# Swarm Training

Shawn Presser

January 2020

**Abstract**

We present a method for training large machine learning models using dozens of independent TPUs. We are able to fine-tune GPT-2 1.5B using 80 preemptible TPUv3-8's on a 10GB dataset in less than a week, a speed comparable to TPU pods, resulting in equivalent training performance at significantly lower costs. Our technique can be applied to any codebase that can interact with TPUs, such as Tensorflow and PyTorch. Finally, we experimentally verify that TPUv2-8's can allocate up to 300GB of RAM during training, showing that TPUs may be the most convenient way to train massive models.

## 1 Introduction

When training very large machine learning models, one problem is that models often require more memory than is available to most GPUs. Even if the model itself fits into available memory, the training process can sometimes consume an order of magnitude more memory due to backpropagation requirements. Gradient checkpointing [1] offers a way to reduce the backpropagation memory cost by trading CPU time for memory usage. Still, for the largest models, researchers have often resulted to using bfloat16 precision [2] rather than full float32 precision, or using Adafactor rather than Adam optimizer, both of which can be harmful to final model quality or training time.

Other techniques for fitting large models into available memory include Megatron [3], which uses model parallelism to split backpropagation across multiple GPUs. Although model parallelism offers impressive scaling performance, it is often time-consuming to implement in practice, requiring intensive analysis of how to partition each type of layer in the model. Additionally, not every type of layer can be partitioned using model parallelism, rendering model parallelism an incomplete solution for certain model architectures.

Ideally, for training massive models, a training technique would satisfy several requirements:

1. No tradeoffs: Use full float32 + Adam optimizer during training.

2. Easy to implement: The overall design of the training system should be conceptually simple to understand and deploy.

3. Realistic: The technique should be deployable on a variety of hardware configurations circa 2020.

4. Cost-scalable: There should be a linear relationship between money and effectiveness. [1]

In this work we show that TPUs satisfy all of these requirements. We demonstrate that GPT-2 1.5B can be fine-tuned in a matter of days by using a "swarm" of TPUs. We also show that the technique is straightforward to implement for other model architectures, e.g. StyleGAN.

[1] Effectiveness is a combination of training speed and final quality. Training speed must be minimized and quality maximized simultaneously.

## 2 Related Work

Hogwild [4] proposes a technique of lock-free training, allowing each device access to shared memory with the possibility of overwriting each other's work. The authors provide emperical evidence that this technique can be hunreds of times faster than traditional lock-step training approaches (where gradients are calculated by each device and then applied to all devices synchronously).

Data parallelism is a popular technique for accelerating machine learning. It distributes a set of training samples among a number of processors, each of which trains in parallel, then the results are combined together. Typically the results take the form of gradient updates, which are broadcasted to all processors after each training step using an all-sum operation. Google Brain conducted an extensive study of data parallelism [5].

## 3 Method

We observe that Hogwild's lock-free technique is closely related to data parallelism: In both cases, a set of processors each train independently on part of the training data, then the results are combined together. The key insight of this paper is to apply Hogwild's lock-free training technique at scale.

Basically, the Swarm Training algorithm is to spawn N training sessions, one for each processor available (GPU, TPU, etc), and to average the model parameters as quickly as possible across all processors. Training and averaging happen simultaneously.

Unlike Hogwild, Swarm Training does not broacast gradient updates. Instead, we continuously average the actual model parameters. This design eliminates much code complexity (a central focus of Swarm Training), freeing engineers to focus on two objectives:

1. Spawn N training sessions in parallel
2. (a) Gather model parameters from all training sessions
2. (b) Compute the average
2. (c) Write the result back to each training session

We will use GPT-2 1.5B as our example. GPT-2 1.5B contains 1,558 million 32-bit floating point values, or roughly 5.8GB of data. Each processor participating in Swarm Training receives its own copy of the model and immediately begins to modify it via training. Our goal is therefore to read 5.8GB of data from each processor, compute the average, and write the averaged result back to each processor, as quickly as possible.

Suppose there are 80 processors participating in swarm training. If we naively apply step 2, we would read all values from each processor, average them together, and then write the results. However, this would require reading 5.8GB * 80 = 464.2GB of data before the final averaged value can be computed, followed by writing 464.2GB of data back to each processor. In practice, it's crucial to minimize the time between reading values from each processor and writing results to each processor. [2]

Instead, we divide up GPT-2 1.5B into "slices", and then perform step 2 on each slice. A slice is simply a subset of the model. For example, GPT-2 1.5B has 48 layers; if you chose to use 48 slices, then each slice would consist of all the model values associated with one layer.[3]

[2] The longer the time, the more training steps are lost.

[3] We used 73 slices for GPT-2 1.5B training, but this number was chosen more or less arbitrarily: "Each slice should take about 10 seconds to process, and 73 slices seem to take about 10 seconds each on our hardware configuration."

## 4 Training Results

## 5 Conclusion

## 6 References

## References


[1] Y. Bulatov, "Fitting larger networks into memory", Jan 2018

[2] S. Wang, "BFloat16: The secret to high performance on Cloud TPUs", Aug 2019

[3] NVIDIA ADLR, "MegatronLM: Training Billion+ Parameter Language Models Using GPU Model Parallelism", Aug 2019

[4] Feng Niu, Benjamin Recht, Christopher Re, Stephen J. Wright, "HOGWILD!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent", Jun 2011

[5] Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, George E. Dahl, "Measuring the Effects of Data Parallelism on Neural Network Training", Jul 2019