Contents

1 Large Batch Training of Convolutional Networks

1.1 Background

- Increasing the global batch while keeping the same number of epochs means that you have fewer iterations to update weights.
  - The straight-forward way to compensate for a smaller number of iterations is to do larger steps by increasing the learning rate (LR).
  - using a larger LR makes optimization more difficult, and networks may diverge especially during the initial phase.
  - “learning rate warm-up
- apply linear scaling and warm-up scheme to train Alexnet on Imagenet
- BUT scaling stopped after B=2K since training diverged for large LRs.

batch size	accuracy
base (256)	57.6%
4K	53.1%
8K	44.8%

- To fix this:
  - replace Local Response Normalization with Batch Normalization (BN) -> AlexNet-BN
  - BN improves model convergence for large LR
  - for B=8K the accuracy gap was decreased from 14% to 2.2%.

1.2 Motivation

- propose a way to analyze the training stability with large LRs: measure the ratio between the norm of the layer weights and norm of gradients update
- if this ratio is:
  - too high, the training may become unstable
  - too small, then weights don’t change fast enough
- This ratio varies a lot between different layers, which makes it necessary to use a separate LR for each layer.
- Propose LARS, two notable differences:
  1. uses a separate learning rate for each layer and not for each weight, which leads to better stability

2. the magnitude of the update is controlled with respect to the weight norm for better control of training speed.

- With LARS, Alexnet-BN and Resnet-50 trained with B=32K without accuracy loss.

1.3 LARS

Algorithm 1 SGD with LARS. Example with weight decay, momentum and polynomial LR decay.

Parameters: base LR γ_0, momentum m, weight decay β, LARS coefficient η, number of steps T
Init: $t = 0, v = 0$. Init weight w_0^l for each layer l
while $t < T$ for each layer l **do**
  $g_t^l \leftarrow \nabla L(w_t^l)$ (obtain a stochastic gradient for the current mini-batch)
  $\gamma_t \leftarrow \gamma_0 * \left(1 - \frac{t}{T}\right)^2$ (compute the global learning rate)
  $\lambda^l \leftarrow \frac{||w_t^l||}{||g_t^l|| + \beta ||w_t^l||}$ (compute the local LR λ^l)
  $v_{t+1}^l \leftarrow m v_t^l + \gamma_{t+1} * \lambda^l * (g_t^l + \beta w_t^l)$ (update the momentum)
  $w_{t+1}^l \leftarrow w_t^l - v_{t+1}^l$ (update the weights)
end while

The LARS algorithm.

1.4 Some key points

- LR warm-up: training starts with small LR, and then LR is gradually increased to the target.
- BN makes it possible to use larger learning rates.
- When λ is large, the update $\|\lambda * \nabla L(\omega_t)\|$can become larger than ω, and this can cause the divergence. This makes the initial phase of training highly sensitive to the weight initialization and to initial LR.
- The paper found that the ratio the L2-norm of weights and gradients $\|\omega\| / \|\nabla L(\omega)\|$ varies significantly between weights and biases, and between different layers.
- The ratio is high during the initial phase, and it is rapidly decrease after few epochs.

Table 2: AlexNet-BN: The norm of weights and gradients at 1st iteration.

Layer	conv1.b	conv1.w	conv2.b	conv2.w	conv3.b	conv3.w	conv4.b	conv4.w
$\|\|w\|\|$	1.86	0.098	5.546	0.16	9.40	0.196	8.15	0.196
$\|\|\nabla L(w)\|\|$	0.22	0.017	0.165	0.002	0.135	0.0015	0.109	0.0013
$\frac{\|\|w\|\|}{\|\|\nabla L(w)\|\|}$	8.48	**5.76**	33.6	83.5	69.9	127	74.6	148
Layer	conv5.b	conv5.w	fc6.b	fc6.w	fc7.b	fc7.w	fc8.b	fc8.w
$\|\|w\|\|$	6.65	0.16	30.7	6.4	20.5	6.4	20.2	0.316
$\|\|\nabla L(w)\|\|$	0.09	0.0002	0.26	0.005	0.30	0.013	0.22	0.016
$\frac{\|\|w\|\|}{\|\|\nabla L(w)\|\|}$	73.6	69	117	**1345**	68	489	93	19