NOISE_GRAD: TRAINING IN 1.58B WITH NO GRADIENT MEMORY

PREPRINT

Will Brickner
wgbrickner@gmail.com

December 7, 2024

ABSTRACT

Training large machine learning models is fantastically computationally and energetically burdensome. With astronomical cost, training runs are a risky endeavor accessible only to a trace portion of humanity. By restricting weights to low precision, inference throughput and energy consumption can be dramatically improved. It was recently shown that LLM inference can be performed in 1.58-bit (ternary) precision without any performance loss [1]. However, training remains unimproved, occurring in `f16` precision. This paper presents an algorithm which trains directly in ternary precision, operates without backpropagation or momentum, and can run concurrently with model inference for similar cost. The gradient representation is exceptionally well-suited to distributed training paradigms. A simple model representation naturally follows, with several remarkable properties. In an optimal implementation, the memory required to train a given model can be reduced by more than 10x.

1 Gradient Estimation

Using the Jacobian vector product, one can compute the alignment between an arbitrary perturbation vector ν and the loss gradient exactly. The alignment can be computed in tandem with $f(x)$, and requires neither $\mathbf{J_f}$ nor ∇f.

$$\alpha_f(\nu) = \mathbf{J_f}\nu = \nabla f \cdot \nu \tag{1}$$

In the continuous domain, the gradient may then be estimated [2]

$$\nu_i \sim \mathcal{N}(0, I) \qquad\qquad \widehat{\nabla} f = \frac{1}{n} \sum \nu_i \, \alpha(\nu_i) \tag{2}$$

In the ternary domain, values are closed in $\mathbb{T} = \{-1, 0, +1\}$. Directions and magnitudes of ternary vectors are highly constrained; vector weighting is not useful. Gradient estimation can be recovered if perturbations ν_i are sparse.

$$\nu_i \sim \text{Bernoulli}(s) \odot \text{U}\{-1, +1\} \tag{3}$$

Remarkably, only the sign of the alignment is required for estimation.

$$\widehat{\nabla} f = \sum \nu_i \operatorname{sgn} \alpha(\nu_i) \tag{4}$$

Convergence is improved by rejecting perturbations with alignment magnitude below the step median:

$$\alpha_\tau(\nu) = \operatorname{sgn} \alpha(\nu) \, \mathbf{1}_{|\alpha(\nu)| \geq \text{median}_j |\alpha(\nu_j)|} \in \mathbb{T} \tag{5}$$

$$\widehat{\nabla} f = \sum \nu_i \, \alpha_\tau(\nu_i) \tag{6}$$

The only hyperparameters are the number and sparsity of perturbations. Summation may be performed with saturation, or in higher precision followed by ternary clamping.

2 Representation Efficiency

One Seed is All You Need Pseudorandom noise has a beautiful property: it is deterministic. Vast sequences can be reproduced with only a seed. This means the perturbation vectors ν_i don't need to be kept in memory, nor transmitted. Perturbation components can be generated as they are used, faster than the speed of memory.

Distributed Training The throughput of distributed training algorithms is generally limited by synchronization, wherein gradients and optimizer states are exchanged between participants. To mitigate this problem, many sophisticated schemes have been devised [3]. Without modification, `noise_grad` gradient steps are encoded using only one tern (1.58 bits) per perturbation, dramatically reducing total communication. Hybridization with existing algorithms may prove a fruitful research direction.

2.1 Models as Steps

With such compact steps, a simple model representation emerges with remarkable properties.

Model Transport To download a model is to download its weights. Because weight initialization is pseudorandom, it is also recovered with only the seed. By expressing a model as its steps, the transport size of a model no longer directly depends on the number of parameters, but on the product of steps and perturbations. With great hubris, one can roughly estimate the size of a ternary `GPT-3 175B` in this format to be between 600KB and 19MB. [1] Size reductions also apply to higher precision models, with size increasing linearly in alignment precision. As discussed later, efficient recovery to weight space relies on perturbation sparsity.

What Can Be Model updates also benefit: to represent additional full-rank training, one can simply specify the additional steps. The initialization can be a point in weight space (i.e. a base model), or a prior sequence of steps.

Unburdened By What Has Been The complete history of model weights can be recovered with product cost in the number of parameters recovered, perturbations used, and steps consumed. Any subset of model weights can be recovered independently with no overhead. Training can be resumed from any previous step in model history. These statements are all true *a priori*. It may also be possible to edit past training steps, e.g. through masking or negation, but this exotic trick requires empirical validation not performed here.

The Burden The reconstruction algorithm is simple, embarrassingly parallel, highly local, and produces its results from noise. It is ideal for modern hardware. However, model step reduction has complexity $\mathcal{O}(nks)$. For large models with high step count, reconstruction becomes impractical, as every weight must be updated with every perturbation vector from every step. Complete reconstruction of a ternary model similar to `GPT-3 175B` would require roughly 10^{19} sums. [2]

Liberation The perturbation vectors have a fortunate property: almost every component is zero. Reducing only non-zero components lowers the reconstruction cost by a factor of the average noise density. Because most steps occur in the ultra-high sparsity regime, reconstruction cost is made tolerable. [3] Unfortunately, the memory access pattern becomes non-local, but modifications to how perturbation noise is generated may admit a local implementation.

[1] $\texttt{steps} \approx \frac{\text{300 B training tokens}}{\text{3.2 M batch size}} = 93750$, $\texttt{samples} \in [32, 1024]$, $\texttt{bits} = \log_2 3 * \texttt{steps} * \texttt{samples} \in [4.74 \times 10^6, 1.5168 \times 10^8]$

[2] $\texttt{ops} \geq \texttt{weights} * \texttt{samples} * \texttt{steps} \in [5.25 \times 10^{17}, 1.68 \times 10^{19}]$

[3] Demonstration will follow in a future revision.

3 Convergence Properties

Because of the discrete geometry of ternary space, gradient steps are also discrete. Weight trajectories are necessarily discontinuous. Loss curves carry greater noise than in high precision networks, and model performance at fixed size is slightly inferior. Larger batch sizes are required, but optimization proceeds aggressively and converges similarly to `Adam`.

To demonstrate convergence behavior, a simple MLP was trained to classify MNIST samples using `noise_grad` and `Adam`. The benchmark model is ReLU MLP of 4 total layers, with layer normalization and a hidden dimension of 256. All weights are confined to ternary, all activations are `f32`. The `Adam` optimizer stores weights in full precision, which are clamped to ternary for the forward pass. This comprises a *Straight Through Estimator* as used in the BitNet paper. It should be noted that the network does not strictly belong to the BitNet class, as the ternary dense layers are not given a continuous scale parameter. [4] This deficiency is likely responsible for the lower accuracy at convergence shared by both runs.

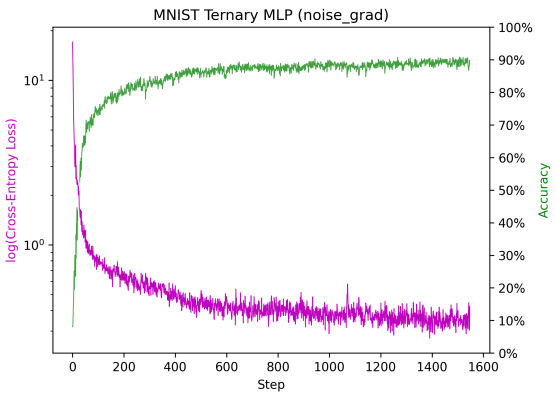

Figure 1: `noise_grad` loss and accuracy curves `{ samples=128, density=`$6 \times 10^{-5} \rightarrow 3.7 \times 10^{-6}$ `}`

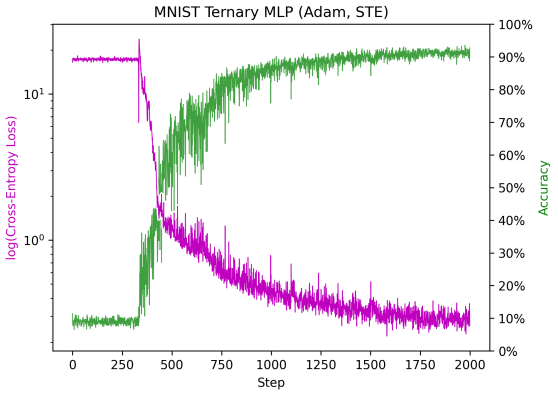

Figure 2: `Adam` optimizer loss and accuracy curves, default parameters.

Empirically, `noise_grad` encounters difficulty in the high-step regime, where only a tiny portion of model parameters remain with suboptimal values. Increasing noise sparsity improves convergence. For this demonstration, the noise density is scheduled crudely in two stages, $6 \times 10^{-5} \rightarrow 1.5 \times 10^{-5}$ when $L < 2$, and $1.5 \times 10^{-5} \rightarrow 3.7 \times 10^{-6}$ when $L < 1$. More principled and flexible methods for sparsity scheduling are needed.

4 Implementation Advice

This work does not provide optimized kernels for `noise_grad` training. Many optimization opportunities exist, some are outlined here.

Ternary Codes Seemingly uniformly, ternary compute kernels use 2-bit encodings to represent ternary values. This approach has the advantage of simplicity. Simple shifting can be used to isolate terns, and arithmetic can be performed more easily. For this simplicity, 2-bit encodings pay a 27% space overhead cost. To minimize memory usage and access, a more efficient encoding is needed. Previous work on the binary representation of ternary numbers offers excellent space efficiency, packing 5 terns per byte [4]. Utilizing this encoding can reduce space overhead to just 0.95%. Even smaller representations are possible when encoding steps, as the distribution of $\alpha_\tau(\nu)$ is biased, evenly split between zero and uniform sign noise. Development of efficient transport encodings for steps is left to future work.

JVP Sparsity The batched JVP is computed alongside an inference pass according to a set of pushforward rules, similar to differentiation rules in reverse mode. High perturbation sparsity means many pushforward rules can be simplified, often reading and producing only a few elements or matrix columns.

[4] A future revision of this preprint will contain amendments allowing co-optimization of ternary and high precision values.

5 Related Work

The TernGrad algorithm performs a stochastic quantization of high precision gradients to ternary, reducing communication in distributed learning contexts [5].

There may be a deep connection between the present work and 1-bit compressed sensing.

References

[1] Shuming Ma, Hongyu Wang, Lingxiao Ma, Lei Wang, Wenhui Wang, Shaohan Huang, Li Dong, Ruiping Wang, Jilong Xue, and Furu Wei. The era of 1-bit llms: All large language models are in 1.58 bits, 2024.

[2] Atılım Güneş Baydin, Barak A. Pearlmutter, Don Syme, Frank Wood, and Philip Torr. Gradients without backpropagation, 2022.

[3] Matthias Langer, Zhen He, Wenny Rahayu, and Yanbo Xue. Distributed training of deep learning models: A taxonomic perspective. *IEEE Transactions on Parallel and Distributed Systems*, 31(12):2802–2818, December 2020.

[4] Olivier Muller, Adrien Prost-Boucle, Alban Bourge, and Frédéric Pétrot. Efficient decompression of binary encoded balanced ternary sequences. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(8):1962–1966, 2019.

[5] Wei Wen, Cong Xu, Feng Yan, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Terngrad: Ternary gradients to reduce communication in distributed deep learning. *CoRR*, abs/1705.07878, 2017.