Instead, by monitoring the largest and smallest values in the calculations (each tensor of the neural network) for some typical data that will be passed through the network, we use a simple linear relationship $y = kx + m$ to map the floating-point values to integers (the quantization operation, $q()$):

$$\text{integer value} = \text{round} \left( \frac{\text{float value}}{\text{scale}} \right) + \text{zero point}$$ (1)

By finding proper values for the scale and zero-point parameters in the above relationship we can make sure that the integer ranges are respected [-128, 127] in the case of int8, and no overflow will occur. Note that we are free to choose these values, e.g., it is legitimate to keep the minimum value fixed (-128), while changing the scale to something else, and vice versa (keep max as 127 and change the scale), or keep the zero-point and set a new scale. This observation is the foundation of the approach outlined here. Using relation (1) above with proper scale and zero-point values yields a much smaller quantization error as shown in Fig. 3 below.

The transformed values are used in the internal calculations, but in the end, we usually map the values back to physical values that make sense to us, which we do by using the inverse mapping (or the dequantize operation, $d()$)

$$\text{float value} = (\text{integer value} - \text{zero point}) \times \text{scale}$$ (2)

Except for transforming the data to integers, we also need to implement integer versions of all the operations in the neural network. In essence, this means that we rewrite the floating-point operations by using the mappings (1) and (2) above, following the idea in [2]. For a simple multiplication between the floating-point values $a$ and $b$, we get

$$\text{mulop} = a \times b = (a_i - z_a) \times (b_i - z_b) \times \text{scale}_{\text{mul}} = (a_i - z_a) \times (b_i - z_b) \times \text{scale}_{\text{mul}}$$ (3)

where, $\text{scale}_{\text{mul}} = \text{scale}_{\text{scale}} \times \text{scale}_{\text{scale}}$.

Now, applying (2) to get the integer result of this operation we have,

$$\text{mulop}_{\text{int8}} = \text{round}(\text{mulop} \times \text{scale}_{\text{scale}}) \times \text{scale}_{\text{mul}}$$ (4)

For full integer quantization all values in the above expression shall be integers, so that they can efficiently be computed and stored on non-FPU MCUs. Using fixed-point multiplications or shifting for the division these operations can be efficiently handled.

**The Nonlinear Integer Optimization Problem**

The activation functions in the Imagimob solution are implemented as Lookup tables (“LUTs”) to avoid the floating-point operations. Using all this, we can run the calculations as integer-only – all we need to do is to quantize the input data and dequantize the output data, shown in Fig. 4 below.

As noted earlier, we are free to choose scale and zero-point values, but we need to select them carefully to keep the quantization error low. As an example, we should choose scale values so that the error after the rounding operation in expression (4) becomes small, namely so that the scale in the numerator and denominator are equal. We can use this as a requirement or "contract" in our implementation:

$$\text{scale}_{\text{int8}} = \text{scale}_{\text{int8}}$$

where we mean that the scales shall be made equal. We can achieve this by using the observation made earlier that we can set one of the scales, e.g., setting $\text{scale}_{\text{int8}}$, to $\text{scale}_{\text{int8}}$, and then adjusting the minimum and maximum values for the integer accordingly. Often, however, the different scale and zero-point values from one operation appear in the next operation and become coupled. Noting down all relationships in the compute graph results in a large system of equations with integer solution. Often the system is overdetermined which leads to an optimization problem. The round-operator is often present in the equations, which makes the problem nonlinear.

**Imagimob’s Solution**

Difficult optimization problems can in computer science sometimes be solved using a heuristic approach, which means that with the help of the computer we try to find an approximate solution to a difficult problem. These solutions do not claim that they find the optimal solution to the problem, but the solution can often be good enough to work in practice.

**References**
