DarwiNN: Efficient Distributed Neuroevolution under Communication Constraints

Gurshaant Singh Malik
University of Waterloo

Lucian Petrica
Xilinx Research Labs
lucianp@xilinx.com

Nachiket Kapre
University of Waterloo

Michaela Blott
Xilinx Research Labs

ABSTRACT

Neuroevolution (NE), defined as the application of evolution-based training methods to Deep Neural Networks (DNNs), has recently demonstrated encouraging results on a variety of learning tasks. NE is highly parallel and relies on DNN inference as its main computational kernel and therefore can potentially leverage large-scale distributed inference-specific hardware infrastructure in the cloud or edge. We introduce chromosome updates (CU), a novel communication-optimized method for distributing NE computation, and DarwiNN, an open-source, GPU-accelerated distributed NE toolbox based on PyTorch, which implements CU and other algorithms for distributing NE.

CCS CONCEPTS

• **Computing methodologies** → **Distributed algorithms**; **Bio-inspired approaches**;

KEYWORDS

Neuroevolution, Distributed Computing, PyTorch, GPU

ACM Reference Format:
Gurshaant Singh Malik, Lucian Petrica, Nachiket Kapre, and Michaela Blott. 2020. DarwiNN: Efficient Distributed Neuroevolution under Communication Constraints. In *Genetic and Evolutionary Computation Conference Companion (GECCO '20 Companion), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3377929.3390007

1 INTRODUCTION

Neuro-Evolution (NE) is an evolution-based alternative to backpropagation for training the weights of deep neural networks (DNNs). NE has been applied successfully to various problems in reinforcement learning (RL) [2], but also to more traditional areas of supervised deep learning such as image classification [1, 3]. As NE methods become better understood and deliver useful results in deep learning, their computational efficiency becomes more important. In this work, we present Chromosome Updates (CU), a communication efficient algorithm for distributing NE computation across a cluster of computers with optimal scalability. Furthermore, we present DarwiNN[1], an open-source, GPU-accelerated and distributed NE toolbox, which implements CU and allows a PyTorch DNN training flow to be easily converted to distributed NE.

GECCO '20 Companion, July 8–12, 2020, Cancún, Mexico

ACM ISBN 978-1-4503-7127-8/20/07.
https://doi.org/10.1145/3377929.3390007

2 BACKGROUND

Evolution Strategies (ES) refers to a class of algorithms which approximates gradient descent through evolution-like dynamics. When training a DNN with a ES-like algorithm, a population of DNN offspring is generated by mutation, achieved by adding gaussian noise to a parent DNN model. The offspring are evaluated through inference on the training dataset, and the resulting fitness (loss) values are combined with the mutation noise to generate gradients for the parent DNN model with respect to the objective function. Two variants of ES, denoted OpenAI-ES[2] and SNES[1] have been utilized to train DNNs for reinforcement learning and image classification respectively. In these works, it was observed that for the large population sizes required for DNN training, mutation and inference computation can be effectively distributed to large numbers of computers, but gradient generation cannot. In [1] an approximate gradient generation technique - semi-updates (SU) - allows effective distributed computation but significantly increases the inter-process communication requirements of the algorithm.

3 CHROMOSOME UPDATES

In the chromosome updates distribution approach (CU), each of N worker processes executes Algorithm 1, where $evalfitness$ performs inference on the training batch and $allgather$ implements the MPI collective of the same name.

Algorithm 1: Chromosome Updates Worker Process

Input: DNN model size M, ES parent $\theta \in \mathbb{R}^M$, population size P, number of workers N, worker index j
Output: Gradient vector $\nabla_\theta F$

Initialize gaussian noise matrix Z[i][j] for $i \in [0, P)$ and $j \in [0, M)$
for $0 \leq i < P/N$ **do**
  for $0 \leq k < M$ **do**
    $\theta_M[k] = \theta[k] + Z[i + jP/N][k]$
  $U_L[i] = evalfitness(\theta_M)$
U = allgather(U_L)
for $0 \leq i < P$ **do**
  for $0 \leq k < M/N$ **do**
    $\nabla_\theta F_L[k] = \nabla_\theta F_L[k] + U[i] * Z[i][k + jM/N]$
$\nabla_\theta F = allgather(\nabla_\theta F_L)$

[1] https://github.com/Xilinx/DarwiNN

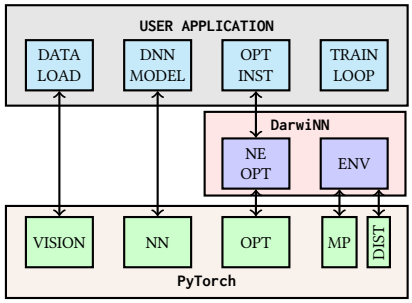

Figure 1: Overview of the DarwiNN software stack

CU requires 2x less communication compared to SU and is also an exact calculation of ES gradients. For most practical values of M and P, generating and storing the entire noise matrix Z at line 1 is unfeasible therefore parts of it must be generated coherently on-demand (e.g. before lines 4 and 9) then discarded.

4 DARWINN

DarwiNN is a GPU-accelerated distributed neuroevolution library designed for *plug-and-play* integration with existing supervised learning flows based on Pytorch. Fig. 1 illustrates how DarwiNN interacts with the user application and PyTorch. Users create a `DarwiNN` environment object (ENV), which provides distributed communication over MPI collectives. The NE optimizer (NE OPT) implements the desired evolution algorithm with GPU acceleration, and refers to the environment for communication. Gradient-based NE optimizers estimate the gradient of the underlying fitness functions and utilize `Pytorch` optimizers for model weight updates. DarwiNN provides distributed implementations for OpenAI-ES[2] and SNES[1] NE optimizers with multiple distribution methods.

5 EVALUATION

We evaluate CU scaling efficiency versus SU by training CNNs with OpenAI-ES in a distributed context at scale. We utilize DNNs from previous work - MNIST_3M [3] and CIF_900K and CIF_8M [1]. The suffix indicates the number of parameters of each DNN. Training scripts were implemented in DarwiNN for both distribution methods. We execute scalability experiments on up to 48 nodes (96 GPUs) of the Dutch National Supercomputer Cartesius. We utilize Infiniband and Ethernet network fabrics to evaluate the effect of network throughput on the scalability of SU and CU.

Weak scaling efficiency is defined as $E_w^N = t_1/t_N$ (t_1 and t_N are the runtimes when executing with 1 and N GPUs respectively). It applies when P/N is constant and we increase N is to increase P. Strong scaling efficiency is defined as $E_s^N = t_1/(i * t_N)$ and applies when P is constant and we increase N to reduce runtime. Fig. 2 (left) illustrates E_w^N for 3 per-GPU population sizes P and in scenarios with/without networking throughput constraints. Fig. 2 (rightmost column) illustrates E_s^N in scenarios with/without networking constraints, on 1 to 4 nodes (2 GPUs per node). CU performs better than SU under communication constraints. We observe that for CU the coherent generation of the noise matrix becomes a bottleneck as the number of workers increases. Additional performance could be obtained by reducing the reseeding time of the GPU PRNG.

REFERENCES

[1] Karel Lenc, Erich Elsen, Tom Schaul, and Karen Simonyan. 2019. Non-Differentiable Supervised Learning with Evolution Strategies and Hybrid Methods. *CoRR* abs/1906.03139 (2019). arXiv:1906.03139 http://arxiv.org/abs/1906.03139

[2] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864* (2017).

[3] Xingwen Zhang, Jeff Clune, and Kenneth O. Stanley. 2017. On the Relationship Between the OpenAI Evolution Strategy and Stochastic Gradient Descent. *ArXiv* abs/1712.06564 (2017).

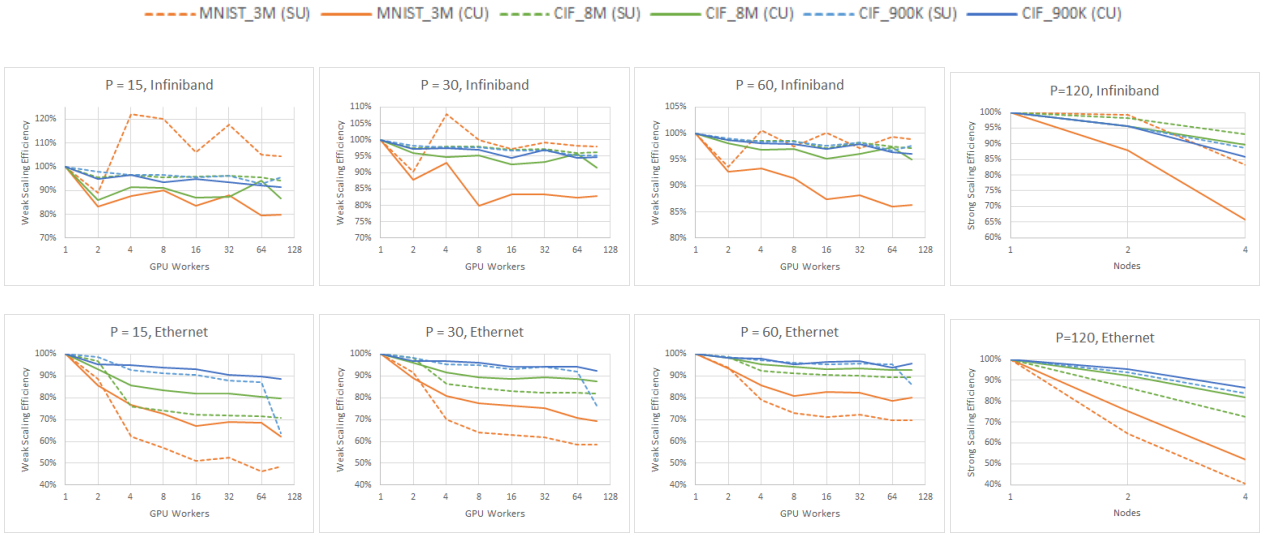

Figure 2: Scaling Efficiency Comparison of SU and CU with and without Communication Consraints