Zeroth-order Optimization for Quantized Transformers

Matin Ansaripour Ekaterina Pankovets Margarita Sagitova

***Abstract*—The impressive capabilities of Large Language Models come with an increasing number of model parameters. As the size of the model grows, so do the costs of training and inference, as well as memory consumption. Recent research suggests using 1-bit models such as BitNet [1], which are significantly more cost-effective at inference time due to their 1-bit weights in transformer layers. However, training such models still requires floating-point computation, which slows down the training of such models. This report explores the application of zeroth-order optimization techniques as an alternative to gradient-based methods. Our objective is to assess the effectiveness of zeroth-order optimization in training BitNet transformer models, examining its potential to reduce computational costs and improve model performance. We find that Random Gradient Estimation (RGE) can achieve performance comparable to the first-order baseline (Adam) while requiring less memory and compute and other methods such as Markov Chain Monte Carlo and Genetic Algorithm are unable to converge.**

I. Introduction

In recent years, Large Language models demonstrated remarkable performance on various tasks. However, using these state-of-the-art models is very expensive due to the large amount of computational and memory resources they demand. One possible solution to this problem is model quantization — a technique for reducing the size of models by converting model weights from high-precision floating-point representations to low-precision floating-point or integer representations. This technique significantly reduces memory usage and the computational cost of large models. But quantization brings benefits only at inference time, whereas training remains very expensive and even more unstable as compared to the full precision models.

To avoid the challenges posed by training quantized models with gradient-based methods, one could explore alternative zeroth-order optimization methods. This project investigates the effects of employing zeroth-order optimization methods in training a strongly quantized variant of transformer architecture called BitNet [1], which demonstrates the competitive performance in language modeling. Specifically, we evaluate the performance of the models trained with Genetic Algorithm, Markov Chain Monte Carlo, and function value-based gradient estimation.

II. Model

The model we use for this project is BitNet [1]. It has the same structure as usual Transformer networks, but instead of the conventional Linear layer, it uses the BitLinear layer, which allows computations with binarized weights. All the other components of the architecture remain in high precision because layer normalization and residual connections anyway contribute negligible computation costs, while input and output embeddings need to stay high-precision to perform transformer prediction sampling.

BitLinear layer. To make the variance of the output after quantization comparable to the full-precision Linear layer output variance, Layer normalization is applied as the first step in BitLinear layer.

Then the inputs are quantized to b-bit precision using use absmax quantization, which scales inputs into the range $[-Q_b, Q_b]$ ($Q_b = 2^{b-1}$):

$$\tilde{x} = \text{Clip}(\frac{Q_b}{\gamma}x, -Q_b + \epsilon, Q_b - \epsilon),$$
$$\text{Clip}(x, a_{min}, a_{max}) = \max(a_{min}, \min(a_{max}, x)),$$

where $\gamma = \|x\|_\infty$ and small floating point constant ϵ is used for numerical stability.

The weights $W \in \mathbb{R}^{n\times m}$ of the layer are binarized as follows:

$$\tilde{W} = \text{Sign}(W - \alpha), \quad \alpha = \frac{1}{nm}\sum_{i,j=1}^{n,m} W_{i,j},$$

where Sign is signum function that returns -1 if the input is less or equal than zero and 1 otherwise.

Finally, the result of multiplication $\tilde{W}\tilde{x}$ is scaled by $\frac{\beta\gamma}{Q_b}$, where $\beta = \frac{1}{nm}\|W\|_1$ to reduce the error between full-precision and quantized weights.

To train 1-bit network with gradient method original paper suggests straight-through estimator [2] to approximate the gradient. This method bypasses the non-differentiable functions, such as the Sign and Clip functions, during the backward pass, making it possible to train the model with gradient descent.

III. Data

In our experiments comparing optimization methods, we utilize the following datasets:

- *Twitter Sentiment Analysis Dataset* [3]: This dataset comprises entities and associated messages. The task is to determine the sentiment of each message towards the entity, categorizing it as Positive, Negative, Neutral, or Irrelevant to the entity. The training set consists of 73,996 samples, while the validation set contains 1,000 samples.
- *Bracket Sequences:* This dataset consists of sequences of opening '(' and closing brackets ')'. The task is predict

the next token in a sequence such that the entire sequence is correct. A sequence is defined as correct if every opening bracket has a corresponding closing bracket. The training set consists of 25,600 samples, and the validation set consists of 2,560 samples. We use this dataset because the correct bracket sequences form a context-free language, which captures some properties of natural language [4]. Therefore, it can reveal nontrivial model capabilities while still being a relatively simple dataset.

IV. Optimization algorithms

Baseline. As a baseline we train the model with Adam algorithm using straight-through estimator technique to bypass the non-differentiable functions.

Genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection. Such algorithms are not popular in deep learning applications, but they were shown to be a competitive alternative for Reinforcement Learning which posses challenge for gradient-based algorithms [5].

In this project we test an extremely simple GA. A *population* of N candidate solutions or *individuals* to an optimization problem is evolved toward better solutions for G *generations*. *Individuals* are neural network parameter vectors $\{\theta_i\}_{i=1}^N$. At every *generation* each *individual* θ_i is evaluated to produce a *fitness score*. Top T *individuals* are selected to be the parents of the next generation. To produce the next generation, a *mutation* process is repeated $N-1$ times, and the last *individual* in a new population is an unmodified copy of the best individual from the previous generation, a technique called elitism.

Mutation process is performed as follows: full-precision network parameters such as input embeddings, last linear layer mapping token embeddings to logits of output and scaling parameters of BitLinear layers are mutated adding Gaussian noise $\theta' = \theta + \sigma\epsilon$ where $\epsilon \sim N(0, I)$. And BitLinear layer weights that take one of two values ± 1 mutated by flipping each of the elements of the weight matrix with probability p.

Monte Carlo Markov Chain (MCMC) optimization is a technique to generate samples such that they can be used to approximate target distribution. In the context of machine learning, this involves sampling model parameters from the distribution that minimizes a loss function [6].

The steps of this optimizer can be summarized as follows: Given the current parameters of the model, we first evaluate the loss function, denoted as l_{cur}. Next, a new configuration of parameters is proposed: for binary weights, each weight is inverted with a probability of $\frac{1}{2}$; other parameters are randomly sampled from a normal distribution. Then, the loss function at the new parameters, denoted as l_{new}, is evaluated. The new configuration of parameters is accepted with the *acceptance probability* given by

$$A = \min\left[1, \exp(-(l_{\text{new}} - l_{\text{cur}}))\right]. \tag{1}$$

This acceptance probability ensures that moves to configurations with lower loss are always accepted, while moves to configurations with higher or equal loss are accepted with a probability that decreases exponentially with the increase in loss.

Function value-based Gradient Estimation.

Function value-based zeroth-Order (ZO) optimization is a gradient-free alternative to First-Order (FO) optimization. Instead of directly using gradients, it approximates FO gradients through function value-based gradient estimates, known as ZO gradient estimates, as detailed in studies by [7]–[10].

There are various methods for estimating ZO gradients. This report focuses on the Randomized Gradient Estimator (RGE) [9], [10], which calculates the finite difference of function values along randomly chosen direction vectors. RGE has been used by [11], [12] for memory-efficient fine-tuning of large language models (LLMs) due to its query efficiency, requiring a minimal number of function queries.

For a scalar-valued function $f(\mathbf{x})$ where $\mathbf{x} \in \mathbb{R}^d$ of dimension d, the RGE, denoted as $\hat{\nabla} f(\mathbf{x})$, is given by the central difference formula:

$$\hat{\nabla} f(\mathbf{x}) = \frac{1}{q} \sum_{i=1}^{q} \left[\frac{f(\mathbf{x} + \mu \mathbf{u}_i) - f(\mathbf{x} - \mu \mathbf{u}_i)}{2\mu} \mathbf{u}_i \right]. \tag{2}$$

Here, $\mathbf{u}_i$ is a random direction vector typically drawn from the standard Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$, q is the number of function queries, and $\mu > 0$ is a small perturbation stepsize, also known as the smoothing parameter. It's important to note that the number of queries q balances the ZO gradient estimation variance and query complexity.

We implemented RGE with Projected Stochastic Gradient Descent (SGD) for training the BitNet model. In our approach, as we said, we perturb the full-precision parameters by the standard Gaussian noise. For the integral parameters, we flip their value between $\{1, -1\}$ by probability p. Then, we estimate the gradients and update the parameters in full precision using gradient descent. At last, for the integral parameters we project the weights to values $-1, 1$ by their sign.

V. Experiments

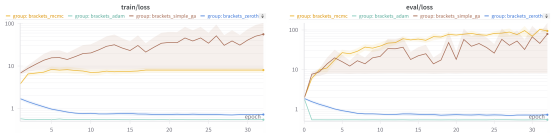

Fig. 1: Train and evaluation loss per epoch across different optimizers on Bracket Sequences dataset (logarithmic scale).

Fig. 2: GPU memory consumption of optimizers throw time.

TABLE I: Mean and standard error of evaluation loss across different optimizers.

Dataset	Adam	ZO	GA	MCMC
Brackets	**0.54 ± 0.01**	0.72 ± 0.04	79.25 ± 71.10	94.45 ± 14.63
Twitter	**0.76 ± 0.07**	4.54 ± 0.32	165.99 ± 17.16	28.68 ± 2.38

We conducted a series of experiments to evaluate the performance of the optimizers. We considered BitNet with 2 heads, a depth of 2, and a width of 64. Also, we assigned a generation task to train on the Bracket Sequences dataset and a sentiment classification task on the Twitter dataset. Initially, we fine-tuned our optimizers using the sweep functionality in Weights & Biases (wandb), performing 10 runs for each dataset. The details of the hyperparameter tuning process are presented in Appendix.

After fine-tuning, we ran each optimizer on the dataset with 5 different random seeds to ensure the robustness and reproducibility of our results (Fig. 1). All experiments were executed using a T4 GPU with 15GB VRAM for 32 epochs on the Bracket Sequences dataset and 6 epochs on the Twitter dataset. To monitor floating-point computation, we measured the following metrics (Table II and Table III):

- **Train/Int MACs**: The number of integer Multiply-Accumulate operations during training.
- **Train/Float MACs Forward**: The number of floating-point Multiply-Accumulate operations during the forward pass.
- **Train/Random Numbers**: The number of random numbers generated during training.
- **Train/Float MACs Backward**: The number of floating-point Multiply-Accumulate operations during the backward pass.

TABLE II: Metrics measurement on Bracket Sequences dataset by considering maximum value among the epochs.

Metric	Adam	ZO	GA	MCMC
float MACs forward	2.16E+09	8.79E+09	1.35E+11	5.49E+08
float MACs backward	4.32E+09	0	0	0
int MACs	0	2.58E+10	3.96E+11	1.61E+09
random numbers	0	8.04E+05	1.56E+06	5.03E+04

TABLE III: Metrics measurement on Twitter dataset by considering maximum value among the epochs.

Metric	Adam	ZO	GA	MCMC
float MACs forward	3.82E+09	1.80E+10	1.41E+11	4.40E+09
float MACs backward	7.64E+09	0	0	0
int MACs	0	1.04E+11	8.18E+11	2.56E+10
random numbers	0	6.58E+07	6.37E+07	2.06E+06

VI. Discussion

On the Bracket Sequences dataset, we observed comparable results between Adam and Random Gradient Estimation (RGE), while Markov Chain Monte Carlo (MCMC) and simple Genetic Algorithm (GA) showed divergent behavior. BitNet models are quantized models and they do not fully illustrate the spectrum distribution of models. Since GA and MCMC try to find the optimal model by model generation, BitNet models might not give substantial insight to the algorithm to search and find a suitable model. In this way, they can not converge properly.

Since RGE showed good convergence capability, it requires further exploration. Notably, RGE achieved comparable results with Adam while requiring fewer memory resources (Fig. 5). Adam needs more memory for full gradient computation, whereas RGE estimates gradients using finite differences along randomly chosen direction vectors. This allows us to have more parallel data point computation in a single run while maintaining the approximation of the RGE.

For the Twitter dataset, all the optimizers except Adam demonstrated divergent behavior, i.e. the models were moving away from the optimal solution during training. Since this dataset represents a harder and more non-convex task, it is likely that our previous statement for Bracket Sequences about GA and MCMC holds here. For RGE, it might be possible that we need to increase the number of random vectors to approximate better which our limited resources did not allow us to investigate.

One key feature of gradient-free optimizers is their low-cost computation with low memory consumption. Note that some of their larger memory consumption in Fig. 5 and also more MACs operations in Table II and Table III is because we use much larger batch size for these optimizers (see the Appendix). Since gradient-free optimizers have a lower memory footprint they allow us to use a larger batch size. Therefore, since they are seeing more data points at the time, they converge fast. So, it might be a good idea to investigate these optimizers using full-precision models since they can give a proper model distribution to these optimizers.

Another point to mention is that all the gradient-free optimizers did not use any backward-passes, which are very heavy in floating point operations and this fulfills the purpose of reducing floating point operations.

VII. Conclusion and Future Work

This study explored the use of zeroth-order optimization techniques for training quantized transformer models, revealing that while Adam remains superior in performance, Randomized Gradient Estimation (RGE) shows promise with lower resource demands. However, Genetic Algorithm (GA) and Monte Carlo Markov Chain (MCMC) methods struggled with convergence, highlighting the need for further refinement. Future work will focus on investigating GA and MCMC for full-precision models to potentially improve their efficacy and explore the application of RGE on other datasets and deeper BitNet architectures to better understand its scalability and adaptability.

REFERENCES

[1] H. Wang, S. Ma, L. Dong, S. Huang, H. Wang, L. Ma, F. Yang, R. Wang, Y. Wu, and F. Wei, "Bitnet: Scaling 1-bit transformers for large language models," 2023. [Online]. Available: https://arxiv.org/abs/2310.11453

[2] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013. [Online]. Available: https://arxiv.org/abs/1308.3432

[3] "Twitter sentiment analysis," https://www.kaggle.com/datasets/jp797498e/twentity-sentiment-analysis, accessed: April 2024.

[4] J. Ebrahimi, D. Gelda, and W. Zhang, "How can self-attention networks recognize dyck-n languages?" 2020. [Online]. Available: https://arxiv.org/abs/2010.04303

[5] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," 2017. [Online]. Available: http://www.bepress.com/sagmb/vol4/iss1/art2

[6] C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan, "An introduction to mcmc for machine learning," *Machine Learning*, vol. 50, no. 1, pp. 5–43, Jan 2003. [Online]. Available: https://doi.org/10.1023/A:1020281327116

[7] A. D. Flaxman, A. T. Kalai, and H. B. McMahan, "Online convex optimization in the bandit setting: gradient descent without a gradient," 2004.

[8] S. Ghadimi and G. Lan, "Stochastic first- and zeroth-order methods for nonconvex stochastic programming," 2013.

[9] J. C. Duchi, M. I. Jordan, M. J. Wainwright, and A. Wibisono, "Optimal rates for zero-order convex optimization: the power of two function evaluations," 2014.

[10] Y. Nesterov and V. Spokoiny, "Random gradient-free minimization of convex functions," *Foundations of Computational Mathematics*, vol. 17, 11 2015.

[11] S. Malladi, T. Gao, E. Nichani, A. Damian, J. D. Lee, D. Chen, and S. Arora, "Fine-tuning language models with just forward passes," 2024.

[12] Y. Zhang, P. Li, J. Hong, J. Li, Y. Zhang, W. Zheng, P.-Y. Chen, J. D. Lee, W. Yin, M. Hong, Z. Wang, S. Liu, and T. Chen, "Revisiting zeroth-order optimization for memory-efficient llm fine-tuning: A benchmark," 2024.

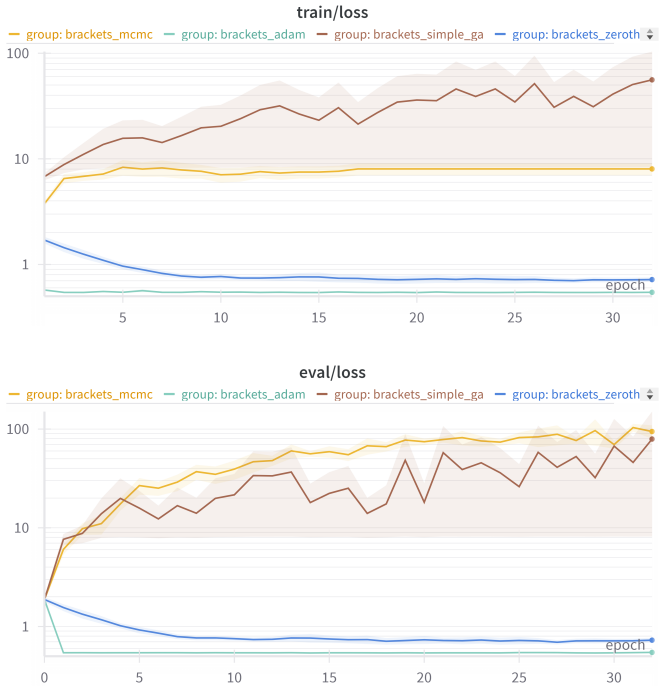

Fig. 3: Train and evaluation loss per epoch across different optimizers on Bracket Sequences dataset (logarithmic scale).

APPENDIX

In the following tables, you can find the details about the tuned hyperparameters that we used during training. Also, the search space for the hyperparameter tuning is mentioned.

Parameter	Twitter dataset	Brackets dataset	Fine-tuning range
lr	0.01224	0.00761	0.0005 - 0.1
beta1	0.45830	0.78673	0.2 - 0.8
beta2	0.11998	0.90802	0.01 - 1.0
weight_decay	0.30642	0.45334	0.01 - 0.5
warmup_steps	128	-1	{-1, 128, 1024}
max_grad_norm	7	2	1 - 10
batch_size	256	256	{256, 2048}

TABLE IV: Hyperparameters for Adam optimizer on Twitter and Brackets datasets with fine-tuning ranges

Parameter	Twitter dataset	Brackets dataset	Fine-tuning range
bin_mutation_prob	0.72075	0.6071	0.2 - 0.8
emb_mutation_scale	0.16520	0.1027	0.01 - 1.0
population_size	32	32	{32, 48, 64, 128}
treshold	16	16	{8, 16, 32}
batch_size	2048	256	{256, 2048}

TABLE V: Hyperparameters for Genetic Algorithm on Twitter and Brackets datasets with fine-tuning ranges

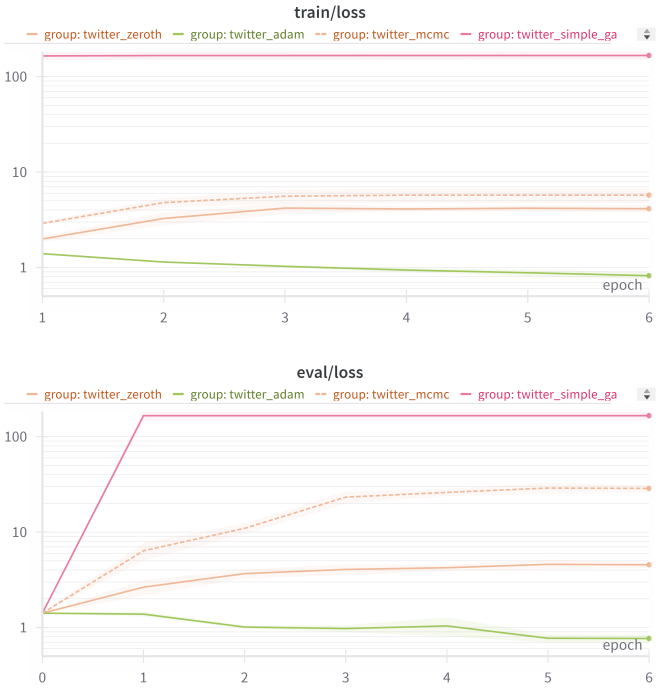

Fig. 4: Train and evaluation loss per epoch across different optimizers on Twitter dataset (logarithmic scale).

Fig. 5: GPU memory consumption of optimizers throw time.

Parameter	Twitter dataset	Brackets dataset	Fine-tuning range
bin_mutation_prob	0.75592	0.67910	0.2 - 0.8
emb_mutation_scale	0.02961	0.05710	0.01 - 1.0
batch_size	2048	256	{256, 2048}

TABLE VI: Hyperparameters for MCMC optimizer on Twitter and Brackets datasets with fine-tuning ranges

Parameter	Twitter dataset	Brackets dataset	Fine-tuning ranges
lr	0.00013	0.00074	0.00001 - 0.001
bin_mutation_prob	0.33162	0.21711	0.2 - 0.8
momentum	0.73738	0.53924	0.0 - 0.9
random_vec	32	16	{8, 16, 32}
batch_size	256	256	{256, 2048}

TABLE VII: Hyperparameters for RGE optimizer on Twitter and Brackets datasets with fine-tuning ranges

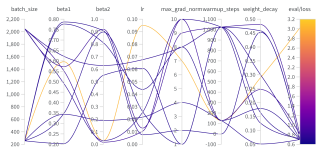

Fig. 6: Hyperparameter tuning for Adam on Brackets dataset.

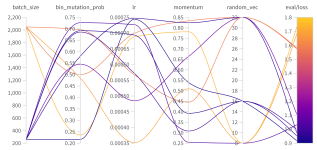

Fig. 7: Hyperparameter tuning for RGE on Brackets dataset.

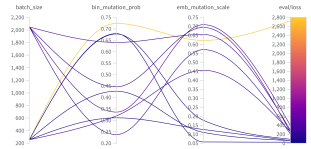

Fig. 8: Hyperparameter tuning for MCMC on Brackets dataset.

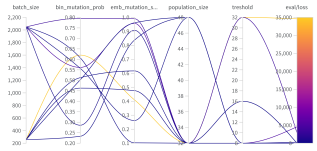

Fig. 9: Hyperparameter tuning for GA on Brackets dataset.

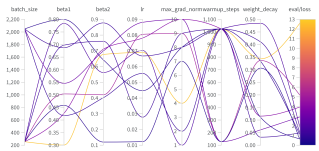

Fig. 10: Hyperparameter tuning for Adam on Twitter dataset.

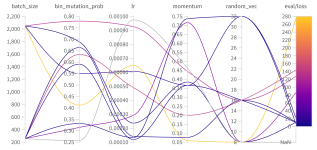

Fig. 11: Hyperparameter tuning for RGE on Twitter dataset.

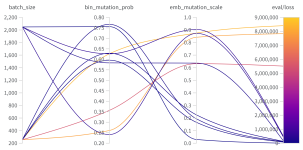

Fig. 12: Hyperparameter tuning for MCMC on Twitter dataset.

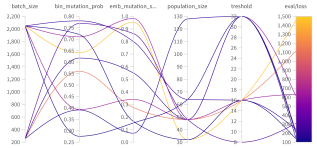

Fig. 13: Hyperparameter tuning for GA on Twitter dataset.