Quantization

Sparsification vs Quantization

network sparsification

reduce redundancy in the number of weights

network quantization

reduces redundancy in network representation

Advantages of Quantization: 1. Smaller Storage

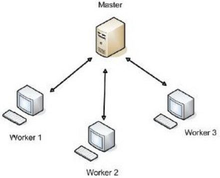
- original model size: S
- $\underbrace{S/32}_{\text{binary}} \rightarrow \underbrace{S/16}_{\text{ternary}} \rightarrow \underbrace{mS/32}_{m\text{-bit}}$

Advantages: 2. Saves Communication Bandwidth

How to speed up training?

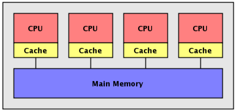
single machine

distributed processing



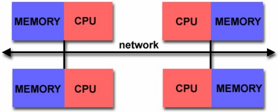
Distributed Architectures

1. shared memory: variables stored in a shared address space



   - e.g., servers, high-end workstations, multicore processors
   - data can be conveniently accessed (as in single-processor machines) ✓
   - less scalable

2. distributed memory: each node has its own memory

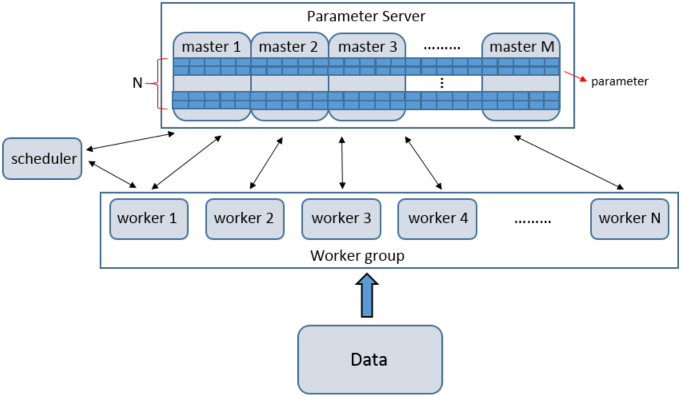


   - nodes connected by high-speed communication network
   - scalable ✓

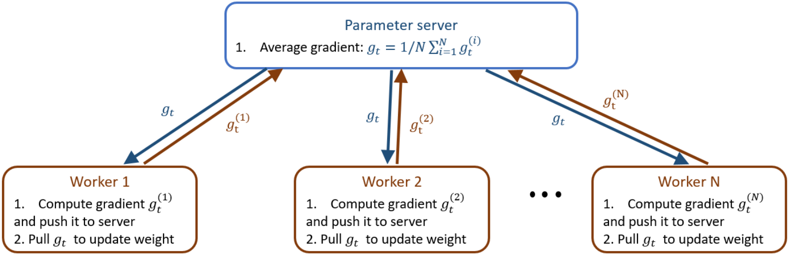
Typical Implementation: Parameter Server

- data set $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$

n is big $\rightarrow$ split the n samples over N machines

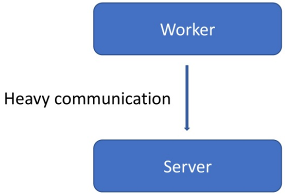


Quantization Saves Communication Bandwidth



1. worker
   - computes gradient w.r.t. weight and send to parameter server
   - pull averaged gradient from server and update weights
2. parameter server
   - synchronize and average gradients and send back to all workers

Problem with Distributed Training



Example

- latency for accessing main memory: around 70ns
- transmission latency in data center: between 0.1ms to 1ms

How to reduce communication cost?

quantized network → small storage → less communication

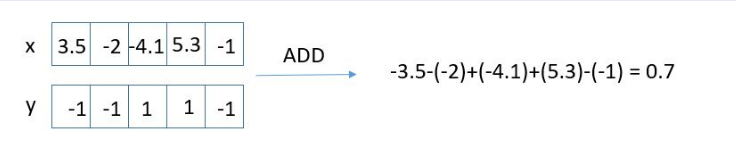
Advantages: 3. Faster Inference

$\mathbf{x}, \mathbf{y}$ are full-precision

- $\mathbf{x} \cdot \mathbf{y}$: floating point multiplication

$\mathbf{x}$ binary, $\mathbf{y}$ full-precision

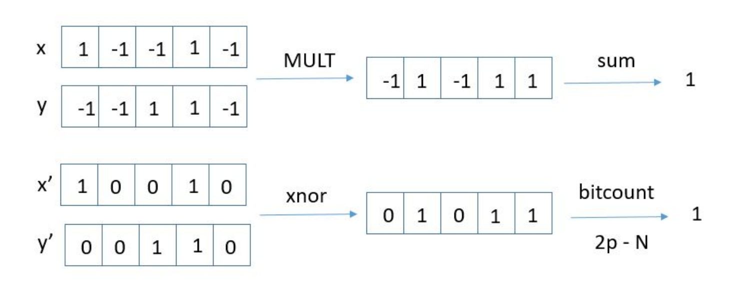
- $\mathbf{x} \cdot \mathbf{y}$: floating point addition



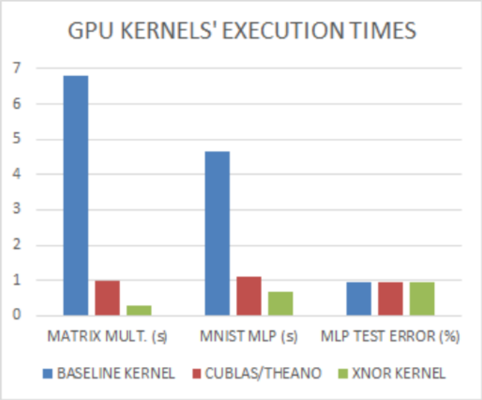
Advantages: 3. Faster Inference...

$\mathbf{x}$ binary, $\mathbf{y}$ binary

- $\mathbf{x} \cdot \mathbf{y} = 2 \times \text{bitcount}(\text{xnor}(\mathbf{x} \cdot \mathbf{y})) - N$



Efficiency of Binary Operations



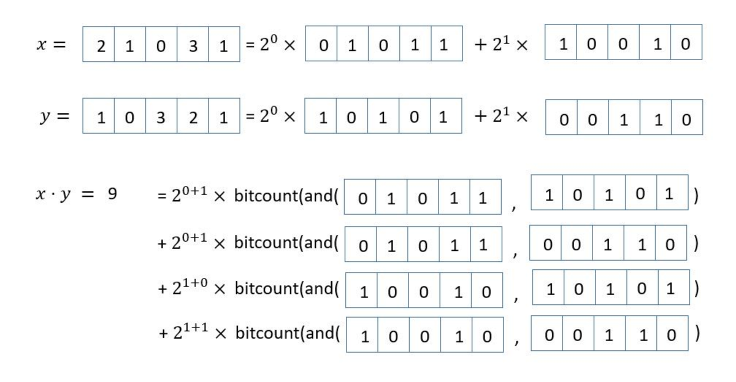
[from (Hubara et al., 2016)]

- left: time to perform a $8192 \times 8192 \times 8192$ (binary) matrix multiplication on a GTX750 Nvidia GPU
- middle: time to run the MLP on MNIST
- right: MLP accuracy

Advantages: 3. Faster Inference...

$\mathbf{x}$ is M-bit, $\mathbf{y}$ is K-bit

- $\mathbf{x} = \sum_{m=0}^{M-1} c_m(\mathbf{x}) 2^m, [c_m(\mathbf{x})]_i, \in \{0, 1\}$
- $\mathbf{y} = \sum_{k=1}^{K-1} c_k(\mathbf{y}) 2^k, [c_k(\mathbf{y})]_i \in \{0, 1\}$
- $\mathbf{x} \cdot \mathbf{y} = \sum_{m=0}^{M-1} \sum_{k=0}^{K-1} 2^{m+k} \text{bitcount}(\text{and}(c_m(\mathbf{x}), c_k(\mathbf{y})))$



Quantizing Deep Networks

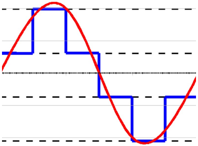
What to quantize?

1. weights
2. activations
3. gradients

How does that affect convergence?

Quantizing Weights

Weight Quantization



32-bit $\rightarrow$ fewer bits

binarization

- 1 bit: quantize to -1 or $+1$

ternarization

- 2 bits: quantize to $-1, 0$ or $+1$

m-bit

1. linear quantization: quantize each weight to $\{-1, -\frac{k-1}{k}, \ldots, -\frac{1}{k}, 0, \frac{1}{k}, \ldots, \frac{k-1}{k}, 1\}$
2. logarithmic quantization: $\{-1, -\frac{1}{2}, \ldots, -\frac{1}{2^{k-1}}, 0, \frac{1}{2^{k-1}}, \ldots, \frac{1}{2}, 1\}$
   - higher resolution to small numbers

How to Obtain Quantized Weight?

post-training quantization

- train $\rightarrow$ compress ($\rightarrow$ retrain)
- e.g., from a pre-trained model

quantization-aware training

- train a quantized network from scratch
- train and compress the network simultaneously

Training Procedure

Variation of backprop

at iteration t

1. quantization: turn full-precision weight $\mathbf{w}^t$ to $\hat{\mathbf{w}}^t$
2. forward propagation using $\hat{\mathbf{w}}^t$
    - sometimes hidden unit activations are also binarized
3. backpropagate the gradient $\nabla \ell(\hat{\mathbf{w}}^t)$
4. update (full-precision) weight as $\mathbf{w}^t = \mathbf{w}^{t-1} - \eta \nabla \ell(\hat{\mathbf{w}}^t)$

Binarization: BinaryConnect

(Courbariaux et al., 2015)

BinaryConnect: Training deep neural networks with binary weights during propagations

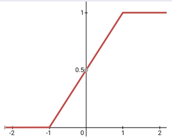
Deterministic

$$b = \text{Binarize}(w) = \text{sign}(w) = \begin{cases} +1 & w \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Stochastic

$$\text{Binarize}(w) = \begin{cases} +1 & \text{with probability } p = \sigma(w) \\ -1 & \text{with probability } 1 - p \end{cases}$$

$\sigma(x) = \text{clip}(\frac{x+1}{2}, 0, 1)$ (hard sigmoid)



Binarization: Binary Weight Network

(Rastegari et al., 2016)

XNOR-Net: ImageNet classification using binary convolutional neural networks

- add a scaling parameter α
  - binarize $\mathbf{w}$ to $\alpha\mathbf{b}$, $\mathbf{b}$: in $\{-1,+1\}^n$
  - one shared scaling parameter for the whole layer

how to find α and $\mathbf{b}$?

- find the best approximation of $\mathbf{w}$ by $\alpha\mathbf{b}$ (minimize $\|\mathbf{w}-\alpha\mathbf{b}\|$)
- simple closed-form solution

$$\alpha = \frac{\|\mathbf{w}\|_1}{n}, \quad \mathbf{b} = \text{sign}(\mathbf{w})$$

Ternarization: Ternary Connect

(Lin et al., 2016b)

Neural networks with few multiplications

- $w \geq 0$: stochastically quantized to 1 with probability w, and 0 otherwise
- $w < 0$: stochastically quantized to -1 with probability $-w$, and 0 otherwise

$$\text{Ternarize}(w) = \begin{cases} \text{sign}(w) & w.p. |w| \\ 0 & \text{otherwise} \end{cases}$$

Ternarization: Ternary Weight Networks

(Li et al, 2016)

Ternary weight networks

- add a scaling parameter

$$\text{Ternarize}(\mathbf{w}) = \alpha \mathbf{b}, \quad \alpha > 0, \mathbf{b} \in \{-1, 0, 1\}^n$$

how to find α and $\mathbf{b}$?

- minimize $\min_{\alpha,\mathbf{b}} \|\mathbf{w} - \alpha \mathbf{b}\|^2$
- difficult to solve
- use a heuristic to find threshold Δ and return

$$\text{Ternarize}(\mathbf{w}) = \alpha \mathbf{I}_\Delta(\mathbf{w})$$

$$[\mathbf{I}_\Delta(\mathbf{w})]_i = \begin{cases} 1 & w_i > \Delta \\ -1 & w_i < -\Delta \\ 0 & \text{otherwise} \end{cases}$$

Ternarization: Trained Ternary Quantization (TTQ)

positive and negative weights may have different scales

(Zhu et al., 2016)

Trained ternary quantization

- use different scaling parameters (α and β) for positive and negative weights

$$\text{Ternarize}(\mathbf{w}) = \alpha \mathbf{I}^{+}_{\Delta}(\mathbf{w}) + \beta \mathbf{I}^{-}_{\Delta}(\mathbf{w}), \quad \alpha > 0, \beta > 0$$

$$[\mathbf{I}^{+}_{\Delta}(\mathbf{w})]_i = \begin{cases} 1 & \text{if} \quad w_i > \Delta \\ 0 & \text{otherwise} \end{cases}, \quad [\mathbf{I}^{-}_{\Delta}(\mathbf{w})]_i = \begin{cases} -1 & \text{if} \quad w_i < -\Delta \\ 0 & \text{otherwise} \end{cases}$$

- again, threshold is set by heuristic

Loss-Aware Weight Binarization

all these methods do not consider the loss during quantization

(Hou et al, 2017)

Loss-aware binarization of deep networks

Loss-aware formulation

$$\min_{\hat{\mathbf{w}}} \quad \ell(\hat{\mathbf{w}})$$
$$\text{s.t.} \quad \hat{\mathbf{w}}_l = \alpha_l \mathbf{b}_l,\ \alpha_l > 0,\ \mathbf{b}_l \in \mathcal{Q}^{n_l},\quad l = 1, \dots, L$$

how to solve this optimization problem?

Proximal Algorithm

$$\min_{\mathbf{x}} \underbrace{f(\mathbf{x})}_{\text{smooth}} + \underbrace{g(\mathbf{x})}_{\text{nonsmooth}}$$

proximal gradient descent

$$\mathbf{x}_{t+1} = \arg\min_{\mathbf{x}} f(\mathbf{x}_t) + \nabla f(\mathbf{x}_t)^\top (\mathbf{x} - \mathbf{x}_t) + \frac{1}{2\eta}\|\mathbf{x} - \mathbf{x}_t\|^2 + g(\mathbf{x})$$

- similar to standard gradient descent, except that the nonsmooth g can now be handled
- uses only first-order information (of f)

can also use second-order information (Hessian)

proximal Newton [Lee et.al, 2014]

$$\mathbf{x}_{t+1} = \arg\min_{\mathbf{x}} \nabla f(\mathbf{x}_t)^\top (\mathbf{x} - \mathbf{x}_t) + (\mathbf{x} - \mathbf{x}_t)^\top \mathbf{H} (\mathbf{x} - \mathbf{x}_t) + g(\mathbf{x})$$

- $\mathbf{H}$: estimate of the Hessian (of f) at $\mathbf{x}_t$

Loss-Aware Weight Binarization...

$$\begin{aligned} \min_{\hat{\mathbf{w}}} \quad & \ell(\hat{\mathbf{w}}) \\ \text{s.t.} \quad & \hat{\mathbf{w}}_l = \alpha_l \mathbf{b}_l, \ \alpha_l > 0, \ \mathbf{b}_l \in \mathcal{Q}^{n_l}, \quad l = 1, \dots, L \end{aligned}$$

deep networks

- nonconvex objective and inhomogeneous curvature
- $\rightarrow$ use second-order information: captures local curvature

Hessian is expensive!

- approximate by diagonal Hessian
- readily available in popular optimizers such as Adam

Subproblem at Each Iteration

$$\begin{aligned} \min_{\hat{\mathbf{w}}} \quad & \nabla \ell(\hat{\mathbf{w}}^{t-1})^\top (\hat{\mathbf{w}} - \hat{\mathbf{w}}^{t-1}) + \frac{1}{2}(\hat{\mathbf{w}} - \hat{\mathbf{w}}^{t-1})^\top \mathbf{D}^{t-1} (\hat{\mathbf{w}} - \hat{\mathbf{w}}^{t-1}) \\ \text{s.t.} \quad & \hat{\mathbf{w}}_l = \alpha_l^t \mathbf{b}_l^t,\ \alpha_l^t > 0,\ \mathbf{b}_l^t \in \mathcal{Q}^{n_l},\ \ l = 1, \ldots, L \end{aligned}$$

- $\mathbf{D}^{t-1}$: approximate diagonal Hessian

Procedure

1. gradient descent with adaptive learning rate defined by Hessian

$$\mathbf{w}_l^t \leftarrow \hat{\mathbf{w}}_l^{t-1} - \nabla_l \ell(\hat{\mathbf{w}}^{t-1}) \oslash \underbrace{\text{diag}(\mathbf{D}_l^{t-1})}_{\text{curvature}}$$

2. quantize $\mathbf{w}_l^t$ to $\hat{\mathbf{w}}_l^t$ by minimizing a scaled difference

$$\hat{\mathbf{w}}_l^t \leftarrow \min_{\hat{\mathbf{w}}_l^t} \frac{1}{2} \sum_{l=1}^{L} \| \underbrace{\hat{\mathbf{w}}_l^t - \mathbf{w}_l^t}_{\text{difference}} \|^2_{\underbrace{\mathbf{D}_l^{t-1}}_{\text{curvature}}}$$

   - $\mathbf{d}_l^{t-1} \equiv \text{diag}(\mathbf{D}_l^{t-1})$

Low curvature

- loss is not sensitive to weight, quantization error can be less penalized

High curvature

- quantization has to be more accurate

Binarization: $Q = \{-1, +1\}$ (32 bits $\to$ 1 bit)

$$\hat{\mathbf{w}}_l^t \leftarrow \min_{\hat{\mathbf{w}}_l^t = \alpha_l^t \mathbf{b}_l^t} \frac{1}{2} \sum_{l=1}^{L} \|\hat{\mathbf{w}}_l^t - \mathbf{w}_l^t\|^2_{\mathbf{D}_l^{t-1}}$$

Loss-Aware Binarization (LAB)

$$\alpha_l^t = \frac{\|\mathbf{d}_l^{t-1} \odot \mathbf{w}_l^t\|_1}{\|\mathbf{d}_l^{t-1}\|_1}, \quad \mathbf{b}_l^t = \mathrm{sign}(\mathbf{w}_l^t)$$

When $\mathbf{D}_l^{t-1} = \lambda \mathbf{I}$

- curvature is the same for all dimensions in the lth layer
- solution reduces to BWN ($\alpha_l^t = \|\mathbf{w}_l^t\|_1 / n_l, \mathbf{b}_l^t = \mathrm{sign}(\mathbf{w}_l^t)$)

When $\alpha_l^t = 1$

- solution reduces to BinaryConnect ($\alpha_l^t = 1, \mathbf{b}_l^t = \mathrm{sign}(\mathbf{w}_l^t)$)

Ternarization: $\mathcal{Q} = \{-1, 0, +1\}$ (32 bits $\rightarrow$ 2 bits)

(Hou & Kwok, 2018)

Loss-aware weight quantization of deep networks

Loss-Aware Ternarization (LAT)

$\hat{\mathbf{w}}_l^t = \alpha \mathbf{b}$

- for fixed $\mathbf{b}$, $\alpha = \frac{\|\mathbf{b} \odot \mathbf{d}_l^{t-1} \odot \mathbf{w}_l^t\|_1}{\|\mathbf{b} \odot \mathbf{d}_l^{t-1}\|_1}$
- for fixed α, $\mathbf{b} = \mathbf{I}_{\alpha/2}(\mathbf{w}_l^t)$
  - $\mathbf{I}_\Delta(\mathbf{x})$ (with threshold Δ) returns a vector such that $[\mathbf{I}_\Delta(\mathbf{x})]_i = 1$ if $x_i > \Delta$, -1 if $x_i < -\Delta$, and 0 otherwise

When $\mathbf{D}_l^{t-1} = \lambda \mathbf{I}$

- curvature is the same for all dimensions in the lth layer
- α_l^t reduces to Ternary Weight Network

α and b: Can be Computed Exactly

1: **Input:** full-precision weight $\mathbf{w}_l^t$, diagonal entries of the approximate Hessian $\mathbf{d}_l^{t-1}$.
2: $\mathbf{s} = \arg \text{sort}(|\mathbf{w}_l^t|)$;
3: $\mathbf{c} = \text{cum}(\text{perm}_\mathbf{s}(|\mathbf{d}_l^{t-1} \odot \mathbf{w}_l^t|)) \oslash \text{cum}(\text{perm}_\mathbf{s}(\mathbf{d}_l^{t-1})) \oslash 2$;
4: $\mathcal{S} = \text{find}(([\text{perm}_\mathbf{s}(|\mathbf{w}_l^t|)]_{[1:(n-1)]} - \mathbf{c}_{[1:(n-1)]}) \odot ([\text{perm}_\mathbf{s}(|\mathbf{w}_l^t|)]_{[2:n]} - \mathbf{c}_{[1:n-1]}) < 0)$;
5: $\alpha_l^t = 2 \arg\max_{c_j : j \in \mathcal{S}} c_j^2 \cdot [\text{cum}(\text{perm}_\mathbf{s}(\mathbf{d}_l^{t-1}))]_j$;
6: $\mathbf{b}_l^t = \mathbf{I}_{\alpha_l^t/2}(\mathbf{w}_l^t)$;
7: **Output:** $\hat{\mathbf{w}}_l^t = \alpha_l^t \mathbf{b}_l^t$.

- $\text{perm}_\mathbf{s}(\mathbf{x})$: returns $[x_{s_1}, x_{s_2}, \ldots x_{s_n}]$ where s is an indexing vector whose entries are a permutation of $\{1, \ldots, n\}$
- $\text{cum}(\mathbf{x}) = [x_1, \sum_{i=1}^2 x_i, \ldots, \sum_{i=1}^n x_i]$ returns partial sums for elements in $\mathbf{x}$

sorting may be expensive

α and $\mathbf{b}$: Can be Computed using Iteration

alternate the updates of α and $\mathbf{b}$

while $|\alpha - \alpha_{\text{old}}| > \epsilon$ **do**

$\quad \alpha_{\text{old}} = \alpha$;

$\quad \alpha = \frac{\|\mathbf{b} \odot \mathbf{d}_l^{t-1} \odot \mathbf{w}_l^t\|_1}{\|\mathbf{b} \odot \mathbf{d}_l^{t-1}\|_1}$;

$\quad \mathbf{b} = \mathbf{I}_{\alpha/2}(\mathbf{w}_l^t)$;

end while

Output: $\hat{\mathbf{w}}_l^t = \alpha \mathbf{b}$.

Ternarization Variant

Use different scaling parameters for positive and negative weights (Trained Ternary Quantization)

Loss-Aware Ternarization (LAT2)

The optimal $\hat{\mathbf{w}}_l^t$ is of the form $\alpha_l^t \mathbf{p}_l^t + \beta_l^t \mathbf{q}_l^t$, where $\alpha_l^t = \frac{\|\mathbf{p}_l^t \odot \mathbf{d}_l^{t-1} \odot \mathbf{w}_l^t\|_1}{\|\mathbf{p}_l^t \odot \mathbf{d}_l^{t-1}\|_1}$, $\mathbf{p}_l^t = \mathbf{I}^+_{\alpha_l^t/2}(\mathbf{w}_l^t)$, $\beta_l^t = \frac{\|\mathbf{q}_l^t \odot \mathbf{d}_l^{t-1} \odot \mathbf{w}_l^t\|_1}{\|\mathbf{q}_l^t \odot \mathbf{d}_l^{t-1}\|_1}$, and $\mathbf{q}_l^t = \mathbf{I}^-_{\beta_l^t/2}(\mathbf{w}_l^t)$.

- $\mathbf{I}^+_\Delta(\mathbf{x})$ considers only the positive threshold, i.e., $[\mathbf{I}^+_\Delta(\mathbf{x})]_i = 1$ if $x_i > \Delta$, and 0 otherwise
- $\mathbf{I}^-_\Delta(\mathbf{x})$ considers only the negative threshold, i.e., $[\mathbf{I}^-_\Delta(\mathbf{x})]_i = 1$ if $x_i < -\Delta$, and 0 otherwise

m-bit Quantization

- linear quantization
  $\mathcal{Q} = \left\{-1, -\frac{k-1}{k}, \dots, -\frac{1}{k}, 0, \frac{1}{k}, \dots, \frac{k-1}{k}, +1\right\}$
- logarithmic quantization
  $\mathcal{Q} = \left\{-1, -\frac{1}{2}, \dots, -\frac{1}{2^{k-1}}, 0, \frac{1}{2^{k-1}}, \dots, \frac{1}{2}, +1\right\}$

Loss-Aware Quantization (LAQ)

- optimal $\hat{\mathbf{w}}_l^t$ has the form $\alpha\mathbf{b}$
- when $\mathbf{b}$ is fixed, $\alpha = \frac{\|\mathbf{b}\odot\mathbf{d}_l^{t-1}\odot\mathbf{w}_l^t\|_1}{\|\mathbf{b}\odot\mathbf{d}_l^{t-1}\|_1}$
- when α is fixed, projects each entry of $\frac{\mathbf{w}_l^t}{\alpha}$ to the nearest element in $\mathcal{Q}$

Experiment: Feedforward Neural Networks

1. **MNIST**
   - 28×28 gray images from 10 digit classes
   - 4-layer multi-layer perceptron
2. **CIFAR-10**
   - 32×32 color images from 10 object classes
   - VGG-like CNN
3. **CIFAR-100**
   - 32×32 color images from 100 object classes
   - VGG-like CNN
4. **SVHN**
   - 32×32 color images from 10 digit classes
   - VGG-like CNN

Results: Testing Error

			MNIST	CIFAR-10	CIFAR-100	SVHN
no binarization		full-precision	1.11	10.38	39.06	2.28
binarization		BinaryConnect	1.28	9.86	46.42	2.45
		BWN	1.31	10.51	43.62	2.54
		LAB	1.18	10.50	43.06	2.35
ternarize	1 scal.	TWN	1.23	10.64	43.49	2.37
		LBNN	2.12	35.84	-	-
		LAT	1.15	10.47	39.10	2.30
		LAT (iter)	1.14	10.38	39.19	2.30
	2 scal.	TTQ	1.20	10.59	42.09	2.38
		LAT2	1.20	10.45	39.01	2.34
		LAT2 (iter)	1.19	10.48	38.84	2.35
3-bit quantization		DoReFa-Net	1.31	10.54	45.05	2.39
		LAQ3 (linear)	1.20	10.67	38.70	2.34
		LAQ3 (log)	1.16	10.52	38.50	2.29

Weight-ternarized networks (on **MNIST**, **CIFAR-100** and **SVHN**)

- perform better than weight-binarized networks
- comparable to the full-precision networks

Among Ternarization Schemes

			MNIST	CIFAR-10	CIFAR-100	SVHN
no binarization		full-precision	1.11	10.38	39.06	2.28
binarization		BinaryConnect	1.28	9.86	46.42	2.45
		BWN	1.31	10.51	43.62	2.54
		LAB	1.18	10.50	43.06	2.35
ternarize	1 scal.	TWN	1.23	10.64	43.49	2.37
		LBNN	2.12	35.84	-	-
		LAT	1.15	10.47	39.10	2.30
		LAT (iter)	1.14	10.38	39.19	2.30
	2 scal.	TTQ	1.20	10.59	42.09	2.38
		LAT2	1.20	10.45	39.01	2.34
		LAT2 (iter)	1.19	10.48	38.84	2.35
3-bit quantization		DoReFa-Net	1.31	10.54	45.05	2.39
		LAQ3 (linear)	1.20	10.67	38.70	2.34
		LAQ3 (log)	1.16	10.52	38.50	2.29

- LAT and its variants have the lowest errors

Ternarization Using Two Scaling Parameters

			MNIST	CIFAR-10	CIFAR-100	SVHN
no binarization		full-precision	1.11	10.38	39.06	2.28
binarization		BinaryConnect	1.28	9.86	46.42	2.45
		BWN	1.31	10.51	43.62	2.54
		LAB	1.18	10.50	43.06	2.35
ternarize	1 scal.	TWN	1.23	10.64	43.49	2.37
		LBNN	2.12	35.84	-	-
		LAT	1.15	10.47	39.10	2.30
		LAT (iter)	1.14	10.38	39.19	2.30
	2 scal.	TTQ	1.20	10.59	42.09	2.38
		LAT2	1.20	10.45	39.01	2.34
		LAT2 (iter)	1.19	10.48	38.84	2.35
3-bit quantization		DoReFa-Net	1.31	10.54	45.05	2.39
		LAQ3 (linear)	1.20	10.67	38.70	2.34
		LAQ3 (log)	1.16	10.52	38.50	2.29

- LAT2 always better than TTQ
- outperforms one scaling parameter only on **CIFAR-100**
  - capacities of deep networks often larger than needed $\rightarrow$ crudely quantized weights may be enough
  - weight quantization is a form of regularization

3-bit Quantization Algorithms

		MNIST	CIFAR-10	CIFAR-100	SVHN
no binarization	full-precision	1.11	10.38	39.06	2.28
binarization	BinaryConnect	1.28	9.86	46.42	2.45
	BWN	1.31	10.51	43.62	2.54
	LAB	1.18	10.50	43.06	2.35
ternarize, 1 scal.	TWN	1.23	10.64	43.49	2.37
	LBNN	2.12	35.84	-	-
	LAT	1.15	10.47	39.10	2.30
	LAT (iter)	1.14	10.38	39.19	2.30
ternarize, 2 scal.	TTQ	1.20	10.59	42.09	2.38
	LAT2	1.20	10.45	39.01	2.34
	LAT2 (iter)	1.19	10.48	38.84	2.35
3-bit quantization	DoReFa-Net	1.31	10.54	45.05	2.39
	LAQ3 (linear)	1.20	10.67	38.70	2.34
	LAQ3 (log)	1.16	10.52	38.50	2.29

- LAQ3 with logarithmic quantization is best
- outperforms others on **CIFAR-100** and **SVHN**
  - more quantization bits useful when the weight-quantized network does not have enough capacity

Experiments: Recurrent Neural Network (LSTM)

task

- input: sequence of characters
- predict: the next character at each time step

1. **War and Peace**
   - almost entirely English text
   - 3258K characters, and a vocabulary size of 87
2. source code of the **Linux Kernel**
   - consists of 621K characters and has a vocabulary size of 101
3. **Penn Treebank**
   - 50 different characters, including English characters, numbers, and punctuations

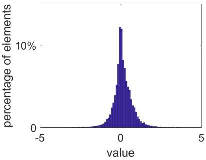
Results

Testing cross-entropy values

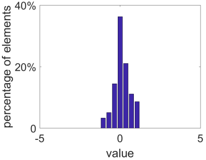
			War & Peace	Linux	Treebank
no binarization		full-precision	1.268	1.326	1.083
binarization		BinaryConnect	2.942	3.532	1.737
		BWN	1.313	1.307	1.078
		LAB	1.291	1.305	1.081
ternarization	1 scaling	TWN	1.290	1.280	1.045
		LAT_e	1.248	1.256	1.022
		LAT_a	1.253	1.264	1.024
	2 scaling	TTQ	1.272	1.302	1.031
		LAT2_e	1.239	1.258	1.018
		LAT2_a	1.245	1.258	1.015
3-bit quantization		DoReFa-Net	1.349	1.276	1.017
		LAQ3 (linear)	1.282	1.327	1.017
		LAQ3 (log)	1.268	1.273	1.009
4-bit quantization		DoReFa-Net	1.328	1.320	1.019
		LAQ4 (linear)	1.294	1.337	1.046
		LAQ4 (log)	1.272	1.319	1.016

Experiments: Logarithmic vs Linear Quantization

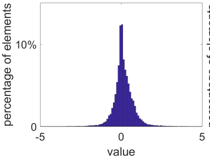
- distributions of the full-precision weights are bell-shaped.
- logarithmic quantization can give finer resolutions to many of the weights which have small magnitudes.



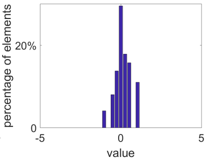
Full-precision weights.



Quantized weights.



Full-precision weights.



Quantized weights.

Variant: Add More Terms to the Objective

(Zhou et al., 2018)

Explicit Loss-Error-Aware Quantization for Low-Bit Deep Neural Networks

- combines weight approximation error and quantization's impact on loss function

$$\begin{aligned} \min_{\hat{\mathbf{w}}} \quad & \ell(\hat{\mathbf{w}}) + a_1 L_p(\mathbf{w}, \hat{\mathbf{w}}) + a_2 E(\mathbf{w}, \hat{\mathbf{w}}) \\ \text{s.t.} \quad & \hat{\mathbf{w}}_l = \alpha_l \mathbf{b}_l,\ \alpha_l > 0,\ \mathbf{b}_l \in \mathcal{Q}^{n_l},\ \ l = 1, \dots, L \end{aligned}$$

- $L_p(\mathbf{w}, \hat{\mathbf{w}})$: difference in losses between quantized and full-precision models

$$L_p(\mathbf{w}, \hat{\mathbf{w}}) = |\ell(\hat{\mathbf{w}}) - \ell(\mathbf{w})|$$

- $E(\mathbf{w}, \hat{\mathbf{w}})$: approximation error between quantized weight and full-precision counterpart

$$E(\mathbf{w}, \hat{\mathbf{w}}) = \|\mathbf{w} - \hat{\mathbf{w}}\|^2$$

Quantizing Activations

Beyond Quantizing Weights

what else can be quantized? $\rightarrow$ activation

How to Quantize Activation?

forward pass

easy (as in weight quantization)

backward pass

how to backpropagate gradient?

Example (threshold function)

- $f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$
- $\frac{df}{dx} = 0 \rightarrow$ network will not learn anything

Straight-Through Estimator

straight-through estimator (Hinton 2012)

- backpropagate as if the activation had been the identity function
- simply copy the gradient w.r.t. the discrete output directly as an estimator of the gradient with respect to the input argument

Binarized Neural Network

(Hubara et al., 2016)

Binarized Neural Networks

- BinaryConnect: binarize weights
- BNN: binarize both weights and activations

- forward: $r_o = \text{sign}(r_i)$
- backward: $\frac{\partial c}{\partial r_i} = \frac{\partial c}{\partial r_o} \mathbf{I}_{|r_i| \leq 1}$
  - gradient becomes zero when r_i is too large

XNOR-Net

(Rastegari et al., 2016)

XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks

- BWN: binarize weights
- XNOR-Net: binarize both weights and activations

- approximate weight $\mathbf{w}$ as $\alpha\mathbf{b}$ ($\mathbf{b}$ binary)
- approximate activation $\mathbf{x}$ as $\beta\mathbf{h}$ ($\mathbf{h}$ binary)
- minimize the difference between $\mathbf{w}'\mathbf{x}$ and $(\alpha\mathbf{b})'(\beta\mathbf{h})$

$$\begin{array}{ll} \min_{\alpha,\mathbf{b},\beta,\mathbf{h}} & \|\mathbf{w}'\mathbf{x} - \alpha\beta\mathbf{b}'\mathbf{h}\|^2 \\ \text{s.t.} & \alpha, \beta > 0, \mathbf{b}, \mathbf{h} \in \{-1, 1\}^{n_l} \end{array}$$

- solution

$$\alpha = \frac{\|\mathbf{w}\|_1}{n_l},\ \mathbf{b} = \text{sign}(\mathbf{w});\ \ \beta = \frac{\|\mathbf{x}\|_1}{n_l},\ \mathbf{h} = \text{sign}(\mathbf{x})$$

BWN vs XNOR-Net

	Network Variations		Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)
Standard Convolution	Real-Value Inputs 0.11 -0.21 ... -0.34 -0.25 0.61 ... 0.52	Real-Value Weights 0.12 -1.2 ... 0.41 -0.2 0.5 ... 0.68	+ , − , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs 0.11 -0.21 ... -0.34 -0.25 0.61 ... 0.52	Binary Weights 1 -1 ... 1 -1 1 ... 1	+ , −	~32x	~2x	%56.8
BinaryWeight Binary Input (**XNOR-Net**)	Binary Inputs 1 -1 ... -1 -1 1 ... 1	Binary Weights 1 -1 ... 1 -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

[from (Rastegari et al., 2016)]

Experimental Results

Classification accuracy (%)									
Binary-weight				Binary-input-binary-weight				Full-precision	
BWN		BC [11]		XNOR-Net		BNN [11]		AlexNet [1]	
Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
56.8	**79.4**	35.4	61.0	**44.2**	**69.2**	27.9	50.42	56.6	80.2

[from (Rastegari et al., 2016)]

m-Bit Activation: DoReFa-Net

activation functions like ReLU can be unbounded

(Zhou et al., 2016)

DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients

- assume the output of the previous layer has passed through a bounded activation function in $[0, 1]$

$$r_o = \text{quantize}_m(r_i) = \frac{1}{2^m - 1}\text{round}((2^m - 1)r_i)$$

- quantizes $r_i \in [0, 1]$ to the nearest number in $\{0, \frac{1}{2^m-1}, \ldots, \frac{2^m-2}{2^m-1}, 1\}$

Experimental Results

Table 2: Comparison of prediction accuracy for ImageNet with different choices of bitwidth in a DoReFa-Net. W, A, G are bitwidths of weights, activations and gradients respectively. Single-crop top-1 accuracy is given. Note the BNN result is reported by (Rastegari et al., 2016), not by original authors. We do not quantize the first and last layers of AlexNet to low bitwidth, as BNN and XNOR-Net do.

W	A	G	Training Complexity	Inference Complexity	Storage Relative Size	AlexNet Accuracy
1	1	6	7	1	1	0.395
1	1	8	9	1	1	0.395
1	1	32	-	1	1	0.279 (BNN)
1	1	32	-	1	1	0.442 (XNOR-Net)
1	1	32	-	1	1	0.401
1	1	32	-	1	1	0.436 (initialized)
1	2	6	8	2	1	0.461
1	2	8	10	2	1	0.463
1	2	32	-	2	1	0.477
1	2	32	-	2	1	0.498 (initialized)
1	3	6	9	3	1	0.471
1	3	32	-	3	1	0.484
1	4	6	-	4	1	0.482
1	4	32	-	4	1	0.503
1	4	32	-	4	1	0.530 (initialized)
8	8	8	-	-	8	0.530
32	32	32	-	-	32	0.559

Parameterized Clipping

- DoReFa-Net: assumes that output of the previous layer has passed through a bounded activation function in $[0, 1]$
- alternatively, can perform clipping

setting a global clipping rule may be sub-optimal

(Choi et al., 2018)

PACT: Parameterized Clipping Activation for Quantized Neural Networks

1. clip activation

$$\text{PACT}(x) = 0.5(|x| - |x - \alpha| + \alpha) = \begin{cases} 0 & x \in (-\infty, 0) \\ x & x \in [0, \alpha) \\ \alpha & x \in [\alpha, +\infty) \end{cases}$$

   - α can be learned by SGD

2. truncated activation output is then linearly quantized to m bits

Experimental Results

- weights quantized with DoReFa
- activations quantized with PACT

Network	FullPrec	DoReFa				PACT			
		2b	3b	4b	5b	2b	3b	4b	5b
CIFAR10	0.916	0.882	0.899	0.905	0.904	0.897	0.911	0.913	0.917
SVHN	0.978	0.976	0.976	0.975	0.975	0.977	0.978	0.978	0.979
AlexNet	0.551	0.536	0.550	0.549	0.549	0.550	0.556	0.557	0.557
ResNet18	0.702	0.626	0.675	0.681	0.684	0.644	0.681	0.692	0.698
ResNet50	0.769	0.671	0.699	0.714	0.714	0.722	0.753	0.765	0.767

[from (Choi et al., 2018)]

Quantizing Gradients

Motivation

- quantized weights + activations $\rightarrow$ speed up forward pass

what about backward pass?

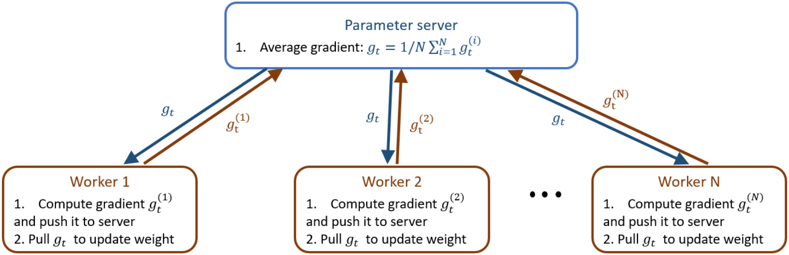
BNN and XNOR-Net

- weights are binarized, gradients are in full precision
- backward-pass still requires convolution between 1-bit numbers and 32-bit floating-points

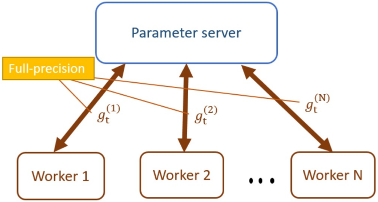
Another Motivation: Distributed Training

quantized networks

- small storage and fast inference
- training can still be time-consuming $\rightarrow$ distributed training



Problem with Distributed Training...



communication of gradients still expensive

Compressing Gradient: Gradient Sparsification

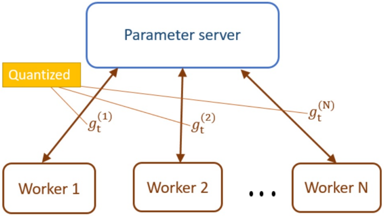
(Aji & Heafield 2017)

Sparse communication for distributed gradient descent

top-k operator

- only the k largest coordinates are transmitted
- achieves 22% speedup on 4 GPUs for training a large neural machine translation model

Compressing Gradient: Gradient Quantization



Example (32-bit full-precision gradient)

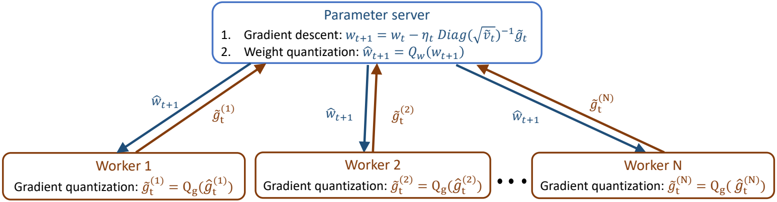
- gradient quantized to m bits $\rightarrow$ communication cost reduced $32/m$ times

how to quantize gradients?

$$\mathbf{g}_t \mapsto \underbrace{\|\mathbf{g}_t\|_\infty}_{\text{magnitude}} \cdot \underbrace{\text{sign}(\mathbf{g}_t)}_{\text{sign}} \odot \underbrace{\mathbf{q}_t}_{\text{value}}$$

- $\mathbf{q}_t \in (\mathcal{S}_g)^d$ where $\mathcal{S}_g = \{-B_k, \dots, -B_1, B_0, B_1, \dots, B_k\}$

Distributed Weight and Gradient Quantization



1. worker
   - computes full-precision gradient w.r.t. quantized weight
   - quantizes the gradient and send to parameter server
2. parameter server
   - synchronize and average quantized gradients
   - updates the full-precision weight
   - quantize weight and send back to all workers

Gradient Quantization in DoReFa-Net

- gradients are unbounded
- gradients may have significantly larger value range than activations

$$\text{Quantize}(dr) = \underbrace{2\text{max}(|g|)}_{\text{scale}} \left[\text{quantize}_m \left(\frac{dr}{\underbrace{2\max(|dr|)}_{\text{normalize}}} + \frac{1}{2} \right) - \frac{1}{2} \right]$$

- dr: back-propagated gradient of the output r
- $\text{quantize}_m(x) = \frac{1}{2^m - 1}\text{round}((2^m - 1)x)$:
  - quantized values in $\{-1, -\frac{2^m-2}{2^m-1}, \ldots, -\frac{1}{2^m-1}, \frac{1}{2^m-1}, \ldots, \frac{2^m-2}{2^m-1}, 1\}$

Gradient Quantization in DoReFa-Net...

- add extra noise to alleviate quantization noise
  - $N(m) = \frac{\sigma}{2^m - 1}$, where $\sigma \sim \text{Uniform}(-0.5, 0.5)$

$$2\text{max}(|g|)\left[\text{quantize}_m\left(\frac{dr}{2\max(|dr|)} + \frac{1}{2} + N(m)\right) - \frac{1}{2}\right]$$

- empirically, succeed in quantizing gradients to less than 8 bits, while still achieving comparable prediction accuracy

Convergence Analysis of Quantized Networks

(Hou et al, 2019)

Analysis of quantized models

Setting

Online learning

- continually adapts the model with a sequence of observations
- at time t
  - algorithm picks a model with parameter $\mathbf{w}^t \in \mathcal{S}$
  - algorithm then incurs a loss $f^t(\mathbf{w}^t)$

regret and average regret

$$R(T) = \sum_{t=1}^{T} f^t(\mathbf{w}^t) - f^t(\mathbf{w}^*), \quad \frac{R(T)}{T}$$

- $\mathbf{w}^* = \arg\min_{\mathbf{w} \in \mathcal{S}} \sum_{t=1}^{T} f^t(\mathbf{w})$: best model parameter in hindsight

Assumptions

(A1) f_t is convex

(A2) f_t is twice differentiable with Lipschitz-continuous gradient

(A3) f_t has bounded gradient

- $\|\nabla f_t(\mathbf{w})\| \leq G$ and $\|\nabla f_t(\mathbf{w})\|_\infty \leq G_\infty$ for all $\mathbf{w} \in \mathcal{S}$

(A4) $\|\mathbf{w}_m - \mathbf{w}_n\| \leq D$ and $\|\mathbf{w}_m - \mathbf{w}_n\|_\infty \leq D_\infty$ for all $\mathbf{w}_m, \mathbf{w}_n \in \mathcal{S}$

Loss-Aware Weight Quantization (LAQ)

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \text{Diag}(\sqrt{\hat{\mathbf{v}}_t})^{-1}\hat{\mathbf{g}}_t$$

- $\hat{\mathbf{v}}_t$: moving average of (squared) $\hat{\mathbf{g}}_t$

LAQ

With full-precision gradients and $\eta_t = \eta/\sqrt{t}$,

$$\frac{R(T)}{T} \leq O\left(\frac{d}{\sqrt{T}}\right) + \frac{\sqrt{d}LD\alpha\Delta_w}{2}$$

- T: # iterations; d: dimension; Δ_w: quantization resolution

- speed: $O(1/\sqrt{T})$
- error: $\frac{\sqrt{d}LD\alpha\Delta_w}{2}$
  - if the full-precision weight may be outside the representable range: $LD\sqrt{D^2 + \frac{d\alpha^2\Delta_w^2}{4}}$

Loss-aware Weight Quantization

Theorem (Full-Precision Gradient)

- *speed:* $O(1/\sqrt{T})$; *error:* $\frac{\sqrt{d}LD\alpha\Delta_w}{2}$

Theorem (Quantized Gradient)

- *speed: slowed by a factor of* $\sqrt{\frac{1+\sqrt{2d-1}}{2}\Delta_g + 1}$
  - Δ_g: *gradient quantization resolution*
- *error: no change*

problems

1. deep networks typically have a large d
2. distributed learning prefers small number of bits for gradients $\rightarrow$ large Δ_g

Gradient Clipping

- clip gradient by a maximum magnitude of $c\sigma$
  - c: clipping factor
  - σ : standard deviation of elements in $\mathbf{g}_t$

$$\text{Clip}(\mathbf{g}_{t,i}) = \begin{cases} \mathbf{g}_{t,i} & |\mathbf{g}_{t,i}| \leq c\sigma \\ \text{sign}(\mathbf{g}_{t,i}) \cdot c\sigma & \text{otherwise} \end{cases}$$

- found to be empirically useful when gradients are ternarized (Wen et al., 2018)

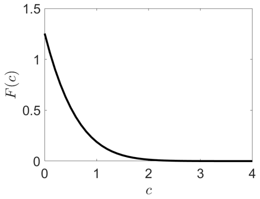
Weight Quantization with Quantized Clipped Gradient

Theorem (Quantized Gradient)

- *speed: slowed by a factor* $\sqrt{\frac{1+\sqrt{2d-1}}{2}\Delta_g + 1}$
- *error: no change*

Theorem (Quantized Clipped Gradient)

- *speed: slows down by a factor* $\sqrt{(2/\pi)^{\frac{1}{2}} c\Delta_g + 1}$ *(indep of d!)*
- *error: an extra error term* $\sqrt{d}D\sigma(2/\pi)^{\frac{1}{4}}\sqrt{F(c)}$

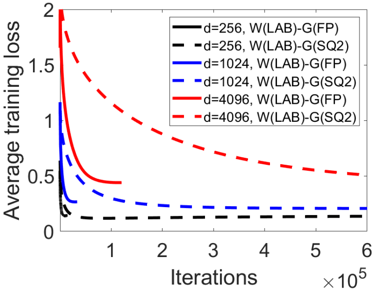


smaller c

- faster convergence
- larger $F(c) \rightarrow$ larger error

Synthetic Data: Vary d

- linear model
- weights: quantized to 1 bit (LAB)
- gradient: full-precision (FP) or quantized to 2 bits (SQ2)

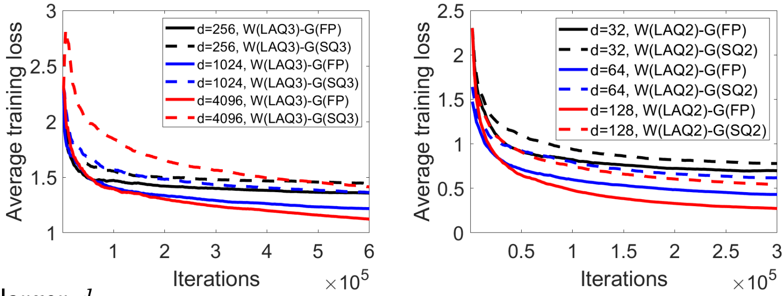


larger d

- a larger loss upon convergence
- slower convergence particularly for quantized gradients

CIFAR-10: Vary d

- single hidden-layer perceptron: d is number of hidden units
- Cifarnet (CNN): d is number of filters in each conv layer
- weights: quantized to 1 bit (LAB)
- gradients: full-precision (FP) or quantized to 2 bits (SQ2)

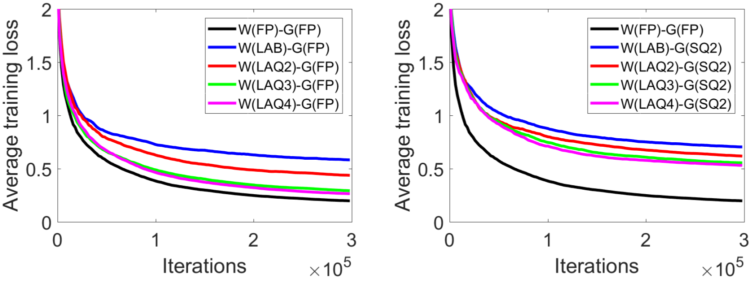


larger d

- slower convergence particularly for quantized gradients
- does not necessarily lead to a larger loss upon convergence

CIFAR-10: Weight Quantization Resolution Δ_w

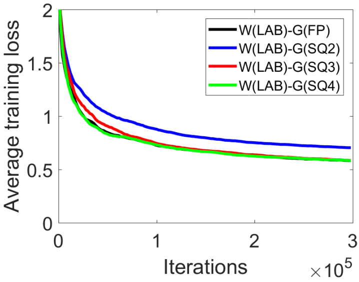
- weight: full-precision (W(FP)) or W(LAB), W(LAQ2), W(LAQ3), W(LAQ4)
- gradient: full-precision (G(FP)) or 2-bit quantized (G(SQ2)) without clipping

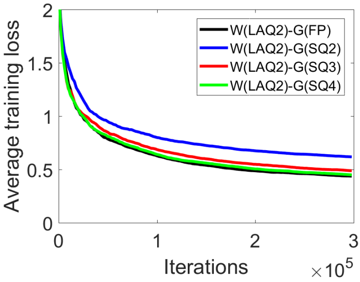


- weight-quantized: larger training losses than full-precision
- more weight bits are used $\rightarrow$ smaller is final loss

CIFAR-10: Gradient Quantization Resolution Δ_g

- weight: binarized (W(LAB)) or 2-bit quantized (W(LAQ2))
- gradients (not clipped): G(FP); G(SQ2); G(SQ3); G(SQ4)





- fewer bits $\rightarrow$ larger final error
- degradation negligible when 4 bits are used for gradients

Testing Accuracy on CIFAR-10: Cifarnet

- weights: 1-bit (LAB), m-bit (LAQm)
- gradients: full-precision (FP) or $m = \{2, 3, 4\}$-bit (SQm)
- two workers

gradient \ weight	FP	LAB	LAQ2	LAQ3	LAQ4
FP	83.74	80.37	82.11	83.14	83.35
SQ2 (no clipping)	81.40	78.67	80.27	81.27	81.38
SQ2 (clip,$c = 3$)	82.99	80.25	81.59	83.14	83.40
SQ3 (no clipping)	83.24	80.18	81.63	82.75	83.17
SQ3 (clip, $c = 3$)	83.89	80.13	81.77	82.97	83.43
SQ4 (no clipping)	83.64	80.44	81.88	83.13	83.47
SQ4 (clip, $c = 3$)	83.80	79.27	81.42	82.77	83.43

- degradation is small when 3 or 4 bits are used
- more gradient bits $\rightarrow$ clipping is not needed

ImageNet: Top-1 and Top-5 Accuracies

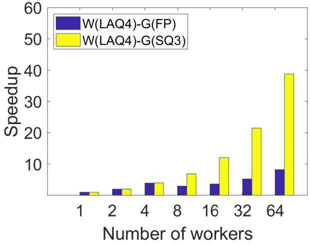
weight	gradient	$N = 2$		$N = 4$		$N = 8$	
		top-1	top-5	top-1	top-5	top-1	top-5
FP	FP	55.08	78.33	55.45	78.57	55.40	78.69
LAQ4	FP	53.79	77.21	54.22	77.53	54.73	78.12
	SQ3 (no clipping)	52.48	75.97	52.87	76.40	53.18	76.62
	SQ3 (clip, $c = 3$)	54.13	77.27	54.23	77.55	54.34	78.07
LAQ6	FP	54.23	77.54	54.41	77.56	54.75	78.18
	SQ3 (no clipping)	52.64	76.08	53.00	76.38	53.08	76.81
	SQ3 (clip, $c = 3$)	54.21	77.32	54.53	77.85	54.61	78.10

- quantized clipped gradient
  - outperforms quantized non-clipped gradient
  - comparable accuracy as full-precision gradient
- LAQ6 using quantized clipped gradients has less than 1% top-1 accuracy drop compared to using full-precision weights

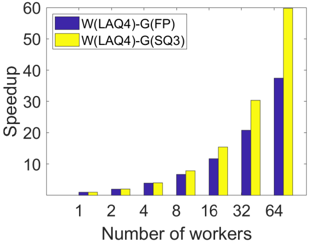
ImageNet: Speedup

- 16-node GPU cluster (each node has 4 1080ti GPUs)
- weight: quantized to 4 bits (LAQ4)
- gradient: full-precision (FP) or quantized to 3 bits (SQ3)

1Gbps Ethernet



10Gbps Ethernet



- small bandwidth: communication is bottleneck → quantizing gradient significantly faster
- larger bandwidth: difference in speedups smaller
- quantized grad (1Gbps) similar speedup as full-precision grad (10Gbps)