Dynamic-Precision Training of Deep Neural Network Models

Paul Estano

With Silviu I. Filip & Elisa Riccietti

November 15, 2022

Universite de Rennes 1 - IRISA - Inria

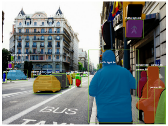
Overview

1. Context & Motivation
2. Neural Network Learning in Low/Mixed-Precision
3. Optimization in Dynamic-Precision

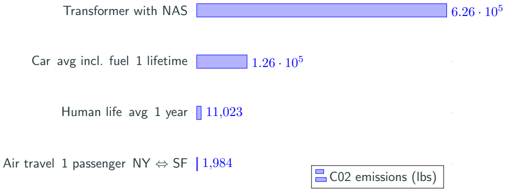
Context & Motivation

DNNs are (almost) everywhere

- *Deep Learning* (DL) has enabled many breakthroughs in recent years.



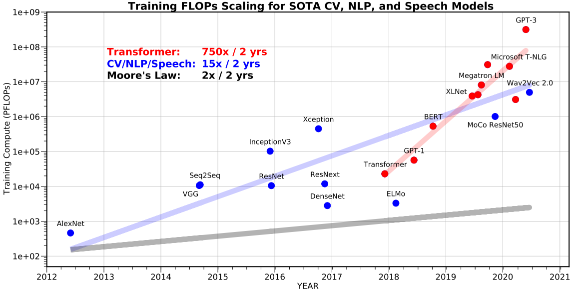
- The development and the use of these algorithms have a consequential impact on the environment:



(Taken from [Strubell et al., 2019])

DNNs are getting bigger

Training DNNs is computationally and memory expensive[1].



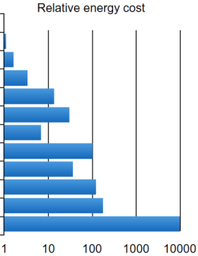
How can we make it more efficient and scalable?

[1] https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8

DNN Quantization

- **Idea**: Use less bits to represent numbers during DNN training.

Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FB Add	0.4
32b FB Add	0.9
8b Mult	0.2
32b Mult	3.1
16b FB Mult	1.1
32b FB Mult	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

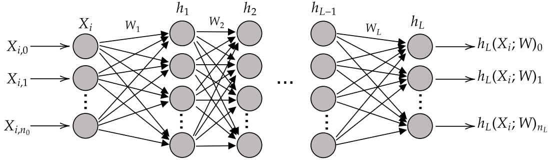


Taken from [Horowitz et al., 2014]

- **How do we use smaller number formats without hurting performance?**

Neural Network Learning in Low/Mixed-Precision

Neural Network Learning

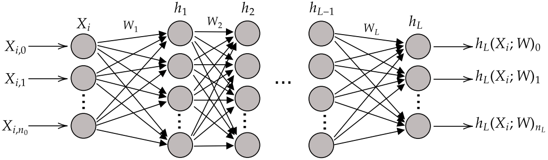


- We study a neural network composed of L layers h_ℓ:

$$h_\ell(X; W) = \phi_\ell(W_\ell h_{\ell-1}(X; W))$$

$W = \{W_\ell\}_{\ell=1}^{L}$ a set of weight matrices, $\{\phi_\ell\}_{\ell=1}^{L}$ a set of non-linear functions.

Neural Network Learning

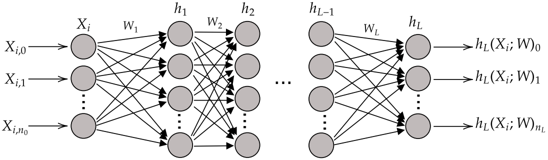


- We study a neural network composed of L layers h_ℓ:

$$h_\ell(X;W) = \phi_\ell(W_\ell h_{\ell-1}(X;W))$$

  $W = \{W_\ell\}_{\ell=1}^L$ a set of weight matrices, $\{\phi_\ell\}_{\ell=1}^L$ a set of non-linear functions.

Neural Network Learning



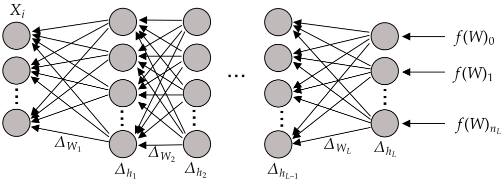
- We study a neural network composed of L layers h_ℓ:

$$h_\ell(X; W) = \phi_\ell(W_\ell h_{\ell-1}(X; W))$$

  $W = \{W_\ell\}_{\ell=1}^{L}$ a set of weight matrices, $\{\phi_\ell\}_{\ell=1}^{L}$ a set of non-linear functions.
- Given a training set $\{(X_i, Y_i)\}_{i=1}^{N}$ and an *error* function, we want to minimize the *empirical risk* f:

$$\arg\min_W f(W) = \arg\min_W \frac{1}{N} \sum_{i=1}^{N} error(h_L(X_i; W), Y_i)$$

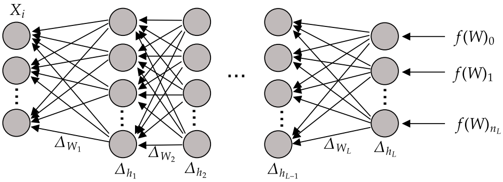
The cost of Neural Network Training



$$W_\ell^{(k+1)} \leftarrow W_\ell^{(k)} - \gamma \frac{\partial f(W^{(k)})}{\partial W_\ell^{(k)}}$$

Requires several quantities:

The cost of Neural Network Training

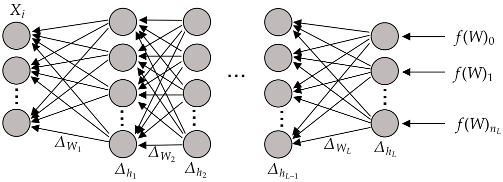


$$W_\ell^{(k+1)} \leftarrow W_\ell^{(k)} - \gamma \frac{\partial f(W^{(k)})}{\partial W_\ell^{(k)}}$$

Requires several quantities:

- Forward activations (like for inference).

The cost of Neural Network Training

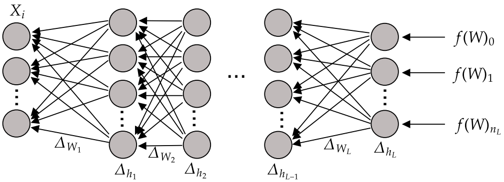


$$W_\ell^{(k+1)} \leftarrow W_\ell^{(k)} - \gamma \frac{\partial f(W^{(k)})}{\partial W_\ell^{(k)}}$$

Requires several quantities:

- Forward activations (like for inference).
- Backward gradients (*backpropagation*):
  - With respect to activations Δ_{h_ℓ}
  - With respect to weights Δ_{W_ℓ}

The cost of Neural Network Training



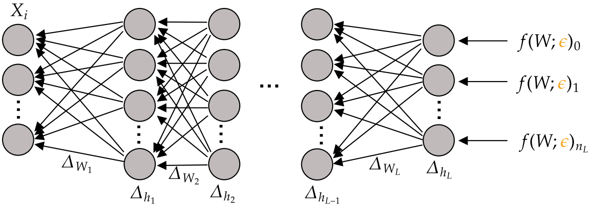
$$W_\ell^{(k+1)} \leftarrow W_\ell^{(k)} - \gamma \frac{\partial f(W^{(k)})}{\partial W_\ell^{(k)}}$$

Requires several quantities:

- Forward activations (like for inference).
- Backward gradients (*backpropagation*):
  - With respect to activations Δ_{h_ℓ}
  - With respect to weights Δ_{W_ℓ}

Can we use mixed-precision to reduce the cost of training?

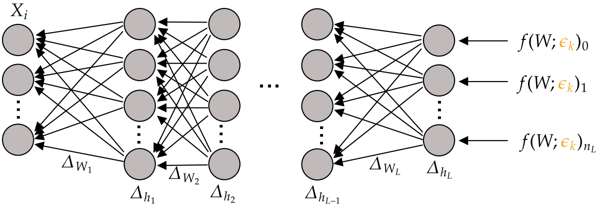
Neural Network Learning in Low-Precision



- Operations are accessible up to a precision ϵ. We now look for:

$$\arg\min_{W} f(W) \text{ s.t. } |\hat{f}(W,\epsilon) - f(W)| \leqslant \epsilon$$

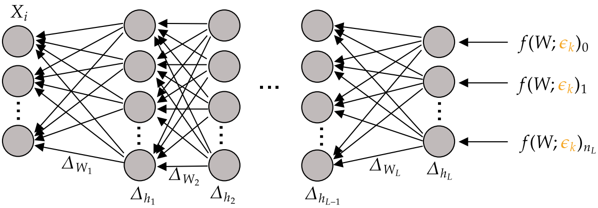
Neural Network Learning in Mixed-Precision



- Operations are accessible up to a precision ϵ_k. We now look for:

$$\arg\min_{W} f(W) \text{ s.t. } |\hat{f}(W,\epsilon_k) - f(W)| \leqslant \epsilon_k$$

Neural Network Learning in Mixed-Precision

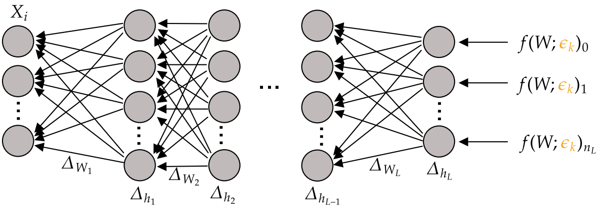


- Operations are accessible up to a precision ϵ_k. We now look for:

$$\arg\min_W f(W) \text{ s.t. } |\hat{f}(W,\epsilon_k) - f(W)| \leqslant \epsilon_k$$

- Reducing precision during training is hard:
  - vanishing & exploding gradients
  - small parameter updates

Neural Network Learning in Mixed-Precision



- Operations are accessible up to a precision ϵ_k. We now look for:

$$\arg\min_W f(W) \text{ s.t. } |\hat{f}(W,\epsilon_k) - f(W)| \leqslant \epsilon_k$$

- Reducing precision during training is hard:
  - vanishing & exploding gradients
  - small parameter updates
- Large dynamic range required $\rightarrow$ use floating-point

Recent results in MP Training

Comparison of recent MP training methods:

Method	Formats ((Exponent, Mantissa) / Width)				Top-1 Accuracy	
	w	Δ_W	Δ_{h_ℓ}	Acc.	FP32	Proposed
SWALP [Yang et al., 2019]	8	8	8	32	70.3	65.8
S2FP8 [Cambier et al., 2020]	(5,2)/(8,23)	(5,2)	(5,2)	(8,23)	70.3	69.6
HFP8 [Sun et al., 2019]	(4,3)	(6,9)	(5,2)	(6,9)	69.4	69.4
BM8 [Fox et al., 2021]	(2,5)	(6,9)	(4,3)	31	69.7	69.8
FP8-SEB [Park et al., 2022]	(4,3)	(4,3)	(4,3)	(8,23)	69.7	69.0
FP134 [Lee et al., 2022]	(3,4)	(3,4)	(3,4)	(8,23)	69.8	69.8

Results are ImageNet accuracy (%) using ResNet18 (adapted from[Lee et al., 2022]).

Notable Ideas:

- Shared scaling factor/bias at block or tensor level
  $\rightarrow$ shift dynamic range at runtime

Recent results in MP Training

Comparison of recent MP training methods:

Method	Formats ((Exponent, Mantissa) / Width)				Top-1 Accuracy	
	w	Δ_W	Δ_{h_ℓ}	Acc.	FP32	Proposed
SWALP [Yang et al., 2019]	8	8	8	32	70.3	65.8
S2FP8 [Cambier et al., 2020]	(5,2)/(8,23)	(5,2)	(5,2)	(8,23)	70.3	69.6
HFP8 [Sun et al., 2019]	(4,3)	(6,9)	(5,2)	(6,9)	69.4	69.4
BM8 [Fox et al., 2021]	(2,5)	(6,9)	(4,3)	31	69.7	69.8
FP8-SEB [Park et al., 2022]	(4,3)	(4,3)	(4,3)	(8,23)	69.7	69.0
FP134 [Lee et al., 2022]	(3,4)	(3,4)	(3,4)	(8,23)	69.8	69.8

Results are ImageNet accuracy (%) using ResNet18 (adapted from[Lee et al., 2022]).

Notable Ideas:

- Shared scaling factor/bias at block or tensor level
  $\rightarrow$ shift dynamic range at runtime
- Scale the loss function before back propagation
  $\rightarrow$ shifts gradients in a representable range to avoid under/overflows

Recent results in MP Training

Comparison of recent MP training methods:

Method	Formats ((Exponent, Mantissa) / Width)				Top-1 Accuracy	
	w	Δ_W	Δ_{h_ℓ}	Acc.	FP32	Proposed
SWALP [Yang et al., 2019]	8	8	8	32	70.3	65.8
S2FP8 [Cambier et al., 2020]	(5,2)/(8,23)	(5,2)	(5,2)	(8,23)	70.3	69.6
HFP8 [Sun et al., 2019]	(4,3)	(6,9)	(5,2)	(6,9)	69.4	69.4
BM8 [Fox et al., 2021]	(2,5)	(6,9)	(4,3)	31	69.7	69.8
FP8-SEB [Park et al., 2022]	(4,3)	(4,3)	(4,3)	(8,23)	69.7	69.0
FP134 [Lee et al., 2022]	(3,4)	(3,4)	(3,4)	(8,23)	69.8	69.8

Results are ImageNet accuracy (%) using ResNet18 (adapted from[Lee et al., 2022]).

Notable Ideas:

- Shared scaling factor/bias at block or tensor level
  $\rightarrow$ shift dynamic range at runtime
- Scale the loss function before back propagation
  $\rightarrow$ shifts gradients in a representable range to avoid under/overflows
- Ad-hoc rounding used in the quantizer: e.g. stochastic [Yang et al., 2019] & hysteresis [Lee et al., 2022]
- Most of these papers do not use any ad-hoc optimization method!

Current Limitations

- Use of *QPytorch* [Zhang et al., 2019] $\rightarrow$ not an adequate simulation of low precision hardware.
  Goal: Use MPtorch [Tatsumi et al., 2021]. Work in progress based on Pytorch and CUDA to simulate low-precision/mixed-precision computation.

Current Limitations

- Use of *QPytorch* [Zhang et al., 2019] $\rightarrow$ not an adequate simulation of low precision hardware.
  Goal: Use MPtorch [Tatsumi et al., 2021]. Work in progress based on Pytorch and CUDA to simulate low-precision/mixed-precision computation.
- Lack of theoretical results and use of heuristics.
  Goal: Have convergence guarantees in low/mixed-precision.

Optimization in Dynamic-Precision

Optimization in dynamic precision

- **Goal**: a method with
  - an automatic strategy to set the precision
  - good convergence properties

Optimization in dynamic precision

- **Goal**: a method with
  - an automatic strategy to set the precision
  - good convergence properties

Optimization in dynamic precision

- **Goal**: a method with
  - an automatic strategy to set the precision
  - good convergence properties
- [Conn et al., 2000, Ch. 10.6] introduces a *trust-region* based algorithm satisfying such requirements in a *classical setting*:

$$\min_x f(x)$$

Optimization in dynamic precision

- **Goal**: a method with
  - an automatic strategy to set the precision
  - good convergence properties
- [Conn et al., 2000, Ch. 10.6] introduces a *trust-region* based algorithm satisfying such requirements in a *classical setting*:

$$\min_x f(x)$$

- We investigate the extension of this scheme to a stochastic setting:

$$\min_x \mathbb{E}\left[f(x)\right]$$

Optimization in dynamic precision

- **Goal**: a method with
  - an automatic strategy to set the precision
  - good convergence properties
- [Conn et al., 2000, Ch. 10.6] introduces a *trust-region* based algorithm satisfying such requirements in a *classical setting*:

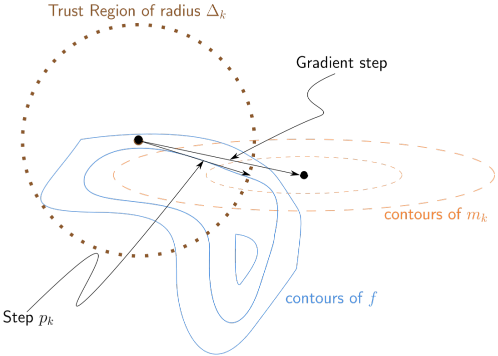
$$\min_x f(x)$$

- We investigate the extension of this scheme to a stochastic setting:

$$\min_x \mathbb{E}\left[f(x)\right]$$

- Notable application: training of deep neural networks

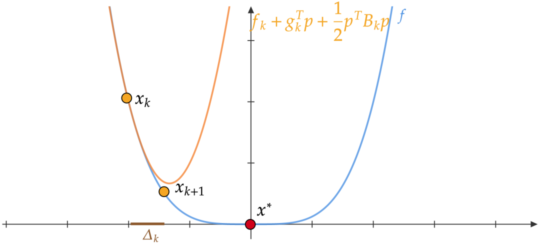
Trust-Region (TR) Methods [Nocedal et al., 2006]



At every iteration:

- Generate a step p_k with the help of a model m_k of the loss f
- The model is *trusted* inside of a region of radius Δ_k
- Δ_k varies based on the model performance in previous iterations

Trust-Region (TR) Methods [Nocedal et al., 2006]

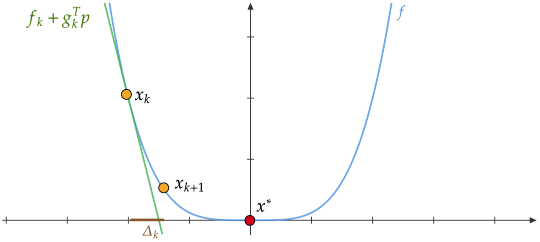


With $f_k = f(x_k)$, g_k and B_k resp. gradient and hessian of f

- TR methods often minimize a *quadratic model*

$$p_k = \underset{\|p\| \leqslant \Delta_k}{\arg\min}\, m_k(p) = \underset{\|p\| \leqslant \Delta_k}{\arg\min}\, f_k + g_k^T p + \frac{1}{2} p^T B_k p$$

Trust-Region (TR) Methods [Nocedal et al., 2006]

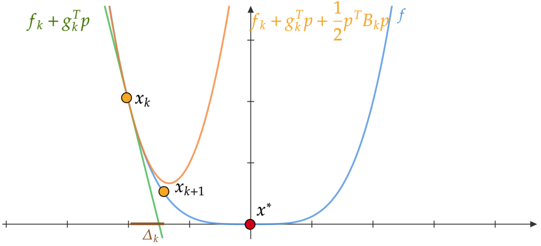


With $f_k = f(x_k)$, g_k and B_k resp. gradient and hessian of f

- TR methods often minimize a *quadratic model*, but using a *linear model* also works

$$p_k = \underset{\|p\| \leqslant \Delta_k}{\arg\min}\, m_k(p) = \underset{\|p\| \leqslant \Delta_k}{\arg\min}\, f_k + g_k^T p$$

Trust-Region (TR) Methods [Nocedal et al., 2006]



With $f_k = f(x_k)$, g_k and B_k resp. gradient and hessian of f

- TR methods often minimize a *quadratic model*, but using a *linear model* also works

$$p_k = \underset{\|p\| \leqslant \Delta_k}{\arg\min}\, m_k(p) = \underset{\|p\| \leqslant \Delta_k}{\arg\min}\, f_k + g_k^T p + \frac{1}{2} p^T B_k p$$

Basic Trust-Region

$$\min_x f(x)$$

Basic Trust-Region Algorithm [Conn et al., 2000, Sec. 6.1]

for $k = 0, 1, 2, \ldots$ **do**

1. Compute model $m_k(p)$
2. Compute the step p_k solving the sub-problem

$$\arg\min_{\|p\| \leqslant \Delta_k} m_k(p)$$

3. Trust-region update: $\rho_k = \dfrac{\overbrace{f(x_k) - f(x_k + p_k)}^{\text{actual reduction}}}{\underbrace{m_k(0) - m_k(p_k)}_{\text{predicted reduction}}}$

If $\rho_k \geqslant \eta_2$	Then $\Delta_{k+1} \leftarrow 2\Delta_k$	Accept $x_k + p_k$
If $\eta_2 > \rho_k \geqslant \eta_1$	Then $\Delta_{k+1} \leftarrow \Delta_k$	Accept $x_k + p_k$
If $\rho_k < \eta_1$	Then $\Delta_{k+1} \leftarrow 0.5\Delta_k$	Reject $x_k + p_k$

end for

where η_1, η_2 are constants satisfying $0 < \eta_1 \leqslant \eta_2 < 1$

Dynamic-Precision Trust-Region [Conn et al., 2000, Ch. 10.6]

- Operations are accessible up to a precision ϵ_k:

$$|\hat{f}(x, \epsilon_k) - f(x)| \leqslant \epsilon_k$$

- Want to solve $\min_x f(x)$ by having access just to $\hat{f}(x, \epsilon_k)$

Dynamic-Precision Trust-Region [Conn et al., 2000, Ch. 10.6]

- Operations are accessible up to a precision ϵ_k:

$$|\hat{f}(x, \epsilon_k) - f(x)| \leqslant \epsilon_k$$

- Want to solve $\min_x f(x)$ by having access just to $\hat{f}(x, \epsilon_k)$
- How can we choose the precision ϵ_k?

Dynamic-Precision Trust-Region [Conn et al., 2000, Ch. 10.6]

- Operations are accessible up to a precision ϵ_k:

$$|\hat{f}(x, \epsilon_k) - f(x)| \leqslant \epsilon_k$$

- Want to solve $\min_x f(x)$ by having access just to $\hat{f}(x, \epsilon_k)$
- How can we choose the precision ϵ_k?

  $\rightarrow$ guarantee monotonic decrease of the objective function f

Dynamic-Precision Trust-Region [Conn et al., 2000, Ch. 10.6]

- Operations are accessible up to a precision ϵ_k:

$$|\hat{f}(x, \epsilon_k) - f(x)| \leqslant \epsilon_k$$

- Want to solve $\min_x f(x)$ by having access just to $\hat{f}(x, \epsilon_k)$
- How can we choose the precision ϵ_k?
  $\rightarrow$ guarantee monotonic decrease of the objective function f

Theory (intuition)

In mixed-precision $\rho_k \geqslant \eta_1$ gives us:

$$\hat{f}(x_k, \epsilon_k) - \hat{f}(x_k + p_k, \epsilon_k) \geqslant \eta_1 [m_k(0) - m_k(p_k)] > 0$$

Dynamic-Precision Trust-Region [Conn et al., 2000, Ch. 10.6]

- Operations are accessible up to a precision ϵ_k:

$$|\hat{f}(x, \epsilon_k) - f(x)| \leqslant \epsilon_k$$

- Want to solve $\min_x f(x)$ by having access just to $\hat{f}(x, \epsilon_k)$
- How can we choose the precision ϵ_k?
  $\rightarrow$ guarantee monotonic decrease of the objective function f

Theory (intuition)

In mixed-precision $\rho_k \geqslant \eta_1$ gives us:

$$\hat{f}(x_k, \epsilon_k) - \hat{f}(x_k + p_k, \epsilon_k) \geqslant \eta_1 [m_k(0) - m_k(p_k)] > 0$$

By definition of $\hat{f}$:

$$\begin{aligned} f(x_k) - f(x_k + p_k) &\geqslant \hat{f}(x_k, \epsilon_k) - \hat{f}(x_k + p_k, \epsilon_k) - 2\epsilon_k \\ &\geqslant \underbrace{\eta_1 [m_k(0) - m_k(p_k)]}_{>0} - \underbrace{2\epsilon_k}_{?} > 0 \end{aligned}$$

Dynamic-Precision Trust-Region [Conn et al., 2000, Ch. 10.6]

- Operations are accessible up to a precision ϵ_k:

$$|\hat{f}(x, \epsilon_k) - f(x)| \leqslant \epsilon_k$$

- Want to solve $\min_x f(x)$ by having access just to $\hat{f}(x, \epsilon_k)$
- How can we choose the precision ϵ_k?
  $\rightarrow$ guarantee monotonic decrease of the objective function f

Theory (intuition)

In mixed-precision $\rho_k \geqslant \eta_1$ gives us:

$$\hat{f}(x_k, \epsilon_k) - \hat{f}(x_k + p_k, \epsilon_k) \geqslant \eta_1 [m_k(0) - m_k(p_k)] > 0$$

By definition of $\hat{f}$:

$$\begin{aligned} f(x_k) - f(x_k + p_k) &\geqslant \hat{f}(x_k, \epsilon_k) - \hat{f}(x_k + p_k, \epsilon_k) - 2\epsilon_k \\ &\geqslant \underbrace{\eta_1 [m_k(0) - m_k(p_k)]}_{>0} - \underbrace{2\epsilon_k}_{?} > 0 \end{aligned}$$

$$\implies \epsilon_k < \eta_1 \frac{1}{2} [m_k(0) - m_k(p_k)]$$

Dynamic-Precision Trust-Region [Conn et al., 2000, Ch. 10.6]

$$\min_x f(x) \text{ s.t. } |\hat{f}(x, \epsilon_k) - f(x)| \leqslant \epsilon_k$$

Trust-Region with Dynamic-Precision

for $k = 0, 1, 2, \ldots$ **do**

1. Compute model $m_k(p)$
2. Compute the step p_k solving the sub-problem

$$\arg\min_{\|p\| \leqslant \Delta_k} m_k(p)$$

if $\eta_1 \frac{1}{2}[m_k(0) - m_k(p_k)] > \epsilon_k$ **then**

3. Trust-region update: $\rho_k = \frac{\hat{f}(x_k, \epsilon_k) - \hat{f}(x_k + p_k, \epsilon_k)}{m_k(0) - m_k(p_k)}$

If $\rho_k \geqslant \eta_2$	Then $\Delta_{k+1} \leftarrow 2\Delta_k$	Accept $x_k + p_k$ & $\epsilon_{k+1} = \epsilon_k$
If $\eta_2 > \rho_k \geqslant \eta_1$	Then $\Delta_{k+1} \leftarrow \Delta_k$	Accept $x_k + p_k$ & $\epsilon_{k+1} = \epsilon_k$
If $\rho_k < \eta_1$	Then $\Delta_{k+1} \leftarrow 0.5\Delta_k$	Reject $x_k + p_k$ & $\epsilon_{k+1} = \epsilon_k$

else

Reduce ϵ_k and go to 1.

end if

end for

Dynamic-Precision Trust-Region

Limitations of the classical setting:

- small dimension: use a quadratic model
- deterministic setting: imposes a monotonic decrease of the loss

We adapt the previous algorithm to a stochastic setting:

- use batch training: stochasticity
- use a linear model: no hessian
- consider a non-monotonous loss function: ρ evaluated F times/epoch

Can be fine tuned for the training of neural networks:

- scale the loss: avoid vanishing gradients
- introduce momentum and weight-decay

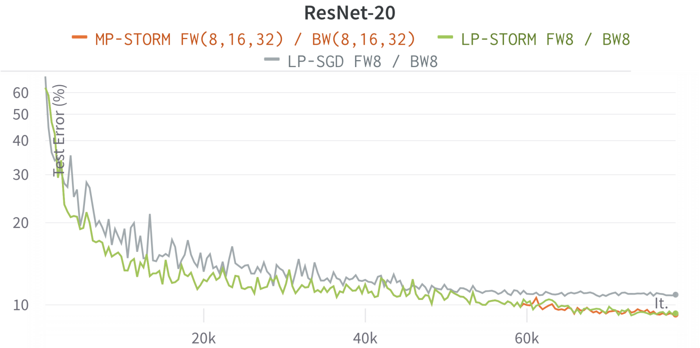
$$\min_x f(x) + \lambda \|x\|_2^2 \text{ s.t. } |\hat{f}(x, \epsilon_k) - f(x)| \leqslant \epsilon_k$$

Gradient estimate is $G_{k+1} = G_k + \mu \dfrac{\partial f(x)}{\partial x}$ & $G_0 = \dfrac{\partial f(x)}{\partial x}$

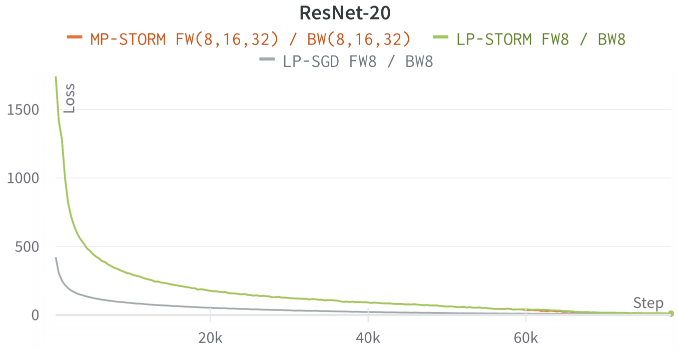
Setup:

- ResNet-20 [Zhang et al., 2015]
- On CIFAR-10 (data augmented)
- 200 Epochs with batch size $B = 128$
- **TR**: $TR_{init} = 0.1$
- **TR**: Floating Point on 8, 16, 32 bits (E4M3, E5M10, IEEE 754)
- **SGD**: $LR_{init} = 0.1$ with cosine annealing scheduler
- **SGD**: Floating Point on 8 bits
- **TR&SGD**: Loss Scaling is used ($f \times 1024$)
- **TR&SGD**: Quantization with *QPytorch* [Zhang et al., 2019]

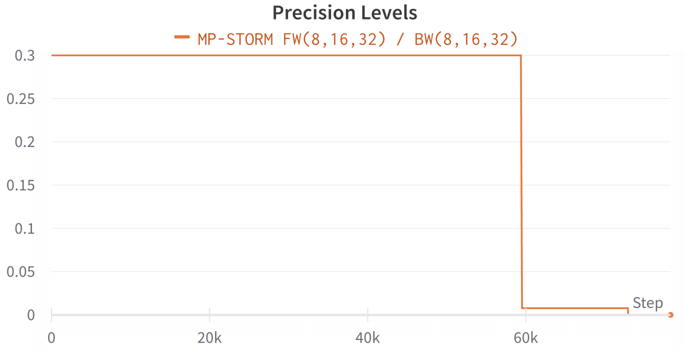
Initial Results: Test Errors



Initial Results: Loss



Initial Results: Precision



Limitations

- Convergence theory unclear (yet).
- Requires 2 evaluation of the loss/iteration.
- Mixed-precision does not seem to have an effect.
- TR methods are cumbersome

Limitations

- Convergence theory unclear (yet).
- Requires 2 evaluation of the loss/iteration.
- Mixed-precision does not seem to have an effect.
- TR methods are cumbersome
  $\rightarrow$ **Could we extend simpler (first-order) methods instead?**

Limitations

- Convergence theory unclear (yet).
- Requires 2 evaluation of the loss/iteration.
- Mixed-precision does not seem to have an effect.
- TR methods are cumbersome
  $\rightarrow$ **Could we extend simpler (first-order) methods instead?**
  $\rightarrow$ **Idea:** extend SGD to a mixed-precision setting

Mixed-Precision training of DNNs: Summary & Currrent Work

- SOTA methods usually use heuristics or focus on quantization aspects.
- We extended a classical optimization algorithms:
  - A trust-region method allowing us to jointly tune the step-size and the precision.

Current work:

- Running experiments:
  - Trust-Region with momentum
  - Trust-Region with weight-decay
  - Trust-Region with momentum and weight-decay
- Convergence of Stochastic Trust-Region
- Calculation error of a multi-layer perceptron
- Stochastic Gradient Descent in mixed-precision

Thank you! Questions?

Backup

Mixed-Precision Stochastic Gradient Descent: reminders 1

We extend SGD convergence proof [Ghadimi et al., 2013] to a mixed-precision problem

Definition

- We want to solve:

$$f^* = \inf_{x \in \mathbb{R}^n} f(x)$$

  With f potentially non-convex

- We only have access to $\hat{f}$ such that:

$$|\hat{f}(x_k, \epsilon_k) - f(x_k)| \leq \epsilon_k$$

- We only have access to a *stochastic gradient* $G(x_k, \xi_k)$ where ξ_k is a random variable in $\mathbb{R}^d$

Mixed-Precision Stochastic Gradient Descent: reminders 2

Assumption

- *f is L-smooth:*

$$\|\nabla f(x) - \nabla f(y)\| \leqslant L\|x - y\|.$$

- *$G(x_k, \xi_k)$ is an unbiased estimator of $\nabla f(x_k)$ with bounded variance:*

$$\mathbb{E}[G(x_k, \xi_k)] = \nabla f(x_k)$$
$$\mathbb{E}[\|G(x_k, \xi_k) - \nabla f(x_k)\|] \leqslant \sigma^2$$

- *The precision ϵ_k, the step-size γ_k and the probability mass function $P_R(k)$ are such that $\gamma_k \leq \frac{2(1-\eta)}{L}$ and:*

$$\textit{Prob}\{R = k\} = \frac{2(1-\eta)\gamma_k - L\gamma_k^2}{\sum_{k=1}^{N} 2(1-\eta)\gamma_k - L\gamma_k^2} \ \& \ \epsilon_k \leqslant \gamma_k \frac{\eta}{2}\|G(x_k, \xi_k)\|^2.$$

Mixed-Precision Stochastic Gradient Descent (ongoing work)

Theorem (Informal)
Under previous assumptions we have:

$$\mathbb{E}[\|\nabla f(x_R)\|^2] \leqslant \frac{\hat{f}(x_1, \epsilon_1) - \hat{f}^* - \epsilon_1 + L\sigma^2 \sum_{k=1}^N \gamma_k^2}{\sum_{k=1}^N 2(1-\eta)\gamma_k - L\gamma_k^2} \leqslant O(\frac{1}{\sqrt{N}}),$$

where the expectation is taken with respect to R *and* $\xi_{[N]} = (\xi_1, ..., \xi_N)$

MP-SGD convergence proof: a brief overview

Proof.
We extend [Ghadimi et al., 2013].
By definition of $\hat{f}(x_k)$, we have:

$$\hat{f}(x_{k+1}, \epsilon_{k+1}) - \hat{f}(x_k, \epsilon_k) \leq f(x_{k+1}) - f(x_k) + \epsilon_{k+1} + \epsilon_k$$

Using the assumptions on f & summing up:

$$\sum_{k=1}^{N}(\gamma_k - \frac{L}{2}\gamma_k^2)\|\nabla f(x_k)\|^2 \leq \hat{f}(x_1, \epsilon_1) - \hat{f}^* - \sum_{k=1}^{N}(\gamma_k - L\gamma_k^2)\langle \nabla f(x_k), \delta_k \rangle + \sum_{k=1}^{N} \frac{L}{2}\gamma_k^2 \|\delta_k\|^2 + \sum_{k=1}^{N} 2\epsilon_k - \epsilon_1$$

With $\delta_k = G(x_k, \xi_k) - \nabla f(x_k)$. The problematic term is $\sum_{k=1}^{N} 2\epsilon_k$.
Using the constraints on ϵ_k and $P_R(k)$ we find the final result $\square$

Challenges

- The constraint is very restrictive and does not allow many iterations in low-precision $\rightarrow$ Is there a way to relax this? (Potentially decreasing SGD convergence rate?)
- We now need to tune the learning-rate $\rightarrow$ Could we extend some adaptive method (Adagrad, ADAM...)?

Neural Network Error Analysis

What are we looking for?

- We consider the computation of

$$h_\ell(x) = \phi_\ell(W_\ell h_{\ell-1}(x)) \in \mathbb{R}^{n_\ell}$$

  with $h_0(x) = x \in \mathbb{R}^{n_0}$, where $\phi_\ell : \mathbb{R}^{n_\ell} \mapsto \mathbb{R}^{n_\ell}$ is an activation function and $W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ is a matrix of weights.
- **Goal:** We are trying to model the error propagation through the network layers inside the forward path. Where ϵ_ℓ is the error committed at layer ℓ

Error analysis of DNNs: a theoretical result

Theorem (Informal)

$$\varepsilon_\ell^h = \kappa_{\phi_\ell}(v_\ell) \circ \kappa_{v_\ell} \circ (\varepsilon_\ell^W + \overline{\varepsilon}_{\ell-1}^h) + \varepsilon_\ell^\phi,$$

With

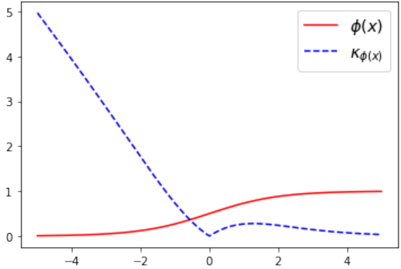
- ε_ℓ^h, ε_ℓ^W, ε_ℓ^ϕ, $\overline{\varepsilon}_{\ell-1}^h$, *various errors committed at layer* ℓ *and* $\ell - 1$
- κ_{ϕ_ℓ}, *condition number of activation function*
- κ_{v_ℓ}, *condition number of the weights/inputs vector product*

- The errors are amplified by the condition numbers of each layers.

Error analysis of DNNs: the intuition behind

⋆ PE Need some help with the def of ϕ condition number.

◇ ER Use tanh rather than sigmoid



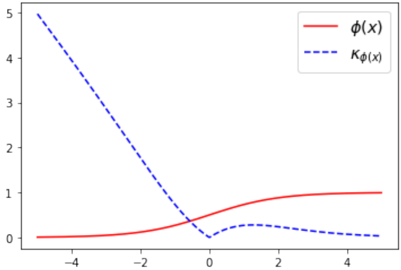
Sigmoid activation function $\phi(x)$ and its condition number $\kappa_{\phi(x)}$

Idea: When $\|x\|$ big, the output error is small. When $\|x\|$ is small the output error is big.

Error analysis of DNNs: the intuition behind

⋆ PE Need some help with the def of ϕ condition number.

◇ ER Use tanh rather than sigmoid



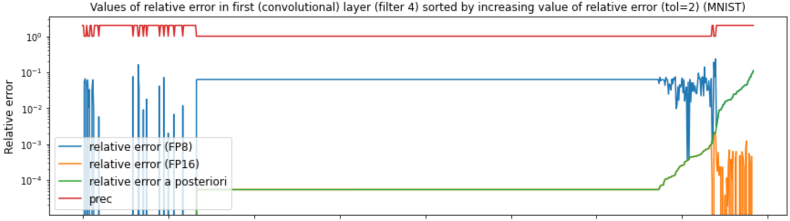
Sigmoid activation function $\phi(x)$ and its condition number $\kappa_{\phi(x)}$

Idea: When $\|x\|$ big, the output error is small. When $\|x\|$ is small the output error is big.

Our approach exploits this activation functions property to apply mixed-precision.

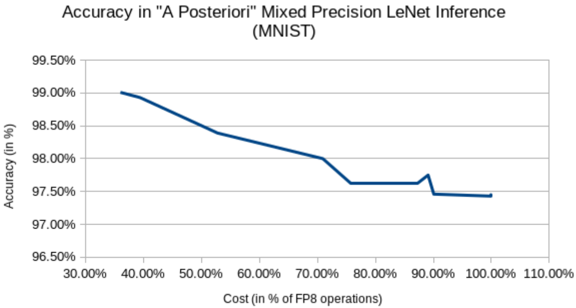
Error analysis of DNNs: Experimental validation 1

We select the precision at every dot product of every matrix multiplication based on a thresholding of the condition-numbers. write the rule



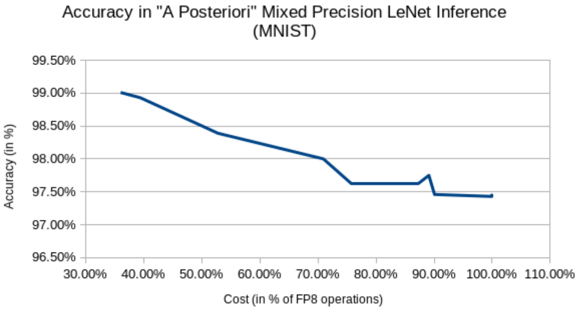
Error analysis of DNNs: Experimental validation 2

We vary the threshold (i.e. the ratio of FP8 operations) to observe its impact on the overall accuracy of a simple neural network (LeNet-5) for a common task (MNIST)



Error analysis of DNNs: Experimental validation 2

We vary the threshold (i.e. the ratio of FP8 operations) to observe its impact on the overall accuracy of a simple neural network (LeNet-5) for a common task (MNIST)



Future work: could we use this to accelerate DNNs inference?

Error analysis of DNNs: some intuition

Theory (intuition)

$$\begin{aligned}\hat{h}_\ell(x) &= \phi_\ell(v_\ell) \circ (\mathbf{1} + \kappa_{\phi_\ell}(v_\ell) \circ \delta v_\ell) \circ (\mathbf{1} + \Delta\phi_\ell) \\ &= h_\ell(x) \circ (\mathbf{1} + \underbrace{\kappa_{\phi_\ell}(v_\ell) \circ \delta v_\ell + \Delta\phi_\ell + \kappa_{\phi_\ell}(v_\ell) \circ \delta v_\ell \circ \Delta\phi_\ell}_{\Delta h_\ell})\end{aligned}$$

With δv_ℓ

STORM: Precision-switch policy

$$m(W_k, s) = f_k + g(W_k)^T s$$

To refine precision ϵ_k of $\hat{f}$:

select

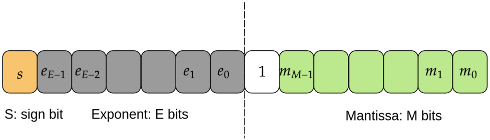
$$\epsilon_k^+ \in (0, \eta[m(W_k, 0) - m(W_k, s_k)] \text{ with } \eta > 0$$

i.e. for a linear model with $s_k = -\Delta_k \frac{g(W_k)}{\|g(W_k)\|_2}$

$$\epsilon_k^+ \in (0, -\eta\Delta_k \|g(W_k)\|_2]$$

Floating-Point 101

- Floating-point formats offer various trade-offs in terms of range, precision & performance

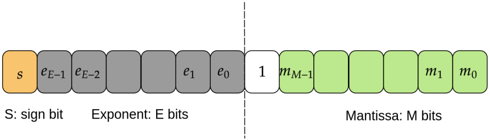


$$x = (-1)^s \times 1.m \times 2^{e-\text{BIAS}}$$

Format	Mantissa size	Exponent size	Bias	Range	Unit Roundoff	TFLOPS on H100	IEEE-754
fp64	52	11	1023	$10^{\pm 308}$	1×10^{-16}	48	Yes
fp32	23	8	127	$10^{\pm 38}$	1×10^{-8}	48	Yes
fp16	10	5	15	$10^{\pm 5}$	5×10^{-4}	400	Yes
tfloat32 (tf32)	10	8	127	$10^{\pm 38}$	5×10^{-4}	800	No
bfloat16 (bf16)	7	8	127	$10^{\pm 38}$	4×10^{-3}	800	No
fp8	3	4	7	$10^{\pm 2}$	6×10^{-2}	1600	No
	2	5	15	$10^{\pm 5}$	1×10^{-1}		

Floating-Point 101

- Floating-point formats offer various trade-offs in terms of range, precision & performance



$$x = (-1)^s \times 1.m \times 2^{e-\text{BIAS}}$$

Format	Mantissa size	Exponent size	Bias	Range	Unit Roundoff	TFLOPS on H100	IEEE-754
fp64	52	11	1023	$10^{\pm 308}$	1×10^{-16}	48	Yes
fp32	23	8	127	$10^{\pm 38}$	1×10^{-8}	48	Yes
fp16	10	5	15	$10^{\pm 5}$	5×10^{-4}	400	Yes
tfloat32 (tf32)	10	8	127	$10^{\pm 38}$	5×10^{-4}	800	No
bfloat16 (bf16)	7	8	127	$10^{\pm 38}$	4×10^{-3}	800	No
fp8	3	4	7	$10^{\pm 2}$	6×10^{-2}	1600	No
	2	5	15	$10^{\pm 5}$	1×10^{-1}		

- Default format for DNN applications is fp32