A Cost of Affine Quantizer

A.1 Cross-terms due to zero-points

Consider two real numbers r_1 and r_2 and their product $r_3 = r_1 \cdot r_2$. Using the affine mapping from (2) to represent this, we get:

$$s_3(q_3 - z_3) = s_1(q_1 - z_1) \cdot s_2(q_2 - z_2), \tag{12}$$

which can be expressed as

$$q_3 = z_3 + \frac{s_1 s_2}{s_3}\big[q_1 q_2 - q_1 z_2 - q_2 z_1 + z_1 z_2\big]. \tag{13}$$

The cross-terms in (13) add complexity and often require special handling to remain efficient. While the added cost can be amortized over several accumulations of a matrix multiplication or convolution operation, it would still require optimizations[8], both algorithmic and kernel-level.

By eliminating zero-points, the cross-terms vanish and the operation simplifies to:

$$q_3 = \frac{s_1 s_2}{s_3}\big[q_1 q_2\big]. \tag{14}$$

A.2 Real-valued scale-factors

With positive real scale-factors, the constant multiplier $s_1 s_2 / s_3$ in (14), empirically found to be in the interval (0, 1) (Jacob et al., 2017), can be expressed in the normalized form $2^{-n} s_0$ where n is a non-negative integer and s_0 is in the interval [0.5, 1). In other words, the accumulator (storing $q_1 q_2$) needs to be scaled by a fixed-point multiplier that approximates s_0 and right-shifted by n bits (with round-to-nearest):

$$q_3 = 2^{-n} s_0 \big[q_1 q_2\big]. \tag{15}$$

However, by constraining scale-factors s_1, s_2, s_3 to strict power-of-2, the scaling operation reduces to a rather simple bit-shift (with round-to-nearest):

$$q_3 = 2^{-f}\big[q_1 q_2\big]. \tag{16}$$

B Log Threshold Training

Initially, it may seem that with the definition of a gradient with respect to the raw threshold, backpropagation and gradient descent could be immediately used to train it. However, just as training weights in a vanilla neural network requires care in the choice of optimizer and learning rate, here too care must be taken to ensure training stability and convergence. There are three main properties we would like our training procedure to satisfy: numerical stability, scale invariance, and convergence. We discuss each of these issues and the engineering tweaks used to solve them here.

[8]Some of which are covered in (Jacob et al., 2016a; 2017; Krishnamoorthi, 2018).

B.1 Numerical Stability

One obvious problem with training raw thresholds $t \in \mathbb{R}^+$ is that gradient updates could potentially bump a threshold to a negative value, causing $\log_2 t$ and therefore scale-factor s to diverge. If this happens even once, the network as a whole will break. An easy solution is to train $\log_2 t$ as opposed to t itself, since its domain is $\log_2 t \in \mathbb{R}$. Using log thresholds is convenient because it already appears in the expression for $s(t)$. However, the most important benefit is described in Section B.2, where the log representation makes ensuring scale invariance very easy.

B.2 Scale Invariance

For a given input distribution we prefer that the threshold gradients have similar magnitudes regardless of the position of the threshold itself. This *threshold scale invariance* is useful for making sure training is not too slow when the thresholds are far from their optimal values. Similarly, the properties of our threshold gradients should not depend on the scale of the input distribution. This *input scale invariance* is important because it ensures that quantized training behaves the same way for the different weights and activations in the network, even if the variance of their distributions vary over many orders of magnitude.

Unfortunately, neither of these scale invariances hold. Far from improving, Figure 7 shows that in moving from raw threshold training (left) to log threshold training (middle), both scale invariance properties of the threshold gradients actually degrade.

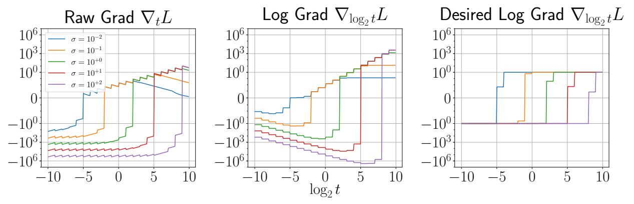

Figure 7. Gradients of L_2-loss with respect to raw threshold (left) or log threshold (middle, right) versus log threshold, for Gaussian(σ) inputs of varying σ. Desired (normed) gradients for the log threshold case are shown on the right.

Threshold scale invariance: Updates to the log threshold would be threshold scale invariant if the gradients on both sides of the negative-to-positive jump were flat, as seen in the right plot of Figure 7. However, this is not the case for log threshold gradients (center plot of Figure 7). On the left-of-jump side, as $\log_2 t$ decreases, gradients of (hence

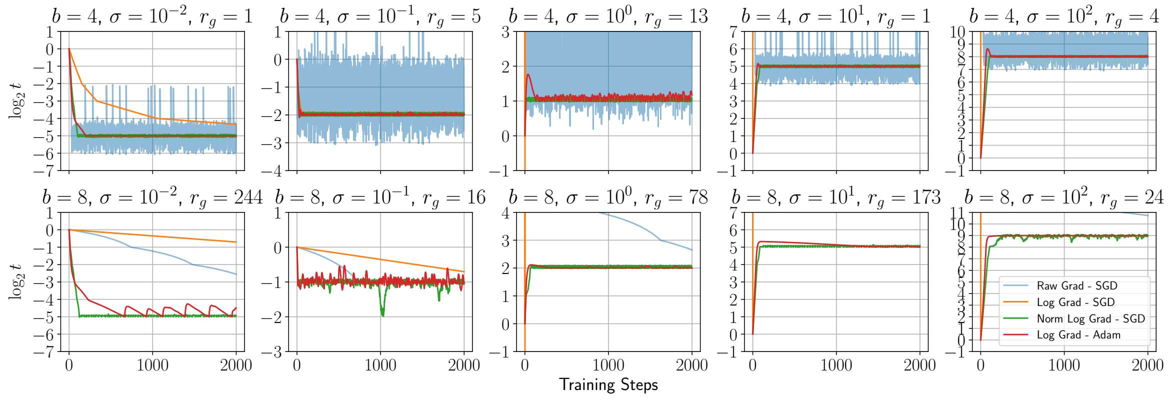

Figure 8. Raw, log and normed log threshold training on L_2-loss for 2000 steps with learning rate $\alpha = 0.1$. We compare different bit-widths - 4 (top) and 8 (bottom), and Gaussian(σ) inputs of varying σ - smallest (left) to largest (right). The empirical value of r_g is estimated from the last few hundred steps of Adam.

updates to) $\log_2 t$ get exponentially smaller, meaning it will converge very slowly to lower optimal values (see the log grad SGD case in the left plots of Figure 8). Similarly, on the right-of-jump side, as $\log_2 t$ increases, updates to $\log_2 t$ increase exponentially, meaning it will converge very quickly and possibly unstably to higher optimal values (see the log grad SGD case in the right plots of Figure 8). In the raw threshold domain, we would like gradients of (hence updates to) t to scale proportional to t. This is also not the case for the left-of-jump side of raw threshold gradients (left plot of Figure 7). In other words, the raw and log threshold gradients are swapped from what we would prefer on the left-of-jump sides.

Input scale invariance: Updates to the log threshold are input scale invariant if the gradients are threshold scale invariant and x-axis shifted copies for varying input scales, as seen in the right plot of Figure 7. However, this is not the case for log threshold gradients (center plot of Figure 7) as the gradient magnitudes depend on the scale of the input. In fact when accounting for the threshold scale dependence, the gradient magnitudes depend quadratically on the scale of the input.

Normed gradients: While neither raw or log threshold gradients have the desired properties of scale invariance, only minimal modifications to our log threshold gradient is needed to get these properties to hold (see desired log threshold gradient on the right of Figure 7). In particular, if we normalize the gradient g_i by its bias-corrected moving average variance, we achieve a close approximation of the desired gradients $\tilde{g}_i$, shown in (17). To improve stability, we can encapsulate (17) in a clipping function to guarantee no large gradients, shown in (18).

Yet another desired property highlighted in Figure 7 is that near the jump, the ratio of the gradient magnitudes to either side of the jump is to be preserved between the original and normed gradient cases. This is important for the convergence dynamics of the system discussed in Section B.3. In dynamic situations, the gradient normalization solution (17) approximates this feature as well.

$$\begin{aligned} v_i &\leftarrow \beta v_{i-1} + (1-\beta) g_i^2 \\ \hat{v}_i &\leftarrow \frac{v_i}{1-\beta^i} \\ \tilde{g}_i &\leftarrow \frac{g_i}{\sqrt{\hat{v}_i} + \epsilon} \qquad (17) \\ \tilde{g}_i &\leftarrow \tanh\left(\frac{g_i}{\sqrt{\hat{v}_i} + \epsilon}\right) \qquad (18) \end{aligned}$$

Figure 8 shows training curves on the toy L_2 quantization error problem across various bit-widths, input scales, and optimization algorithms. Raw gradient with SGD fails for large σ and converges too slowly for small σ, as we would expect from Sections B.1 and B.2. Additionally, they have b, σ-dependent stability once converged. Switching from raw to log threshold gradients, we see that log gradient with Adam performs well, yet log gradient with SGD performs poorly, with weak convergence rates for small σ and divergence for large σ. However, after performing gradient normalization (18), normed log gradient with SGD performs well, demonstrating that lack of proper gradient norming is the main issue preventing convergence using standard gradient descent. Besides the differing convergence rates, another characteristic becomes immediately obvious - stability after convergence. For example, raw gradient method tends to oscillate wildly between multiple integer-level log thresholds, whereas normed log gradient method is better behaved and tends to stay within a single integer log threshold band.

Adam optimizer: While gradient norming (18) led to good results with SGD, we note that Adam without this gradient norming also works quite well. It is easy to see why this is -

Adam has built-in gradient norming (Kingma & Ba, 2014). Thus we can avoid redefining the gradients by simply using an optimizer that includes adaptive gradients, such as Adam or RMSprop (Hinton et al., 2012). While RMSprop appears to superficially resemble (18) more closely than Adam, we suspect Adam has better behavior in the absence of gradient clipping due to its use of moments to smooth the gradients. To use Adam safely, we derive rough bounds on the learning rate and momentum parameters to ensure the oscillations seen in Figure 8 for log gradient with Adam do not exceed a single integer bin. This is important because if they move across bins often, the network may have more trouble adapting to the changing distributions from a given quantized layer, in an effect that may be similar to the motivation for batch normalization (Ioffe & Szegedy, 2015).

B.3 Convergence

One primary cause of the sharp gradient jumps seen in Figure 7 is our insistence on power-of-2 scaling. In the forward pass, features downstream from the quantized layer are completely unaware of intermediate non-power-of-2 scale-factors so there are sharp jumps at integral $\log_2 t$, similar to what might be observed when using the STE for traditional quantization. The net effect is a bang-bang like operation.

In more detail, for a given input distribution there is some critical integer threshold $\log_2 t^*$ before which the gradients are negative (causing positive threshold updates) and after which the gradients are positive. This negative feedback will force the threshold to oscillate around $\log_2 t^*$. The gradients g_l and g_h on either side of $\log_2 t^*$ tend to be fairly constant within a distance 1 of $\log_2 t^*$ due to power-of-2 scaling. For simplicity, assume $|g_l| > |g_h|$ so that the ratio $r_g = -g_l/g_h > 1$. As r_g grows, we would expect the following behavior: the threshold stays in the higher bin for a while, slowly decaying until reaching the lower bin, at which point a large $|g_l|$ causes it to jump back to the higher bin, where it begins a slow decay again. This behavior can be observed in the left plots of Figure 8 and are shown in more detail in Figure 9.

If normed log gradients and SGD are used together, the dynamics are fairly simple. Let $\log_2 t_i \leftarrow \log_2 t_{i-1} - \alpha \tilde{g}_i$ be the SGD update on normed log gradient $\tilde{g}_i$ (18). Then because $|\tilde{g}_i| \leq 1$ by design, a given jump in the sawtooth-like pattern will have magnitude bounded by learning rate α. Thus by selecting $\alpha \ll 1$, we can ensure convergence within a threshold bin.

However in our experiments, we used the implementationally simpler approach of unnormed log gradients with the Adam optimizer. While simpler to implement, the analysis is more complicated due to the second-order nature of the optimizer. Adam has three key hyperparameters: α, β_1, β_2 and operates by keeping track of a moving mean of gradients $m_i \leftarrow \beta_1 m_{i-1} + (1-\beta_1) g_i$ and a moving variance $v_i \leftarrow \beta_1 v_{i-1} + (1-\beta_1) g_i^2$ before applying update rule $\theta_i \leftarrow \theta_{i-1} - \alpha \cdot m_i / \sqrt{v_i}$. In practice, bias correction is used to get $\hat{m}_i, \hat{v}_i$, but when considering settling dynamics for $i \to \infty$, this bias correction is insignificant. Typical values are $\alpha \approx 10^{-3}, \beta_1 \approx 0.9, \beta_2 \approx 0.999$.

In Appendix C, a detailed analysis of convergence for Adam is carried out. From this analysis a simple set of guidelines emerge. First, the learning rate is set to guarantee $\alpha < 0.1/\sqrt{p}$. Next, we ensure $1/e < \beta_1 < 1$ to satisfy the limits of our analysis. Finally, we make sure $r_g \approx p \ll 1/(1-\beta_2) \Rightarrow 1-\beta_2 \ll 1/p$. These results are summarized in Table 4. For simplicity, we use $\alpha = 0.01, \beta_1 = 0.9, \beta_2 = 0.999$ for all of our training.

Table 4. Guidelines for log threshold training with Adam, assuming $b = 2^{b-1} - 1$ for signed data.

Bit-width	b	4	8
α	$\leq \frac{0.1}{\sqrt{2^{b-1}-1}}$	≤ 0.035	≤ 0.009
β_1	$\geq 1/e$	$\geq 1/e$	$\geq 1/e$
β_2	$\geq 1 - \frac{0.1}{2^{b-1}-1}$	≥ 0.99	≥ 0.999
Steps	$\approx \alpha^{-1} + (1-\beta_2)^{-1}$	≈ 100	≈ 1000

C Analysis of Adam Convergence

Let T be the period of oscillations at convergence. If we assume $T \ll 1/(1-\beta_2)$, then we can treat the moving variance estimate as if it is a constant $v_i = ((T-1)g_h^2 + g_l^2)/T \approx g_l^2(1/r_g^2 + 1/T)$. However, we cannot make the same assumption for the relationship between T and β_1. Instead, based on our earlier discussion in Section B.3 of the bang-bang behavior, we assume that a gradient g_l is seen for a single step, then g_h is seen for $T-1$ steps. Then for a given cycle of this behavior, $m_i = \beta_1^i(\beta_1 m_0 + (1-\beta_1)g_l) + (1-\beta_1^i)g_h$, where m_0 is the steady-state minimum mean during the cycle. Because this is steady-state, we can solve for m_0 and m_i:

$$\begin{aligned} m_i &= \beta_1^i(\beta_1 m_0 + (1-\beta_1)g_l) + (1-\beta_1^i)g_h \\ m_T = m_0 &= \beta_1^T(\beta_1 m_0 + (1-\beta_1)g_l) + (1-\beta_1^T)g_h \\ m_0 &= \frac{\beta_1^T(1-\beta_1) - (1-\beta_1^T)/r_g}{1-\beta_1^{T+1}} g_l \qquad (19) \\ \frac{m_i}{g_l} &= \beta_1^{i+1} \frac{\beta_1^T(1-\beta_1) - (1-\beta_1^T)/r_g}{1-\beta_1^{T+1}} \\ &\quad + \beta_1^i(1-\beta_1+\frac{1}{r_g}) - \frac{1}{r_g} \qquad (20) \end{aligned}$$

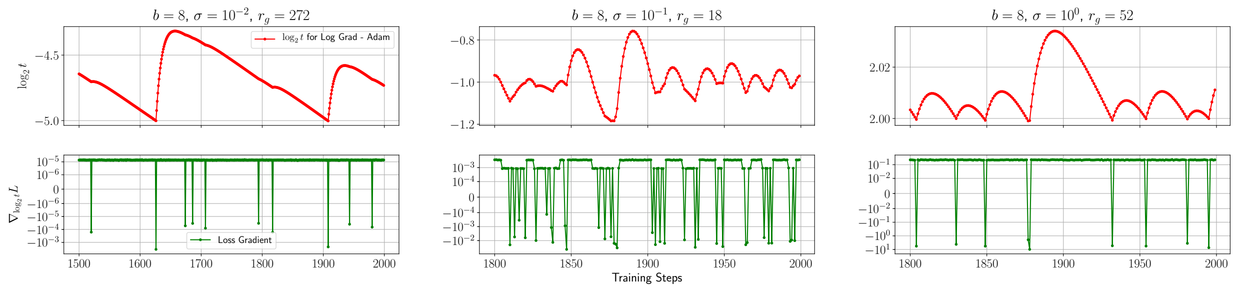

Figure 9. Close up of Figure 8 for the Adam-trained log threshold gradient on a few select settings.

Adam updates look like $\theta_i \leftarrow \theta_{i-1} - \alpha \cdot m_i/\sqrt{v_i}$ or $\theta_i \leftarrow \theta_0 - \alpha \sum_{j=0}^{i} m_j/\sqrt{v_j}$. We can solve for T by finding when $\theta_T = \theta_0$ or $\sum_{i=0}^{T} m_i/\sqrt{v_i} = 0$. As an intermediate step, we find:

$$\begin{aligned}\Delta_t\theta &= \sum_{i=0}^{t} \frac{m_i}{\sqrt{v_i}} \\ &= \sum_{i=0}^{t} \frac{\beta_1^i\left(\beta_1 \frac{\beta_1^T(1-\beta_1)-(1-\beta_1^T)/r_g}{1-\beta_1^{T+1}} + 1 - \beta_1 + \frac{1}{r_g}\right) - \frac{1}{r_g}}{\sqrt{1/r_g^2 + 1/T}} \\ &= \frac{1}{\sqrt{\frac{1}{r_g^2}+\frac{1}{T}}}\left[\frac{1-\beta_1^{t+1}}{1-\beta_1}\left(\beta_1 \frac{\beta_1^T(1-\beta_1)-(1-\beta_1^T)/r_g}{1-\beta_1^{T+1}}\right.\right. \\ &\qquad \left.\left. + 1 - \beta_1 + \frac{1}{r_g}\right) - \frac{t+1}{r_g}\right] \qquad (21)\end{aligned}$$

Now, we set $\Delta_T\theta = 0$:

$$\begin{aligned}0 &= \frac{1}{\sqrt{\frac{1}{r_g^2}+\frac{1}{T}}}\left[\frac{1-\beta_1^{T+1}}{1-\beta_1}\left(\beta_1 \frac{\beta_1^T(1-\beta_1)-(1-\beta_1^T)/r_g}{1-\beta_1^{T+1}}\right.\right. \\ &\qquad \left.\left. + 1 - \beta_1 + \frac{1}{r_g}\right) - \frac{T+1}{r_g}\right] \\ &= \beta_1^{T+1} - \frac{\beta_1(1-\beta_1^T)}{r_g(1-\beta_1)} + 1 - \beta_1^{T+1} + \frac{1-\beta_1^{T+1}}{r_g(1-\beta_1)} - \frac{T+1}{r_g} \\ T &= r_g \qquad (22)\end{aligned}$$

The worst case happens when r_g is large, so if we substitute $T \leftarrow r_g$ and assume $r_g \gg 1$, we get:

$$\begin{aligned}\Delta_t\theta &\approx \sqrt{r_g}\left[\frac{1-\beta_1^{t+1}}{1-\beta_1}\left(\beta_1 \frac{\beta_1^{r_g}(1-\beta_1)-(1-\beta_1^{r_g})/r_g}{1-\beta_1^{r_g+1}}\right.\right. \\ &\qquad \left.\left. + 1 - \beta_1 + \frac{1}{r_g}\right) - \frac{t+1}{r_g}\right] \qquad (23) \\ &= \sqrt{r_g}\left[\frac{1-\beta_1^{t+1}}{1-\beta_1}c_1 - \frac{t+1}{r_g}\right] \qquad (24)\end{aligned}$$

where we replace the large expression in (23) with c_1 in (24). We now solve for the critical point of $\Delta_t\theta$ to determine $t_{max} = \text{argmax}_t \Delta_t\theta$.

$$\begin{aligned}0 &= \frac{d}{dt}\Delta_t\theta \\ &= \sqrt{r_g}\left[\frac{\ln(\beta_1^{-1})\beta_1^{t_{max}+1}}{1-\beta_1}c_1 - \frac{1}{r_g}\right] \\ \beta_1^{t_{max}+1} &= \frac{1}{\ln(\beta_1^{-1})}\frac{1-\beta_1}{r_g \cdot c_1} \qquad (25) \\ &= \frac{1}{\ln(\beta_1^{-1})}\frac{1-\beta_1^{r_g+1}}{1+r_g} \\ t_{max} &= \log_{\beta_1}\left(\frac{1}{\ln(\beta_1^{-1})}\frac{1-\beta_1^{r_g+1}}{1+r_g}\right) - 1 \qquad (26)\end{aligned}$$

Plugging (25) and (26) into (24),

$$\begin{aligned}\Delta_{t_{max}}\theta \approx \sqrt{r_g}\left[\frac{c_1}{1-\beta_1} - \frac{1}{r_g\ln(\beta_1^{-1})}\right. \\ \left. - \frac{1}{r_g}\log_{\beta_1}\left(\frac{1}{\ln(\beta_1^{-1})}\frac{1-\beta_1^{r_g+1}}{1+r_g}\right)\right] \qquad (27)\end{aligned}$$

To simplify this expression, note that $\beta_1 < 1$ and $r_g \gg 1$ so $1 - \beta_1^{r_g} \approx 1$. Then $c_1/(1-\beta_1) \approx 1 + 1/r_g \approx 1$ and:

$$\Delta_{t_{max}}\theta \approx \sqrt{r_g}\left[1 + \frac{1+\ln(r_g \ln\beta_1^{-1})}{r_g \ln\beta_1}\right] \qquad (28)$$

Further, if $1/e < \beta_1 < 1$, then the right term is negative and the expression has a simple upper bound:

$$\Delta_{t_{max}}\theta < \sqrt{r_g} \tag{29}$$

In practice, we notice that sometimes noise can can cause θ to stay on the high-gradient side of the threshold boundary for multiple steps, causing the momentum to build up. Thus, to be safe, we recommend designing for $\Delta_{t_{max}}\theta < 10\sqrt{r_g}$.

A rough estimate for the number of steps needed for convergence is $\mathcal{O}(\Delta\lceil\log_2 t\rfloor/(\alpha|\tilde{g}|))$. Because of adaptive gradients, $|\tilde{g}|$ should be close to 1, provided we allow enough time for historical variance to decay - $\mathcal{O}(1/(1-\beta_2))$ steps[9]. Thus, the overall number of steps would be $\mathcal{O}(\Delta\lceil\log_2 t\rfloor/\alpha + \Delta\lceil\log_2 t\rfloor/(1-\beta_2))$. Assuming calibration is used, $\Delta\lceil\log_2 t\rfloor$ should be close to 1, giving the simplified expression $\mathcal{O}(1/\alpha + 1/(1-\beta_2))$ steps.

Finally, we address how to approximate r_g. The operation of crossing a threshold boundary moves some fraction f of inputs $\{x_i\}$ from the $n \le \lfloor x/s\rceil \le p$ case to the $\lfloor x/s\rceil < n$ or $\lfloor x/s\rceil > p$ cases (assume only $\lfloor x/s\rceil > p$ for simplicity from here on). Using the toy L_2-loss model (9),

$$\nabla_{(\log_2 t)}L = s^2\ln 2 \cdot \begin{cases} \left(\left\lfloor\frac{x}{s}\right\rceil - \frac{x}{s}\right)^2 & \text{if } n \le \left\lfloor\frac{x}{s}\right\rceil \le p, \\ n(n-x/s) & \text{if } \left\lfloor\frac{x}{s}\right\rceil < n, \\ p(p-x/s) & \text{if } \left\lfloor\frac{x}{s}\right\rceil > p \end{cases} \tag{30}$$

we see that for any given x_i, the ratio r_{gi} between the gradients in the outer and inner cases is $p(p-x_i/s)/(\lfloor x_i/s\rceil - x_i/s)^2$. But since x_i recently switched cases, $(p - x_i/s) < 1$. As a rough estimate, we might expect $r_{gi} \approx (1/2p)/(1/12) \approx 6p$. Averaged over the entire input, $r_g \approx 6fp \lessapprox p$. The $10\times$ over-design helps address some uncertainty in this measure as well.

Figure 9 shows a re-run of Figure 8 for the case of Adam optimization on log threshold gradients. These plots allow us to validate our Adam convergence analysis above. First we note that $p = 2^{8-1} - 1 = 127$, which is an approximate upper bound on r_g and well within the $10\times$ over-design principle. Next, notice that $T \approx r_g$. For example, in the $\sigma = 10^{-2}$ case, $T \approx 280$ while $r_g \approx 272$.

Most importantly, we expect the max log-threshold deviation to be upper-bounded by $\alpha\sqrt{r_g} = (1.6, 0.4, 0.7)$ from left to right if our original assumptions hold - that we visit the lower threshold bin for one step and stay in the upper bin for $T-1$ steps. While the bound holds for all σ, it is close to not holding for $\sigma = 10^{-1}$. A brief inspection reveals why this is the case - the log threshold spends far more than one step in the lower threshold bin per period, violating our one-step assumption. This violation can be explained by looking at the gradients, which show that the lower threshold bin sometimes has positive gradients, depending on the randomness of the input Gaussian vector. These phenomena motivate our suggestion to over-design by $10\times$. The cost in additional steps needed to reach convergence seems like a worthwhile trade-off.

[9] This is a problem when historical gradient magnitudes were higher, as is usually the case when $\Delta\lceil\log_2 t\rfloor < 0$, as seen in the small σ plots of Figure 8.

D Best or Mean Validation

We run validation every 1000 training steps and save the best top-1 score checkpoint. This approach was initially driven by a desire to better understand convergence and stability properties with our method, but we continued using it since intermediate validation was not too expensive for 5 epochs of retraining. However a valid concern is that this intermediate validation introduces a positive bias to our results through cherry-picking. To quantify this, we compare the positive-biased validation method to simply taking the average of validation scores at fixed intervals: 20%, 40%, 60%, 80% and 100% of the fifth epoch. As noted in Table 5, the differences between these methods on the top-1 accuracy are 0.1% and 0.2% for MobileNet v1 and VGG 16 respectively, suggesting that cherry-picking only results in a minor positive bias on our reported accuracy.

Table 5. Best validation (cherry-picked) is compared to the average of five validations (at pre-determined steps) in the last epoch, for two networks.

	Accuracy (%)		Epochs
	top-1	top-5	
MobileNet v1 1.0 224			
	70.982	89.886	4.2
	70.986	89.860	4.4
	71.076	89.930	4.6
	71.000	89.870	4.8
	71.022	89.944	5.0
Mean	**71.0**	**89.9**	
Best	**71.1**	**90.0**	2.1
VGG 16			
	71.448	90.438	4.2
	71.462	90.456	4.4
	71.434	90.436	4.6
	71.500	90.426	4.8
	71.458	90.456	5.0
Mean	**71.5**	**90.4**	
Best	**71.7**	**90.4**	0.9