REVISITING DISTRIBUTED SYNCHRONOUS SGD

Jianmin Chen, Rajat Monga, Samy Bengio & Rafal Jozefowicz
Google Brain
Mountain View, CA, USA
{jmchen,rajatmonga,bengio,rafalj}@google.com

1 THE NEED FOR A LARGE SCALE DEEP LEARNING INFRASTRUCTURE

The recent success of deep learning approaches for domains like speech recognition (Hinton et al., 2012) and computer vision (Ioffe & Szegedy, 2015) stems from many algorithmic improvements but also from the fact that the size of available training data has grown significantly over the years, together with the computing power, in terms of both CPUs and GPUs.

While a single GPU often provides algorithmic simplicity and speed up to a given scale of data and model, there exist an operating point where a distributed implementation of training algorithms for deep architectures becomes necessary.

2 ASYNCHRONOUS STOCHASTIC GRADIENT DESCENT

In 2012, Dean et al. (2012) presented their approach for a distributed stochastic gradient descent algorithm. It consists of two main ingredients. First, the parameters of the model can be distributed on multiple servers, depending on the architecture. This set of servers are called the *parameter servers*. Second, there can be multiple workers processing data in parallel and communicating with the parameter servers. Each worker processes a mini-batch of data independently of the other ones, as follows:

- it fetches from the parameter servers the most up-to-date parameters of the model needed to process the current mini-batch;
- it then computes gradients of the loss with respect to these parameters;
- finally, these gradients are sent back to the parameter servers, which then updates the model accordingly.

Since each worker communicates with the parameter servers independently of the others, this is called *Asynchronous Stochastic Gradient Descent* (or Async-SGD). A similar approach was later proposed by Chilimbi et al. (2014).

In practice, it means that while a worker computes gradients of the loss with respect to its parameters on a given mini-batch, other workers also interact with the parameter servers and thus potentially update its parameters; hence when a worker sends back its gradients to the parameter server, these gradients are usually computed w.r.t. the parameters of an old version of the model. When a model is trained with N workers, each update will be $N-1$ steps old on average.

While this approach has been shown to scale very well up to a few dozens of workers for some models, experimental evidence shows that increasing the number of workers sometimes hurt the training of the model with the noise introduced by the discrepancy between the model used to compute gradients and the model actually updated.

3 REVISITING SYNCHRONOUS SGD AND ITS VARIANTS

Both Dean et al. (2012) and Chilimbi et al. (2014) use versions of Async-SGD where the main potential problem is that each worker computes gradients over a potentially old version of the model. In order to remove this discrepancy, we propose here to reconsider a synchronous version of distributed stochastic gradient descent (Sync-SGD), where the parameter servers wait for all workers to send their gradients, aggregate them, and send the updated parameters to all workers afterward, making sure that the actual algorithm is a true mini-batch stochastic gradient descent, where the actual batch size is the sum of all the mini-batch sizes of the workers.

While this approach solves the discrepency problem, it also introduces two potential problems: the effective size of the batch is now significantly larger, and the actual update time now depends on the slowest worker. In order to alleviate the latter, we introduce *backup workers* Dean & Barroso (2013) as follows: instead of having N workers, we use a few more, say 5% more, but as soon as the parameter servers receive gradients from N of the workers, they stop waiting and send back the updated parameters, while the slower workers' gradients will be dropped when they arrive.

We also experimented with with a *mixed* approach by grouping certain number of workers together, synchronize intra group and do asynchronous updates among different groups. This turned out to be worse than Sync-SGD so we are not providing more details.

4 IMAGENET EXPERIMENTS

We conducted experiments on the ImageNet Challenge dataset (Russakovsky et al., 2015), where the task is to classify images out of 1000 categories. We used the latest *Inception* model from Szegedy et al. (2016) and trained it in several conditions, including using from 50 to 200 workers, each of which runs on a k40 GPU, and using *asynchronous SGD*, *synchronous SGD* and *synchronous SGD with backups*. All the experiments in this paper are using the TensorFlow system Abadi et al. (2015).

Number of workers	Test Accuracy (%)	Time (hrs)
25	78.94	184.9
50	78.83	97.67
100	78.44	51.97
200	78.04	22.94

Table 1: Comparison of test accuracy and time to convergence using asynchronous SGD training with different numbers of workers. The test accuracy figure is also given in Figure 3

Table 1 shows the test accuracy when training using asynchronous SGD with different numbers of workers. As can be seen, when the number of workers is doubled, the time to convergence is almost halved, however, while the accuracy loss from 25 to 50 workers is only 0.11%, it is much worse after 50.

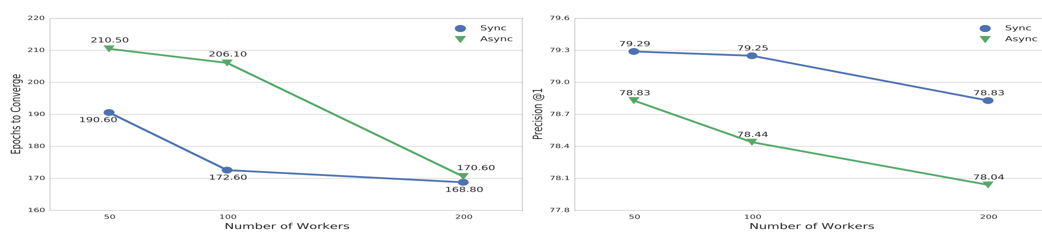

Figure 1: Comparison of test accuracy and number of epochs to converge for synchronous and asynchronous SGD.

Figure 1 shows the comparison of the test accuracy and epochs to converge between synchronous and asynchronous SGD with different numbers of workers. With synchronous SGD, all gradients are summed before updating the parameters: with N workers, it means effectively about N times bigger updates than with a single worker trained with SGD. Since we used RMSProp with momentum, the effective learning rate increase is less compared with single worker SGD as the parameters are only updated once instead of N times with asynchronous training.

Figure 1 shows that synchronous training can achieve about 0.5 to 0.9 percent higher accuracy, needs fewer epochs to converge, and scales better as there is only 0.04% accuracy loss from 50 to 100 workers.

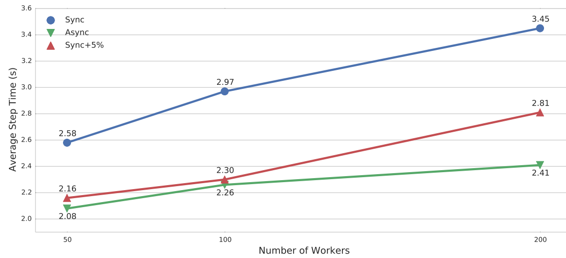

Figure 2: Comparison of the step time for synchronous, synchronous with backup and asynchronous training, for 50, 100 and 200 workers.

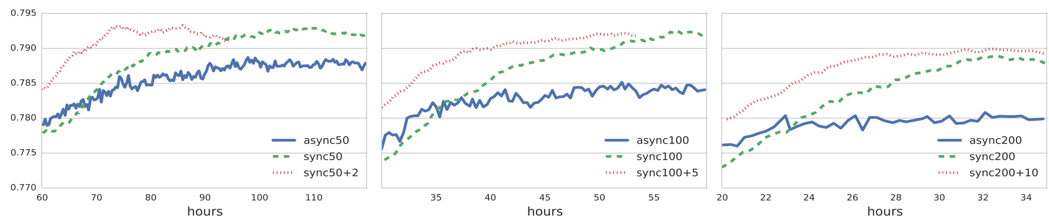

Figure 3: Test accuracies with respect to training time, for synchronous, synchronous with backup and asynchronous training for 50, 100 and 200 workers. Note that to make things clearer, we only show the accuracy range from 0.770 to 0.795

However, the most important concern about synchronous training is the overhead expected from the synchronization of the workers in a large scale distributed system:

- After computing the gradients, each worker needs to wait for all of workers to compute their gradients in order to update the parameters which are then fetched to process the next batch; This will increase the step time to process a batch while no such barrier is needed in the asynchronous setting.
- Since workers can run at different speed, the overall speed is governed by the slowest worker.
- The underlying hardware resources in the data centers are shared with other jobs so workers could be slowed or preemptied by other jobs besides broken hardware.

In the asynchronous training mode, gradient clipping is needed for stabilization, which requires each worker to collect all gradients, compute the global norm and then clip all gradients accordingly. However, synchronization turns out to be very stable so gradient clipping is no longer needed, which means that we can pipeline the update of parameters in different layers: applying gradients of upper layers can happen concurrently with computing the lower layers' gradients. Hence the real overhead is only the time to apply the bottom layer's gradients. Besides, the overhead of global clipping is also saved in synchronous training. Since each worker has exactly the same amount of computation and network traffic, the variation of step time is relatively low. However, there are usually a few workers that can be significantly slower or even dead due to network or hardware malfunctions. Backup workers are useful to remove such long tail events. Figure 2 shows the average step time of the 3 configurations: Sync is about 20-40% slower than async but with backup workers, sync is almost as fast as async with up to 100 workers.

Figure 3 shows the test accuracy over time of all 3 configurations with different numbers of workers. As shown in the figure, adding backup workers resulted in faster training and for 50 workers with 2 backups, sync can converge 25% faster than the async version with 0.48% better precision.

5 Other Experiments

We also conducted experiments on large scale character language models (LMs) trained on the One Billion Word Benchmark dataset (Chelba et al. (2013)), which is a popular benchmark for evaluating

LMs. The goal is to predict the next character of a sequence given a history of past words. In order to make it computationally efficient, we started with the best pre-trained model from Jozefowicz et al. (2016) that takes characters as inputs and assigns probabilities to the next words. Since we were interested in predicting characters, the last layer of the model was replaced with a smaller LSTM that tries to predict the next word one character at a time. The resulting architecture works over an unbounded vocabulary as it can consume any input and produce any output. More details about the experimental setting is available in Section 5.5 of Jozefowicz et al. (2016). Using 32 synchronous workers gave us a few percent improvement of the final performance: below 49 per-word perplexity. Training with Async-SGD was significantly less stable and required using much lower learning rate due to occasional explosions of the training loss. With synchronous SGD these issues disappear even when the learning rate was 100 times larger than in the best configuration.

Finally, we explored distributed training on the DRAW model (Gregor et al. (2015)). This is a very difficult optimization problem involving recurrent neural networks, attention mechanism and a very complicated loss function. The models were trained on MNIST dataset. We found that both synchronous and asynchronous training with 10 workers is significantly faster than using a single worker. Additionally, synchronous training allowed for using larger learning rates, which resulted in 40-50% faster convergence speed with the same amount of work.

6 Conclusion and Future Work

Distributed training strategies for deep learning architectures will become ever more important as the size of datasets increases. We have shown in this work that synchronous SGD with backup workers is a viable and scalable strategy. We are currently experimenting with different kinds of datasets, including word-level language models where parts of the model (the embedding layers) are often very sparse, which involves very different communication constraints. We are also working on further improving the performance of synchronous training like combining gradients from multiple workers sharing the same machine before sending them to the parameter servers to reduce the communication overhead.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`.

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation*, 2014.

J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. A. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems, NIPS*, 2012.

Jeffrey Dean and Luiz Andr Barroso. The tail at scale. *Communications of the ACM*, 56:74–80, 2013. URL `http://cacm.acm.org/magazines/2013/2/160173-the-tail-at-scale/fulltext`.

Karol Gregor, Ivo Danihelka, Alex Graves, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 29:82–97, 2012.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML*, 2015.

R. Jozefowicz, O. Vinyals, M. Schuster, N. Shazeer, and Y. Wu. Exploring the limits of language modeling. In *ArXiv 1602.02410*, 2016.

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. In *International Journal of Computer Vision*, 2015.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *ArXiv 1512.00567*, 2016.