CS 433 Parallel and Distributed Programming

By: Wangwenqian Yangguanrou

January 2, 2023

I. FUNDAMENTALS OF FIXED-POINT QUANTIFICATION

A. Post-training quantization

Post-training quantization (PTQ) algorithms take a pre-trained FP32 network and convert it directly into a fixed-point network without the need for the original training pipeline. These methods can be data-free or may require a small calibration set, which is often readily available.In most cases, PTQ is sufficient for achieving 8-bit quantization with close to floating-point accuracy.[5]

B. Uniform quantization

Uniform quantization is the most commonly used quantization scheme because it permits efficient implementation of fixed-point arithmetic.

1. *Asymmetric quantization*

The scale factor s and the zero-point z are used to to map a floating point value to the integer grid, whose size depends on the bit-width b. The scale factor is commonly represented as a floating-point number and specifies the step-size of the quantizer. The zero-point is an integer that ensures that real zero is quantized without error. Let$[\beta, \alpha]$ be the range of real values chosen for quantization.

Starting from a real-valued vector x we first map it to the unsigned integer grid $\{0, ..., 2^b - 1\}$[6]:

$$x_{int} = clamp(\lfloor \frac{x}{s} \rceil + z; 0, 2^b - 1) \tag{1}$$

$$s = \frac{\alpha - \beta}{2^b - 1} \tag{2}$$

$$z = -round(\frac{\beta}{s}) - 2^{b-1} \tag{3}$$

To approximate the real-valued input x, a de-quantization step is performed:

$$x \approx \hat{x} = s(x_{int} - z) \tag{4}$$

Combining the two steps above we can provide a general definition for the quantization function, q(·), as:

$$\hat{x} = q(x; s, z, b) = s \left[clamp(\lfloor \frac{x}{s} \rceil + z; 0, 2^b - 1) - z \right] \tag{5}$$

2. *Symmetric quantization*

Symmetric quantization is a simplified version of the general asymmetric case. The symmetric quantizer restricts the zero-point to 0.

$$x \approx \hat{x} = sx_{int} \tag{6}$$

$$x_{int} = clamp(\lfloor \frac{x}{s} \rceil; 0, 2^b - 1) \text{for unsigned integers} \tag{7}$$

$$x_{int} = clamp(\lfloor \frac{x}{s} \rceil; -2^{b-1}, 2^{b-1} - 1) \text{for signed integers} \tag{8}$$

C. Batch normalization folding

For on-device inference, batch normalization operations are folded into the previous or next linear layers in a step called batch normalization folding. This removes the batch normalization operations entirely from the network, as the calculations are absorbed into an adjacent linear layer. Besides reducing the computational overhead of the additional scaling and offset, this prevents extra data movement and the quantization of the layer's output.

Assumed the parameter of convolution(linear) layer is w_c, b_c, the running variance, running mean, weight and bias of the batchnorm layer is rv, rm, w, b. The relation of input x and output y of conv(linear)+bn modules is:

$$y = w \cdot \frac{w_c \cdot x + b_c - rm}{\sqrt{rv + 10^{-8}}} + b$$

. Then the parameter of convolution(linear) layer after folding is:

$$w_c \leftarrow \frac{w \cdot w_c}{\sqrt{rv + 10^{-8}}}$$

$$b_c \leftarrow \frac{w \cdot (b_c - rm)}{\sqrt{rv + 10^{-8}}} + b$$

D. Quantization granularity

There are several choices for sharing quantization parameters among tensor elements. We refer to this choice as quantization granularity.

- per-tensor quantization: a single set of quantization parameters per tensor, one for the weights and one for activations.
- per-channel quantization: a separate quantizer for individual segments of a tensor (e.g., output channels of a weight tensor).

E. Learning Quantization Ranges

Quantization range setting refers to the method of determining clipping thresholds of the quantization grid, q_{min} and q_{max}. The key trade-off in range setting is between clipping and rounding error and their impact on the final task loss for each quantizer being configured. Quantization ranges are treated differently for weight quantization and activation quantization[3]:

- For weights, the basic idea is simply to set a := min w, b := max w.
- For activations, ranges depend on the inputs to the network. To estimate the ranges, we collect [a; b] ranges seen on activations during training and then aggregate them via exponential moving averages (EMA) with the smoothing parameter being close to 1 so that observed ranges are smoothed across thousands of training steps.

There are also other quantization range setting methods like Mean squared error (MSE), Cross entropy, BN based range setting which are more difficult to implement but may have better results.

F. Bias Correction

There's an inherent bias in the mean and variance when doing common min max quantization on weight values. Use W_c to represent weight values per channel, W_c^q to represent weight values after quantization per channel. The compensate constant is as follows.

$$\epsilon_c = \frac{||W_c - E(W_c)||_2}{||W_c^q - E(W_c^q)||_2}$$

Therefore, the weight values after quantization can be scale to submit to the mean and variace of weight values before quantization.

$$w \leftarrow \epsilon_c(w - E(W_c^q)) + E(W_c)$$

II. PTQ QUANTIZATION METHOD

A. ACIQ

ACIQ is a method optimizing the quantization mean-square-error between quantified values and original values. It assumes that the tensor values satisfied laplace or guassian distribution and shows that clipping the value with some α bound optimizes the noise.

1. **Basic ACIQ Clipping Value**

Assumed that the distribution of tensor values $f(x)$ submits to Laplace distribution $L(\mu, b)$, the tensor values are clipped into range $[\mu - \alpha, \mu + \alpha]$ and quantized into M bits. ACIQ aims to optimize mean-square-error of quantization: $E[(X - Q(X))^2]$. It supposes $\mu = 0$ in the following steps, because it does not lose generality since this mean can always be subtracted and added.

In the model, the quantization step is

$$\delta = \frac{2\alpha}{2^M} \tag{9}$$

. For index $i \in [0, 2^M - 1]$, it assumes that all values fall in $[-\alpha + i\delta, -\alpha + (i+1)\delta]$ are rounded to the mid point q_i in the bin. The MSE to optimize is as follows.

$$E[(X - Q(X))^2] = \int_{-\infty}^{-\alpha} f(x)(x+\alpha)^2 dx + \sum_{i=0}^{2^M-1} \int_{-\alpha+i\delta}^{-\alpha+(i+1)\delta} f(x)(x - q_i)^2 dx + \int_{\alpha}^{\infty} f(x)(x-\alpha)^2 dx \tag{10}$$

The MSE can be rewritten in $E[(X - Q(X))^2] = QuantizationNoise + ClippingNoise$, where

$$QuantizationNoise = \sum_{i=0}^{2^M-1} \int_{-\alpha+i\delta}^{-\alpha+(i+1)\delta} f(x)(x - q_i)^2 dx \tag{11}$$

$$ClippingNoise = \int_{-\infty}^{-\alpha} f(x)(x+\alpha)^2 dx + \int_{\alpha}^{\infty} f(x)(x-\alpha)^2 dx \tag{12}$$

Assumed that the tensor values satisfy laplace distribution $f(x) = \frac{1}{2b} e^{-\frac{|x|}{b}}$. Then the clipping noise can be calculated directly:

$$\begin{aligned} ClippingNoise =& 2\int_{\alpha}^{\infty} f(x)(x-\alpha)^2 dx \\ =& e^{-\frac{x}{b}}[2\alpha b - 2b^2 - \alpha^2 - x^2 - 2(b-\alpha)x]\Big|_{\alpha}^{\infty} \\ =& 2b^2 \cdot e^{-\frac{\alpha}{b}} \end{aligned}$$

The quantization noise is approximated by Piece-Wise Linear function. Since the quantization step δ is very small and the laplace distribution function $f(x)$ is smooth, $f(x)$ can be approximated by a linear function $g(x) = f(q_i) + f'(q_i)(x - q_i), x \in [-\alpha + i\delta, -\alpha + (i+1)\delta]$. Then the quantization noise can be calculated as follows:

$$
\begin{aligned}
QuantizationNoise &\approx \sum_{i=0}^{2^M-1} \int_{-\alpha+i\delta}^{-\alpha+(i+1)\delta} g(x)(x-q_i)^2 dx \\
&= \sum_{i=0}^{2^M-1} \int_{-\alpha+i\delta}^{-\alpha+(i+1)\delta} f(q_i)(x-q_i)^2 dx + \sum_{i=0}^{2^M-1} \int_{-\alpha+i\delta}^{-\alpha+(i+1)\delta} f'(q_i)(x-q_i)^3 dx \\
&= \sum_{i=0}^{2^M-1} \frac{1}{3} f(q_i)(x-q_i)^3 \Big|_{-\alpha+i\delta}^{-\alpha+(i+1)\delta} + \frac{1}{4} f'(q_i)(x-q_i)^4 \Big|_{-\alpha+i\delta}^{-\alpha+(i+1)\delta} \\
&= \sum_{i=0}^{2^M-1} \frac{1}{3} f(q_i)(x-q_i)^3 \Big|_{-\alpha+i\delta}^{-\alpha+(i+1)\delta} \\
&= \frac{2}{3} \cdot (\frac{\delta}{2})^3 \sum_{i=0}^{2^M-1} f(q_i) = \frac{(2\alpha)^3}{12 \cdot 2^{3M}} \sum_{i=0}^{2^M-1} f(q_i) = \frac{2 \cdot \alpha^3}{3 \cdot 2^{3M}} \sum_{i=0}^{2^M-1} f(q_i) \\
&\approx \frac{2 \cdot \alpha^3}{3 \cdot 2^{3M}} \sum_{i=0}^{2^M-1} \frac{1}{2\alpha} = \frac{\alpha^2}{3 \cdot 2^{2M}}
\end{aligned}
$$

The second equation holds because q_i is the mid point of $[-\alpha + i\delta, -\alpha + (i+1)\delta]$. The sencond $\approx$ holds because the smooth distribution $f(x)$ on $[-\alpha, \alpha]$ can be simulated by a uniform distribution $f(x) \approx \frac{1}{2\alpha}$. The authors also plot the simulation figure to prove this approximation reasonable.

Therefore,

$$
E[(X - Q(X))^2] \approx 2b^2 \cdot e^{-\frac{\alpha}{b}} + \frac{\alpha^2}{3 \cdot 2^{2M}} \tag{13}
$$

The optimal of $E[(X - Q(X))^2]$ is reached when

$$
\frac{dE[(X - Q(X))^2]}{d\alpha} = -2b \cdot e^{-\frac{\alpha}{b}} + \frac{2\alpha}{3 \cdot 2^{2M}} = 0
$$

. Solving this equation given bit width M, the clipping values $\alpha = c(M) \cdot b$, in which parameter $c(M)$ is as follows.

M	4	5	6	7	8	9
$c(M)$	5.03	6.2	7.41	8.64	9.89	11.16

TABLE I: ACIQ $\alpha = c(M) \cdot b$ Mapping Table

2. **Fused ReLU**

Assumed the distribution of tensor values satisfy $Laplace(0, b)$.(ProV A) For those activation right before a ReLU function, they can be clipped to a smaller dynamic range $[0, \alpha]$, which results in more precise quantization. The authors optimize the MSE on the positive half as follows.

$$
E[(X - Q(X))^2] = \sum_{i=0}^{2^M-1} \int_{i\delta}^{(i+1)\delta} f(x)(x-q_i)^2 dx + \int_{\alpha}^{\infty} f(x)(x-a)^2 dx \approx \frac{\alpha^2}{24 \cdot 2^{2M}} + b^2 \cdot e^{-\frac{\alpha}{b}} \tag{14}
$$

We explain eq.14 further. In eq.14, the clipping noise is $\frac{1}{2}$ of the clipping noise above(II A). As for quantization noise, the quantization step here is $\delta = \frac{\alpha}{2^M}$, so the quantization noise is

$$\frac{2}{3} \cdot (\frac{\delta}{2})^3 \sum_{i=0}^{2^M-1} f(q_i) = \frac{\alpha^3}{12 \cdot 2^{3M}} \sum_{i=0}^{2^M-1} f(q_i) \approx \frac{\alpha^2}{24 \cdot 2^{2M}}$$

Therefore, for Fused ReLU, the optimal of $E[(X - Q(X))^2]$ is reached when

$$\frac{dE[(X - Q(X))^2]}{d\alpha} = -b \cdot e^{-\frac{\alpha}{b}} + \frac{\alpha}{12 \cdot 2^{2M}} = -b \cdot e^{-\frac{\alpha}{b}} + \frac{\alpha}{3 \cdot 2^{2(M+1)}} = 0$$

.

Solving this equation given bit width M, the clipping values $\alpha_F = c_F(M) \cdot b$, in which parameter $c_F(M) = c(M+1)$.

M	3	4	5	6	7	8
$c_F M$	5.03	6.2	7.41	8.64	9.89	11.16

TABLE II: ACIQ $\alpha_F = c_F(M) \cdot b$ Mapping Table

3. Notice about Bias Correction Problem I F

The bias correction used in codes provided by ACIQ([1]) is $w \leftarrow \epsilon_c(w - E(W_c^q)) + E(W_c)$, while the bias correction formula in paper is $w \leftarrow \epsilon_c(w + E(W_c) - E(W_c^q))$. We think the one used in codes is right, the one in paper is wrong.

B. AdaRound

1.**Motivation:** "Rounding-to-nearest", the predominant approach for all neural network weight quantization, is not optimal for post-training quantization.[4]

2.**Method:** AdaRound is a better weight-rounding mechanism for post-training quantization that adapts to the data and the task loss. The aim is to minimize the performance loss incurred due to quantization. Assuming per-layer weight quantization, the quantized weight $\hat{w}_i^{(l)}$ is

$$\hat{w}_i^{(l)} \in \left\{ w_i^{(l),floor}, w_i^{(l),ceil} \right\}, \tag{15}$$

where

$$w_i^{(l),floor} = s^{(l)} \cdot clip\left(\lfloor \frac{w_i^{(l)}}{s^{(l)}} \rfloor, n, p \right) \tag{16}$$

and $w_i^{(l),ceil}$ is similarly defined by replacing $\lfloor \cdot \rfloor$ with $\lceil \cdot \rceil$ $\Delta w_i^{(l)} = w^{(l)} - \hat{w}_i^{(l)}$ denotes the perturbation due to quantization.

Finding the optimal rounding procedure can be formulated as the following binary optimization problem

$$\underset{\Delta w}{argmin}\, \mathbb{E}\left[\mathcal{L}(x, y, w + \Delta w) - \mathcal{L}(x, y, w) \right] \tag{17}$$

Through some mathematical derivation, approximation, assumptions, relaxations, we use the following asymmetric reconstruction formulation

$$\underset{V}{argmin} \|f_a(Wx) - f_a(\widetilde{W}\hat{x})\|_F^2 + \lambda f_{reg}(V), \tag{18}$$

where $\hat{x}$ is the layer's input with all preceding layers quantized, f_a is the activation function, $\|\cdot\|_F^2$ denotes the Frobenius norm and $\widetilde{W}$ are the soft-quantized weights that we optimize over

$$\widetilde{W} = s \cdot clip\left(\lfloor \frac{W}{s} \rfloor + h(V), n, p\right) \tag{19}$$

$V_{i,j}$ is the continuous variable that we optimize over and $h(V_{i,j})$ can be any differentiable function that takes values between 0 and 1, i.e., $h(V_{i,j}) \in [0, 1]$. The additional term $f_{reg}(V)$ is a differentiable regularizer that is introduced to encourage the optimization variables $h(V_{i,j})$ to converge towards either 0 or 1, i.e., at convergence $h(V_{i,j}) \in \{0, 1\}$. AdaRound employs a rectified sigmoid as $h(V_{i,j})$. The rectified sigmoid is defined as

$$h(V_{i,j}) = clip(\sigma(V_{i,j})(\zeta - \gamma) + \gamma, 0, 1), \tag{20}$$

where $\sigma(\cdot)$ is the sigmoid function and, ζ and γ are stretch parameters, fixed to 1.1 and -0.1, respectively. For regularization we use

$$f_{reg}(V) = \sum_{i,j} 1 - |2h(V_{i,j}) - 1|^{\beta}, \tag{21}$$

where we anneal the parameter β. This allows most of the $h(V_{i,j})$ to adapt freely in the initial phase (higher β) to improve the MSE and encourages it to converge to 0 or 1 in the later phase of the optimization (lower β), to arrive at the binary solution that we are interested in.

III. EXPERIMENTS

A. ACIQ

1. **Methods:** We refer to codes provided by [1] in our codes. The overall implementation flow is as follows:

- Load data and pretrained model.
- Replacing original Conv2d, Linear, ReLU, Avg/MaxPool and Batchnorm(FrozenBatchnorm) layers with corresponding quantization layers: Conv2dWithId, LinearWithId, ReLUWithId, Avg/MaxPoolWithId and BatchnormWithId(FrozenBatchnormWithId). These "WithId" layers inherit from the corresponding layers. They first forward the inputs as the super class do. Then they use aciq related quantizer to quantify the output, prepare for the next layers.
- Mark ReLU, Absorb BatchNorm and Quantify layers' weights: 1) In order to improve the quantization precision, we mark the conv2d and batchnorm layers right before the ReLU layers, so that we can choose the $c_F(M)$ but not $c(M)$ to calculate α and the activation(output) can be quantified on $[0, \alpha]$.

  2) If the batch normalization layer is right next to the convolution layer or linear layer, batch normalization is absorbed in the weights of the convolution layer or linear layer.

  3) Quantify the weights of Conv2d and Linear Layers with common minmax quantizer. After quantization, do bias correction as I F.
- Implement ACIQ range clipping:

  1) Calculating the parameter b assuming the input tensor values submit to laplace distribution $f(x) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}}$. b is calculating by $b = \sum_i |x_i - \mu|$.

  2) Check the alpha mapping table to calculate α with parameter b. If the activation is "before_ReLU", use $c_F(M)$ instead of $c(M)$.

  3) Clip tensor value to $[\mu - \alpha, \mu + \alpha]$. If the activation is "before_ReLU", clip tensor value to $[0, max(\mu, 0) + \alpha]$.

- Simulate quantization:

  1) For all the weight values, using naive $[min, max]$ as clipping range.

  2) For all the activation values, using ACIQ to clip range.

- Evaluation: Obtain the accuracy results of the quantization model on the validation dataset.

- Notice

  – We implement FrozenBatchnormWithId according to BatchnormWithId in source codes of ACIQ[1].

  – We also modify the mark ReLU function for deeplabv3 and retinanet.

2. **Experiment setup**

- Set bit width=8 for both activations and weights.

- We use common quantization method(MinMax) with bias correction to quantify weights of Conv2d and Linear Layers.

- We use ACIQ to specify activation's quantization range.

- For the weights and activations that satisfy the condition, we quantify them per channel.

- We use Pytorch for all our experiments.

B. AdaRound

1. **Methods:** We refer to Base-quantization for most of the structure of our AdaRound codes. However, it has a big mistake in optimizing the parameter V. Therefore, in the step "AdaRound" below, we mainly refer to the implement in Aimet. The overall implementation flow is as follows:

- Load data and pretrained model.

- Inplace quantization layers:

  Replacing original Conv2d and Linear layers with corresponding quantization layers to implement the quantization of weights and activations. Besides, if the batch normalization layer is right next to the convolution layer, batch normalization is absorbed in the weights of the convolution layer. In detail, "AdaRoundQuantizer" is implemented for weight quantization and "AsymmetricQuantizer" is implemented for activations quantization, which uses normal uniform affine quantization.

- Calibration:

  1) Quantization parameters are calibrated offline by processing the weights and activations generated by running inference on a sample dataset.

  2) For activations, we collect [a;b] ranges seen on activations during the calibration step and then aggregate them via exponential moving averages (EMA). For weights, we simply set a := min w, b := max as introduced earlier.

  3) After collecting the [a;b] ranges, we use the parameters to update the scale and zero point parameter according to Eq.(2) and Eq.(3). At the end of this stage, the quantization parameters are no longer changed.

- AdaRound:

  AdaRound uses Eq.(19) to quantize weights. For each weight, we need the value of the $V_{i,j}$ to do calculation. Eq.(18) defines the final objective that can be optimized via stochastic gradient descent. We implement AdaRound layer by layer as mention in paper as follows:

  1) Get all the layers need to train in the order of forwarding. We use hook api in pytorch to add each layer's name one by one following the order of forwarding.

  2) Determine the layers who has an activation function just after it. In this part we refer to the "mark ReLU" method used in codes of ACIQ.

  3) For each layer in the layer list above, do the following steps:

  - Prepare data for just that layer. We use hook api in pytorch to get the input data $\hat{x}$ of quantization flow, the output data Wx of the original flow.
  - Use Adam optimizer to optimize only V of the layer. Only V of the layer require gradient.
  - For each iteration, we calculate the reconstruction loss and round loss in Eq.(18) and optimize the parameter V each batch. The parameter β used in round loss calculation updates for each iteration. When the optimization of this layer is finished, we stop changing the parameter's value and set it $requires\_grad = False$.

- Evaluation: Obtain the accuracy results of the quantization model on the validation dataset.

2. **Experiment setup**

- Set bit width=8 for both activations and weights.
- Although the paper recommend to use 1k+ unlabeled samples from validation set to train AdaRound for 10k iterations, we have not that much time to train so many data and for so many iterations. So in the calibration stage, we use (128 for resnet, 1280 for retinanet, 32 for deeplabv3) unlabeled samples from validation set as the sample dataset (for it is hard to select at most 128 samples from train set with even categories). In AdaRound training step, we use (128 for resnet, 10 for retinanet, 32 for deeplabv3) unlabeled samples from validation set and Adam optimizer with default hyper-parameters.
- Initialize parameter V by solving the equation $h(V) = \frac{W}{s} - \lfloor \frac{W}{s} \rfloor$.
- We use linear annealing scheduler for temperature β from 20 to 2.
- We use Pytorch for all our experiments.

IV. RESULTS AND ANALYSIS

	ResNet-50			RetinaNet_resnet50_fpn							Deeplabv3_resnet50	
Method	p1	p5	time	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	time	Miou	time
Baseline	76.13	92.86	10 min 15 sec	36.4	55.7	38.2	19.3	40.0	49.0	5 min 26 sec	77.83	1 min 30 sec
ACIQ	76.05	92.86	15 min 2 sec	36.0	55.3	37.9	19.1	39.8	49.3	34 min 2 sec	77.67	5 min 32 sec
AdaRound	76.01	92.83	10 min 47 sec	36.4	55.8	38.2	19.3	40.0	48.8	8 min 31 sec	77.84	2 min 23 sec

Analysis:

- The performance of the quantized model is slightly worse than the original model on some metrics, while the others reach the baseline. The results are reasonable, because 8bits is enough to make the quantized value very close to the original value, except for small clipping noise and rounding noise, thus the accuracy will drop a bit.

- The inference time of the quantized model is longer than the original model, because of the extra quantization and dequantization operations. The quantization will accelerate the inference process with hardware support.
- We experiment on retinanet using different numbers of images for calibration and AdaRound: 1) 64 for calibrate and 64 for AdaRound; 2) 1280 for calibrate and 10 for AdaRound. The results show that 1280 for calibrate and 10 for AdaRound perform better. Therefore, performance on resnet and deeplabv3 may be higher if we use 1280 imgs to calibrate (However we have no time to experiment).

V. IMPROVEMENT DIRECTIONS

A. ACIQ

We mainly report the optimization problems we find out on aciq and some methods we try to fix it.

After reading the optimization of fused ReLU, we find it unreasonable to optimize the positive half. The authors aim to optimize on $[0, \alpha]$, but they are actually optimizing on $[\mu, \mu+\alpha]$ by optimizing $E[(X - Q(X))^2] = \sum_{i=0}^{2^M-1} \int_{i\delta}^{(i+1)\delta} f(x)(x-q_i)^2 dx + \int_{\alpha}^{\infty} f(x)(x-\alpha)^2 dx$ with assumption $\mu = 0$, which is only a part of values that would pass the ReLU function when $\mu > 0$.

Assumed that we want to clip to $[0, \alpha]$, the α in which is always larger than μ of $f(x)$. The correct MSE it need to optimize without the assumption $\mu = 0$ is as follows.

$$E[(X - Q(X))^2] = \frac{\alpha^2}{24 \cdot 2^{2M}} + b^2 \cdot e^{-\frac{\alpha-\mu}{b}}$$

The $c_c(M, b)$ of $\alpha = c_c(M, b)b$ is the solution of equation:

$$\frac{\alpha}{3 \cdot 2^{2M+2}} = b \cdot e^{-\frac{\alpha-\mu}{b}}$$

We read the codes provided in ACIQ[1], it uses $[0, min(\mu, 0) + \alpha_F]$ as the clipping range in reality.

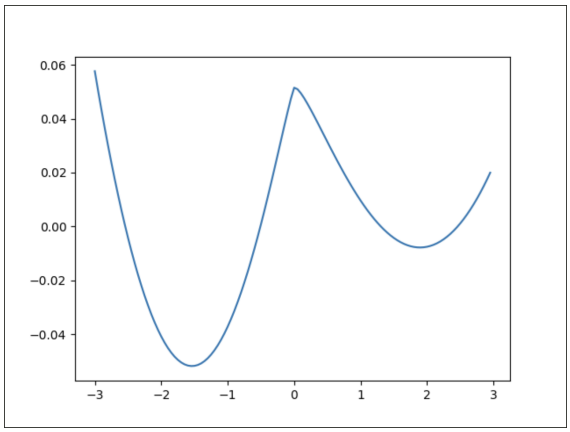
In order to fix this optimization problem, we try the polynomial approximation method and use it in our experiment. It doesn't work well(maybe there're still sth. wrong in our calculation), so we doesn't show it in our results. In short, we just elaborate our idea here.

We believe that $c_c(M, \frac{\mu}{b})$ can be simulated by a function $G(c_F(M), \frac{\mu}{b}) = c_F(M) - g(\frac{\mu}{b})$, for $\frac{\mu}{b}$ in a relatively small range. The most accurate simulation may be able to calculated with Taylor expansion, but we have no time to calculate it and just try some parameter to minimize $|3 \cdot 2^{2M+2} \cdot e^{-G(c_F(M), \frac{\mu}{b}) + \frac{\mu}{b}} - G(c_F(M), \frac{\mu}{b})|$ for $\frac{\mu}{b} \in [-3, 3]$. For $\frac{\mu}{b} \notin [-3, 3]$, we believe the distribution is far from 0, due to which the basic aciq optimization over $[-\infty, +\infty]$ is a good simulation(for those $\frac{\mu}{b} < -3$, $[0, max]$ may be enough clip range. We also analyze $\frac{\mu}{b}$ appear in activation in deeplabv3. It shows that 97% of $\frac{\mu}{b}$ are in $[-3, 3]$.

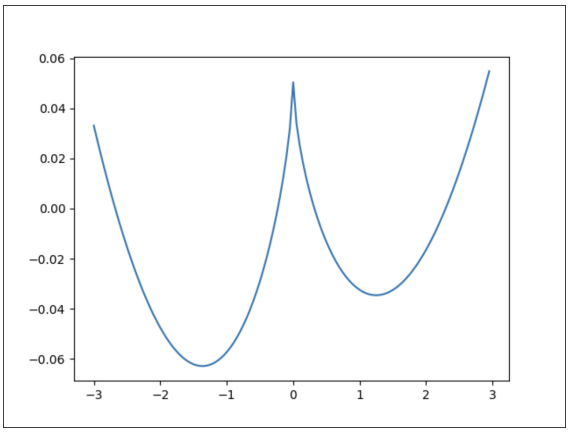
The $g(\frac{\mu}{b})$ we use in $G(c_F(M), \frac{\mu}{b})$ is shown as follows.

```
def tiao(d): # 4bit
    k = -0.92
    return k*(0.996*d+0.011*abs(d)**0.65+0.001*d**2-0.0045)
def tiao2(d): # 8bit
    k = -0.89
    return k*(0.963*d+0.019*abs(d)**1.3+0.001*d**2-0.008)
```

FIG. 1: $g(\frac{\mu}{b})$ for $M = 4$ and $M = 8$



(a)Fig1



(b)Fig2

FIG. 2: Bias of $3 \cdot 2^{2M+2} \cdot e^{-G(c_F(M),\frac{\mu}{b})+\frac{\mu}{b}} - G(c_F(M), \frac{\mu}{b})$ for $M = 4$ (left 2(a)) and $M = 8$ (right 2(b))

B. AdaRound

By referring to relevant papers like [2], we briefly discuss some improvement directions of AdaRound:

- AdaRound forces the quantized weights to be within ± 1 of their round-to-nearest values. We may relax this implicit constraint by allowing the quantized weights to take more values.
- Integer programming: some parts of the network may allow lower precision compared to other layers. We may optimally allocate the bit-widths for each layer, while constraining accuracy degradation.
- A common practice Batch Normalization folding can reduce the amount of MAC operations. However, the reduction in bit-width after quantization can cause the model's internal statistics to deviate further from those of the full precision model. To compensate for this deviation, [2] proposed Batch Normalization Tuning to update BN statistics. First, reconstruct the BN layers then re-tune the BN layers' statistics (by a few iterations of running-mean to re-collect the statistics). Finally, re-absorb (re-fuse) the BN layers into the weight layers.
- We may try to implement per-channel quantization for AdaRound and try other more sophisticated quantization range setting methods.

VI. DIVISION

Wenqian Wang: ACIQ + Help implement training codes of AdaRound
Guanrou Yang: AdaRound

VII. REFERENCE

[1] NRon Banner, Yury Nahshan, and Daniel Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment. *arXiv preprint arXiv:1810.05723*, 2019.

[2] Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. Improving post training neural quantization: Layer-wise calibration and integer programming. *arXiv preprint arXiv:2006.10518*, 2020.

[3] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[4] Markus Nagel, Rana Ali Amjad, Mart Van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. In *International Conference on Machine Learning*, pages 7197–7206. PMLR, 2020.

[5] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

[6] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602*, 2020.