LEARNED STRAIGHT-THROUGH ESTIMATORS FOR BINARIZED NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Binarized neural networks (BNNs) training requires an approximation of the gradient of the sign function, whose derivative is zero almost everywhere. Current methods replace the sign gradient with some hand-crafted surrogate, such as the straight-through estimator (STE) and its variants. The STE shape is decided in advance, whether held constant or varied by a progressive schedule; this is despite the distributions it acts change over layers and epochs. To address this, we propose the Learned STE (LSTE), which lets the training objective tune the shape of the STE in every quantizer. Specifically, we give each weight quantizer and each activation quantizer one learnable parameter that controls the shape of their STE. These parameters act only in the backward pass, so the current mini-batch gives them no gradient; we train them instead on the loss of the next mini-batch, which depends on the weight update they shape. The forward pass is unchanged, so LSTE combines with different forward quantizers and adds no cost at inference. The same mechanism also selects between published estimators by learning the ratio that mixes them, and shows that the preferred one depends on the quantizer. Experiments on CIFAR-10, ImageNet, and 1-bit language models of the Llama family trained on C4, show that LSTE improves on state-of-the-art methods.

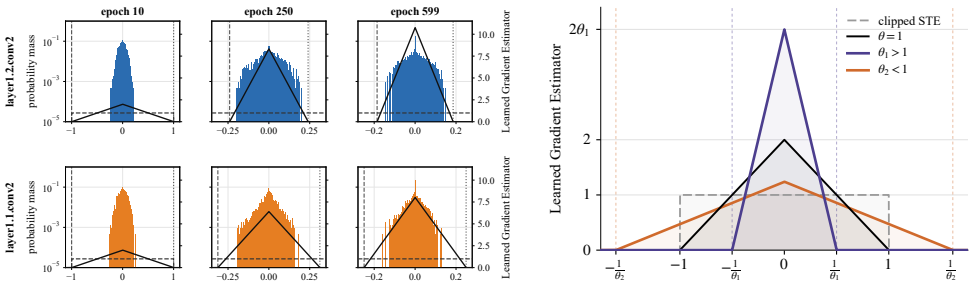
1 INTRODUCTION

Deep neural networks have advanced many vision and language tasks (He et al., 2016; Touvron et al., 2023), at a cost in memory and computation that resource-constrained devices cannot meet. Quantization reduces costs both by representing weights and activations with fewer bits, and it leaves the architecture unchanged (Jacob et al., 2018). Binarization is its extreme form: with 1-bit weights and 1-bit activations, a convolutional layer needs up to $32\times$ less memory, and XNOR and bitcount operations compute its convolution up to $58\times$ faster (Rastegari et al., 2016), compared to the 32-bit version. Quantization-aware training trains such a network from scratch, with a quantized forward pass and a full-precision backward pass (Jacob et al., 2018). Early work trains binarized neural networks (BNNs) this way for image classification (Courbariaux et al., 2015; Hubara et al., 2016), and recent work trains language models the same way (Wang et al., 2023; Panferov et al., 2025).

A gap in accuracy remains between BNNs and their full-precision counterparts (Rastegari et al., 2016; Xu et al., 2021), and part of it arises in the backward pass. The derivative of the sign function is zero almost everywhere, so backpropagation passes no gradient to the real-valued weights and activations that it binarizes. A gradient estimator replaces this derivative during training: the straight-through estimator (STE) passes the gradient unchanged (Bengio et al., 2013), and later gradient estimators clip it, reshape it, or rescale it (Hubara et al., 2016; Liu et al., 2018; Qin et al., 2020; Xiang et al., 2024). Prior work designs each of these gradient estimators by hand before training and gives the same one to every quantizer of the network; a schedule (Qin et al., 2020; Wu et al., 2023) may change the gradient estimator over time, but in the same way in every layer and in both quantizer types.

The shape of a gradient estimator decides how much gradient each real-valued weight and activation receives. Two features of that shape stand out in the gradient estimators in common use: the support, the range of values outside which it passes no gradient at all, and the peak at zero, the largest gradient it passes. A value outside the support receives no gradient, so it barely changes and its sign rarely flips, which Xu et al. (2021) and Xiang et al. (2024) identify as a cause of slow convergence. The real-valued weights and activations, however, follow distributions that differ across layers and move

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.



(a) Learned estimators and the weights they act on.

(b) Effect of the learned parameter.

Figure 1: (a) The distribution of the real-valued weights that a quantizer binarizes, with the learned gradient estimator that acts on them, for two weight quantizers at three epochs during training. Both start from the same estimator and reach different shapes, each one matched to the values that its own layer holds. (b) The learned gradient estimator for three settings of its parameter θ: a larger θ raises its peak and narrows its support, trading the gradient near zero against the range that receives it. The dashed rectangle is the clipped STE.

during training, so no single gradient estimator suits every quantizer at every iteration. Figure 1a illustrates this for the weight quantizers of two layers: their distributions differ, and the gradient estimator that our method learns settles on a different shape in each of them.

We would therefore like the training objective itself to choose the gradient estimator of each quantizer, but standard backpropagation provides no signal for this choice: a parameter of the forward pass changes the output of its layer, so the current loss depends on it, and backpropagation supplies its gradient; in contrast, a parameter of a gradient estimator acts in the backward pass alone, its gradient is exactly zero, and gradient descent on the current loss leaves it where it started.

This paper proposes the learned straight-through estimator (LSTE), a gradient estimator whose shape the training objective learns, separately in every quantizer. We give one learnable parameter to the weight quantizer and one to the activation quantizer of every binarized layer. The parameter controls the shape of the gradient estimator, raising its peak and narrowing its support as it grows (Figure 1b), and at its initial value the gradient estimator is the piecewise linear one of Liu et al. (2018). We train these parameters on the loss of the next mini-batch: they shape the weight update of the current mini-batch, and that update decides the weights at which we evaluate the next one, so we keep the computational graph of the update and differentiate the next loss through it. Keeping only the last update makes the rule a one-step hypergradient (Luketina et al., 2016; Baydin et al., 2018), with the short-horizon bias that Wu et al. (2018) analyze. LSTE changes the backward pass alone, so it applies to any forward quantizer that binarizes with the sign function, and it adds no parameter and no operation at inference.

The same construction trains any parameter that acts in the backward pass alone, which lets us ask which published gradient estimator the training objective prefers: we mix the clipped STE with the piecewise linear gradient estimator, hold both of them fixed, and learn only the mixing ratio between them. The answer differs between the two forward quantizers we study, and between the two quantizer types of one network.

In summary, the main contributions of this work are as follows.

- We propose LSTE, which gives every quantizer one learnable parameter that controls the shape of its gradient estimator, and we train these parameters on the loss of the next mini-batch through the kept weight update.
- We introduce a learned mixing ratio between two published gradient estimators, and we find that the preferred one depends on the forward quantizer and can differ between the two quantizer types of one network.
- Extensive experiments demonstrate that LSTE achieves state-of-the-art performance across three standard benchmarks for BNN training. Specifically, a LSTE-trained binarized ResNet-18 obtains 62.4% top-1 accuracy on ImageNet, surpassing the previous state-of-the-art methods in the same

architecture ($+0.4\%$ over SURGE and $+1.0\%$ over RBONN), and LSTE lowers the validation loss of 1-bit Llama models on C4 at every model size.

2 Related Work

BinaryConnect binarizes the weights of a network and keeps real-valued weights for the update (Courbariaux et al., 2015), and Hubara et al. (2016) binarize the activations as well. The sign function keeps only the sign of each entry and discards its magnitude, so the binarized weights and activations differ from the real-valued ones. A first direction of work limits the effect of this quantization error in the forward pass, by scaling the binarized weights (Rastegari et al., 2016), by transforming the weights before the sign function (Qin et al., 2020; Xu et al., 2021), or by adding a real-valued shortcut around each binarized convolution (Liu et al., 2018). Other methods in this direction train a parameter of the quantizer, such as the scaling factor, the clipping level, or the step size (Bulat & Tzimiropoulos, 2019; Choi et al., 2018; Esser et al., 2020). Each of these parameters appears in the forward pass, so backpropagation provides its gradient. For language models, BitNet binarizes the weights (Wang et al., 2023) or ternarizes them (Ma et al., 2024), and QuEST binarizes the weights and the activations after a Hadamard transform (Panferov et al., 2025).

A second direction of work acts on the backward pass. The derivative of the sign function is zero almost everywhere, so backpropagation passes no gradient to the real-valued weights and activations. A gradient estimator substitutes a coefficient for this derivative. The STE uses the identity (Bengio et al., 2013), whose behaviour Yin et al. (2019) analyze, the clipped STE sets the gradient to zero outside $[-1, 1]$ (Hubara et al., 2016), and later gradient estimators use the derivative of a differentiable approximation of the sign function, such as a piecewise polynomial (Liu et al., 2018) or a hyperbolic tangent (Qin et al., 2020). Some of these gradient estimators follow a schedule, and the authors design this schedule by hand before training and give the same one to every layer (Qin et al., 2020; Wu et al., 2023). Other methods rescale or correct the backpropagated gradient (Lee et al., 2021; Xiang et al., 2024; Huang et al., 2026), or apply the clipped STE after a Hadamard transform (Panferov et al., 2025). Each of these gradient estimators is fixed before training, and every layer uses the same one.

Closer to our work are methods that adapt the gradient estimator during training. They learn the sharpness or the slope of a soft approximation of the quantizer (Gong et al., 2019; Darabi et al., 2018), derive an adaptive STE from a bilevel formulation of training (Le et al., 2022), or estimate a backward gain for each group of weights from the local sensitivity of the weight quantizer (Yi et al., 2026). MetaQuant learns the gradient estimator of the weights with an auxiliary network that all layers share, and trains this network on the loss of the next mini-batch (Chen et al., 2019). To obtain the gradient of this loss with respect to the network, it backpropagates the loss through the weight quantizer with the clipped STE.

Unlike these methods, we add no auxiliary network and learn the gradient estimator of both the weight quantizers and the activation quantizers: we give each quantizer one learnable parameter that controls the shape of its gradient estimator. These parameters act only in the backward pass, so we train them on the loss of the next mini-batch, which depends on the weight update they shape, with a one-step hypergradient (Luketina et al., 2016; Baydin et al., 2018; Shaban et al., 2019; Chandra et al., 2022). We backpropagate this loss through each quantizer with its own learned gradient estimator rather than with a fixed STE.

3 Background

Forward pass. Consider the k-th binarized layer of a network. It holds a real-valued weight vector $W_k \in \mathbb{R}^d$ and receives an input activation vector $x_k \in \mathbb{R}^d$ computed by the previous layer. BNNs replace both vectors by their signs and, following XNOR-Net (Rastegari et al., 2016), compute

$$\widehat{W}_k = \operatorname{sign}(W_k) \in \{-1, +1\}^d, \qquad \widehat{x}_k = \operatorname{sign}(x_k) \in \{-1, +1\}^d, \tag{1a}$$

$$f_b(x_k; W_k) = \alpha_k \left(\widehat{W}_k \circledast \widehat{x}_k\right), \qquad \alpha_k = \frac{\|W_k\|_1}{d}, \tag{1b}$$

where $\circledast$ denotes the inner product of two binarized vectors, which XNOR and bitcount operations compute, and the scalar α_k minimizes the quantization error $\|W_k - \alpha \widehat{W}_k\|_2^2$ over $\alpha > 0$. As in

Bi-Real Net (Liu et al., 2018), the binarized activation carries no scale. Training performs the forward pass with $\widehat{W}_k$ and $\widehat{x}_k$ and updates the real-valued W_k.

For language models, QuEST (Panferov et al., 2025) applies an orthogonal Hadamard transform before the sign function, which brings the distribution of the entries closer to a Gaussian and preserves inner products. Let $H \in \mathbb{R}^{d\times d}$ denote the block-diagonal matrix whose blocks are normalized Hadamard matrices, let $v \in \mathbb{R}^d$ be a vector that a quantizer binarizes, one row of the weight matrix or one token of the input, and let the scalar $s^\star$ minimize the quantization error $\mathbb{E}\big[(\xi - s\ \mathrm{sign}(\xi))^2\big]$ over $s > 0$, where $\xi \sim \mathcal{N}(0,1)$ is a standard Gaussian random variable. QuEST computes

$$\rho = s^\star \cdot \mathrm{RMS}\big(Hv\big), \qquad z = \frac{Hv}{\rho}, \qquad \widehat{v} = \rho\, H^\top \mathrm{sign}(z), \tag{2}$$

where RMS denotes the root mean square of the entries of a vector.

Backward pass. The derivative of the sign function is zero almost everywhere, so backpropagation passes no gradient to W_k and x_k. Let $u \in \mathbb{R}$ denote an entry of W_k or x_k, called a latent value, let $\widehat{u} = \mathrm{sign}(u)$ be its binarized value, and let $\mathcal{L}$ denote the training loss. A gradient estimator replaces the derivative $\partial\widehat{u}/\partial u$ in the chain rule by a coefficient $c : \mathbb{R} \to \mathbb{R}$,

$$\frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial \widehat{u}} \cdot \frac{\partial \widehat{u}}{\partial u} \approx \frac{\partial \mathcal{L}}{\partial \widehat{u}} \cdot c(u). \tag{3}$$

The identity STE (Bengio et al., 2013) passes the gradient unchanged, $c(u) = 1$, and the clipped STE (Hubara et al., 2016) sets it to zero outside $[-1, 1]$, $c(u) = \mathbf{1}[|u| \le 1]$, where $\mathbf{1}[\cdot]$ denotes the indicator function. For the activations, Bi-Real Net (Liu et al., 2018) uses the derivative of a piecewise polynomial approximation F of the sign function,

$$c(u) = \frac{\partial F(u)}{\partial u} = \begin{cases} 2 + 2u, & \text{if } -1 \le u < 0, \\ 2 - 2u, & \text{if } 0 \le u < 1, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

In the backward pass of (2), QuEST treats ρ as a constant and applies the clipped STE with a clipping level $\tau > 0$, a hyperparameter, to every entry of z, the normalized vector in the Hadamard domain,

$$\frac{\partial \mathcal{L}}{\partial v} \approx H^\top \Big(\big(H\, \tfrac{\partial \mathcal{L}}{\partial \widehat{v}}\big) \odot c(z) \Big), \qquad c(u) = \mathbf{1}[|u| \le \tau]\,, \tag{5}$$

where $\odot$ denotes the elementwise product.

4 Method

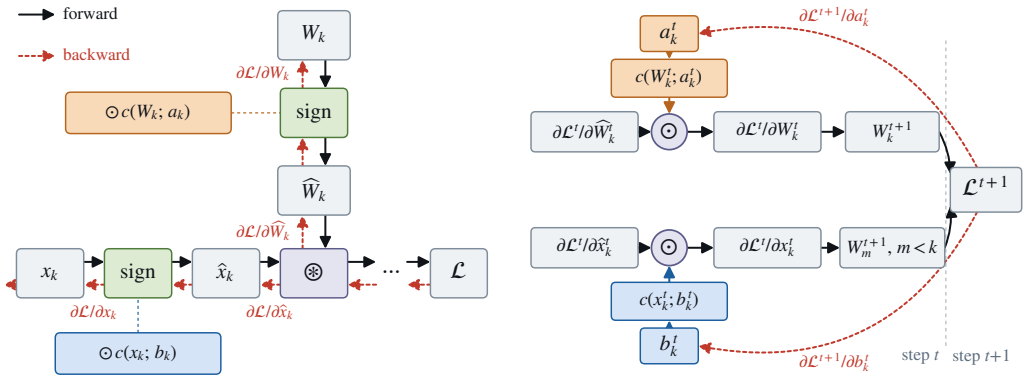
4.1 The Learned Gradient Estimator

The real-valued weights and activations have different distributions in different layers, and these distributions change during training (Figure 1a). Prior gradient estimators cannot follow them: either a fixed estimator applies the same coefficient in every quantizer at every iteration, or a hand-designed schedule changes that coefficient over time in the same way for all layers (Qin et al., 2020; Wu et al., 2023). Therefore, in neither case does the shape of the gradient estimator depend on how the latent values of that quantizer are distributed during training. LSTE therefore gives every quantizer its own parameter, which controls the shape of its gradient estimator and which we learn during training.

Definition. Consider a network with $N \in \mathbb{N}$ binarized layers, indexed by $k \in \{1, \dots, N\}$. For each such layer we introduce two learnable parameters $a_k, b_k \in \mathbb{R}_{>0}$, which we call the LSTE parameters of layer k. We write θ for either parameter, so that $\theta = a_k$ in a weight quantizer and $\theta = b_k$ in an activation quantizer. Attaching θ to the piecewise linear coefficient (4) gives the coefficient of the learned gradient estimator,

$$c(u;\theta) = \begin{cases} \theta\,(2 + 2\theta u), & \text{if } -1/\theta \le u < 0, \\ \theta\,(2 - 2\theta u), & \text{if } 0 \le u < 1/\theta, \\ 0, & \text{otherwise,} \end{cases} \quad = \quad \begin{cases} 2\theta\big(1 - \theta|u|\big), & \text{if } |u| < 1/\theta, \\ 0, & \text{otherwise,} \end{cases} \tag{6}$$

where $u \in \mathbb{R}$ is a latent value. The learned gradient estimator of layer k uses $c(\cdot\,; a_k)$ in its weight quantizer and $c(\cdot\,; b_k)$ in its activation quantizer (Section 4.2). At $\theta = 1$, the coefficient reduces to (4), and we initialize $a_k = b_k = 1$, so every quantizer starts from that fixed estimator.



(a) Forward and backward pass of a binarized layer.

(b) Two consecutive iterations.

Figure 2: (a) The forward pass (black) contains no LSTE parameter. In the backward pass (red), $c(W_k; a_k)$ and $c(x_k; b_k)$ replace the derivatives of the two sign functions (8). (b) Solid arrows carry the gradients of iteration t to the updated weights and to $\mathcal{L}^{t+1}$. Dotted arrows are the derivatives $\partial\mathcal{L}^{t+1}/\partial a_k^t$ and $\partial\mathcal{L}^{t+1}/\partial b_k^t$, which follow the same path backwards. The parameter a_k^t reaches $\mathcal{L}^{t+1}$ only through W_k^{t+1}, and b_k^t through W_m^{t+1} of every layer $m < k$.

Effect of the parameter. The LSTE parameter θ sets both the peak height 2θ at $u = 0$ and the support $|u| < 1/\theta$ (Figure 1b). A larger θ makes the coefficient taller and its support narrower, so more of the gradient reaches the latent values near zero and a latent value with $|u| > 1/\theta$ receives none, and a smaller θ makes the coefficient lower and its support wider, so latent values farther from zero keep receiving gradient. Its integral is 2 for every θ (Appendix D.2), so the parameter trades the gradient that reaches the latent values near zero against the gradient that reaches the rest, at a constant total.

4.2 The Weight Update

Figure 2a shows where the learned gradient estimator acts in the backward pass of a binarized layer. Let t index the training iterations and let $\mathcal{L}^t$ denote the training loss on the mini-batch of iteration t. For simplicity, we assume plain SGD (though any optimizer can be used) with learning rate $\eta > 0$,

$$W_k^{t+1} = W_k^t - \eta \frac{\partial \mathcal{L}^t}{\partial W_k^t}. \tag{7}$$

Following (3), the learned gradient estimator computes

$$\frac{\partial \mathcal{L}}{\partial W_k} = \frac{\partial \mathcal{L}}{\partial \widehat{W}_k} \odot \frac{\partial \widehat{W}_k}{\partial W_k} \approx \frac{\partial \mathcal{L}}{\partial \widehat{W}_k} \odot c(W_k; a_k), \qquad \frac{\partial \mathcal{L}}{\partial x_k} = \frac{\partial \mathcal{L}}{\partial \hat{x}_k} \odot \frac{\partial \hat{x}_k}{\partial x_k} \approx \frac{\partial \mathcal{L}}{\partial \hat{x}_k} \odot c(x_k; b_k). \tag{8}$$

Substituting the first estimate of (8) into (7) gives the weight update of LSTE, and expanding the gradient that it contains through the activation quantizer of layer $k+1$ gives, for $k < N$,

$$W_k^{t+1} = W_k^t - \eta \frac{\partial \mathcal{L}^t}{\partial \widehat{W}_k^t} \odot c\big(W_k^t; a_k^t\big), \qquad \frac{\partial \mathcal{L}^t}{\partial \widehat{W}_k^t} \approx \Big(\frac{\partial \mathcal{L}^t}{\partial \hat{x}_{k+1}^t} \odot c\big(x_{k+1}^t; b_{k+1}^t\big)\Big) \frac{\partial x_{k+1}^t}{\partial \widehat{W}_k^t}. \tag{9}$$

Repeating the second transition at layers $k+2, \ldots, N$ brings one coefficient $c(x_j^t; b_j^t)$ into the update for every layer $j > k$. The update of W_k therefore depends on the LSTE parameters only through a_k^t and $b_{k+1}^t, \ldots, b_N^t$ (Appendix D.4): a_k affects the update of W_k alone, whereas b_k affects the update of W_m for every layer $m < k$.

4.3 Learning the LSTE Parameters

The LSTE parameters appear only in the backward pass, so the loss of the current mini-batch does not depend on them and its derivatives with respect to them are zero (Appendix D.3). They shape the

Algorithm 1: One training iteration of LSTE.

Input: mini-batch of iteration t, weights W^t, LSTE parameters a^{t-1}, b^{t-1}, graph $\mathcal{G}^{t-1}$

1 Compute the loss $\mathcal{L}^t$ with W^t via Eq. (1a) and Eq. (1b)

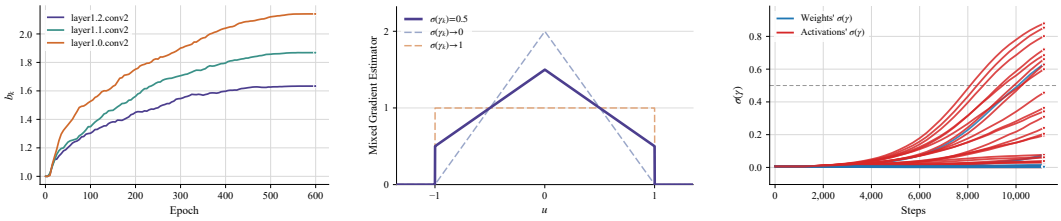
2 Compute $\partial\mathcal{L}^t/\partial a^{t-1}$ and $\partial\mathcal{L}^t/\partial b^{t-1}$ through $\mathcal{G}^{t-1}$ via Eq. (22) and Eq. (31)

3 Update a^t and b^t via Eq. (10)

4 Compute $\partial\mathcal{L}^t/\partial W^t$ via Eq. (8), and keep its graph $\mathcal{G}^t$

5 Update W^{t+1} via Eq. (9)

Output: W^{t+1}, a^t, b^t, $\mathcal{G}^t$



(a) b_k parameters during training. (b) The mixture (11). (c) The mixing ratio on C4.

Figure 3: (a) b_k of three quantizers of ResNet-20 on CIFAR-10 during training. (b) The mixture (11) (solid) and its two limits (dashed), the piecewise linear coefficient c_{br} and the clipped STE c_{clip}. (c) The mixing ratio $\sigma(\gamma_k)$ of the 30M language model on C4, for the weight quantizers (blue) and the activation quantizers (red).

update (9) of the weights, and the updated weights determine the loss $\mathcal{L}^{t+1}$ of the next mini-batch. Figure 2b shows this dependence over two consecutive iterations. We therefore update them by gradient descent on $\mathcal{L}^{t+1}$, with learning rates $\eta_a > 0$ and $\eta_b > 0$,

$$a_k^{t+1} = a_k^t - \eta_a \frac{\partial \mathcal{L}^{t+1}}{\partial a_k^t}, \qquad b_k^{t+1} = b_k^t - \eta_b \frac{\partial \mathcal{L}^{t+1}}{\partial b_k^t}. \tag{10}$$

After every update we clamp both parameters to a small positive value, so that the peak and the support in (6) stay defined. Since these derivatives are with respect to the loss of the next mini-batch, we keep the computational graph of the weight update (9), which we denote by $\mathcal{G}^t$ and which records W^{t+1} as a function of a^t and b^t. We compute $\mathcal{L}^{t+1}$ and its weight gradient with a^t and b^t before updating them. Appendices D.6 and D.7 derive $\partial\mathcal{L}^{t+1}/\partial a_k^t$ and $\partial\mathcal{L}^{t+1}/\partial b_k^t$ in detail.

Training procedure. Algorithm 1 describes one training iteration, where a^t and b^t collect the LSTE parameters of all layers. It uses two backward passes: the first continues through $\mathcal{G}^{t-1}$ and gives the gradients of the LSTE parameters, and the second computes the weight gradient with the updated parameters and records $\mathcal{G}^t$ for the next iteration.

5 Analysis

In this section, we introduce a mixture of the piecewise linear coefficient (4) and the clipped STE in the backward pass, and let the training objective decide which of the two it prefers in each quantizer. We first follow the LSTE parameters during training, and then use the mixture to select the gradient estimator of each quantizer type.

Trajectories of the LSTE parameters. Figure 3a follows b_k during training, and Appendix A.1 follows a_k and b_k together for every binarized layer. The quantizers separate during the first epochs and keep their order for the rest of training, so neither one parameter shared by all layers nor a schedule that changes every layer alike can reproduce these trajectories.

Mixing two gradient estimators. Let $c_{\mathrm{clip}}(u) = \mathbf{1}[|u| \le 1]$ denote the clipped STE (Hubara et al., 2016) and let $c_{\mathrm{br}}(u)$ denote the piecewise linear coefficient (4), where u is a latent value. For a

quantizer of layer k, we replace the coefficient in (8) by the mixture

$$c_{\text{mix}}(u;\gamma_k) = \sigma(\gamma_k)\, c_{\text{clip}}(u) + \big(1-\sigma(\gamma_k)\big)\, c_{\text{br}}(u), \qquad \sigma(\gamma_k) = \frac{1}{1+e^{-\gamma_k}}, \tag{11}$$

where $\gamma_k \in \mathbb{R}$ is a learned logit. The mixing ratio $\sigma(\gamma_k) \in (0,1)$ weights the two coefficients: the mixture approaches c_{br} as $\sigma(\gamma_k) \to 0$ and c_{clip} as $\sigma(\gamma_k) \to 1$ (Figure 3b). Like a_k and b_k, the logit appears only in the backward pass, so we learn it with the rules of Section 4.3, which apply unchanged (Appendix A.2).

Setup and results. The two coefficients are comparable: each one passes gradient on $|u| \leq 1$ and nowhere else, and both integrate to 2 (Appendix A.3), so the mixing ratio changes the shape of the coefficient and not how much gradient it passes. We set $\eta_a = \eta_b = 0$, so that the logit is the only parameter of the backward pass that changes. On the vision models, the mixing ratio of every quantizer ends below $\frac{1}{2}$, so LSTE builds on the piecewise linear coefficient in both quantizer types (Appendix A). Figure 3c reports the mixing ratio on the 30M model on C4, where we mix the weight quantizers and the activation quantizers in parallel and start every logit at $\gamma_k = -5$, so that the mixture starts at the coefficient that the vision models selected. The two quantizer types then separate: every weight ratio but one stays near 0, while the activation ratios spread upward and about half of them end above $\frac{1}{2}$. LSTE therefore trains the language models with the clipped STE in the activation quantizers and the piecewise linear coefficient in the weight quantizers (Section 6.3).

6 Experiments

6.1 Datasets and Implementation Details

Datasets. We evaluate LSTE on CIFAR-10 (Krizhevsky, 2009) and on ImageNet-1K (Russakovsky et al., 2015) for image classification, and on C4 (Raffel et al., 2020) for language modelling, where we train from scratch and give each model 100 training tokens per non-embedding parameter, following Panferov et al. (2025). More details of the datasets, the data augmentation and the evaluation metrics are provided in Appendix C.1.

Implementation details. On CIFAR-10 we evaluate LSTE with ResNet-18, ResNet-20 (He et al., 2016) and VGG-Small (Simonyan & Zisserman, 2015). On ImageNet we evaluate it with ResNet-18 and ResNet-34 (He et al., 2016), following the architecture of Bi-Real Net (Liu et al., 2018). With ResNet-18 we also follow the architecture of ReActNet (Liu et al., 2020). The two architectures add different numbers of real-valued parameters, so we compare within each of them, against the methods that report results on it. On C4 we evaluate it with decoder-only transformers of the Llama family (Touvron et al., 2023) of 30, 50, 100 and 200 million non-embedding parameters, where we keep the forward quantizer (2) of Panferov et al. (2025) and replace only their gradient estimator (5). More training details are provided in Appendices C.2 and C.3.

6.2 Image Classification

CIFAR-10. Table 1 displays the performance comparison on CIFAR-10 with the ResNet-18, ResNet-20 and VGG-Small backbones. We compare LSTE with DoReFa, XNOR-Net, DSQ, IR-Net, RBNN, ReCU, SURGE and OvSW, which Table 1 cites. We can see that LSTE outperforms all other methods on every backbone. Specifically, LSTE obtains a 0.1% performance increase over OvSW with ResNet-18, a 0.2% performance increase over SURGE with ResNet-20, and a 0.2% performance increase over OvSW with VGG-Small.

ImageNet. Table 2 displays the performance comparison on ImageNet with the ResNet-18 and ResNet-34 backbones. We compare LSTE with BNN, XNOR-Net, XNOR++, Bi-Real, IR-Net, RBNN, ReCU and OvSW in the Bi-Real architecture, and with RBONN and SURGE in the ReActNet architecture, which Table 2 cites. We can see that LSTE is leading in both the Top-1 and the Top-5 accuracy. Specifically, with ResNet-18 LSTE obtains a 0.2% Top-1 performance increase over OvSW in the Bi-Real architecture and a 0.4% performance increase over SURGE in the ReActNet architecture, and with ResNet-34 it obtains a 0.2% performance increase over OvSW.

Table 1: A performance comparison with state-of-the-art methods on CIFAR-10. W/A denotes the bit width of the weights and of the activations.

Backbone	Method	W/A	Top-1 (%)
ResNet-18	Full-precision	32/32	94.8
	IR-Net (Qin et al., 2020)	1/1	91.5
	RBNN (Lin et al., 2020)	1/1	92.2
	ReCU (Xu et al., 2021)	1/1	92.8
	SURGE (Huang et al., 2026)	1/1	93.1
	OvSW (Xiang et al., 2024)	1/1	93.2
	LSTE (ours)	1/1	**93.3**
ResNet-20	Full-precision	32/32	92.1
	DoReFa (Zhou et al., 2016)	1/1	79.3
	DSQ (Gong et al., 2019)	1/1	84.1
	IR-Net (Qin et al., 2020)	1/1	86.5
	ReCU (Xu et al., 2021)	1/1	87.4
	OvSW (Xiang et al., 2024)	1/1	87.7
	SURGE (Huang et al., 2026)	1/1	88.0
	LSTE (ours)	1/1	**88.2**
VGG-Small	Full-precision	32/32	94.1
	XNOR-Net (Rastegari et al., 2016)	1/1	89.8
	DoReFa (Zhou et al., 2016)	1/1	90.2
	IR-Net (Qin et al., 2020)	1/1	90.4
	RBNN (Lin et al., 2020)	1/1	91.3
	DSQ (Gong et al., 2019)	1/1	91.7
	ReCU (Xu et al., 2021)	1/1	92.2
	SURGE (Huang et al., 2026)	1/1	92.5
	OvSW (Xiang et al., 2024)	1/1	92.8
	LSTE (ours)	1/1	**93.0**

6.3 Language Modelling

Table 3 displays the performance comparison on C4, where we compare LSTE with QuEST (Panferov et al., 2025). On this forward quantizer we use LSTE to choose the gradient estimator of each quantizer rather than to reshape it during training: the learned mixing ratio of Section 5 settles on the piecewise linear coefficient (4) in the weight quantizers and on the clipped STE (5) in the activation quantizers. At 30 and 50 million parameters we train the baseline ourselves, so that the two columns differ only in the gradient estimator; at 100 and 200 million we read the baseline from the curve of Panferov et al. (2025). We can see that LSTE reduces the validation loss at each size of the model. Specifically, LSTE obtains a reduction of 0.029 at 30M, 0.031 at 50M, 0.034 at 100M and 0.040 at 200M.

6.4 Ablation Study

LSTE makes two changes to the backward pass: the piecewise linear coefficient (6) replaces the identity in the weight quantizer, and every quantizer learns the parameter of that coefficient. We abbreviate that coefficient as PWL in Table 4. We ablate each change on ImageNet with ResNet-18 in the ReActNet architecture. As shown in Table 4, the identity STE achieves 61.3% accuracy. Replacing the identity with the fixed coefficient improves performance by 0.3%, learning a_k alone and learning b_k alone each add another 0.5%, and learning both adds 0.8%, less than their sum because both shape the same weight update (9). More ablations, on CIFAR-10 and in the Bi-Real architecture and on the sensitivity to η_a and η_b, are provided in Appendix B.

Table 2: A performance comparison with state-of-the-art methods on ImageNet. W/A denotes the bit width of the weights and of the activations. We report the Top-1 (%) and Top-5 (%) accuracy performances. * denotes the ReActNet architecture with one-stage training.

Backbone	Method	W/A	Top-1 (%)	Top-5 (%)
ResNet-18	Full-precision	32/32	69.6	89.2
	BNN (Hubara et al., 2016)	1/1	42.2	67.1
	XNOR-Net (Rastegari et al., 2016)	1/1	51.2	73.2
	Bi-Real (Liu et al., 2018)	1/1	56.4	79.5
	IR-Net (Qin et al., 2020)	1/1	58.1	80.0
	RBNN (Lin et al., 2020)	1/1	59.9	81.9
	ReCU (Xu et al., 2021)	1/1	61.0	82.6
	OvSW (Xiang et al., 2024)	1/1	61.6	83.1
	LSTE (ours)	1/1	**61.8**	**83.3**
	RBONN* (Xu et al., 2022)	1/1	61.4	83.5
	SURGE* (Huang et al., 2026)	1/1	62.0	83.7
	LSTE * (ours)	1/1	**62.4**	**83.9**
ResNet-34	Full-precision	32/32	73.3	91.3
	XNOR++ (Bulat & Tzimiropoulos, 2019)	1/1	57.1	79.9
	Bi-Real (Liu et al., 2018)	1/1	62.2	83.9
	IR-Net (Qin et al., 2020)	1/1	62.9	84.1
	RBNN (Lin et al., 2020)	1/1	63.1	84.4
	ReCU (Xu et al., 2021)	1/1	65.1	85.8
	OvSW (Xiang et al., 2024)	1/1	65.5	86.1
	LSTE (ours)	1/1	**65.7**	**86.3**

Table 3: Validation loss of Llama models trained from scratch on C4 with 1-bit weights and 1-bit activations, where lower is better. * denotes that the result is read from a published figure.

Size	QuEST	QuEST + LSTE (ours)
30M	3.938	**3.909**
50M	3.791	**3.760**
100M	3.602*	**3.568**
200M	3.423*	**3.383**

Table 4: Ablation study on ImageNet with ResNet-18 in the ReActNet architecture. Each column gives the estimator of one quantizer: Identity STE is $c(u) = 1$, Fixed PWL is (6) at $\theta = 1$, and Learned PWL learns θ.

Weight	Activation	Top-1 (%)
Identity STE	Fixed PWL	61.3
Fixed PWL	Fixed PWL	61.6
Learned PWL	Fixed PWL	62.1
Fixed PWL	Learned PWL	62.1
Learned PWL	Learned PWL	**62.4**

7 Conclusion

We proposed LSTE, a gradient estimator for binarized networks whose shape the training objective learns in every quantizer. One learnable parameter per quantizer sets the peak and the support of the estimator, and because that parameter acts only in the backward pass, we train it on the loss of the next mini-batch through the kept weight update. Applied to the ratio that mixes two published estimators, the same construction reports which of them each quantizer prefers, and the answer differs between the two forward quantizers we study. LSTE improves on state-of-the-art methods on CIFAR-10, on ImageNet and on 1-bit Llama models trained on C4.

AI use statement

We use generative AI tools for two tasks. We use them to edit the paper for readability, after we have written every section ourselves, and we use them to clean and refactor research code that we wrote. We do not use generative AI tools for any task on the required-disclosure list: the method, the derivations, the proofs, the hypotheses, the implementation, the experimental design and the interpretation of the results are our own, and the remaining required-disclosure tasks do not apply to

this work. We have reviewed all AI-assisted work: we accept or reject every wording suggestion, and we verify and test every line of code that an AI tool has touched. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

We have read the ICLR Code of Ethics and we adhere to it. Our experiments use the public CIFAR-10, ImageNet-1K and C4 datasets, and they involve no human subjects and no personal data.

REPRODUCIBILITY STATEMENT

The paper describes LSTE in full. Section 4 defines the learned gradient estimator and the rules that update its two parameters, and Appendix D derives those rules step by step, together with the assumptions that they require. Section 6.1 states the experimental setting, and Appendix C gives the full details: the datasets, the data augmentation, the architectures, the training recipes and every hyper-parameter. Appendix B reports the additional ablations, and Appendix A gives the settings of the analysis. We release our code upon acceptance.

REFERENCES

Atilim Gunes Baydin, Robert Cornish, David Martínez-Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations (ICLR)*, 2018.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Adrian Bulat and Georgios Tzimiropoulos. XNOR-Net++: Improved binary neural networks. In *British Machine Vision Conference (BMVC)*, 2019.

Kartik Chandra, Audrey Xie, Jonathan Ragan-Kelley, and Erik Meijer. Gradient descent: The ultimate optimizer. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Shangyu Chen, Wenya Wang, and Sinno Jialin Pan. MetaQuant: Learning to quantize by learning to penetrate non-differentiable quantization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3916–3926, 2019.

Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. PACT: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3123–3131, 2015.

Sajad Darabi, Mouloud Belbahri, Matthieu Courbariaux, and Vahid Partovi Nia. BNN+: Improved binary network training. *arXiv preprint arXiv:1812.11800*, 2018.

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *International Conference on Learning Representations (ICLR)*, 2020.

Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4852–4861, 2019.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Haoyu Huang, Boyu Liu, Linlin Yang, Yanjing Li, Yuguang Yang, Xuhui Liu, Canyu Chen, Zhongqian Fu, and Baochang Zhang. SURGE: Surrogate gradient adaptation in binary neural networks. In *International Conference on Machine Learning (ICML)*, 2026.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 4107–4115, 2016.

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/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Huu Le, Rasmus Kjær Høier, Che-Tsung Lin, and Christopher Zach. AdaSTE: An adaptive straight-through estimator to train binary neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 460–469, 2022.

Junghyup Lee, Dohyung Kim, and Bumsub Ham. Network quantization with element-wise gradient scaling. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Mingbao Lin, Rongrong Ji, Zihan Xu, Baochang Zhang, Yan Wang, Yongjian Wu, Feiyue Huang, and Chia-Wen Lin. Rotated binary neural network. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. Bi-Real Net: Enhancing the performance of 1-bit CNNs with improved representational capability and advanced training algorithm. In *European Conference on Computer Vision (ECCV)*, pp. 722–737, 2018.

Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. ReActNet: Towards precise binary neural network with generalized activation functions. In *European Conference on Computer Vision (ECCV)*, pp. 143–159, 2020.

Jelena Luketina, Mathias Berglund, Klaus Greff, and Tapani Raiko. Scalable gradient-based tuning of continuous regularization hyperparameters. In *International Conference on Machine Learning (ICML)*, 2016.

Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit LLMs: All large language models are in 1.58 bits. *arXiv preprint arXiv:2402.17764*, 2024.

Andrei Panferov, Jiale Chen, Soroush Tabesh, Mahdi Nikdan, and Dan Alistarh. QuEST: Stable training of LLMs with 1-bit weights and activations. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pp. 47820–47836, 2025.

Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. Forward and backward information retention for accurate binary neural networks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2250–2259, 2020.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *European Conference on Computer Vision (ECCV)*, pp. 525–542, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1723–1732, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Hongyu Wang, Shuming Ma, Li Dong, Shaohan Huang, Huaijie Wang, Lingxiao Ma, Fan Yang, Ruiping Wang, Yi Wu, and Furu Wei. BitNet: Scaling 1-bit transformers for large language models. *arXiv preprint arXiv:2310.11453*, 2023.

Xiao-Ming Wu, Dian Zheng, Zuhao Liu, and Wei-Shi Zheng. Estimator meets equilibrium perspective: A rectified straight through estimator for binary neural networks training. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 17055–17064, 2023.

Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. In *International Conference on Learning Representations (ICLR)*, 2018.

Jingyang Xiang, Zuohui Chen, Siqi Li, Qing Wu, and Yong Liu. OvSW: Overcoming silent weights for accurate binary neural networks. In *European Conference on Computer Vision (ECCV)*, 2024.

Sheng Xu, Yanjing Li, Tiancheng Wang, Teli Ma, Baochang Zhang, Peng Gao, Yu Qiao, Jinhu Lü, and Guodong Guo. Recurrent bilinear optimization for binary neural networks. In *European Conference on Computer Vision (ECCV)*, pp. 19–35, 2022.

Zihan Xu, Mingbao Lin, Jianzhuang Liu, Jie Chen, Ling Shao, Yue Gao, Yonghong Tian, and Rongrong Ji. ReCU: Reviving the dead weights in binary neural networks. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 5198–5208, 2021.

Kai Yi, Vignesh Vivekraja, Harshit Khaitan, and Steven Li. JacQuant: STE-free quantization-aware training via learned jacobian surrogates. *arXiv preprint arXiv:2605.25469*, 2026.

Penghang Yin, Jiancheng Lyu, Shuai Zhang, Stanley Osher, Yingyong Qi, and Jack Xin. Understanding straight-through estimator in training activation quantized neural nets. In *International Conference on Learning Representations (ICLR)*, 2019.

Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

A Analysis

This appendix supports Section 5. We give the LSTE parameters of every binarized layer (A.1), derive the update rule of the mixing logit (A.2), list the settings of the experiments (A.3), give the mixing ratio of every quantizer (A.4), repeat the experiment from other starting points (A.5), and report the language models (A.6).

A.1 Trajectories of Every Binarized Layer

Figure 4 places a_k and b_k of the three quantizers of Figure 3a side by side. Figure 5 repeats it with every binarized layer of ResNet-20 on CIFAR-10. Each parameter starts at 1 and reaches a value of its own. For the three quantizers of Figure 4, the weight parameter ends between 2.8 and 5.4 and the activation parameter between 1.6 and 2.1, so both grow, the coefficient of every quantizer becomes taller and narrower than the piecewise linear one, and the weight parameters travel further. Within the stage shown, the two quantizer types order themselves by depth in opposite directions: a_k increases with depth and b_k decreases with it, so no single schedule over depth describes both.

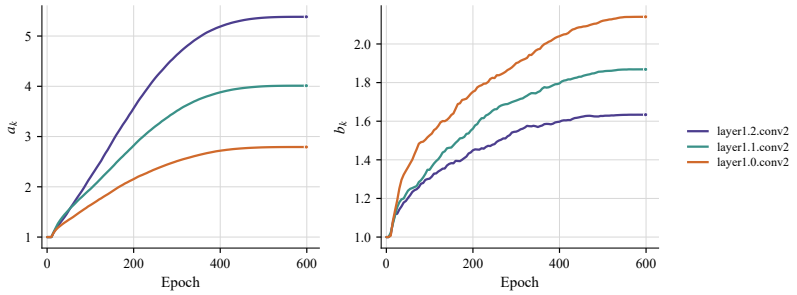
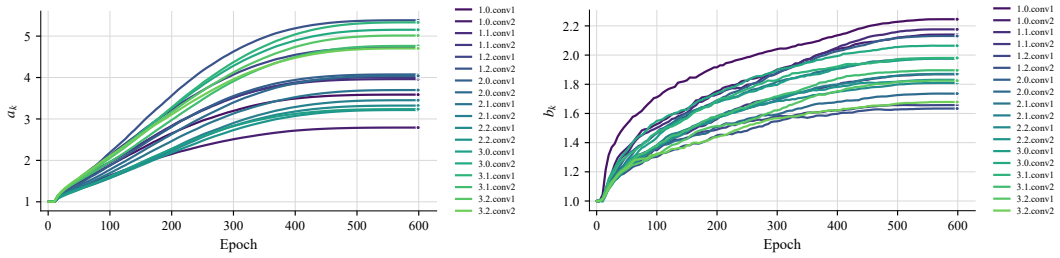


Figure 4: The LSTE parameters a_k (left) and b_k (right) of the three quantizers of Figure 3a, from their common initial value 1. The two panels use different learning rates, so their scales are not comparable.



(a) Weight quantizers.

(b) Activation quantizers.

Figure 5: The LSTE parameters of every binarized layer of ResNet-20 on CIFAR-10 during training, with $\eta_a = 7$ and $\eta_b = 0.05$. Every quantizer starts at $a_k = b_k = 1$.

A.2 Update Rule for the Mixing Logit

The logit γ_k appears only in the backward pass, so the argument of Lemma 3 gives $\partial \mathcal{L}^t / \partial \gamma_k^t = 0$. The logit enters the backward pass at the same place as the parameter of the same quantizer, so Lemma 4 also applies. The derivations of Propositions 5 and 6 therefore hold with $\partial c / \partial \theta$ replaced by $\partial c_{\text{mix}} / \partial \gamma_k$. For fixed u, the mixture (11) is affine in $\sigma(\gamma_k)$, and $\sigma'(\gamma) = \sigma(\gamma)\big(1 - \sigma(\gamma)\big)$, so

$$\frac{\partial c_{\text{mix}}}{\partial \gamma_k}(u; \gamma_k) = \sigma(\gamma_k)\big(1 - \sigma(\gamma_k)\big)\big(c_{\text{clip}}(u) - c_{\text{br}}(u)\big). \tag{12}$$

To show how the rule depends on the boundaries, we write the two branches with their own parameters, $c_{\text{clip}}(u; \theta_{\text{clip}}) = \theta_{\text{clip}}\, \mathbf{1}[|u| \le 1/\theta_{\text{clip}}]$ and the piecewise linear coefficient $c(u; \theta_{\text{br}})$ of (6), so that

(11) is the case $\theta_{\text{clip}} = \theta_{\text{br}} = 1$. Substituting the derivative into the common form (36) gives

$$\frac{\partial \mathcal{L}^{t+1}}{\partial \gamma_k^t} = -\eta\, \sigma'(\gamma_k^t) \sum_i D_i^{t+1}\, r_i^t \left(c_{\text{clip}}(v_i^t; \theta_{\text{clip}}) - c(v_i^t; \theta_{\text{br}})\right), \tag{13}$$

with $(v, r, D) = (W_k, g_k, G_k)$ for a weight quantizer and $(v, r, D) = (x_k, \delta_k, \Phi_k)$ for an activation quantizer. With the update $\gamma_k^{t+1} = \gamma_k^t - \eta_\gamma\, \partial \mathcal{L}^{t+1} / \partial \gamma_k^t$, written over the entries i of the quantizer,

$$\gamma_k^{t+1} = \gamma_k^t + \eta_\gamma \eta\, \sigma'(\gamma_k^t) \sum_{i\,:\,|v_i^t| \le \bar{u}} D_i^{t+1}\, r_i^t \left(c_{\text{clip}}(v_i^t; \theta_{\text{clip}}) - c(v_i^t; \theta_{\text{br}})\right), \tag{14}$$

where $\bar{u} = \max(1/\theta_{\text{clip}}, 1/\theta_{\text{br}})$. The sum runs over the union of the two supports, because the mixture is nonzero wherever either branch is. Where only one branch is nonzero, the difference has the sign of that branch: it is positive where only the clipped branch is nonzero and negative where only the piecewise linear branch is nonzero. In our experiments, $\theta_{\text{clip}} = \theta_{\text{br}} = 1$, so $\bar{u} = 1$.

For $|u| \le 1$, the difference in our experiments is $c_{\text{clip}}(u) - c_{\text{br}}(u) = 1 - (2 - 2|u|) = 2|u| - 1$, which changes sign at

$$|u| = \tfrac{1}{2}. \tag{15}$$

It is negative closer to zero, where the piecewise linear coefficient is larger, and positive farther out, where the clipped STE is larger. As in Appendix D.6, the product $D_i^{t+1} r_i^t$ is positive when the current update moves the latent value v_i in the direction that the next mini-batch also asks for. Such an agreeing latent value decreases γ_k, toward the piecewise linear coefficient, if $|v_i^t| < \frac{1}{2}$, and increases γ_k, toward the clipped STE, if $|v_i^t| > \frac{1}{2}$. In both cases, the logit moves toward the coefficient that gives this latent value the larger value, and a disagreeing latent value acts in the opposite direction. The factor $\sigma'(\gamma_k)$ is largest at $\gamma_k = 0$, so the logit changes fastest at that value, which is the initialization of the vision experiments.

A.3 Settings of the Experiments

Fixed coefficients. The implementation gives each branch of the mixture its own parameter, θ_{clip} for the clipped branch and θ_{br} for the piecewise linear branch. Since the mixture weights the two branches, it is nonzero on the union of their supports, $|u| \le \max(1/\theta_{\text{clip}}, 1/\theta_{\text{br}})$, and between $\min(1/\theta_{\text{clip}}, 1/\theta_{\text{br}})$ and $\max(1/\theta_{\text{clip}}, 1/\theta_{\text{br}})$ only the wider branch contributes. Our experiments set $\theta_{\text{clip}} = \theta_{\text{br}} = 1$ and $\eta_a = \eta_b = 0$, so neither parameter changes during training, both branches are nonzero exactly on $|u| \le 1$ (Figure 3b), and both integrate to 2.

Vision. We start every quantizer from $\gamma_k = 0$, so that $\sigma(\gamma_k) = \frac{1}{2}$ and neither estimator has an advantage, and we mix one quantizer type at a time. The weight experiment uses the mixture on all weight quantizers and the fixed piecewise linear coefficient on all activation quantizers, and the activation experiment does the reverse. With mixtures on both quantizer types, the logits of one type could respond to the coefficients favored by the other, and we could not separate the two results.

Learning rate. The logit has its own learning rate η_γ, and we set it to the learning rate of the parameter it replaces: $\eta_\gamma = 7.0$ in the weight experiment and $\eta_\gamma = 0.7$ in the activation experiment, the values that Table 7 lists for ResNet-20. Everything else follows the CIFAR-10 recipe of Appendix C.2, for 600 epochs.

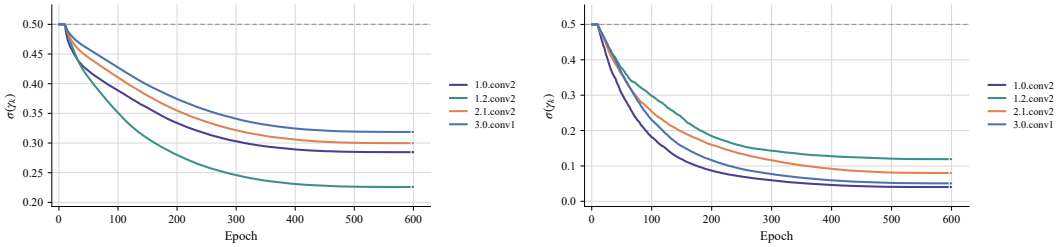
Active parameter update. With $\eta_a = \eta_b = 0$, the parameter update of Algorithm 1 must still run, because the logit is a learned parameter. We check that $\sigma(\gamma_k)$ leaves its initial value during training.

Language models. On the language models we mix the weight quantizers and the activation quantizers in parallel, in one run, and we start every logit from $\gamma_k = -5$, so that $\sigma(\gamma_k) \approx 0.007$ and the mixture starts at the piecewise linear coefficient, which the vision experiments select. We hold the logits fixed for the first 1100 iterations. Because both groups start at that coefficient, the activation result reports a move away from it and the weight result reports the absence of such a move.

Token embedding. On the language model, the token embedding is the only tensor below the activation quantizer of the first attention projection. The kept update contains the embedding, so the logit of that quantizer also receives a gradient (Appendix D.5).

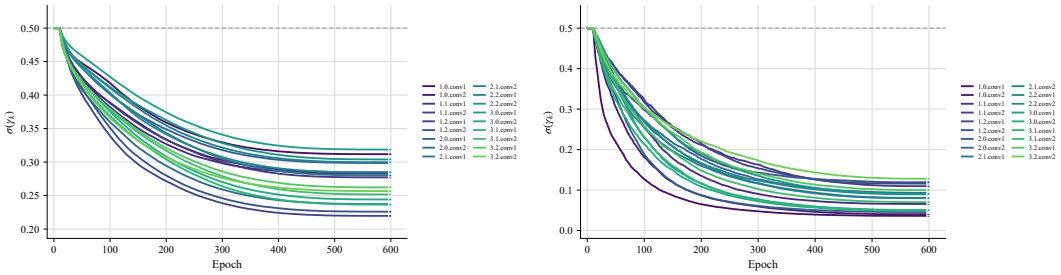
A.4 The Mixing Ratio of Every Quantizer

Figure 6 gives the mixing ratio of three quantizers of ResNet-20 on CIFAR-10, and Figure 7 repeats Figure 6 with every quantizer of ResNet-20. Every curve of both experiments ends below $\frac{1}{2}$: the weight ratios end between 0.22 and 0.32, and the activation ratios between 0.04 and 0.13. Both quantizer types therefore prefer the piecewise linear coefficient, and the activation quantizers prefer it more strongly.



(a) Weight quantizers. (b) Activation quantizers.

Figure 6: The mixing ratio $\sigma(\gamma_k)$ of (11) during training of ResNet-20 on CIFAR-10, from the common start $\frac{1}{2}$ (dashed). A ratio below $\frac{1}{2}$ weights the piecewise linear coefficient more than the clipped STE. Each curve is one quantizer, named by its layer.



(a) Weight quantizers. (b) Activation quantizers.

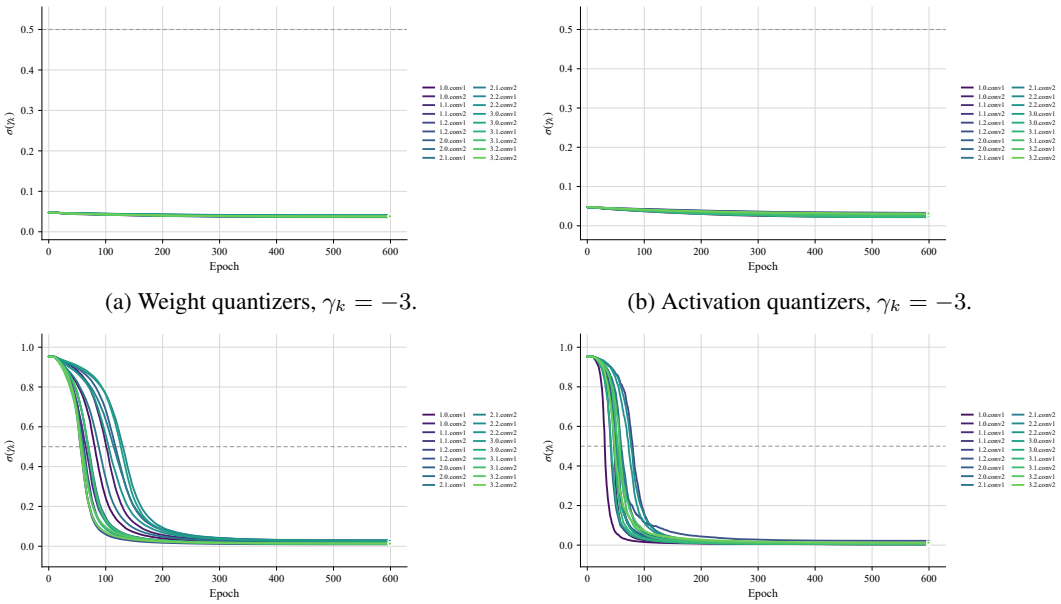
Figure 7: The mixing ratio $\sigma(\gamma_k)$ of every quantizer of ResNet-20 on CIFAR-10, starting from $\frac{1}{2}$ (dashed).

A.5 Other Starting Points

The initialization $\gamma_k = 0$ gives neither coefficient an advantage, and $\sigma'(\gamma_k)$ is largest there, so the logit also moves fastest there. To separate the result from the starting point, Figure 8 repeats both vision experiments from $\gamma_k = -3$, where $\sigma(\gamma_k) \approx 0.05$ and the mixture is close to the piecewise linear coefficient, and from $\gamma_k = 3$, where $\sigma(\gamma_k) \approx 0.95$ and the mixture is close to the clipped STE. From $\gamma_k = -3$ the ratio stays near zero and never crosses $\frac{1}{2}$. From $\gamma_k = 3$ every quantizer crosses $\frac{1}{2}$ within the first third of training and then approaches zero, so the two experiments reach the same side of $\frac{1}{2}$ from all three starting points.

A.6 Language Models

Figure 3c reports the mixing ratio of every quantizer of the 30M model on C4, where u is an entry of the rotated and normalized tensor z of (2). The weight quantizers and the activation quantizers are mixed in parallel, in one run, and every logit starts at $\gamma_k = -5$, so every curve starts at $\sigma(\gamma_k) \approx 0.007$, at the piecewise linear coefficient. The two groups then behave differently. Every weight ratio but one ends below 0.05, so the objective leaves the weight quantizers at the coefficient they started from; the exception ends near 0.6. The activation ratios spread over the whole range, from about 0.02 to about 0.9, and about half of them end above $\frac{1}{2}$, so the objective moves a large part of the activation quantizers toward the clipped STE and none of them further toward the piecewise linear coefficient.



(a) Weight quantizers, $\gamma_k = -3$.

(b) Activation quantizers, $\gamma_k = -3$.

(c) Weight quantizers, $\gamma_k = 3$.

(d) Activation quantizers, $\gamma_k = 3$.

Figure 8: The mixing ratio $\sigma(\gamma_k)$ of every quantizer of ResNet-20 on CIFAR-10 from two other starting points. The dashed line is $\frac{1}{2}$.

This is the opposite of the vision result of Appendix A.4, where every activation ratio ends below 0.13.

B Additional Ablations

Table 4 separates the two LSTE parameters on ImageNet with ResNet-18 in the ReActNet architecture. This appendix repeats that separation on the three CIFAR-10 backbones and on the Bi-Real architecture, and reports how sensitive the result is to the two learning rates η_a and η_b.

B.1 Which Quantizer the Gain Comes From

Tables 5 and 6 repeat Table 4 in the same form, on the three CIFAR-10 backbones and on the Bi-Real architecture.

On CIFAR-10 the three backbones agree with ImageNet. Learning the parameter of one quantizer alone adds between 0.3% and 0.7% over the fixed coefficient, and learning both adds between 0.5% and 1.0%, in every case less than the sum of the two separate gains. The same holds in the Bi-Real architecture on ImageNet, where the two separate gains are 0.4% and 0.5% and the joint gain is 0.7%. Across all five settings, then, each LSTE parameter improves the network on its own, the two together improve it further, and the two gains overlap rather than add.

The two ImageNet tables carry one caveat that the CIFAR-10 table does not. On CIFAR-10 every row of a block uses the learning rates of the last row of that block, so the rows differ only in which parameters are learned. On ImageNet the single-quantizer rows are tuned on their own (Table 7), so each of them gives the best result of its configuration rather than that configuration at the setting of the last row, and the separate gains are upper bounds on what each parameter contributes at the joint setting.

B.2 Learning Rates of the LSTE Parameters

The rules of Section 4.3 introduce two hyper-parameters, η_a and η_b, and nothing else. Table 7 lists the values we use. Two properties of the rules are visible in it. First, η_b is smaller than η_a in every setting,

Table 5: Gradient estimator of each quantizer on CIFAR-10, in the form of Table 4. *Fixed PWL* is (6) at $\theta = 1$, where it is the piecewise linear coefficient (4); *Learned PWL* is (6) with θ learned by (10). Δ is measured against the first row of each block. Every row uses the learning rates of Table 7, with the learning rate of the parameter that is held fixed set to zero.

Backbone	**Weight quant.**	**Activation quant.**	**Top-1 (%)**	Δ
ResNet-20	Fixed PWL	Fixed PWL	87.2	—
	Learned PWL	Fixed PWL	87.9	$+0.7$
	Fixed PWL	Learned PWL	87.9	$+0.7$
	Learned PWL	Learned PWL	**88.2**	$+1.0$
ResNet-18	Fixed PWL	Fixed PWL	92.6	—
	Learned PWL	Fixed PWL	93.1	$+0.5$
	Fixed PWL	Learned PWL	93.0	$+0.4$
	Learned PWL	Learned PWL	**93.3**	$+0.7$
VGG-Small	Fixed PWL	Fixed PWL	92.5	—
	Learned PWL	Fixed PWL	92.8	$+0.3$
	Fixed PWL	Learned PWL	92.9	$+0.4$
	Learned PWL	Learned PWL	**93.0**	$+0.5$

Table 6: Gradient estimator of each quantizer, on ImageNet with ResNet-18 in the Bi-Real architecture. The rows and the labels are those of Table 4.

Weight quantizer	**Activation quantizer**	**Top-1 (%)**
Identity STE	Fixed PWL	61.0
Fixed PWL	Fixed PWL	61.1
Learned PWL	Fixed PWL	61.5
Fixed PWL	Learned PWL	61.6
Learned PWL	Learned PWL	**61.8**

which agrees with Section 4.2: b_k enters the update of every layer below k, whereas a_k enters the update of layer k alone, so the same step size moves the loss of the next mini-batch further through b_k. Second, the rules carry the factor η of (9), so η_a and η_b are not comparable across settings that use different weight learning rates.

Table 8 varies the two learning rates together on CIFAR-10 with ResNet-20 over an order of magnitude in η_a and close to an order of magnitude in η_b. The three settings differ by 0.2%, and the weakest of them already matches the strongest prior method on this backbone.

Table 9 repeats the sweep on ImageNet with ResNet-18 in the Bi-Real architecture, over η_a from 1.0 to 40.0 and η_b from 0.05 to 0.30, under an otherwise identical recipe. The eight runs span 0.16% in Top-1 accuracy, from 61.70% to 61.86%, and every one of them exceeds the 61.6% of the strongest prior method on this backbone in Table 2. The result is therefore a property of the rules and not of a tuned learning rate. On ResNet-34, $\eta_a = 15.0$ with $\eta_b = 0.05$ reaches the same 65.7% as the setting of Table 7, which uses $\eta_a = 30.0$. The ImageNet sweeps are limited by the compute they require, so we report the settings we were able to run.

C Implementation Details

C.1 Datasets and Evaluation

On CIFAR-10 we train on the $50{,}000$ training images and report Top-1 accuracy on the $10{,}000$ test images. We augment each training image with a random crop of 32×32 pixels from the image padded by 4 pixels and with a random horizontal flip, and we normalize every channel with the ImageNet statistics. On ImageNet-1K we train on the 1.28 million training images and report Top-1

Table 7: Learning rates of the LSTE parameters. On CIFAR-10, the single-quantizer rows of Table 5 use the same values with the learning rate of the quantizer that is held fixed set to zero. On ImageNet, the single-quantizer rows of Tables 4 and 6 are tuned on their own and use $\eta_a = 20.0$ with $\eta_b = 0$, and $\eta_b = 0.005$ with $\eta_a = 0$.

Dataset	**Backbone and architecture**	η_a	η_b
CIFAR-10	ResNet-20	7.0	0.7
	ResNet-18	20.0	0.1
	VGG-Small	20.0	0.005
ImageNet	ResNet-18, Bi-Real	7.0	0.05
	ResNet-18, ReActNet	15.0	0.05
	ResNet-34, Bi-Real	30.0	0.05

Table 8: Accuracy on CIFAR-10 with ResNet-20 for three settings of η_a and η_b, under an otherwise identical recipe. The strongest prior method in Table 1 on this backbone reaches 88.0%.

η_a	η_b	**Top-1 (%)**
3.0	0.7	88.0
7.0	0.7	**88.2**
15.0	0.1	88.1

and Top-5 accuracy on the $50{,}000$ validation images. We augment each training image with a random resized crop to 224×224 pixels and a random horizontal flip, and at test time we resize the shorter side to 256 pixels and take the central crop of 224×224. On C4 we evaluate each model once on the whole validation set after the last training step, and we report the cross-entropy loss, the perplexity and the accuracy of the next token. The evaluations that Panferov et al. (2025) run during training use a subset of the validation set, so we do not quote them.

C.2 Vision

We binarize every convolutional layer and every fully connected layer except the first one and the last one, and on ImageNet the downsampling layers also stay real-valued. On CIFAR-10, ResNet-18 and ResNet-20 carry the double shortcut of Liu et al. (2018). We train the ReActNet architecture, which Table 2 marks with *, in one stage, as Xu et al. (2022) and Huang et al. (2026) do. That architecture adds a learned activation reshaping with its own real-valued parameters, which the Bi-Real architecture does not, so we compare within each architecture and not across the two. The forward quantizer follows Xu et al. (2021), which standardizes the weights before computing the scale of (1b); on the language models it follows Panferov et al. (2025), as (2) states.

We train every model from scratch with SGD and momentum 0.9, following Xu et al. (2021): the learning rate rises linearly to 0.1 over the first five epochs, and a cosine schedule anneals it afterwards. Table 10 lists the remaining hyper-parameters, and Table 7 gives η_a and η_b, the only hyper-parameters that LSTE adds. On ImageNet the two architectures differ in their weight decay, which we take from the methods each architecture is compared against.

C.3 Language

We train the language models on C4 tokenized with the Llama 2 tokenizer, over a vocabulary of $32{,}000$ entries. The recipe is the one of Panferov et al. (2025) without modification: AdamW with $\beta = (0.9, 0.95)$, a cosine schedule with 10% warmup, gradient clipping at 1.0, and decoupled weight decay 0.1 applied to every parameter except biases, normalization layers, the token embedding and the output head. The global batch is 512 sequences of 512 tokens, and training runs in `bfloat16` mixed precision. The number of optimization steps is set so that each model sees 100 tokens per non-embedding parameter. The token embedding and the output head stay real-valued; every other

Table 9: Accuracy on ImageNet with ResNet-18 in the Bi-Real architecture for eight settings of η_a and η_b, under an otherwise identical recipe, sorted by accuracy. The strongest prior method in Table 2 on this backbone and architecture reaches 61.6%.

η_a	η_b	**Top-1 (%)**
7.0	0.05	**61.86**
40.0	0.10	**61.86**
1.0	0.05	61.84
15.0	0.30	61.82
5.0	0.05	61.78
1.0	0.10	61.71
17.0	0.10	61.71
17.0	0.05	61.70

Table 10: Vision training hyper-parameters. The two ImageNet columns are the Bi-Real architecture and the ReActNet architecture, which Table 2 marks with *.

	CIFAR-10	**ImageNet**	
		Bi-Real	ReActNet
Epochs	600	200	200
Batch size	256	512	512
Initial learning rate η	0.1	0.1	0.1
Warmup epochs	5	5	5
Momentum	0.9	0.9	0.9
Weight decay	5×10^{-4}	1×10^{-4}	1×10^{-5}
Initialization	$a_k = b_k = 1$	$a_k = b_k = 1$	$a_k = b_k = 1$

linear layer is binarized on both operands. Table 11 lists the shape, the learning rate and the token budget of each size.

The gradient estimators on C4. Section 6.3 holds the LSTE parameters fixed at $a_k = b_k = 1$ and uses the two estimators that Section 5 selects. The activation quantizers use the clipped STE of (5) at the clipping level of Panferov et al. (2025), $\tau = 1.3$. The weight quantizers use the piecewise linear coefficient with a narrower support: we scale its boundary by a constant κ, so that the coefficient is $2a_k(1 - \kappa a_k|u|)_+$ on $|u| \leq 1/(\kappa a_k)$, which leaves the peak $2a_k$ of (6) unchanged and shrinks its support by κ. We use $\kappa = 1.5$ at every model size, so the weight support is $|u| \leq 2/3$ in the rotated and normalized domain of (2). The constant is fixed before training and is not learned, so the derivations of Appendix D apply unchanged with $\partial c/\partial\theta$ evaluated at the scaled boundary.

At 30M and 50M we train both the baseline and LSTE ourselves under this recipe rather than quoting the published numbers, so that the two columns of Table 3 differ only in the gradient estimator. As a check on the reproduction, our baseline run at 30M reaches a validation loss of 3.938 against the 3.945 reported by Panferov et al. (2025). At 100M and 200M we read the baseline from the scaling curve of Panferov et al. (2025).

Table 11: Shapes and hyper-parameters of the Llama-family models of Table 3. The sizes are numbers of non-embedding parameters.

Size	**Blocks**	d_{model}	**Heads**	**Learning rate**	**Tokens**
30M	6	640	5	1.2×10^{-3}	3B
50M	7	768	6	1.2×10^{-3}	5B
100M	8	1024	8	6×10^{-4}	10B
200M	10	1280	10	3×10^{-4}	20B

D DERIVATIONS

This appendix gives the full transitions that Section 4 shortens. We first fix the notation (D.1) and state the properties of the learned gradient estimator (D.2), which Proposition 1 and Lemma 2 collect. We then show that the learnable parameters have no gradient within a mini-batch (D.3), identify which of them shapes the gradient of each layer (D.4), and describe the kept update (D.5). Finally, we derive the rule for the weight parameter (D.6) and for the activation parameter (D.7), which Section 4.3 states, and write them as one rule (D.8).

D.1 NOTATION AND SETTING

The network has N binarized layers, indexed by $k \in \{1, \dots, N\}$ in the order of the forward pass, and a full-precision first layer, which we index by 0. We index training steps by t and write $\mathcal{L}^t$ for the loss on the mini-batch $\mathcal{B}^t$. Layer k holds the real-valued weights W_k^t and receives the real-valued input activations x_k^t, and the forward pass uses $\widehat{W}_k^t = \operatorname{sign}(W_k^t)$ and $\widehat{x}_k^t = \operatorname{sign}(x_k^t)$ as in (1a)–(1b). We treat each tensor as a vector of its entries, we index the entries of W_k^t by i and the entries of x_k^t by j over all samples of the mini-batch, so that a sum over i or over j runs over every entry that the corresponding quantizer binarizes. We compute the scale α_k^t of (1b) from W_k^t without a gradient path, so the backward pass treats it as a constant.

The coefficient $c(u;\theta)$ is (6), with $\theta = a_k$ for the weight quantizer and $\theta = b_k$ for the activation quantizer of layer k. Every statement below holds for $\theta > 0$, which the clamp of Section 4.3 maintains. We write

$$g_{k,i}^t = \frac{\partial \mathcal{L}^t}{\partial \widehat{W}_{k,i}^t}, \qquad \delta_{k,j}^t = \frac{\partial \mathcal{L}^t}{\partial \widehat{x}_{k,j}^t} \tag{16}$$

for the gradients that reach the two binarized tensors of layer k. The two quantizers multiply them by their coefficients, entry by entry,

$$G_{k,i}^t = \frac{\partial \mathcal{L}^t}{\partial W_{k,i}^t} = g_{k,i}^t \, c\big(W_{k,i}^t; a_k^t\big), \qquad s_{k,j}^t = \frac{\partial \mathcal{L}^t}{\partial x_{k,j}^t} = \delta_{k,j}^t \, c\big(x_{k,j}^t; b_k^t\big), \tag{17}$$

where G_k^t is the gradient of the real-valued weights and s_k^t is the gradient that the activation quantizer passes down. For every layer $m \in \{0, \dots, N\}$, we write $G_{m,i}^t = \partial \mathcal{L}^t / \partial W_{m,i}^t$, and the first layer has no coefficient. Plain gradient descent updates every layer as

$$W_{m,i}^{t+1} = W_{m,i}^t - \eta \, G_{m,i}^t. \tag{18}$$

Finally, we write $\mathcal{K} \subseteq \{0, \dots, N\}$ for the set of layers whose update the implementation keeps in the graph, which Section D.5 describes.

D.2 PROPERTIES OF THE LEARNED GRADIENT ESTIMATOR

Let $c_{\mathrm{br}}(v) = \partial F(v)/\partial v = 2(1 - |v|)_+$ denote the Bi-Real coefficient of (4).

Proposition 1 (Properties of the coefficient)**.** *For every $\theta > 0$, the coefficient (6) satisfies*

(i) $c(u;\theta) = \theta \, c_{\mathrm{br}}(\theta u)$ *for all* $u \in \mathbb{R}$*, and* $c(u;1) = c_{\mathrm{br}}(u)$*;*

(ii) $\int_{\mathbb{R}} c(u;\theta) \, \mathrm{d}u = 2$*;*

(iii) c *is continuous in* (u,θ)*, and* $c(\pm 1/\theta;\theta) = 0$*;*

(iv) for every bounded continuous function φ*,* $\int_{\mathbb{R}} c(u;\theta)\,\varphi(u)\,\mathrm{d}u \to 2\varphi(0)$ *as* $\theta \to \infty$*.*

Proof. For (i), $\theta \, c_{\mathrm{br}}(\theta u) = 2\theta(1 - \theta|u|)_+$, which equals (6) on each of its three cases. For (ii), the substitution $v = \theta u$ gives $\int_{\mathbb{R}} \theta \, c_{\mathrm{br}}(\theta u) \, \mathrm{d}u = \int_{\mathbb{R}} c_{\mathrm{br}}(v) \, \mathrm{d}v$, the area of a triangle with base 2 and height 2, which is 2. For (iii), both linear pieces of (6) equal 0 at $u = \pm 1/\theta$ and equal 2θ at $u = 0$. For (iv), the same substitution gives $\int_{\mathbb{R}} c_{\mathrm{br}}(v)\,\varphi(v/\theta)\,\mathrm{d}v$. The integrand converges to $c_{\mathrm{br}}(v)\varphi(0)$, and its absolute value does not exceed the integrable function $c_{\mathrm{br}}(v) \sup|\varphi|$, so dominated convergence gives the limit $2\varphi(0)$. □

Items (ii) and (iv) connect the coefficient to the sign function. The distributional derivative of the sign is 2δ, where δ is the Dirac delta, and its total mass is 2. By (ii), every coefficient in the family has this mass, and by (iv), the coefficient approaches 2δ as θ grows. The parameter therefore decides how the fixed mass spreads around zero, not how much mass there is.

Lemma 2 (Derivative with respect to the parameter)**.** *For every $\theta > 0$ and every u with $|u| \neq 1/\theta$, the derivative $\partial c/\partial\theta$ equals (20). The motion of the boundary $\pm 1/\theta$ adds no term to the derivative of $\int_{\mathbb{R}} c(u;\theta)\,\varphi(u)\,\mathrm{d}u$ for any continuous φ.*

Proof. Let $p(u;\theta) = 2\theta(1-\theta|u|)$, so that $c = p$ for $|u| < 1/\theta$ and $c = 0$ for $|u| > 1/\theta$. In the first region, $\partial c/\partial\theta = \partial p/\partial\theta = 2 - 4\theta|u| = 2(1-2\theta|u|)$, and in the second region the derivative is 0. For the integral, the Leibniz rule gives

$$\begin{aligned}\frac{\partial}{\partial\theta}\int_{-1/\theta}^{1/\theta} p(u;\theta)\,\varphi(u)\,\mathrm{d}u = &\int_{-1/\theta}^{1/\theta} \frac{\partial p}{\partial\theta}(u;\theta)\,\varphi(u)\,\mathrm{d}u \\ &- \frac{1}{\theta^2}\Big(p(1/\theta;\theta)\,\varphi(1/\theta) + p(-1/\theta;\theta)\,\varphi(-1/\theta)\Big),\end{aligned} \tag{19}$$

and the second term is zero because $p(\pm 1/\theta;\theta) = 0$. □

The derivative of the coefficient with respect to its parameter is

$$\frac{\partial c}{\partial\theta}(u;\theta) = \begin{cases} 2\big(1-2\theta|u|\big), & \text{if } |u| < 1/\theta, \\ 0, & \text{otherwise.}\end{cases} \tag{20}$$

It is positive for $|u| < 1/(2\theta)$ and negative for $1/(2\theta) < |u| < 1/\theta$. A larger θ therefore increases the coefficient of latent values closer to zero than $1/(2\theta)$ and decreases the coefficient of the other latent values inside the support. Automatic differentiation gives the derivative of a comparison as zero, so an implementation that writes the coefficient as a linear piece times the mask $\mathbf{1}[|u| < 1/\theta]$ computes exactly (20).

D.3 No Gradient Within a Mini-Batch

Lemma 3 (No gradient within a mini-batch)**.** *For every layer k and step t, $\partial\mathcal{L}^t/\partial a_k^t = 0$ and $\partial\mathcal{L}^t/\partial b_k^t = 0$. Moreover, the real-valued tensors W^t and x^t, the binarized tensors, the scales α^t, and the loss $\mathcal{L}^t$ do not depend on a^t or b^t.*

Proof. The forward pass of step t composes (1a), (1b), and the normalizations and nonlinearities between the layers. None of these operations contains a_k or b_k, so every forward quantity, including $\mathcal{L}^t$, is constant in a^t and b^t, and its derivatives with respect to them are zero. The weights W^t result from the update of step $t-1$, which uses a^{t-1}, b^{t-1} and earlier parameters, and a^t and b^t do not exist before the parameter update of step t. Hence W^t does not depend on them. □

Lemma 3 separates the parameters of LSTE from learned quantizer parameters such as the clipping level of PACT (Choi et al., 2018), the step size of LSQ (Esser et al., 2020) and the slope of BNN+ (Darabi et al., 2018), which enter a differentiable forward map and therefore receive a gradient from the current mini-batch. In the implementation, each quantizer is a custom autograd function whose backward pass multiplies the incoming gradient by the coefficient and returns no gradient for its parameter, which states Lemma 3 to the autograd engine. The backward pass still uses the parameter without detaching it, so the recorded graph of the weight gradient keeps its dependence on the parameter. Algorithm 2 states the two passes of one binarized layer, and the LSTE parameters appear in the second pass only.

D.4 Reach of the Learnable Parameters

Lemma 4 (Reach of the parameters)**.** *At every step t, the following statements hold.*

(i) The gradients g_k^t and δ_k^t can depend only on the activation parameters b_j^t of layers $j > k$.

(ii) The gradient G_k^t can depend only on a_k^t and on b_j^t for $j > k$.

Algorithm 2: One binarized layer k under LSTE.

Forward pass
- **Input:** weights W_k, input activations x_k
- $\widehat{W}_k \leftarrow \operatorname{sign}(W_k)$
- $\widehat{x}_k \leftarrow \operatorname{sign}(x_k)$
- $\alpha_k \leftarrow \|W_k\|_1/d$
- store W_k and x_k for the backward pass
- **Output:** $\alpha_k\,(\widehat{W}_k \circledast \widehat{x}_k)$

Backward pass
- **Input:** g_k, δ_k, the stored W_k and x_k, the LSTE parameters a_k and b_k
- $G_{k,i} \leftarrow g_{k,i}\,c(W_{k,i}; a_k)$ for every entry i
- $s_{k,j} \leftarrow \delta_{k,j}\,c(x_{k,j}; b_k)$ for every entry j
- **Output:** G_k to the optimizer, s_k to the layers below, and no gradient for a_k and b_k

(iii) The gradient G_m^t of any layer $m \in \{0, \dots, N\}$ can depend on b_k^t only if $m < k$.

Proof. The backward pass computes the gradient of a tensor as a sum over the paths from the loss to that tensor in the computational graph, and a parameter enters the gradient only if its quantizer lies on one of these paths. The weights are leaves of the graph of step t, so the weight quantizer of layer j lies only on the paths that end at W_j. The activation quantizer of layer j lies on the paths that continue from $\widehat{x}_j$ to x_j and further down, which end at tensors of layers below layer j.

For (i), the paths from the loss to $\widehat{W}_k$ and to $\widehat{x}_k$ start above layer k and end at layer k. They contain no weight quantizer, because every weight quantizer ends its own path, and they contain the activation quantizers of some layers $j > k$ only. For (ii), $G_{k,i}^t = g_{k,i}^t\,c(W_{k,i}^t; a_k^t)$ by (17), and (i) applies to g_k^t. For (iii), a path that ends at W_m passes the activation quantizer of layer k only if layer k lies above layer m. □

The lemma says "can depend" because a network may contain paths that skip a quantizer, such as the shortcuts of Bi-Real Net (Liu et al., 2018). In a residual network, the gradient of W_m with $m < k$ is the sum of a term that passes through the activation quantizer of layer k and a term that does not. Only the first term depends on b_k, and the derivation of Section D.7 accounts for this split.

Every path in the proof of (iii) passes the activation quantizer of layer k, and the output of that quantizer is s_k^t of (17). The parameter b_k^t therefore enters the gradients of the layers below layer k, and the loss $\mathcal{L}^{t+1}$, only through s_k^t, which is the form that Appendix D.7 uses.

Expanding the gradient of the binarized weights of layer k one layer at a time gives the coefficients that Section 4.2 collects,

$$\frac{\partial \mathcal{L}^t}{\partial \widehat{x}_j^t} = \frac{\partial \mathcal{L}^t}{\partial x_{j+1}^t}\,\frac{\partial x_{j+1}^t}{\partial \widehat{x}_j^t} \approx \Big(\frac{\partial \mathcal{L}^t}{\partial \widehat{x}_{j+1}^t} \odot c\big(x_{j+1}^t; b_{j+1}^t\big)\Big)\frac{\partial x_{j+1}^t}{\partial \widehat{x}_j^t}, \quad j = k+1, \dots, N-1, \tag{21}$$

where the first equality holds because $\widehat{x}_j$ influences the loss only through the output x_{j+1} of layer j, and the estimate is the second one of (8) at layer $j+1$. Applied at $j = k+1, \dots, N-1$, it brings one coefficient $c(x_j^t; b_j^t)$ into the update (9) for every layer $j > k$.

D.5 The Kept Update

The rules differentiate $\mathcal{L}^{t+1}$ with respect to parameters of step t. Standard training applies (18) to detached tensors, so W^{t+1} enters step $t+1$ with no connection to a^t or b^t, and this derivative does not exist. LSTE instead computes G^t with a recorded backward pass, so that the autograd engine stores $G_{k,i}^t = g_{k,i}^t\,c(W_{k,i}^t; a_k^t)$ as a function of a^t and b^t, and it stores (18) as a function of G^t. Backpropagation of $\mathcal{L}^{t+1}$ then passes through the update and reaches them. Because G^t is itself a gradient, this backpropagation differentiates a gradient, which is why the Jacobian (29) appears in (31).

The rules are partial derivatives of $\mathcal{L}^{t+1}$ with respect to a^t and b^t, with every quantity of earlier steps held fixed. The same parameters also shaped the updates of earlier steps, and a full derivative would include those paths as well. LSTE keeps only the last update, as the one-step hypergradient methods of Luketina et al. (2016) and Baydin et al. (2018) do and as Shaban et al. (2019) analyze for bilevel problems. This choice inherits the short-horizon bias that Wu et al. (2018) analyze: the rules favor parameters that reduce the loss of the next mini-batch.

The set $\mathcal{K}$ collects the layers whose update is kept. In our implementation it contains the weights of all binarized layers and of the full-precision first layer. We update normalization parameters, biases and the classifier without a kept graph, so their terms are absent from the sums of Section D.7. In the language experiment of Section 5, the token embedding is the only tensor below the activation quantizer of the first attention projection, so unless $\mathcal{K}$ contains the embedding, the parameters of that quantizer receive a zero gradient; that experiment therefore keeps the embedding in the update.

The derivative $\partial\mathcal{L}^{t+1}/\partial W^{t+1}$ in the rules is itself computed through the estimator, because the derivative of the sign is zero almost everywhere. The rules are therefore derivatives in the computational graph (3) in which we train the weights.

D.6 Derivation of the Rule for the Weight Parameter

Proposition 5 (Rule for the weight parameter)**.** *Under the update (18), for every layer $k \in \mathcal{K}$,*

$$\frac{\partial \mathcal{L}^{t+1}}{\partial a_k^t} = -\eta \sum_i G_{k,i}^{t+1}\, g_{k,i}^t\, \frac{\partial c}{\partial \theta}\big(W_{k,i}^t; a_k^t\big), \qquad G_{k,i}^{t+1} = \frac{\partial \mathcal{L}^{t+1}}{\partial W_{k,i}^{t+1}}. \tag{22}$$

Proof. Step 1*: one tensor carries the dependence.* By Lemma 3, a_k^t does not affect the forward pass of step t. It enters the gradient G_k^t by (17), and it enters no other gradient of step t, because every weight quantizer ends its own path in the proof of Lemma 4 and therefore appears in no incoming gradient g_m^t or δ_m^t. It follows that a_k^t enters W^{t+1} only through W_k^{t+1}. The loss $\mathcal{L}^{t+1}$ is a function of all entries of W^{t+1}, and the chain rule for a scalar parameter sums over the entries of W_k^{t+1},

$$\frac{\partial \mathcal{L}^{t+1}}{\partial a_k^t} = \sum_i \frac{\partial \mathcal{L}^{t+1}}{\partial W_{k,i}^{t+1}} \frac{\partial W_{k,i}^{t+1}}{\partial a_k^t} = \sum_i G_{k,i}^{t+1} \frac{\partial W_{k,i}^{t+1}}{\partial a_k^t}. \tag{23}$$

Step 2*: the gradient of the next mini-batch.* The first backward pass of step $t+1$ uses a^t and b^t, which are in force before the parameter update of step $t+1$ (Algorithm 1). By (17),

$$G_{k,i}^{t+1} = g_{k,i}^{t+1}\, c\big(W_{k,i}^{t+1}; a_k^t\big), \tag{24}$$

where g_k^{t+1} uses the activation parameters b^t.

Step 3*: how the update reacts.* Substituting (17) into (18) and differentiating with the product rule,

$$\frac{\partial W_{k,i}^{t+1}}{\partial a_k^t} = \frac{\partial W_{k,i}^t}{\partial a_k^t} - \eta\, \frac{\partial g_{k,i}^t}{\partial a_k^t}\, c\big(W_{k,i}^t; a_k^t\big) - \eta\, g_{k,i}^t\, \frac{\partial c}{\partial \theta}\big(W_{k,i}^t; a_k^t\big). \tag{25}$$

The first term is zero by Lemma 3, and the second term is zero by Lemma 4(i). Hence

$$\frac{\partial W_{k,i}^{t+1}}{\partial a_k^t} = -\eta\, g_{k,i}^t\, \frac{\partial c}{\partial \theta}\big(W_{k,i}^t; a_k^t\big). \tag{26}$$

Step 4*: combination.* Substituting (26) into (23) gives (22). □

With (20), the rule reads

$$\frac{\partial \mathcal{L}^{t+1}}{\partial a_k^t} = -2\eta \sum_{i\,:\,|W_{k,i}^t|<1/a_k^t} G_{k,i}^{t+1}\, g_{k,i}^t\, \big(1 - 2a_k^t |W_{k,i}^t|\big). \tag{27}$$

Only latent weights inside the current support contribute. With the update (10),

$$a_k^{t+1} = a_k^t + 2\eta_a \eta \sum_{i\,:\,|W_{k,i}^t|<1/a_k^t} G_{k,i}^{t+1}\, g_{k,i}^t\, \big(1 - 2a_k^t |W_{k,i}^t|\big). \tag{28}$$

The product $G_{k,i}^{t+1} g_{k,i}^t$ is positive when the current update moves the latent weight $W_{k,i}$ in the direction that the next mini-batch also asks for, since $c > 0$ inside the support. We call such a weight an agreeing weight. An agreeing weight with $|W_{k,i}^t| < 1/(2a_k^t)$ increases a_k, which by Section D.2 increases its own coefficient. An agreeing weight with $1/(2a_k^t) < |W_{k,i}^t| < 1/a_k^t$ decreases a_k, which also increases its own coefficient. A disagreeing weight acts in the opposite direction. The rule therefore changes a_k so that the coefficient grows on the latent weights where consecutive mini-batches agree and shrinks where they disagree.

D.7 Derivation of the Rule for the Activation Parameter

We write

$$J_{m,k,ij}^t = \frac{\partial G_{m,i}^t}{\partial s_{k,j}^t} \tag{29}$$

for the Jacobian of the gradient at W_m with respect to the gradient s_k^t that the activation quantizer of layer k passes down. By Lemma 4(iii) and the remark after it, the gradient of every layer $m < k$ splits into a term that passes through s_k^t and a term that does not,

$$G_{m,i}^t = \sum_j J_{m,k,ij}^t \, s_{k,j}^t + R_{m,k,i}^t, \tag{30}$$

where $R_{m,k}^t$ collects the paths that avoid the activation quantizer of layer k. The backward pass between s_k^t and W_m multiplies the gradient by the binarized weights, by the derivatives of the normalizations and nonlinearities, which depend only on forward quantities, and by the coefficients of quantizers below layer k. Each of these operations is linear in the gradient and does not depend on b_k^t, so (30) is exact and neither $J_{m,k}^t$ nor $R_{m,k}^t$ depends on b_k^t.

Proposition 6 (Rule for the activation parameter)**.** *Under the update (18),*

$$\frac{\partial \mathcal{L}^{t+1}}{\partial b_k^t} = -\eta \sum_j \Phi_{k,j}^{t+1} \, \delta_{k,j}^t \, \frac{\partial c}{\partial \theta}\big(x_{k,j}^t; b_k^t\big), \qquad \Phi_{k,j}^{t+1} = \sum_{\substack{m \in \mathcal{K} \\ m<k}} \sum_i J_{m,k,ij}^t \, G_{m,i}^{t+1}. \tag{31}$$

Proof. Step 1*: the sum over the layers below.* By Lemma 3 and Lemma 4(iii), b_k^t enters W^{t+1} only through the kept updates of the layers $m \in \mathcal{K}$ with $m < k$, so

$$\frac{\partial \mathcal{L}^{t+1}}{\partial b_k^t} = \sum_{\substack{m \in \mathcal{K} \\ m<k}} \sum_i G_{m,i}^{t+1} \, \frac{\partial W_{m,i}^{t+1}}{\partial b_k^t}. \tag{32}$$

Step 2*: how each update below reacts.* By (18), Lemma 3 and (30),

$$\frac{\partial W_{m,i}^{t+1}}{\partial b_k^t} = -\eta \, \frac{\partial G_{m,i}^t}{\partial b_k^t} = -\eta \sum_j J_{m,k,ij}^t \, \frac{\partial s_{k,j}^t}{\partial b_k^t}, \tag{33}$$

since $J_{m,k}^t$ and $R_{m,k}^t$ do not depend on b_k^t.

Step 3*: how the quantizer reacts.* By (17), and since δ_k^t does not depend on b_k^t by Lemma 4(i),

$$\frac{\partial s_{k,j}^t}{\partial b_k^t} = \delta_{k,j}^t \, \frac{\partial c}{\partial \theta}\big(x_{k,j}^t; b_k^t\big). \tag{34}$$

Step 4*: combination.* Substituting (34) into (33) and then into (32), and exchanging the order of the sums,

$$\frac{\partial \mathcal{L}^{t+1}}{\partial b_k^t} = -\eta \sum_j \Big(\sum_{\substack{m \in \mathcal{K} \\ m<k}} \sum_i J_{m,k,ij}^t \, G_{m,i}^{t+1} \Big) \delta_{k,j}^t \, \frac{\partial c}{\partial \theta}\big(x_{k,j}^t; b_k^t\big), \tag{35}$$

which is (31). □

The rule has the same structure as (22). Its second factor contains only quantities at the activation quantizer of layer k, while its first factor Φ_k^{t+1} collects the next-batch gradients of all kept layers below. The reading of (28) carries over: b_k changes so that the coefficient grows on the latent activations where the gradient passed down in the current mini-batch agrees with Φ_k^{t+1}. The implementation never forms $J_{m,k}^t$ or Φ_k^{t+1}. One backpropagation of $\mathcal{L}^{t+1}$ through the kept update computes (22) and (31) for all quantizers at once.

D.8 One Rule for Every Learnable Parameter of the Backward Pass

Let θ be the parameter of a quantizer, v the latent tensor of that quantizer, r the gradient that reaches its binarized tensor, D^{t+1} the next-batch gradient that the parameter affects, and i an index over the entries of that quantizer. Propositions 5 and 6 are both instances of

$$\frac{\partial \mathcal{L}^{t+1}}{\partial \theta^t} = -\eta \sum_i D_i^{t+1}\, r_i^t\, \frac{\partial c}{\partial \theta}\big(v_i^t; \theta^t\big), \tag{36}$$

with $(\theta, v, r, D) = (a_k, W_k, g_k, G_k)$ for the weight quantizer and $(\theta, v, r, D) = (b_k, x_k, \delta_k, \Phi_k)$ for the activation quantizer. The first factor D^{t+1} is a gradient of the next mini-batch, and the remaining factors come from the update (9) of the current one, in which only the coefficient depends on the parameter. Only $\partial c/\partial\theta$ depends on the choice of coefficient, so the same construction applies to any coefficient that is differentiable in its parameter.

Every rule contains the weight learning rate η. Under SGD for the parameters, the parameter updates therefore shrink with η, and the parameters stop moving when a schedule sends η to zero. When Adam updates the parameters, its normalization removes this factor.

Algorithm 1 evaluates the rules with a shift of one step. At step t, the loss $\mathcal{L}^t$ plays the role of $\mathcal{L}^{t+1}$ in (36), and the kept update of step $t-1$ provides the dependence on a^{t-1} and b^{t-1}. At $t = 1$, no update is kept, so the parameters keep their initial value for one step.