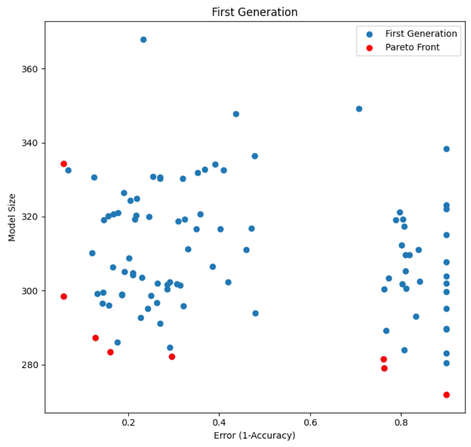

Fig. 8. Distribution of 100 random neuron individuals

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

[2] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A White Paper on Neural Network Quantization. Qualcomm AI Research, San Diego, CA, USA, 2020.

[3] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, ”A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002