Adaptive Distributed Training of Deep Learning Models

Luo Mai, Guo Li, Andrei-Octavian Brabete, Alexandros Koliousis, Peter Pietzuch
Imperial College London
lsds@imperial.ac.uk

1 Introduction

Deep Learning (DL) has revolutionised many application fields [3, 10, 22] but the training of DL models is expensive: it can take half a month [10] to several months [7] to achieve state-of-the-art accuracy in real-world AI challenges [22, 23] on a single GPU. To reduce the time to reach a desired training accuracy (time-to-accuracy), systems often adopt data parallelism [6]: multiple GPUs train the replicas of a model on data partitions, and synchronise replicas for collective progress in training.

Users expect time-to-accuracy to decrease with more GPUs. This is challenging because it requires users to configure a distributed DL system correctly. Training must utilise all devices while the replicas converge to the desired accuracy. In a given training scenario [2, 14], users typically decide how to synchronise replicas by evaluating a wide range of choices [2, 4, 11, 14, 16, 27], exhibiting trade-offs between device utilisation and convergence: e.g. synchronous SGD (S-SGD) [4] improves convergence but may incur network bottlenecks; while asynchronous model averaging [16] keeps device utilisation high but may hurt convergence. Furthermore users must decide on the hyper-parameters of distributed replicas. This is non-trivial because modern DL systems expose many hyper-parameters [17], including the learning rate, batch size and momentum, and each of them affects how replicas converge [9].

Existing distributed DL systems [1, 5, 21] provide little support to users when deciding on synchronisation strategies and hyper-parameters: (i) they usually come with specific implementations of S-SGD, such as parameter servers [12] or all-reduce systems [24]. In training scenarios in which S-SGD is not effective, e.g. when training with stragglers [16], Byzantine nodes [2] or small batch sizes [14], users must implement custom solutions to synchronise replicas, making the process of deciding on synchronisation strategies cumbersome [13] and difficult to assess [27]; (ii) existing systems only support static hyper-parameters derived from empirical experience when training specific models [10]. When the level of parallelism is changed or the model is modified, users must manually adjust hyper-parameters in a trial-and-error fashion [9], resulting in wasted time and resources [8, 18].

We argue that, to help users make effective decisions when scaling training, a distributed DL system must be designed with *adaptiveness* in mind. Similar to self-driving databases [19], which automate the setting of critical performance parameters, such a system should help users (i) configure flexibly how replicas synchronise with another; and (ii) declare *dynamic* hyper-parameters that automatically adapt according to monitored metrics, which reflect the condition of replicas and the utilisation of devices.

```
1  import training as tr
2  import monitoring as mon
3  import communication as comm
4  import control as ctrl
5
6  def adapt_batch_size(ctrl, noise):
7    n_peers = tr.exp_decay(noise, 0.01) / GPU_BATCH_SIZE
8    ctrl.barrier()
9    ctrl.rescale(n_peers)
10   ctrl.barrier()
11
12 def build_driver_program(sample, loss):
13   grads = tr.resnet(sample, loss).auto_diff()
14   avg_grads = comm.all_reduce(grads)
15   optimiser = tr.optimiser(avg_grads)
16   noise = mon.noise(grads, avg_grads)
17   avg_noise = comm.all_reduce(noise)
18   c = ctrl.control(optimiser, avg_noise)
19   c.hook(adapt_batch_size, avg_noise)
```

Listing 1. A dynamic hyper-parameter policy

2 Adaptive Training with KungFu

We have developed KungFu, a system that supports adaptive training of DL models. We describe the design and benefits of KungFu through a use case: a user wants to train a DL model while adapting the number of parallel devices based on the *gradient noise scale* (GNS) [18]. GNS is a statistical measure for the signal-to-noise ratio of gradients. Intuitively, when the ratio is small, using large batches of training data is counter-productive; when the ratio is large, the model is more resilient to large-batch training.

Adaptation primitives. We want to define a high-level abstraction for specifying the adaptation logic of dynamic hyper-parameters. KungFu expresses the adaptation logic as *primitives* declared as part of a driver program, thus requiring only low effort for implementing monitoring and control. Such an abstraction also supports computation over monitored metrics and the synchronisation of changes to hyper-parameters in a distributed environment.

Listing 1 shows how KungFu expresses adaptation: (i) a driver program includes *monitoring operators* that attach to the training operators (e.g. in line 15, the gradient and

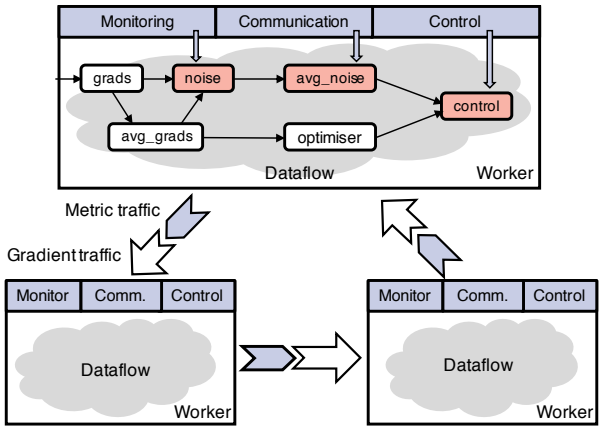

Figure 1. KungFu system overview.

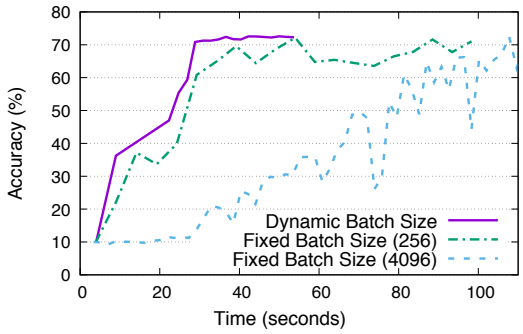

Figure 2. Time-to-accuracy of adaptive and static batch size policies in KungFu (B denotes batch size.)

average-gradient operators attach to a monitoring operator for computing the GNS); (ii) *collective communication operators* compute global metrics (in line 16, an all-reduce operator computes the average noise scale); and (iii) *control operators* continuously evaluate monitored metrics and synchronise modification to hyper-parameters (lines 17–19 register an adaptation function called at each iteration of the training). During control, we transform the noise scale to the number of peers that best utilise the GPUs (line 7). Lines 8–10 create a barrier to re-scale the cluster.

Embedding monitoring and control. We want to design a DL system that can efficiently realise the above abstraction. Today training metrics are often collected by external monitoring systems such as TensorBoard [25] or Prometheus [20]. Since such systems collect logs offline, the metrics become only available after training, and users must derive static hyper-parameter policies from them.

To enable efficient online monitoring and control, our key idea is to embed the execution of monitoring and control primitives within the training infrastructure. Figure 1 shows this idea: each worker has a DL library that performs training using dataflows. It creates a training dataflow augmented with monitoring, communication and control operators, as defined in Listing 1. These operators are executed asynchronously and reuse the result produced by the training operators (i.e. grads and avg_grads), without having to interrupt the training and copy the data. In addition, the communication operators use a networking layer that piggybacks monitoring data with synchronised gradients. This not only improves networking performance, but also reuses existing scale-out mechanisms [13] in both training and monitoring. Finally, the replicated control operators can be treated as an execution barriers. This provides clear semantics to users regarding *when* and *where* the changes to hyper-parameters take place in a distributed setting.

KungFu implementation. We have implemented the above design as part of a distributed training library for TensorFlow [1] called KungFu. The computation of GNS and the associated batch size controller are implemented as TensorFlow operators. To modify batch sizes dynamically, KungFu fixes the batch size per dataflow, guaranteeing high GPU utilisation, and implements a decentralised runtime that allows TensorFlow nodes to join and leave with low overhead. The KungFu runtime also features an efficient collective communication layer so that the distributed measurement of metrics does not lead to bottlenecks.

Experimental results. We implement the sample policy in Listing 1 using KungFu. We evaluate the effectiveness of this policy when training the ResNet-32 model [10] with the CIFAR-10 dataset [15], as implemented by the TensorFlow benchmarks [26]. The experiment runs on a 20-CPU-core server with 4 NVIDIA Titan X GPUs. We compare our *dynamic policy* with two *static policies* that use small and large batch sizes, 256 and 4096, respectively. We fix the learning rate to 0.1 and compare the time to reach the shared maximum test accuracy (in our setting, 72%).

Figure 2 shows the time-to-accuracy results. The small-batch policy benefits from a high statistical efficiency and thus converges 50% faster than the large-batch policy, even though its GPU utilisation is lower; the adaptive policy starts with a small batch size of 128 and gradually reaches 4096 after 4 epochs, achieving the benefits of both small and large-batch training. Hence, it exhibits the best time-to-accuracy, which is 32% faster than the static small-batch policy, demonstrating the low performance overhead of the KungFu approach.

We also evaluate the robustness of convergence for these three policies, and let them train for 9 extra epochs after reaching the target accuracy. The adaptive policy converges to a more stable accuracy compared to the others. This implies that the use of online monitoring metrics helps select hyper-parameters that can best fit in the loss space, thus improving the quality of minima.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.

[2] Dan Alistarh, Zeyuan Allen-Zhu, and Jerry Li. Byzantine Stochastic Gradient Descent. In *32nd International Conference on Neural Information Processing Systems (NIPS)*, 2018.

[3] Sercan Ömer Arik, Mike Chrzanowski, Adam Coates, Greg Diamos, Andrew Gibiansky, Yongguo Kang, Xian Li, John Miller, Jonathan Raiman, Shubho Sengupta, and Mohammad Shoeybi. Deep Voice: Real-time Neural Text-to-Speech. arXiv:1702.07825 [cs.CL], 2017.

[4] Jianmin Chen, Rajat Monga, Samy Bengio, and Rafal Józefowicz. Revisiting Distributed Synchronous SGD. arXiv:1604.00981 [cs.LG], 2016.

[5] Tianqi Chen, Mu Li, Yutian Li, Min Lin, Naiyan Wang, Minjie Wang, Tianjun Xiao, Bing Xu, Chiyuan Zhang, and Zheng Zhang. MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems. arXiv:1512.01274 [cs.DC], 2015.

[6] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marcaurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng. Large Scale Distributed Deep Networks. In *25th International Conference on Neural Information Processing Systems (NIPS)*, 2012.

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL], 2018.

[8] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural Architecture Search: A survey. arXiv:1808.05377 [stat.ML], 2018.

[9] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. arXiv:1706.02677 [cs.CV], 2017.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[11] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A Gibson, Greg Ganger, and Eric P Xing. More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. In *26th International Conferences on Neural Information Processing Systems (NIPS)*, 2013.

[12] Yuzhen Huang, Tatiana Jin, Yidi Wu, Zhenkun Cai, Xiao Yan, Fan Yang, Jinfeng Li, Yuying Guo, and James Cheng. FlexPS: Flexible Parallelism Control in Parameter Server Architecture. *PVLDB*, 11(5):566–579, 2018.

[13] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, Tiegang Chen, Guangxiao Hu, Shaohuai Shi, and Xiaowen Chu. Highly Scalable Deep Learning Training System with Mixed-Precision: Training ImageNet in Four Minutes. arXiv:1807.11205 [cs.LG], 2018.

[14] Alexandros Koliousis, Pijika Watcharapichat, Matthias Weidlich, Luo Mai, Paolo Costa, and Peter Pietzuch. Crossbow: Scaling Deep Learning with Small Batch Sizes on multi-GPU Servers. *PVLDB*, 12(11):1399–1412, 2019.

[15] Alex Krizhevsky. Convolutional deep belief networks on cifar-10, 2010.

[16] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. Asynchronous Decentralized Parallel Stochastic Gradient Descent. In *35th International Conference on Machine Learning (ICML)*, 2018.

[17] Luo Mai, Alexandros Koliousis, Guo Li, Andrei-Octavian Brabete, and Peter Pietzuch. Taming Hyper-parameters in Deep Learning Systems. *SIGOPS Oper. Syst. Rev.*, 53(1):52–58, 2019.

[18] Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An Empirical Model of Large-Batch Training. arXiv:1812.06162 [cs.LG], 2018.

[19] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C Mowry, Matthew Perron, Ian Quah, et al. Self-Driving Database Management Systems. In *8th Biennial Conference on Innovative Data Systems Research (CIDR)*, 2017.

[20] The Prometheus Monitoring System and Time-Series Database. https://github.com/prometheus/prometheus, 2019.

[21] PyTorch. https://pytorch.org, 2019.

[22] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ Questions for Machine Comprehension of Text. arXiv:1606.05250 [cs.CL], 2016.

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[24] Alexander Sergeev and Mike Del Balso. Horovod: Fast and easy distributed deep learning in TensorFlow. arXiv:1802.05799 [cs.LG], 2018.

[25] TensorFlow's Visualization Toolkit. https://github.com/tensorflow/tensorboard, 2019.

[26] TensorFlow Benchmarks. https://github.com/tensorflow/benchmarks, 2019.

[27] Pijika Watcharapichat, Victoria Lopez Morales, Raul Castro Fernandez, and Peter Pietzuch. Ako: Decentralised Deep Learning with Partial Gradient Exchange. In *7th ACM Symposium on Cloud Computing (SoCC)*, 2016.