Comparison and Analysis of Decentralized Federated Learning Algorithms in Deep Learning settings

Jeremy Rack (2568757)
Monseej Purkayastha (7047530)
Georgi Vitanov (7025852)
Saarland Informatics Campus, Saarland University, Saarbrücken, Germany

***Abstract*—This report presents an in-depth exploration of distributed learning methods, primarily focusing on the BEER and MoTEF algorithms and their performance in deep learning settings. Through extensive experimentation, we investigate the efficacy of these methods in a decentralized distributed learning framework. Our results demonstrate that both BEER and MoTEF exhibit promising performance in terms of efficiency and performance. We also analyse the effect of different compression methods that are an integral part of these algorithms. Furthermore, we analyse the scalability of these algorithms in terms of number of nodes or systems in the network, different network topologies as well as batch sizes and other parameters. Our code for the various experiments is hosted in Github.**

I. Introduction

Distributed learning has become crucial for handling large-scale data and computations. It breaks down complex tasks across multiple nodes, accelerating learning and enabling work with datasets too large for single machines. This approach also enhances data privacy by sharing only model weights between entities. Decentralized distributed learning further amplifies these benefits. In this system, nodes operate independently, communicating directly with each other without a central authority. This structure offers increased privacy by keeping data local and provides robustness against individual node failures. Overall, decentralized distributed learning extends the advantages of distributed learning, offering a more resilient and privacy-preserving approach to large-scale machine learning tasks.

II. Methodology

There are many decentralized federated learning algorithms out there. In this report, we focus on two methods; BEER (BEtter comprEssion for decentRalized optimization)[1] and MoTEF (Momentum Tracking and Error Feedback) [2]. BEER introduces improvements in the communication space by incorporating novel compression techniques, MoTEF improves on it further by incorporating momentum.

A. BEER

The BEER learning algorithm is designed for decentralized optimization, effectively utilizing message compression to mitigate communication bottlenecks. Each client running BEER maintains the following variables:

Algorithm 1 BEER(BEtter comprEssion for decentRalized optimization)

Require: weights $\mathbf{x}_0$, learning rate η, mixing parameter γ, compression function $\mathcal{C}_\alpha$, Iterations T, g_i^t local approx. of v, h_i^0 local approx. of x, v_i^0 local approx. of global gradient.

for $t = 0, 1, 2, \dots$ concurrently for each client $i \in [n]$ **do**
  if $t > 0$ **then**
    for neighbors $j : (i, j) \in E$ **do**
      ▷ *Send/Receive updates to weights/gradient*
      send $q_{h,j}^t, q_{g,j}^t$
      receive $q_{h,j}^t, q_{g,j}^t$
      $h_j^t = h_j^{t-1} + q_{h,j}^t$
      $g_j^t = g_j^{t-1} + q_{g,j}^t$
    end for
  end if
  ▷ *Update x with Mixing- and Gradient Step*
  $x_i^{t+1} = x^t + \gamma \sum_{j:(i,j)\in E} w_{ij}(h_j^t - h_i^t) - \eta v_i^t$
  ▷ *Compress Update*
  $q_{h,i}^{t+1} = \mathcal{C}_\alpha(x_i^{t+1} - h_i^t)$
  $h_i^{t+1} = h_i^t + q_{h,i}^{t+1}$
  ▷ *Update v with Mixing and Gradient Difference*
  $\Delta grad_i^{t+1} = \nabla f_i(x_i^{t+1}, \xi_i^{t+1}) - \nabla f_i(x_i^t, \xi_i^{t+1})$
  $v_i^{t+1} = v_i^t + \gamma \sum_{j:(i,j)\in[n]} w_{ij}(g_j^t - g_i^t) + \Delta grad_i^{t+1}$
  ▷ *Compress v Update*
  $q_{g,i}^{t+1} = \mathcal{C}_\alpha(v_i^{t+1} - g_i^t)$
  $g_i^{t+1} = g_i^t + q_{g,i}^{t+1}$
end for

- Weights $\mathbf{x}^t$
- Learning rate η
- Mixing parameter γ
- Compression function $\mathcal{C}_\alpha$
- Iterations T
- Local approximation of v_i^t: g_i^t
- Local approximation of $\mathbf{x}^t$: h_i^t
- Local approximation of the global gradient: v_i^t

Additionally, each client keeps track of the h_j^t and g_j^t values from its neighbors (denoted as j). These values are

only updated with compressed messages from the neighbor nodes and serve as compressed estimates of $\mathbf{x}^t$ and v_i^t. The algorithm employs mixing steps to synchronize the clients. For the gradient step, a global estimate of the gradient v_i^t is used, which is updated via gradient tracking.

B. MoTEF

Algorithm 2 MoTEF(MOmentum Tracking and Error Feedback)

Require: weights $\mathbf{x}_0$, learning rate η, momentum parameter λ, mixing parameter γ, compression function $\mathcal{C}_\alpha$, Iterations T, g_i^t local approx. of v, h_i^0 local approx. of x, v_i^0 local approx. of global gradient, m is the momentum.

for $t = 0, 1, 2, \ldots$ concurrently for each client $i \in [n]$ **do**
  if $t > 0$ **then**
    for neighbors $j : (i, j) \in E$ **do**
      ▷ *Send/Receive updates to weights/gradient*
      send $q_{h,j}^t, q_{g,j}^t$
      receive $q_{h,j}^t, q_{g,j}^t$
      $h_j^t = h_j^{t-1} + q_{h,j}^t$
      $g_j^t = g_j^{t-1} + q_{g,j}^t$
    end for
  end if
  ▷ *Update x with Mixing- and Gradient Step*
  $x_i^{t+1} = x^t + \gamma \sum_{j:(i,j)\in E} w_{ij}(h_j^t - h_i^t) - \eta v_i^t$
  ▷ *Compress Update*
  $q_{h,i}^{t+1} = \mathcal{C}_\alpha(x_i^{t+1} - h_i^t)$
  $h_i^{t+1} = h_i^t + q_{h,i}^{t+1}$
  ▷ *Compute Momentum with Gradient of current Batch*
  $m_i^{t+1} = (1 - \lambda)m_i^t + \lambda \nabla f_i(x_i^{t+1}, \xi_i^{t+1})$
  ▷ *Update v with Mixing and Momentum Difference*
  $v_i^{t+1} = v_i^t + \gamma \sum_{j:(i,j)\in[n]} w_{ij}(g_j^t - g_i^t) + m_i^{t+1} - m_i^t$
  ▷ *Compress v Update*
  $q_{g,i}^{t+1} = \mathcal{C}_\alpha(v_i^{t+1} - g_i^t)$
  $g_i^{t+1} = g_i^t + q_{g,i}^{t+1}$
end for

The only difference between the BEER and MoTEF algorithms is the use of **momentum**. MoTEF uses momentum to stabilize training. It introduces a new momentum parameter λ to scale the smoothing of the gradient. The smoothed gradient is then saved in m^{t+1}. The actual gradient step is then performed using m^{t+1} and the previous smoothed gradient m^t.

C. Experimental setup

We have employed a simple fully-connected 2-layered MLP (shown in Figure 1) as the baseline. The MNIST dataset [3] was utilized for all experiments. We evaluated the model performance using both algorithms for 10 epochs. This parameter (no. of epochs) was kept constant for all further experiments.

III. Results

For our baseline experiments shown in Figure 2, we have used the standard 2-layer connected MLP with a ring topology. The other parameters for the baseline are $\gamma = 0.001, \eta = 0.03, \lambda = 0.9$, compression ratio $= 0.2$. The network size was kept to a modest 4.

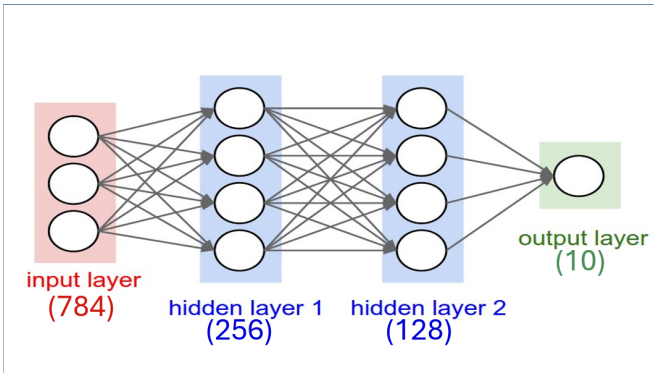


Fig. 1: Base model Architecture

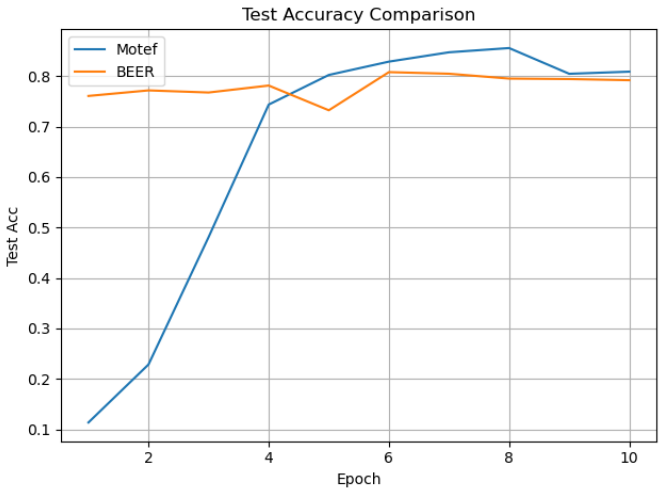


Fig. 2: Baseline results

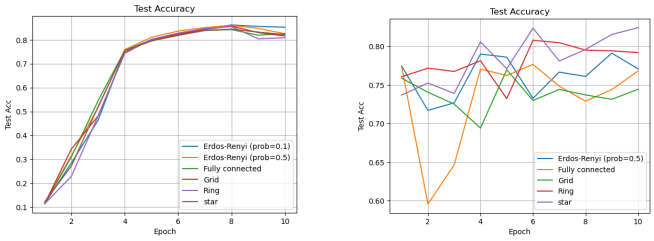
To investigate the impact of network topology, we tested various configurations including ring, star, grid, fully connected, and two versions of Erdős-Rényi graphs with edge probabilities of 0.1 and 0.5. Figure 3 shows model accuracy for the MNIST task using both methods. This was also tested keeping the network size to 4. The results are also summarised in Table I. All the algorithm parameters ($\gamma, \eta, \lambda,$ compression ratio) were kept the same as the baseline experiments.

We also analyzed the effects of using different compression schemes as they are integral to these algorithms. For these experiments shown in Table II, we utilized well-known schemes such as top-k[4], random-k, and 8-bit quantization. Ring topology was used for these experiments. The world size and algorithm hyperparams were kept the same as usual. In Table II, the first entries (before /) are for BEER and the ones after are for MoTEF.

The momentum parameter of MoTEF (λ) is an important addition for stabilising the algorithm's convergence. Hence, we have evaluated MoTEF for different values of λ. We have also

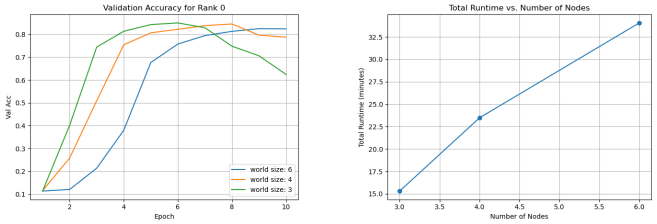
Topology	MoTEF		BEER	
	Total Time (m)	Val Accuracy	Total Time (m)	Val Accuracy
Erdos-renyi (p=0.1)	**6.88**	**85.35**%	-	-
Erdos-renyi (p=0.5)	5.92	82.56%	13.50	77.07%
Fully-connected	7.81	81.62%	17.31	76.80%
Grid	6.14	82.69%	11.73	74.43%
Star	5.73	80.19%	12.50	79.18%
Ring	6.72	80.87%	**12.22**	**82.41**%

TABLE I: MoTEF and BEER Performance across different topologies



(a) (b)

Fig. 3: Topology Analysis: (a) MOTEF Topology, (b) BEER Topology



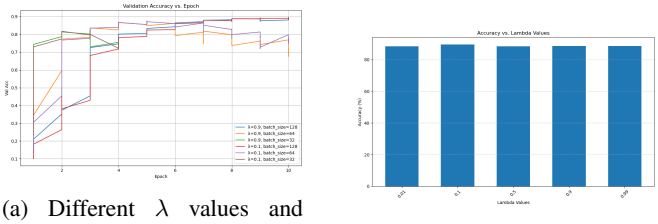
(a) Accuracy vs model size (b) Train times vs model size

Fig. 5: Performance metrics of different graph sizes: (a) Accuracy, (b) Training Times

Compressor	Total Time (m)	Val Accuracy
Random-k	20.52/13.07	89.61/86.22%
Top-k	20.01/15.05	89.42/86.20%
8-bit (quantized)	20.31/14.01	89.65/86.35%

TABLE II: Val accuracy and total training times (MoTEF/BEER) for different compressors

experimented with different batch sizes in parallel as shown in Figure 4a. While keeping batch-size fixed to 128/64/32, we also ran experiments with different values of λ. Figure 4b shows this.



(a) Different λ values and batch sizes (b) Val Accuracy vs λ

Fig. 4: MoTEF performance across (a)λ values and batch sizes; (b)λ values

To test the scalability of the algorithms, we also tested with different different world sizes as well. For this experiment, we used networks consisting of 3,4 and 6 nodes with the standard ring topology, while keeping all parameters the same.

IV. Discussion

Figure 2 shows MoTEF[2] is more stable and monotonic in convergence than BEER[1]. This is confirmed by experiments with different topologies (Figure 3 and Table I). We also see that BEER becomes more unstable with increased connectivity, performing best with ring topology and worst when fully-connected. MoTEF, with its momentum tracking, remains stable across topologies. Table II indicates MoTEF is agnostic to compression schemes for convergence. While BEER converges fast, it diverges quickly after converging and falls into the gradient explosion trap, while MoTEF still keeps improving resulting in better accuracy. BEER is also a lot more sensitive to parameter tuning (η), often resulting in an exploding gradients problem. From Figure 4a, we can see that smaller batch sizes do not constitute a big problem only degrading accuracy marginally, and that MoTEF performs more optimally with smaller $\lambda = 0.1$. This is contrary to our first intuition that the gradient smoothing parameter should be relatively small such that the algorithm still has enough flexibility to improve with each batch ($\lambda = 0.9$). In 4b we can see the direct influence of lambda with a fixed batch-size of 128. It shows that the model performs almost equivalently for all values of λ however best with a small $\lambda(0.1)$, however not too small (0.01). We also tested a simple 2-layer MLP with various world sizes (3, 4, and 6) as shown in Figure 4. Contrary to expectations, increasing the number of nodes did not reduce training times. This may be due to our local CPUs' limited multi-threading capability, which led to resource exhaustion. The experiment could yield better results with multiple GPUs.

V. Summary

This study compares BEER and MoTEF in decentralized federated learning. We have found that MoTEF shows greater stability across topologies while BEER is highly unstable in highly connected networks. The used compression method did not differ much for both algorithms. From all this, we can say that MoTEF proves more robust overall, while BEER requires careful handling. Future work should explore optimizations on advanced hardware.

REFERENCES

[1] H. Zhao, B. Li, Z. Li, P. Richtárik, and Y. Chi, "BEER: Fast $o(1/t)$ rate for decentralized nonconvex optimization with communication compression," in *Advances in Neural Information Processing Systems*, A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, Eds., 2022. [Online]. Available: https://openreview.net/forum?id=I47eFCKa1f3

[2] R. Islamov, Y. Gao, and S. U. Stich, "Near optimal decentralized optimization with compression and momentum tracking," 2024. [Online]. Available: https://arxiv.org/abs/2405.20114

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[4] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified sgd with memory," 2018. [Online]. Available: https://arxiv.org/abs/1809.07599