In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2020)

Population-based Evolutionary Distributed SGD

Amna Shahab
University of Edinburgh

Boris Grot
University of Edinburgh

ABSTRACT

Neural model training is a time-consuming task where exploiting parallelism is of utmost importance. Employing data-parallelism in stochastic gradient descent (SGD) by partitioning the training dataset is a popular approach; however, algorithmic inefficiencies when operating at large minibatch sizes limit the degree of parallelism, a problem termed the *scalability limit*. In the face of this scalability challenge, we propose using Evolutionary Algorithms (EA) as a meta-algorithm together with SGD for training neural models. Our training scheme, Population-based Evolutionary SGD (PESGD) combines local SGD training on each training node with a periodic evolutionary step which selects the best performing models to generate a new population of models for the next iteration. We believe that the complementarity of SGD and EA for neural model training can be exploited well in PESGD.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; *Genetic algorithms*; *Distributed artificial intelligence*.

KEYWORDS

neural networks; machine learning; evolutionary algorithms

1 INTRODUCTION

Neural model training is a large-scale problem with immense commercial value. Minibatch SGD is the de facto standard for the training of neural models. Unfortunately, the staggering time to train the leading models hinders rapid testing and deployment. While modern GPUs have enabled faster training of models, the time to train large models on massive datasets remains intolerably high.

To reduce training times significantly, machine learning (ML) practitioners employ multiple machines to run data-parallel minibatch stochastic gradient descent (DP-SGD). Indeed, recent works report using up to 2K GPUs for distributed training [2]. At this scale, frequent communication of model parameter gradients requires a great deal of network bandwidth, resulting in high communication latencies. In order to reduce communication frequency in DP-SGD, large minibatches are processed per iteration. In fact, the minibatch size in DP-SGD is increased proportionally to the number of GPUs. Problematically, larger minibatches limit the model's ability to generalize, hence requiring more passes over the dataset to reach target accuracy [4] and exhibiting statistical inefficiency [5]. Researchers at Google Brain recently demonstrated the effects of statistical inefficiency in large minibatch DP-SGD on a diverse class of workloads, concluding that DP-SGD fails to reduce the number of training iterations beyond a critical minibatch size [10]. As statistical inefficiency prevents DP-SGD from reducing the number of training iterations and in turn total training time, the scalability of DP-SGD is fundamentally limited.

Evolutionary algorithms (EA) have been employed in optimizing model architectures [8] and training hyper-parameters [7]. At the same time, EA offer an alternative to SGD in directly training the parameters of neural models. This branch of research has observed little attention and studies have been limited to modestly sized models [9]. Instead of training and evaluating a single model as in SGD using exact gradient values, an EA-based approach uses a population of models (same architecture but different parameter values), where the population presents an approximation of the gradient. For models with a large number of parameters, researchers believe that EA with approximate gradients are unable to match the solution quality of SGD with exact gradients [9].

Despite the drawbacks, we observe that EA enjoy several fundamental advantages for scalable training of complex models. EA are advantageous in optimization problems where the optimization surface is highly complex and poorly understood [1]. More importantly, EA are highly amenable to parallelism as they train independent model instances in a population which communicate infrequently, making them suitable for a large-scale distributed training environment. Another critical advantage of training a population of models, with each GPU (or a set of GPUs) training a different model instance, is that there is no need to scale the minibatch size with the number of GPUs.

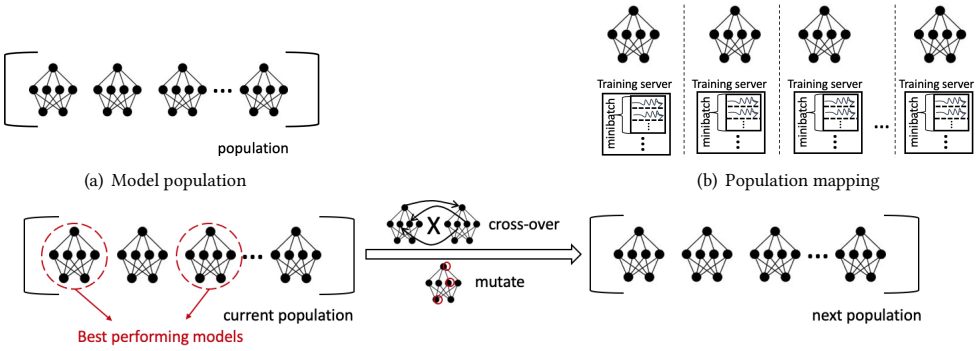
Our insight is that EA can be used together with SGD as a meta-algorithm for training neural models. This training arrangement allows a higher degree of parallelism compared to DP-SGD by virtue of training a model population. Each model in the population trains independently of the other models using local SGD for a defined number of epochs. The findings of the model instances are then consolidated through an evolutionary step the output of which is a new and better performing population of models. The overall training is an iterative process which terminates when the best performing model candidate meets the termination criteria.

Our key insight is that training with EA+SGD is inherently parallel, and thus is more suitable for an ultra large-scale distributed training setup compared to simple DP-SGD. This arrangement has the potential to significantly reduce model training time, which will aid both research and product development.

2 OUR APPROACH

We observe that training multiple and distinct instances of the model on different training GPUs (or set of GPUs) allows for high throughput without increasing the effective minibatch size. The principal challenge of such an approach lies in consolidating the "learning" of the model instances. We find that this problem maps exactly to population-based optimization scenarios where EA operate well. Employing EA together with SGD allows training multiple model instances through SGD independently of each other, which are then periodically consolidated using evolutionary operators. We name this technique Population-based Evolutionary SGD (PESGD).

PESGD mechanics. PESGD works with a population of models. Initially, a population of neural models (same architecture but different parameters) is generated – the population generation step



(a) Model population (b) Population mapping

(c) Evolution step

Figure 1: Model population creation and mapping in PESGD.

(figure 1(a)). Each model instance in a population is independently trained using SGD locally (figure 1(b)), hence using the exact gradient values. Local SGD is run for a pre-defined number of epochs over the entire training dataset – the SGD step. At the end of each SGD step, all of the models in the population are evaluated using the evaluation dataset – the evaluation step. The evaluation scores are analogous to the "fitness" of each candidate model to the target score or the "fitness target". Based on the obtained fitness scores, the top performing models are selected as parents for the next population generation – the selection step. Using the selected parents, a pool of models is then created making use of evolutionary operators such as mutation and cross-over – the evolution step (figure 1(c)). This pool of models is then installed as the new population for next SGD step. The PESGD algorithm is summarized below.

```
while max(fitness_eval) < fitness_target:
do
  1. Population generation step
  2. SGD step
  3. Evalution step
  4. Selection step
  5. Evolution step
done
```

PESGD benefits. PESGD combines the best of both SGD and EA. First, each model in a population trains using local SGD for a defined number of epochs. Operating on exact gradients with local SGD during this period helps preserve precision in the search for a good quality solution. In contrast, with an approximate gradient (e.g., if using EA exclusively), the trajectory towards the eventual solution can be longer.

Second, each model in the population runs a local SGD instance meaning that the minibatch is confined only to that GPU. The effective minibatch size, therefore, does not need to increase with the population size or the number of training machines/units.

Third, each model in the population trains completely independently of the other models during the SGD step. This naturally eliminates worker stalling for each SGD iteration due to stragglers which is a common problem in synchronous DP-SGD [6].

Finally, population-based training maps better to a distributed training setup where network can often be the bottleneck. Within PESGD, each SGD step is completely devoid of communication over the network. The only communication is during the short selection and evolution steps which is negligible in comparison to the per-iteration communication in DP-SGD.

PESGD challenges. While SGD and EA provide complementary benefits for neural model training, careful selection of hyper-parameters for each algorithm is imperative. Existing literature provides extensive studies for hyper-parameter tuning of SGD [3], and highlights the importance of the choice of hyper-parameters for the SGD training trajectory. We believe that similar is true for EA. A suitable selection of population size, evolution frequency, evolutionary operators and population diversity will produce a favourable PESGD trajectory.

3 CONCLUSION

The need for high-quality deep neural models mandates low training times to facilitate rapid exploration of a large number of potential models. For example, training a model in a few hours as opposed to a few weeks would qualitatively change the extent of experimentation that both researchers and practitioners can afford. State-of-the-art SGD-based training algorithms are ineffective at leveraging additional hardware resources in a large-scale distributed setup to reduce training time. In response, we propose supplementing SGD with a meta-algorithm, EA, which lends itself to higher degrees of hardware parallelism. This training approach has the potential to unlock unprecedentedly low model training times through massive parallelism.

REFERENCES

[1] X. Cui et al. 2018. Evolutionary stochastic gradient descent for optimization of deep neural networks. In *NIPS*. 6048–6058.
[2] X. Jia et al. 2018. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv:1807.11205* (2018).
[3] Y. Jiang et al. 2019. Fantastic Generalization Measures and Where to Find Them. *arXiv:1912.02178* (2019).
[4] N. Keskar et al. 2016. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv:1609.04836* (2016).
[5] A. Koliousis et al. 2019. CROSSBOW: scaling deep learning with small batch sizes on multi-gpu servers. *VLDB* 12, 11 (2019), 1399–1412.
[6] S. Li et al. 2018. Near-optimal straggler mitigation for distributed gradient methods. In *2018 IEEE IPDPSW*. IEEE, 857–866.
[7] I. Loshchilov and F. Hutter. 2016. CMA-ES for hyperparameter optimization of deep neural networks. *arXiv:1604.07269* (2016).
[8] R. Miikkulainen et al. 2019. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Elsevier, 293–312.
[9] G. Morse and K. O Stanley. 2016. Simple evolutionary optimization can rival stochastic gradient descent in neural networks. In *GECCO*. 477–484.
[10] C. Shallue et al. 2018. Measuring the effects of data parallelism on neural network training. *arXiv:1811.03600* (2018).