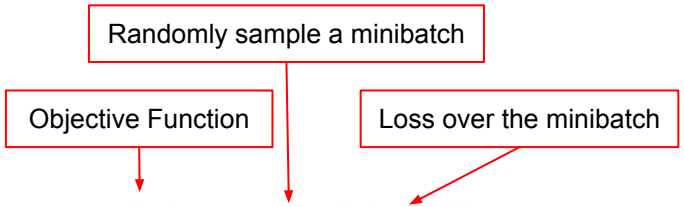
AdaScale SGD: A User-Friendly Algorithm for Distributed Training

Tyler B. Johnson [† 1] **Pulkit Agrawal** [† 1] **Haijie Gu** [1] **Carlos Guestrin** [1]

Published in ICML 2020

Main Idea: A new, dynamic approach for setting the learning rate for large-batch training. AdaScale SGD uses the gradient's variance to adapt the learning rate properly to any batch size, while preserving model quality.

Problem Formulation



- What are we optimizing? $\text{minimize}_{\mathbf{w}\in\mathbb{R}^d} F(\mathbf{w}) := \mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[f(\mathbf{w},\mathbf{x})\right]$
  - F, f are differentiable. Mini-batch gradient is unbiased ($\mathbb{E}_{\mathbf{x}\sim\mathcal{X}}\left[\nabla_{\mathbf{w}} f(\mathbf{w},\mathbf{x})\right] = \nabla F(\mathbf{w})$
- How do we optimize this objective?
  - **SGD:** the canonical algorithm for solving such a problem
    - $\mathbf{g}_t \leftarrow \nabla_{\mathbf{w}} f(\mathbf{w}_t, \mathbf{x}_t)$
    - $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \mathbf{g}_t$
    - $\eta_t = \texttt{lr}(t)$: learning rate schedule, determines step size at each iteration
      - *Different/arbitrary LR schedules can be plugged in to AdaScaleSGD!*
  - **Scaled SGD:** applies multiple batches in parallel (i.e., increasing the batch size)
    - Apply S batches per iteration, update with mean of batch gradients
    - **Issue:** must adapt the learning rate to handle larger batches

Why do we need large batches?

- Training models is expensive. Speeding this up allows us to use more data and bigger models, which expands the capabilities of ML.
- **Data-parallel training:** easy to implement (see torch.nn.DataParallel), but works best if you have large batches.
  - Communication is an issue because serialization/synchronization is required [2]
  - Each GPU must be utilized as much as possible to amortize communication cost
  - Large batches distribute more data to each GPU
- **Properties of Large Batches:**
  - Variance of the gradient is lower (i.e., this is a simple way to achieve variance reduction)
  - Convergence can occur in fewer iterations, but only if we increase the step size!
  - **Goal:** better wall-clock time to convergence with equal or better performance

$$Var(X) = E[X^2] - E[X]^2 = \int_{\mathbf{R}}(x-\mu)^2 f(x)dx = \sum_{i=1}^{N} \frac{(x_i - \overline{x})}{N-1}$$

What did we do before AdaScale SGD?

- **Previous SOTA:** linear scaling rule [3] (or extensive hyperparameter tuning)
  - Assume initial batch size N and learning rate ɲ.
  - For a (larger) batch size of M*N, the learning rate becomes M*ɲ.
  - **Pros:**
    - Initial results are promising. This method clearly works in some cases.
    - Implementation is very simple.
  - **Cons:**
    - Succeeds in limited cases - poor performance/divergence at greater scale
    - This is a “fixed scaling rule” that does not adapt during training

AdaScale SGD

- How do we define variance of the gradient?

$$\boxed{\begin{aligned}\boldsymbol{\Sigma}_{\mathbf{g}}(\mathbf{w}) &= \mathrm{cov}_{\mathbf{x}\sim\mathcal{X}}(\nabla_{\mathbf{w}} f(\mathbf{w},\mathbf{x}), \nabla_{\mathbf{w}} f(\mathbf{w},\mathbf{x}))\,, \\ \text{and} \quad \sigma^2_{\mathbf{g}}(\mathbf{w}) &= \mathrm{tr}(\boldsymbol{\Sigma}_{\mathbf{g}}(\mathbf{w}))\,.\end{aligned}}$$

  - Finds the covariance of the gradient
    - Gradient is a vector, so covariance yields a matrix where every entry (i, j) represents the covariance of vector positions i and j within the gradient vector.
  - Sum diagonal entries of the covariance matrix (i.e., the matrix trace)
  - **Result:** a sum of the variance computed at each vector position in the gradient
    - This variance is computed across different minibatch samples
  - *Data parallelism will reduce this variance*

AdaScale SGD

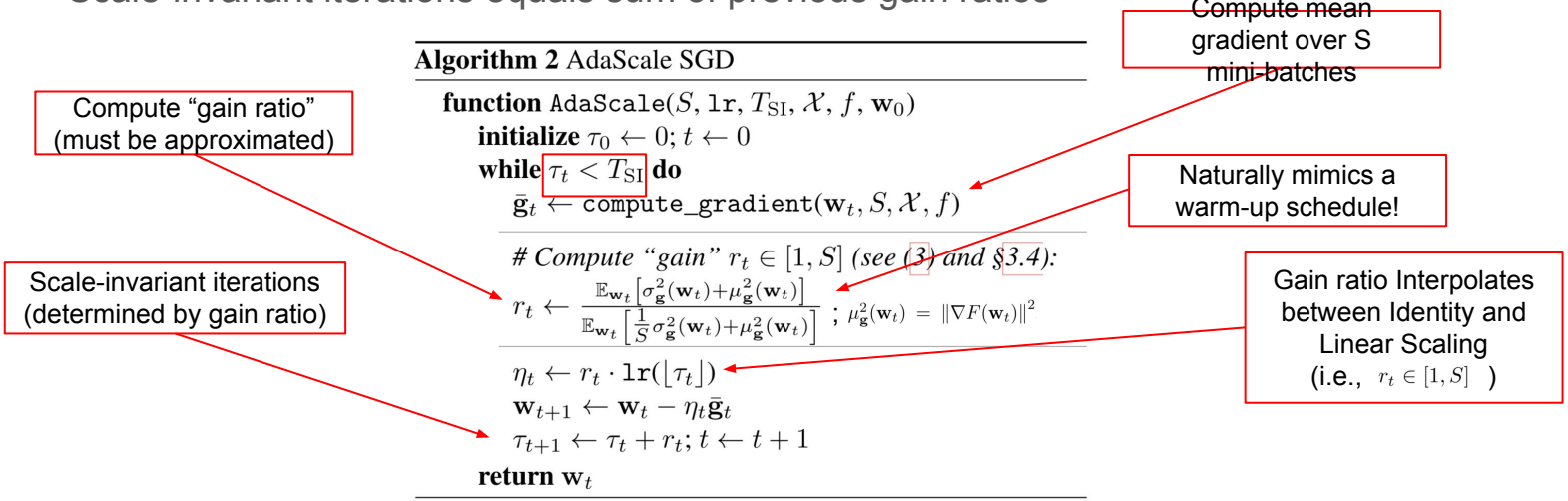
- Consider two simple scaling rules of the form $(S, lr_1, T_1) \rightarrow (lr_S, T_S)$
  - Where lr is learning rate and T is total iterations performed
- **Identity:** $lr_S = lr_1$, $T_S = T_1$ (i.e., keep hyperparameters the same)
  - This rule is optimal when mini-batch gradients have zero variance
  - If variance is small, we don't get significant benefits from scaling training to large batches
    - Reducing variance has no impact on the aggregated gradient
  - This rule does not reduce the number of iterations
- **Linear:** $lr_S = S*lr_1$, $T_S = \lceil T_1/S \rceil$(i.e., same rule from [3])
  - Treats SGD as perfectly parallelizable (i.e., performing S batch updates simultaneously is the same as doing in sequence)
  - Becomes more optimal (i.e., linear speedup) as gradient variance approaches infinity
  - Increasing S leads to a reduction in gradient variance, which can lead to a speed up

AdaScale SGD

- **Problem:** variance of gradient will not be zero or infinity in practice, and the variance is changing throughout training!
  - How do we choose which rule to use?
  - Can we interpolate between them dynamically based on the gradient's variance?
    - We want to get something between linear and identity scaling!
- **Proposed Solution:** AdaScale SGD

AdaScale SGD

- Dynamically interpolates between linear and identity scaling based on the expected gradient variance.
  - A batch size of S*<base_batch_size> is used at every iteration during training
  - Learning rate is multiplied by “gain ratio” at each iteration
    - *Idea:* represents the number of single batch iterations within each update
  - Scale-invariant iterations equals sum of previous gain ratios



What is the gain ratio?

- **Controls the interpolation between identity and linear scaling**
- Approximate gain ratio with per-batch and aggregated gradients that are readily available during training!

$$r_t \leftarrow \frac{\mathbb{E}_{\mathbf{w}_t}\left[\sigma^2_{\mathbf{g}}(\mathbf{w}_t)+\mu^2_{\mathbf{g}}(\mathbf{w}_t)\right]}{\mathbb{E}_{\mathbf{w}_t}\left[\frac{1}{S}\sigma^2_{\mathbf{g}}(\mathbf{w}_t)+\mu^2_{\mathbf{g}}(\mathbf{w}_t)\right]}$$

$$\eta_t \leftarrow r_t \cdot \texttt{lr}(\lfloor \tau_t \rfloor)$$

Approximate

$$r_t = \frac{\mathbb{E}[\frac{1}{S}\sum_{i=1}^{S}\|\mathbf{g}_t^{(i)}\|^2]}{\mathbb{E}[\|\bar{\mathbf{g}}_t\|^2]}$$

  - Controls the number of scale-invariant iterations that have occured
  - Scales the learning rate based on this number (naturally mimics warm-up)
  - *Allows linear scaling to be dampened when gradient variance is not too high*
- **Must be approximated** (i.e., true form uses the full gradient)
  - Approximate it with gradient expectation computed over all current/prior batches
  - This is computed using an **exponentially decaying average** of the two terms within the gain ratio expression: $\frac{1}{S}\sum_{i=1}^{S}\|\mathbf{g}_t^{(i)}\|^2$ $\|\bar{\mathbf{g}}_t\|^2$
  - AdaScale is relatively robust to setting the weight of this exponential average

Anything else?

- Keep momentum constant at 0.9 throughout training
  - Adapting momentum could be future work
  - It is a less-critical hyperparameter to tune for momentum-SGD according to the authors
- Different learning rate schedules can be used
  - AdaScale gives you warm-up and appropriate scaling to large batches
  - Scaling is applied to the learning rates that come from the provided schedule

Experiments

- **AS:** AdaScale SGD
- **LSW:** Linear Scaling w/ Warmup
- **LSW+:** LR schedule of LSW + Iterations of AS

Table 2: Comparison of final model quality. *Shorthand:* AS=AdaScale, LSW=Linear scaling with warm-up, LSW+=Linear scaling with warm-up and additional steps, gray=model quality significantly worse than for $S = 1$ (5 trials, 0.95 significance), N/A=training diverges, Elastic↑/↓=elastic scaling with increasing/decreasing scale (see Figure 4). Linear scaling leads to poor model quality as the scale increases; AdaScale preserves model performance for nearly all cases.

Task	S	Total batch size	Validation metric			Training loss			Total iterations		
			AS	LSW	LSW+	AS	LSW	LSW+	AS	LSW	LSW+
cifar10	1	128	94.1	94.1	94.1	0.157	0.157	0.157	39.1k	39.1k	39.1k
	8	1.02k	94.1	94.0	94.0	0.153	0.161	0.145	5.85k	4.88k	5.85k
	16	2.05k	94.1	93.6	94.1	0.150	0.163	0.136	3.36k	2.44k	3.36k
	32	4.10k	94.1	92.8	94.0	0.145	0.177	0.128	2.08k	1.22k	2.08k
	64	8.19k	93.9	76.6	93.0	0.140	0.272	0.140	1.41k	611	1.41k
imagenet	1	256	76.4	76.4	76.4	1.30	1.30	1.30	451k	451k	451k
	16	4.10k	76.5	76.3	76.5	1.26	1.31	1.27	33.2k	28.2k	33.2k
	32	8.19k	76.6	76.1	76.4	1.23	1.33	1.24	18.7k	14.1k	18.7k
	64	16.4k	76.5	75.6	76.5	1.19	1.35	1.20	11.2k	7.04k	11.2k
	128	32.8k	76.5	73.3	75.5	1.14	1.51	1.14	7.29k	3.52k	7.29k
	Elastic↑	various	76.6	75.7	–	1.15	1.37	–	11.6k	7.04k	–
	Elastic↓	various	76.6	74.1	–	1.23	1.45	–	13.6k	9.68k	–
speech	1	32	79.6	79.6	79.6	2.03	2.03	2.03	84.8k	84.8k	84.8k
	4	128	81.0	80.9	81.0	5.21	4.66	4.22	22.5k	21.2k	22.5k
	8	256	80.7	80.2	80.7	6.74	6.81	6.61	12.1k	10.6k	12.1k
	16	512	80.6	N/A	N/A	7.33	N/A	N/A	6.95k	5.30k	6.95k
	32	1.02k	80.3	N/A	N/A	8.43	N/A	N/A	4.29k	2.65k	4.29k
transformer	1	2.05k	27.2	27.2	27.2	1.60	1.60	1.60	1.55M	1.55M	1.55M
	16	32.8k	27.4	27.3	27.4	1.60	1.60	1.59	108k	99.0k	108k
	32	65.5k	27.3	27.0	27.3	1.59	1.61	1.59	58.9k	49.5k	58.9k
	64	131k	27.6	26.7	27.1	1.59	1.63	1.60	33.9k	24.8k	33.9k
	128	262k	27.4	N/A	N/A	1.59	N/A	N/A	21.4k	12.1k	21.4k
yolo	1	16	80.2	80.2	80.2	2.65	2.65	2.65	207k	207k	207k
	16	256	81.5	81.4	81.9	2.63	2.66	2.47	15.9k	12.9k	15.9k
	32	512	81.3	80.5	81.7	2.61	2.81	2.42	9.27k	6.47k	9.27k
	64	1.02k	81.3	70.1	80.6	2.60	4.02	2.51	5.75k	3.23k	5.75k
	128	2.05k	81.4	N/A	N/A	2.57	N/A	N/A	4.07k	1.62k	4.07k

Improvements over LSW:

- Model converges with larger batch sizes
- As S increases, AdaScale preserves model quality with fewer iterations
- Performs well in a variety of domains (i.e., it adapts to the training scenario)
- Robust to sudden changes in batch size (elastic scaling)
- Nearly no failure cases in the presented results

Citations

[1] Johnson, Tyler B., et al. "AdaScale SGD: A User-Friendly Algorithm for Distributed Training." arXiv preprint arXiv:2007.05105 (2020).

[2] Assran, Mahmoud, et al. "Advances in Asynchronous Parallel and Distributed Optimization." arXiv preprint arXiv:2006.13838(2020).

[3] Goyal, Priya, et al. "Accurate, large minibatch sgd: Training imagenet in 1 hour." arXiv preprint arXiv:1706.02677 (2017).