Fixing a Common Error in Power of Two Quantization

Matt Schoenbauer
Department of Computer Science
Columbia University
New York, NY
mms2371@columbia.edu

Austin Pham
Department of Computer Science
Columbia University
New York, NY
ap4460@columbia.edu

***Abstract*—Power-of-Two Quantization is a common technique in quantized deep learning, enabling fast hardware operations via the replacement of multiplications with bit-shifts. In this paper we show that the formula used in over a dozen existing publications for approximating a real number by a power of two is flawed. We propose an alternative approximation that incurs little additional computational cost, and prove its efficacy. We display the performance of this formula on linear and power-of-two quantizers tested on Resnets, MobileNet, and MobileVit with the CIFAR-10 dataset.**

***Index Terms*—Quantization, Deep Learning, Power-Of-Two**

I. Introduction

A significant body of research in the field of deep neural network quantization focuses on various forms of "power of two" quantization. By restricting model weights or model quantization parameters to powers of two, faster hardware implementation can be achieved by replacing multiplication operations with bit-shift operations. The formula used in almost every paper[1] on power of two quantization to approximate a positive number x with a power of two is

$$PO2(x) = 2^{\text{round}(\log_2(x))} \tag{1}$$

[1], [15], [11], [19], [13], [9], [17], [3], [8], [14], [18], [16], [5]. This formula does not always give the power of two nearest to x. For example, if we set $x = 2.9$, we see that

$$PO2(x) = 2^{\text{round}(\log_2(x))} = 2^{\text{round}(1.536)} = 2^2 = 4.$$

This indicates that there is an issue with the formula, since the power of two nearest to 2.9 is 2, not 4. Since the rounding happens in log-space, the formula has an upward bias.

In this paper we contribute the following:

- A new power-of-two quantization formula, $PO2_+$.
- Proofs showing that $PO2_+$ achieves superior quantization error over $PO2$.
- Empirical demonstrations on linear and power-of-two quantizers tested on Resnets, MobileNet, and MobileVit.

[1]There are some papers ([4], [2]) that correctly compute the nearest power of two, but none provide the critique we give here.

II. Literature Review

The two primary methods for quantizing deep neural networks are post-training quantization and quantization-aware training (see [12] for more details). Furthermore, the two primary methods for quantizing network weights are *linear quantization* and *logarithmic quantization*. Linear quantization of a vector x is defined by the formula

$$\text{LinearQuant}(x) = \Delta \cdot \text{clip}(\text{round}(x/\Delta), a, b),$$

where Δ, a, and b, are either set by the user or determined from the data. Linear quantization evenly spaces the quantization weights between the range $[a, b]$ with interval Δ. A common case for using the $PO2$ formula for linear quantization is to restrict Δ to be a power of two, as is done in ([1], [19], [18]). This is also known as *dyadic* quantization [20]. Logarithmic quantization is given by the formula

$$\text{LogQuant}(x) = \text{sign}(x) \cdot \Delta \cdot \text{clip}(PO2(|x|/\Delta), a, b),$$

where the hyperparameters have a similar meaning. This is the most common use of the $PO2$ formula in the literature ([15], [11], [13], [17], [3], [8], [14], [18], [16]). In our experiments, we compare the use of the $PO2$ formula with our improved $PO2_+$ formula for both LinearQuant and LogQuant for both PTQ and QAT.

III. Improved power-of-two quantization formula

We propose a new formula to calculate the power of two approximation to x:

$$PO2_+(x) = 2^{\text{round}\left(\log_2\left(\sqrt{8/9}\cdot x\right)\right)} \tag{2}$$

This formula does not make the same errors that $PO2$ makes. For example, we can see that

$$PO2_+(2.9) = 2^{\text{round}(\log_2(2.734))} = 2^{\text{round}(1.451)} = 2^1 = 2.$$

In Appendix A, we prove the following Theorem:

Theorem 1. *For all $x > 0$, $PO2_+(x) = 2^{n^*}$, where*

$$n^* \in \text{argmin}_{n\in\mathbb{Z}}|2^n - x|.$$

This theorem shows that $PO2_+$ always gives the nearest power of two to x (or one of the nearest, if there is a tie). This means

that the $PO2_+$ formula is optimal for parameters that will be added or multiplied with other values when computing the model output. To see this, we note that if x is a weight value and y is an input value to the layer, then

$$|xy - PO2_+(x)y|^2 = |y| \cdot |x - PO2_+(x)|$$

The left term is the quantization error for this layer's output, and the right hand term is the optimized term in Theorem 1, scaled by a constant. Similarly, we have

$$|x + y - (PO2_+(x) + y)|^2 = |x - PO2_+(x)|,$$

so that the $PO2_+$ formula gives the optimal approximation for added parameters (i.e. biases) as well.

In certain operations (such as batch normalization), quantized parameters will divide layer outputs. In this case the optimal nearby power of two formula is given by

$$PO2_+^{inv}(x) = 2^{\text{round}\left(\log_2\left(\sqrt{9/8}\cdot x\right)\right)} \tag{3}$$

A justification of this is given in the following Theorem, which is proven in the Appendix.

Theorem 2. *For all $x > 0$, $PO2_+^{inv}(x) = 2^{n^*}$, where*

$$n^* \in \text{argmin}_{n\in\mathbb{Z}} \left| \frac{1}{2^n} - \frac{1}{x} \right|.$$

Thus different operations yield different optimal formulas for power of two approximations. There is in fact an operation that gives the original $PO2$ formula as the optimal approximation, as described in the Theorem below. The proof of this Theorem is brief, so we include it below.

Theorem 3. *For all $x > 0$, $PO2(x) = 2^{n^*}$, where*

$$n^* \in \text{argmin}_{n\in\mathbb{Z}} \left|\log_2(2^n) - \log_2(x)\right|.$$

Proof. We have

$$\left|\log_2(2^n) - \log_2(x)\right| = \left|n - \log_2(x)\right|.$$

Clearly $n = \text{round}(\log_2(x))$ minimizes this term across integers. □

However, it is not particularly common to multiply or add log weight values to outputs, so the PO2 formula does not seem optimal in many practical situations.

How much does this new formula improve quantization error? The following Theorem shows that the standard $PO2$ formula is incorrect approximately 8.5% of the time.

Theorem 4. *Let X be a random variable such that*

$$\tilde{X} := \log_2(|X|) - \lfloor \log_2(|X|) \rfloor$$

is uniformly distributed on $[0, 1]$. Then

$$\mathbb{P}[PO2(|X|) \neq PO2_+(|X|)] = \log_2(\sqrt{9/8}) \approx 0.08496. \tag{4}$$

See the Appendix for the proof of this Theorem.

Thus this formula gives a small but noticeable improvement in quantization error. One may argue that an improvement only 8.5% of the time does not warrant a change in the implementation, but this optimization is unlike many accuracy improving optimizations in machine learning. Often model improvements come at the cost of extra computational power, extra model variance, and significant extra complexity. This formula only involves applying a single scale before quantization, thus creating minimal computational and mental complexity to the computation, all for a clearly better formula according to standard metrics.

IV. Methodology and Experiment Setup

Our goal now is to compare the $PO2$ and $PO2_+$ formulas across multiple quantizers, models, and datasets. We cover each in detail here.

A. Quantizers

The linear quantizer we will be working with is described in Algorithm 1 in [1], which we will refer to in our experiments as lin. By replacing the $PO2$ operation with the $PO2_+$ operation, we create lin_+. The power-of-two quantizer in [15] will serve as our representative example of this type of quantizer, which we will refer to as $po2$. By scaling the input to the log term in the definition of $LogQuant$ in this paper by $\sqrt{8/9}$, we obtain $po2_+$. We quantize each model with 2, 3, and 4 bits, and estimate gradients using the straight through estimator.

B. Models

We experiment with each of these quantizers on the following models: ResNet[20,32,44,56] [6], MobileNet [7], and MobileVit [10]. For each model, we only quantize model weights (not activations or biases), and exclude the first and last layers from the quantization setup.

C. Datasets

We experiment with the CIFAR-10 dataset. We intend to do experiments with Imagenet in the future, but could not currently due to computational limitations.

D. Training

To implement Quantization Aware Training, we defined our own PyTorch quantized convolution layer. There is no dequantize method defined as we only need to simulate quantization, meaning the weights remain in a float32 quantized representation. In addition to train loss and accuracy, we collected quantization error over all quantized layers at the end of each epoch.

We ran both quantization-aware training and post-training quantization on each model, dataset, and quantizer. Both techniques began with a fully trained full-precision model. The PTQ implementation was straightforward, as we simply quantized each of the weights as described above. For QAT, we used a straight-through estimator for gradient updates, and trained using SGD for 164 epochs and a base learning rate of 0.1.

Given the magnitude of configurations, we used distributed training to speed up our experimentation. First, we adjusted

hyperparameters like learning rate to account for an increased number of GPUs. Issues of convergence were solved when using synchronized batch normalization. Standard batch normalization only normalizes the data within each device (GPU) while synchronized batch normalization normalizes the input within the whole mini-batch allowing for multi-GPU training. In addition, we found that when collecting metrics, Distributed Data Parallel will add additional samples to the dataset if the size of the dataset is not equally divisible by `batch_size` $\times$ `num_processors`. This has the effect that the calculated metric value will be slightly biased towards those replicated samples, leading to a wrong result. To fix this, we used the `torchmetrics` module which enables automatic metrics synchronization between multiple devices for distributed PyTorch applications.

V. Experimental results

We display the relative change in test accuracy when swapping out the $PO2$ formula with the $PO2_+$ formula in Table I for each model and quantization setup we considered. Similar data for quantization error is available in Table II. We can see the impact of the $PO2_+$ formula on quantization error throughout training in Figure 3.

VI. Discussion

We know mathematically that the $PO2_+$ formula gives superior quantization error over the $PO2$ formula for weights multiplied and added to outputs. It may seem very odd that for QAT, the average quantization error does not necessarily improve when using the $PO2_+$ formula. This is because the quantization error depends heavily on the distribution of the weights throughout training, which depends on the training dynamics, which of course depends in a complex way on the quantization techniques. The theorems apply to a static weight (essentially the PTQ case), and one cannot reliably make predictions about the relationship between quantization error for two models throughout training using different quantization techniques. Although the improvement is not uniform, we do still see that in the majority of cases, the $PO2_+$ quantizers do give superior quantization error.

Note that there are several examples in which the lin_+ quantization technique does not yield improved quantization error over the lin technique for post-training quantization. This does not contradict Theorem 1, since the quantization algorithm from [1] uses a stepwise optimization to find the appropriate scale. This is an approximation algorithm, and the quantization formula changes the trajectory of the optimization at each step, so it is much more difficult to make theoretical guarantees about the results. We do still see quantization error improvement a majority of the time. Theorem 1 only guarantees improved quantization error in the $po2$ PTQ case. We indeed see this for every model and bit count in Table II.

Perhaps the greater practical concern is that it does not seem that the $PO2_+$ formula always results in superior test accuracy. As we described above, the $PO2_+$ formula does not guarantee improved quantization error throughout training, so we should not expect improved test accuracy for QAT. However, for PTQ $po2$, we should expect this, if we believe that improved quantization error directly translates to improved test error. But this is not the case: since neural network loss surfaces are nonconvex, we cannot expect that by keeping the weights closer to their trained values we will also keep the performance closer to its peak.

There may be another reason that the test performance values for the $PO2_+$ models was not necessarily as strong as those of the $PO2$ models. As discussed in Section III, there are different ways of measuring quantization error, each of which leads to different optimal power-of-two approximations. Perhaps the absolute error of Theorem 1 is not the optimal way to measure quantization error, and instead the log error of Theorem 3 is better. This metric essentially measures the magnitude of the quotient of the weight and its approximation, or the relative change in scaling one weight to its approximation or vice versa. If this is true, then this requires serious reconsiderations for many techniques in quantized deep learning, where absolute (or MSE) loss is considered the gold standard [12].

An important limitation of these results is that only one test was run for each configuration. In order to clear up these results, we need to gather all of our metrics over multiple seeds, and study the distributions of these metrics around these seeds. Single runs are noisy estimates of the expected behavior on new models, so our results are expected to be a little murky. This is especially important since our expected change is small (bringing an improvement for only 8.5% of weights), so any measurements of test and quantization error must be precise. We intend to run these tests in preparation for publication.

VII. Conclusion

We proposed a new method for power-of-two approximation, and proved its effectiveness. We experimented with it for standard neural network quantization tasks. The empirical results were mixed, and we pointed to further experimentation and theoretical inquiries that may resolve the uncertainty.

Acknowledgments

We would like to thank Daniele Moro from Google Research for his encouragement, excitement, and direction on this project.

Algorithm	Quantizer	Bits	MobileNet	MobileVit	ResNet20	ResNet32	ResNet44	ResNet56
PTQ	lin+	2	15.11	–	–	-24.67	-15.45	–
		3	-1.06	-3.56	-6.07	-2.66	-9.22	-5.74
		4	-0.23	-7.53	-1.47	-0.3	-0.58	-1.36
	po2+	2	–	–	–	–	–	–
		3	3.55	32.58	2.78	4.65	5.22	2.01
		4	-0.81	1.78	1.32	-1.33	2.47	-1.05
QAT	lin+	2	–	–	-3.67	0.03	0.3	-0.15
		3	-7.1	1.32	-0.21	0.0	0.03	-0.63
		4	-1.95	-0.44	-0.23	0.15	-0.05	0.02
	po2+	2	-3.68	1.71	1.31	-0.05	0.14	-0.08
		3	-14.64	-0.16	0.13	0.18	-0.21	-0.29
		4	2.38	-0.17	0.11	-0.16	0.32	0.11

TABLE I

RELATIVE TEST ACCURACY CHANGES AFTER REPLACING THE $PO2$ FORMULA WITH THE $PO2\_+$ FORMULA. POSITIVE VALUES INDICATE AN IMPROVEMENT IN ACCURACY. MISSING ENTRIES INDICATE THAT BOTH RESULTING MODELS HAD LESS THAN 12% ACCURACY, IMPLYING THAT THE QUANTIZED MODELS WOULD NEVER BE USEFUL IN PRACTICE.

Algorithm	Quantizer	Bits	MobileNet	MobileVit	ResNet20	ResNet32	ResNet44	ResNet56
PTQ	lin+	2	-5.98	-6.79	-3.66	-5.43	-5.61	-6.64
		3	-0.97	-1.85	-8.15	-2.62	-4.33	-4.94
		4	6.73	3.74	-1.18	1.28	1.27	0.73
	po2+	2	-0.03	-0.01	-0.01	-0.01	-0.01	-0.01
		3	-3.29	-1.82	-3.07	-2.68	-2.62	-2.28
		4	-8.35	-8.31	-8.41	-8.43	-8.59	-8.41
QAT	lin+	2	–	-94.19	-5.21	-3.61	-4.92	-3.96
		3	-6.15	-5.16	-5.75	-3.94	-2.98	-3.74
		4	9.04	9.32	2.49	2.69	2.44	2.26
	po2+	2	2.96	-1.73	-1.37	0.9	-6.85	-0.1
		3	1.32	-0.22	0.35	5.89	7.14	4.7
		4	-5.35	-9.25	-3.03	-3.08	-3.41	-2.99

TABLE II

RELATIVE QUANTIZION ERROR CHANGES AFTER REPLACING THE $PO2$ FORMULA WITH THE $PO2_+$ FORMULA. NEGATIVE VALUES INDICATE AN IMPROVEMENT IN QUANTIZATION ERROR. MISSING ENTRIES INDICATE THAT BOTH RESULTING MODELS HAD LESS THAN 12% ACCURACY, IMPLYING THAT THE QUANTIZED MODELS WOULD NEVER BE USEFUL IN PRACTICE.

REFERENCES

[1] Sungmin Bae, Piotr Zielinski, and Satrajit Chatterjee. A closer look at hardware-friendly weight quantization. *arXiv preprint arXiv:2210.03671*, 2022.

[2] Jingyong Cai, Masashi Takemoto, and Hironori Nakajo. A deep look into logarithmic quantization of model parameters in neural networks. In *Proceedings of the 10th International Conference on Advances in Information Technology*, pages 1–8, 2018.

[3] Sung-En Chang, Yanyu Li, Mengshu Sun, Runbin Shi, Hayden K-H So, Xuehai Qian, Yanzhi Wang, and Xue Lin. Mix and match: A novel fpga-centric deep neural network quantization framework. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 208–220. IEEE, 2021.

[4] Brian Chmiel, Ron Banner, Elad Hoffer, Hilla Ben Yaacov, and Daniel Soudry. Logarithmic unbiased quantization: Practical 4-bit training in deep learning. 2021.

[5] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[8] Edward H Lee, Daisuke Miyashita, Elaina Chai, Boris Murmann, and S Simon Wong. Lognet: Energy-efficient neural networks using logarithmic computation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5900–5904. IEEE, 2017.

[9] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration:: A survey. 2021.

[10] Sachin Mehta and Mohammad Rastegari. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021.

[11] Daisuke Miyashita, Edward H Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*, 2016.

[12] 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.

[13] Prateeth Nayak, David Zhang, and Sek Chai. Bit efficient quantization for deep neural networks. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 52–56. IEEE, 2019.

[14] Sangyun Oh, Hyeonuk Sim, Sugil Lee, and Jongeun Lee. Automated log-scale quantization for low-cost deep neural networks. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 742–751. IEEE Computer Society, 2021.

[15] Dominika Przewlocka-Rus, Syed Shakib Sarwar, H Ekin Sumbul,

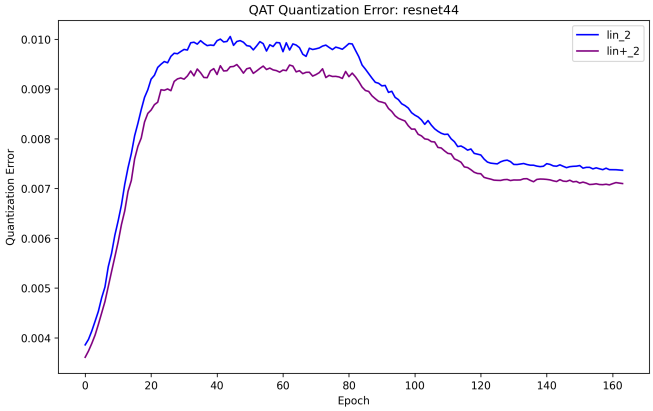

Fig. 1. Quantization error with lin_+.

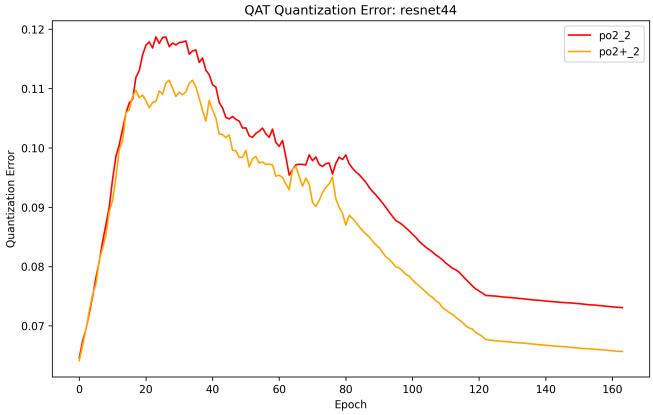

Fig. 2. Quantization error with $po2_+$.

Fig. 3. Here we see that the $PO2_+$ formula gives us improved quantization error throughout training. This data was collected for ResNet44 with 2-bit quantization using both the lin_+ and $po2_+$ formulas. This aligns with the mathematical results in Theorem 1.

Yuecheng Li, and Barbara De Salvo. Power-of-two quantization for low bitwidth and hardware compliant neural networks. *arXiv preprint arXiv:2203.05025*, 2022.

[16] Takeo Ueki, Keisuke Iwai, Takashi Matsubara, and Takakazu Kurokawa. Learning accelerator of deep neural networks with logarithmic quantization. In *2018 7th International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 634–638. IEEE, 2018.

[17] Stefan Uhlich, Lukas Mauch, Kazuki Yoshiyama, Fabien Cardinaux, Javier Alonso García, Stephen Tiedemann, Thomas Kemp, and Akira Nakamura. Differentiable quantization of deep neural networks. *CoRR*, abs/1905.11452, 2019.

[18] Jiawei Xu, Yuxiang Huan, Yi Jin, Haoming Chu, Li-Rong Zheng, and Zhuo Zou. Base-reconfigurable segmented logarithmic quantization and hardware design for deep neural networks. *Journal of Signal Processing Systems*, 92(11):1263–1276, 2020.

[19] Hongyi Yao, Pu Li, Jian Cao, Xiangcheng Liu, Chenying Xie, and Bingzhang Wang. Rapq: Rescuing accuracy for power-of-two low-bit post-training quantization. *arXiv preprint arXiv:2204.12322*, 2022.

[20] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, et al. Hawq-v3: Dyadic neural network quantization. In *International Conference on Machine Learning*, pages 11875–11886. PMLR, 2021.

APPENDIX

Proof of Theorem 1. In this proof it will be convenient to write the formula for $PO2_+$ as

$$PO2_+(x) = 2^{\text{round}\left(\log_2\left(\sqrt{8/9}\cdot x\right)\right)} = 2^{\text{round}(\log_2(x/1.5)+0.5)}.$$

First note that

$$2^{\lfloor \log_2(x)\rfloor} \leq 2^{\log_2(x)} = x \leq 2^{\lceil (\log_2(x)\rceil}.$$

Since 2^n is an increasing function of n, we must have $n^* \in \{\lfloor \log_2(x)\rfloor, \lceil \log_2(x)\rceil\}$.

In the case that $\log_2(x) \in \mathbb{Z}$, then $\{\lfloor \log_2(x)\rfloor = \lceil \log_2(x)\rceil = \text{round}(\log_2(x))$, then

$$PO2_+(x) = 2^{\text{round}(\log_2(x/1.5))+0.5)} = 2^{\text{round}(\log_2(x)-\log_2(1.5)+0.5)} = 2^{\text{round}(\log_2(x)-0.085)} = 2^{\log_2(x)} = x.$$

In this case $PO2_+$ clearly gives the correct value.

If $\log_2(x) \notin \mathbb{Z}$, then $\lfloor \log_2(x)\rfloor = \lceil \log_2(x)\rceil + 1$, so that

$$|2^{\lfloor \log_2(x)\rfloor} - x| = x - 2^{\lfloor \log_2(x)\rfloor}$$

and

$$|2^{\lceil \log_2(x)\rceil} - x| = 2^{\lceil \log_2(x)\rceil} - x = 2^{\lfloor \log_2(x)\rfloor+1} - x = 2\cdot 2^{\lfloor \log_2(x)\rfloor} - x.$$

Thus we have

$$|2^{\lfloor \log_2(x)\rfloor} - x| < |2^{\lceil \log_2(x)\rceil} - x| \iff 2x < 3\cdot 2^{\lfloor \log_2(x)\rfloor}$$

Thus we are done if we can show that

$$\text{round}(\log_2(x/1.5)+0.5) = \lfloor \log_2(x)\rfloor \iff 2x < 3\cdot 2^{\lfloor \log_2(x)\rfloor}.$$

To see this, note that $\text{round}(x+0.5) = \lceil x\rceil = \lfloor x\rfloor + 1$ when $x \notin \mathbb{Z}$. Thus we have

$$\begin{aligned}
\text{round}(\log_2(x/1.5)+0.5) = \lfloor \log_2(x)\rfloor &\iff \lfloor \log_2(x/1.5)\rfloor + 1 = \lfloor \log_2(x)\rfloor && (5)\\
&\iff \lfloor \log_2(x) - \log_2(1.5)\rfloor + 1 = \lfloor \log_2(x)\rfloor && (6)\\
&\iff \log_2(x) - \log_2(1.5) < \lfloor \log_2(x)\rfloor && (7)\\
&\iff \frac{2x}{3} < 2^{\lfloor \log_2(x)\rfloor} && (8)\\
&\iff 2x < 3\cdot 2^{\lfloor \log_2(x)\rfloor}. && (9)
\end{aligned}$$

□

Proof of Theorem 2. By setting $x' = 1/x$, we can apply Theorem 1 to see that $\text{round}(\log_2(\sqrt{8/9}/x))$ is the power of two nearest to $1/x$. Thus

$$-\text{round}(\log_2(\sqrt{8/9}/x)) = \text{round}(-\log_2(\sqrt{8/9}/x)) = \text{round}(\log_2(\sqrt{9/8}x))$$

is a satisfactory candidate for n^*. This is the formula given by $PO2_+^{inv}$. □

Proof of Theorem 4. We assume wlog that $X > 0$. In order to prove the statement, we will show that

$$PO2(X) \neq PO2_+(X) \iff \tilde{X} \in \left[1/2 - \log_2(\sqrt{9/8}), 1/2\right],$$

Since

$$\mathbb{P}\left[\tilde{X} \in \left[1/2 - \log_2(\sqrt{9/8}), 1/2\right]\right] = \log_2(\sqrt{9/8}).$$

We will make use of the equality

$$\text{round}(\log_2(X)) = \text{round}(\log_2(X) - \lfloor \log_2(X)\rfloor) + \lfloor \log_2(X)\rfloor = \text{round}(\tilde{X}) + \lfloor \log_2(X)\rfloor. \tag{10}$$

Furthermore, we define

$$\tilde{X}_+ = \log_2(X) - \log_2(\sqrt{9/8}) - \lfloor \log_2(X) - \log_2(\sqrt{9/8})\rfloor$$

so that we have and

$$\text{round}\left(\log_2(\sqrt{8/9}X)\right) = \text{round}(\tilde{X}_+) + \lfloor \log_2(X) - \log_2(\sqrt{9/8})\rfloor. \tag{11}$$

From this we can see that

$$PO2(X) = PO2_+(X) \iff \text{round}\left(\log_2(\sqrt{8/9}X)\right) = \text{round}\left(\log_2(X)\right) \iff \lfloor \log_2(X) - \log_2(\sqrt{9/8})\rfloor = \lfloor \log_2(X)\rfloor.$$

To see the second equivalence, note that both equations hold if and only if the fractional parts of $\log_2(X) - \log_2(\sqrt{9/8})$ and $\log_2(X)$ are on the same side of 1/2. We will consider 3 separate cases: $\tilde{X} \in [0, 1/2 - \log_2(\sqrt{9/8})]$, $\tilde{X} \in [1/2 - \log_2(\sqrt{9/8}), 1/2]$, $\tilde{X} \in [1/2, 1]$.
Case 1: $\tilde{X} \in [0, 1/2 - \log_2(\sqrt{9/8})]$. Here we have

$$\lfloor \log_2(X) + \log_2(\sqrt{8/9}) \rfloor = \lfloor \log_2(X) \rfloor,$$

so that $PO2(X) = PO2_+(X)$.
Case 2: $\tilde{X} \in [1/2 - \log_2(\sqrt{9/8}), 1/2]$. Here we have

$$\lfloor \log_2(X) + \log_2(\sqrt{8/9}) \rfloor = \lfloor \log_2(X) \rfloor + 1,$$

so that $PO2(X) \neq PO2_+(X)$.
Case 3: $\tilde{X} \in [1/2, 1]$. Here we have

$$\lfloor \log_2(X) + \log_2(\sqrt{8/9}) \rfloor = \lfloor \log_2(X) \rfloor,$$

so that $PO2(X) = PO2_+(X)$. □