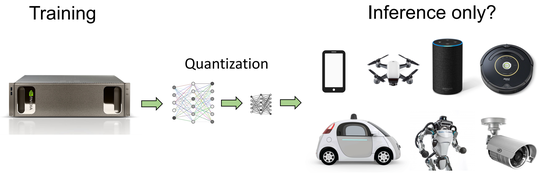
Training Quantized Nets: A Deeper Understanding

Hao Li[1]*, Soham De[1]*, Zheng Xu[1], Christoph Studer[2], Hanan Samet[1], Tom Goldstein[1]

University of Maryland, College Park[1], Cornell University[2]

Motivation



How to train quantized models on low-power devices?

- Floating-point arithmetic may not be available.
- Training without floating-point weights can greatly save power and reduce the hardware requirement (Google TPU).

Goal

- Study quantized training methods from a theoretical perspective.
- Understanding the differences in behavior, and reasons for success or failure, of various methods.

Main Results

✓ We proved that both BinaryConnect (BC) [1] and Stochastic Rounding (SR) [2] are capable of solving **convex** discrete problems **up to a level of accuracy that depends on the quantization level**.

✓ We address the issue of why algorithms that maintain floating-point representations, like BC, work so well, while fully quantized training methods like SR stall before training is complete

- ✓ **BC, like SGD, can efficiently concentrate on minimizers**
- ✓ **SR cannot concentrate on minimizers on decreasing step size**

Training Quantized Neural Networks

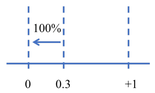
- **Semi-quantized: BinaryConnect (BC)**

$$w_b^t = Q(w_r^t)$$
$$w_r^{t+1} = w_r^t - \alpha_t \nabla \tilde{f}(w_b^t)$$

w_b : binary weights
w_r : real weights
Q : quantization fn
Δ : quantization error

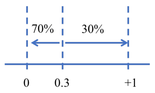
- **Full-quantized: Deterministic Rounding (R)**

$$Q_d(w) = \text{sign}(w) \cdot \Delta \cdot \left\lfloor \frac{|w|}{\Delta} + \frac{1}{2} \right\rfloor$$
$$w_b^{t+1} = Q_d\left(w_b^t - \alpha_t \nabla \tilde{f}(w_b^t)\right)$$



- **Full-quantized : Stochastic Rounding (SR)**

$$Q_s(w) = \Delta \cdot \begin{cases} \lfloor \frac{w}{\Delta} \rfloor + 1 & \text{for } p \le \frac{w}{\Delta} - \lfloor \frac{w}{\Delta} \rfloor, \\ \lfloor \frac{w}{\Delta} \rfloor & \text{otherwise,} \end{cases}$$
$$w_b^{t+1} = Q_s(w_b^t - \alpha_t \nabla \tilde{f}(w_b^t))$$



Convergence under Convexity Assumptions

Convergence of Stochastic Rounding

- **Theorem 1** Assume that F is μ-strongly convex and the learning rates are given by $\alpha_t = \frac{1}{\mu(t+1)}$

$$\mathbb{E}[F(\bar{w}^T) - F(w^\star)] \le \frac{(1+\log(T+1))G^2}{2\mu T} + \frac{\sqrt{d}\Delta G}{2}$$

SR converges until it reaches an "accuracy floor", which is determined by the quantization error Δ.

Convergence of BinaryConect

- **Theorem 4** Assume that F is μ-strongly convex and the learning rates are given by $\alpha_t = \frac{1}{\mu(t+1)}$

$$\mathbb{E}[F(\bar{w}^T) - F(w^\star)] \le \frac{(1+\log(T+1))G^2}{2\mu T} + \frac{DL_2\sqrt{d}\Delta}{2}$$

BC converges until it reaches an "accuracy floor", which is determined by the quantization error Δ and L_2 (0 if F is quadratic).

G : Bounded variance; d : Dimension; D: Diameter of domain; L_2 : Lipschitz bound on Hessia $\|\nabla^2 f_i(x) - \nabla^2 f_i(y)\| \le L_2\|x-y\|$

- Corollary 1
  - Accumulating the real-valued weights in BC allows it to converge to the ***true minimizer*** of quadratic losses.
  - When the function behaves like a quadratic on the distance scale Δ, one would expect BC to effectively concentrate on minimizers.

Non-Convex Problems: Asymptotic Analysis

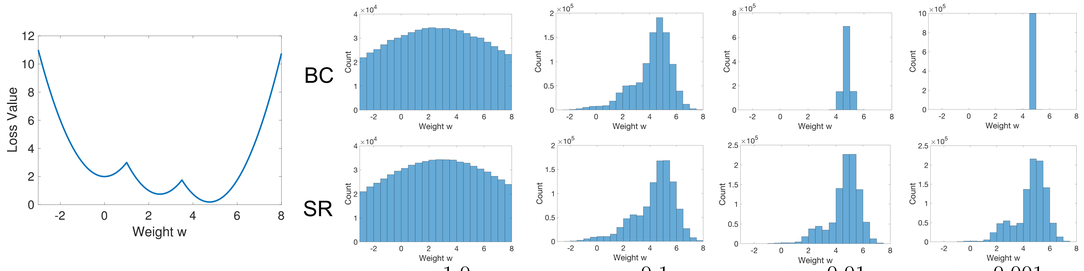
Exploration-Exploitation Tradeoff

➢ **Continuous-valued SGD & BC**

- When the learning rate is large, the algorithm *explores* by moving quickly between states.
- When the learning rate is small, *exploitation* happens.

➢ **Stochastic Rounding lacks this important tradeoff**

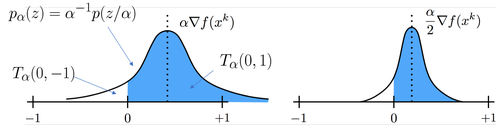
- As the step-size gets small and the algorithm slows down, the quality of the iterates does *not* improve.



Effect of shrinking the learning rate in SR vs BC on a toy problem. Histograms plot the distribution of the quantized weights over 10^6 iterations. As the learning rate shrinks, the BC distribution concentrates on a minimizer, while the SR distribution stagnates.

Stochastic Rounding as a Markov Chain

SR starts at some state x, and moves to a new state y with some transition probability $T(x,y)$ that depends only on x and the learning rate α. For fixed α, this is a **Markov process** with transition matrix $T_\alpha(x,y)$



As α gets smaller, the distribution of the perturbation gets "squished", making the algorithm less likely to move. The "squishing" effect **does not effect the relative probability of moving left or right.**

$$T_\alpha(0,1|x^1 \neq x^0) \approx \frac{T_\alpha(0,1)}{T_\alpha(0,-1)+T_\alpha(0,1)} = \frac{\int_0^\infty p(x)x\,dx}{\int_{-\infty}^0 p(x)x\,dx + \int_0^\infty p(x)x\,dx}$$

Theorem 5 Let $p_{x,k}$ denote the distribution function of the k th entry in the stochastic gradient $\nabla\tilde{f}(x)$. Assume $\int_\nu^\infty p_{x,k}(z)\,dz \le \frac{C_1}{\nu^2}$ for all x, k, v and both $\int_0^{C_2} p_{x,k}(z)\,dz > 0$ and $\int_{-C_2}^0 p_{x,k}(z)\,dz > 0$ for constants C_1 and C_2. Define matrix

$$\tilde{U}(x,y) = \begin{cases} \int_0^\infty p_{x,k}(z)\frac{z}{\Delta}\,dz, \textit{if } x \textit{ and } y \textit{ differ only at coordinate } k, \textit{and } y_k = x_k + \Delta \\ \int_{-\infty}^0 p_{x,k}(z)\frac{z}{\Delta}\,dz, \textit{if } x \textit{ and } y \textit{ differ only at coordinate } k, \textit{and } y_k = x_k - \Delta \\ 0, \textit{otherwise,} \end{cases}$$

and the associated Markov chain transition matrix

$$\tilde{T}_{\alpha_0} = I - \alpha_0 \cdot \text{diag}(1^T\tilde{U}) + \alpha_0\tilde{U},$$

where α_0 is the largest constant that makes $\tilde{T}_{\alpha_0}$ non-negative. Suppose $\tilde{T}_\alpha$ has a stationary distribution $\tilde{\pi}$. Then, for sufficiently small α, T_α has stationary distribution π_α, and

$$\lim_{\alpha\to 0}\pi_\alpha = \tilde{\pi}$$

This satisfies $\tilde{\pi}(x) > 0$ for any state x and is **not concentrated on minimizers of** f.

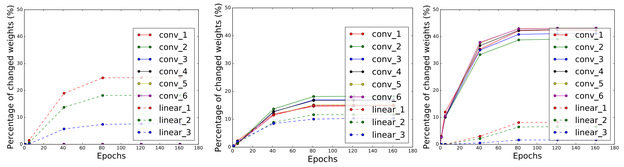
Experiments

Table 1: Top-1 test error after training with full-precision (ADAM), binarized weights (R-ADAM, SR-ADAM, BC-ADAM), and binarized weights with big batch size (Big SR-ADAM).

	CIFAR-10				CIFAR-100	ImageNet
	VGG-9	VGG-BC	ResNet-56	WRN-56-2	ResNet-56	ResNet-18
ADAM	7.97	7.12	8.10	6.62	33.98	36.04
BC-ADAM	10.36	8.21	8.83	7.17	35.34	52.11
Big SR-ADAM	16.95	16.77	19.84	16.04	50.79	77.68
SR-ADAM	23.33	20.56	26.49	21.58	58.06	88.86
R-ADAM	23.99	21.88	33.56	27.90	68.39	91.07

- BC-ADAM has comparable performance to the full-precision model trained by ADAM.
- SR-ADAM outperforms R-ADAM, which verifies the effectiveness of SR.
- Keeping track of the real-valued weights to quantize the binary weights in BC-ADAM seems to really help empirically.

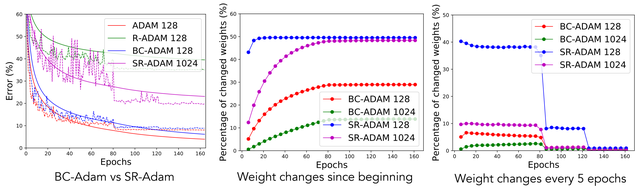
Exploration vs Exploitation Tradeoffs



R-ADAM BC-ADAM SR-ADAM

- SR-ADAM explores aggressively; it changes more weights in the conv layers than both R-ADAM and BC-ADAM, and keeps changing weights until nearly 40% of the weights differ from their starting values.
- The BC method never changes more than 20% of the weights, indicating that it stays near a local minimizer and explores less.

Big Batch Training



BC-Adam vs SR-Adam Weight changes since beginning Weight changes every 5 epochs

- Our theory predicts that we can improve the performance of SR by increasing the batch size, which shrinks the variance of the gradient distribution without changing the mean and concentrates more of the gradient distribution towards downhill directions, making the algorithm more greedy.

Reference

[1] M. Courbariaux, Y. Bengio, J. P. David. BinaryConnect: Training deep neural networks with binary weights during propagations. NIPS 2015

[2] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan. Deep learning with limited numerical precision. ICML 2015