Distributed Gradient Descent for Dense Data

RESEARCH SUPERVISOR

DR .S. SATHYANARAYANAN

ASSISTANT PROFESSOR
SCHOOL OF ENGINEERING, CSE

Presented by, Rithu G
Reg.no: 24011501015

SHIV NADAR
UNIVERSITY
CHENNAI

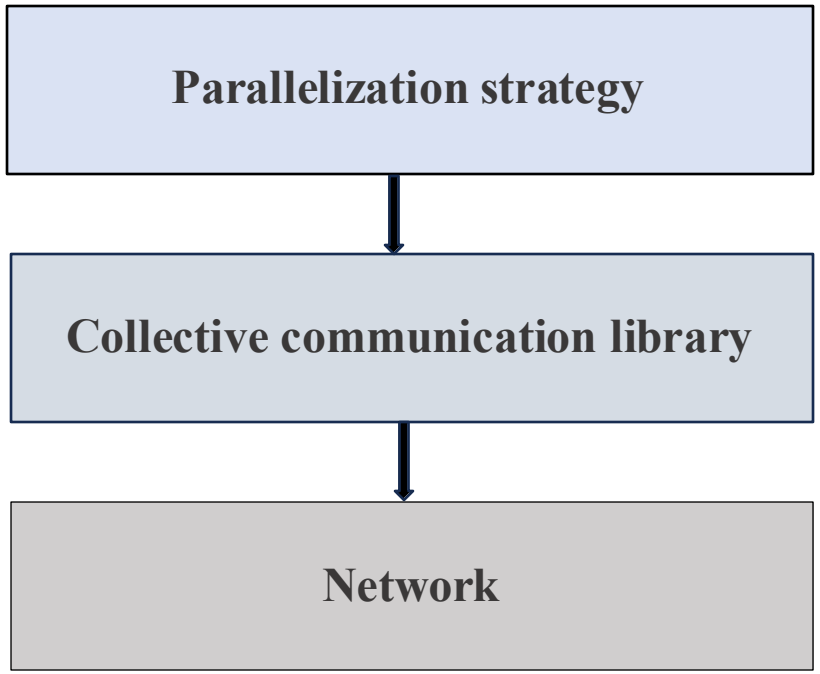
Objective

- Design a Distributed Gradient Descent algorithm tailored for a dense dataset.
- Analyze the data for logics that can be integrated to optimize the algorithm.

Problem statement

- Training deep neural networks on dense, high-dimensional data is computationally and communication-intensive.
- This project explores distributed gradient descent to reduce communication overhead.
- Distributed Gradient Descent does not offer the same efficiency as Vanilla Gradient Descent
- Thus, understanding the stock market dataset and integrate its logic into our algorithm is essential
- The stock market domain is chosen for implementation and evaluation due to its dense, structured data.

Architecture of distributed training system



Parallelization Strategies

Data Parallelism

- The dataset (OHLCV) is split across multiple processing units.
- Replica of our model is stored in each device.
- Gradients are computed in parallel and aggregated after each batch using All Reduce and updates parameters.
- Speeds up training and scales well for large, dense datasets.
- Enables efficient use of GPU or distributed resources.

Dataset

1	Symbol	Series	Prev Close	Open	High	Low	Last	Close	Volume
2	MUNDRAPC	EQ	440	770	1050	770	959	962.9	27294366
3	MUNDRAPC	EQ	962.9	984	990	874	885	893.9	4581338
4	MUNDRAPC	EQ	893.9	909	914.75	841	887	884.2	5124121
5	MUNDRAPC	EQ	884.2	890	958	890	929	921.55	4609762
6	MUNDRAPC	EQ	921.55	939.75	995	922	980	969.3	2977470
7	MUNDRAPC	EQ	969.3	985	1056	976	1049	1041.45	4849250
8	MUNDRAPC	EQ	1041.45	1061	1099.5	1050	1084	1082.45	2848209
9	MUNDRAPC	EQ	1082.45	1089	1109.7	1051	1090.1	1081.3	1749516
10	MUNDRAPC	EQ	1081.3	1100	1134	1078	1100	1102.4	2247904

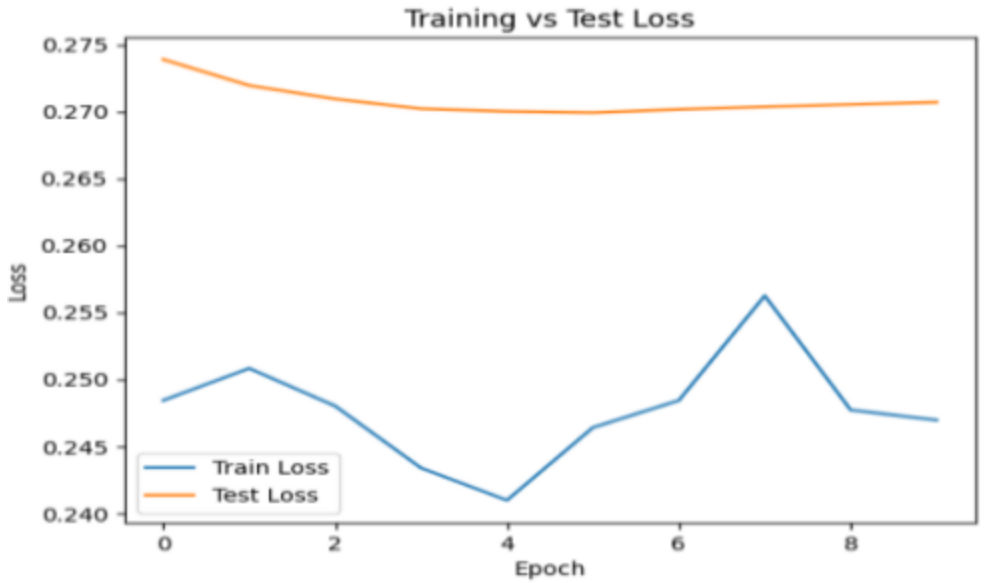
Architecture

```python
import torch.nn as nn

class StockNN(nn.Module):
    def __init__(self, input_dim):
        super(StockNN, self).__init__()
        self.layers = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.ReLU(),
            nn.Dropout(0.3),

            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Dropout(0.2),

            nn.Linear(32, 1)
        )

    def forward(self, x):
        return self.layers(x)

```

Approach

Base Approach

- This code represents a baseline neural network, trained with a simple architecture, Adam optimizer, and MSE loss.
- This setup serves as a starting point to evaluate the model's ability to fit data before applying improvements.

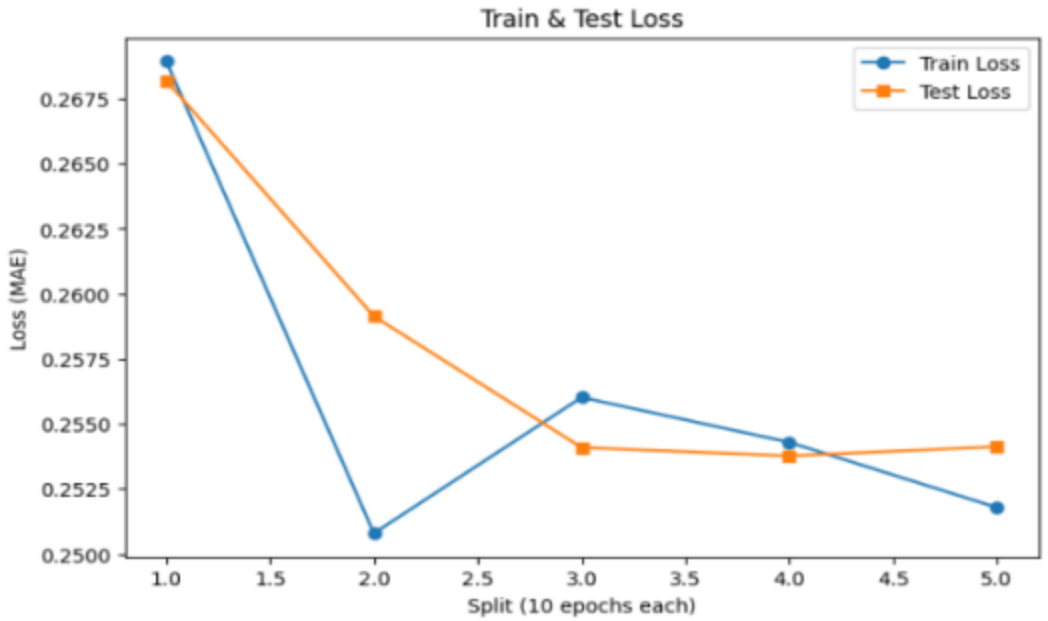


SHIV NADAR UNIVERSITY CHENNAI

Approach

Epoch Split Approach

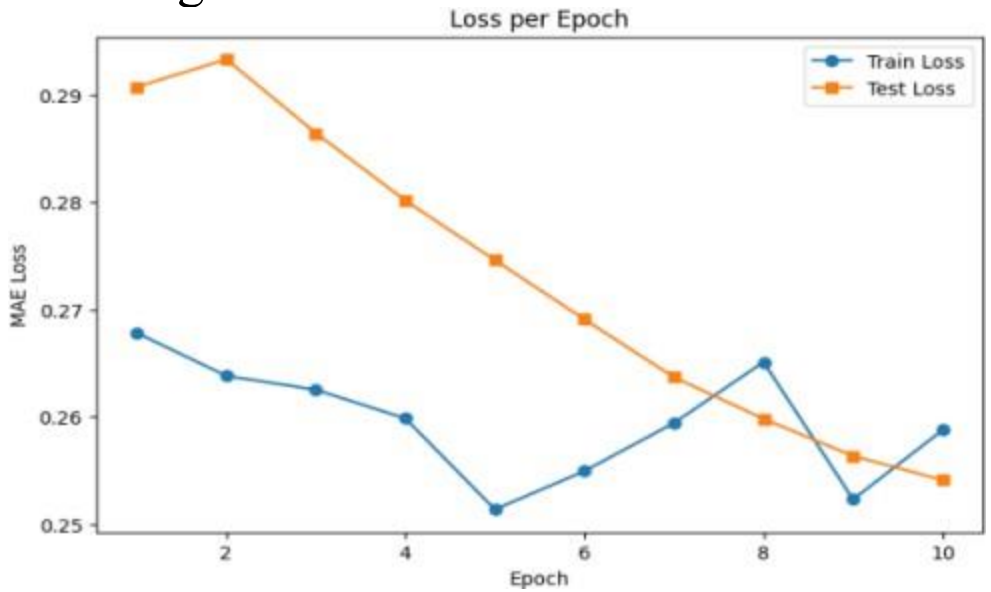
- For each epoch, you collected the minimum training loss and minimum test loss observed across all nodes.
- Both training and test losses decrease steadily from Epoch 1 to Epoch 2, showing effective learning without signs of overfitting.



Approach

Data Split Approach

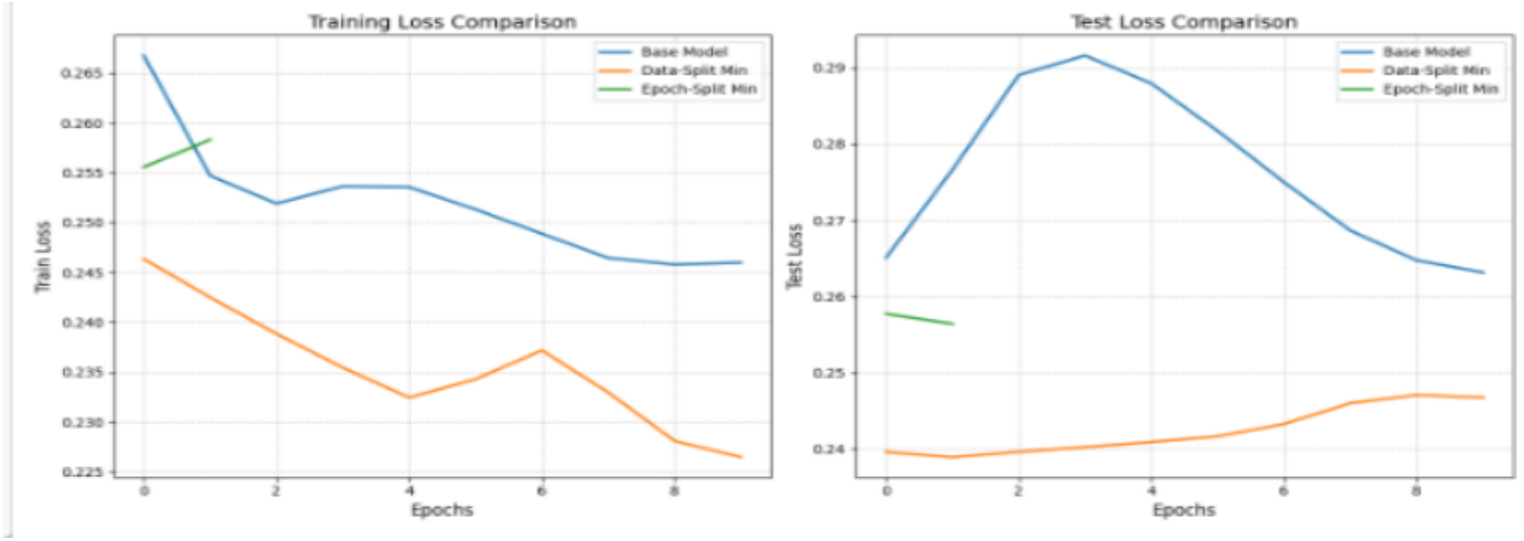
- The dataset is divided into 5 equal subsets, and the model is trained separately on each subset for 10 epochs.
- Learning rate scheduler adjusts training by halving the rate every 5 epochs for stable convergence.




```
Total rows in dataset: 110238
Rows per subset: 22047
Subset 1: rows 0 to 22046, total = 22047
Subset 2: rows 22047 to 44093, total = 22047
Subset 3: rows 44094 to 66140, total = 22047
Subset 4: rows 66141 to 88187, total = 22047
Subset 5: rows 88188 to 110234, total = 22047
```

Comparison:

- Data diversity matters**:** The Data Split Model shows that exposing the model to varied subsets accelerates convergence and slightly improves overall performance.
- Epoch-wise splitting boosts efficiency**:** The Epoch Split Model achieves the lowest training loss consistently, suggesting that structured training schedules can significantly enhance learning efficiency.



SHIV NADAR UNIVERSITY CHENNAI

Particle Swarm Optimization

- N particles are initialized with random positions and velocities.
- The fitness function is calculated for each particle to evaluate solution quality.
- Particle positions and velocities are updated using the standard PSO update equations.
- This process is repeated for X iterations until the swarm converges or a stopping condition is reached.

$$v_{i(t+1)} = \omega\, v_{i(t)} + \alpha\left(pbest_i - x_{i(t)}\right) + \beta\left(gbest - x_{i(t)}\right)$$

$$x_{i,t+1} = x_{i,t} + v_{i,t+1}$$

$v\_i(t)$=*Velocity of particle i at iteration t*
$x\_i(t)$ =*Current position*
ω=*Weightage of previous velocity*
α=*Exploitation Weightage*
β=*Exploration Weightage*
$pbest\_i$= *personal best position of particle i*
gbest= global best position among all particles

PSO With Gradients

- Velocity is nudged with gradient information in GBPSO.
- Gradient points towards the steepest descent.

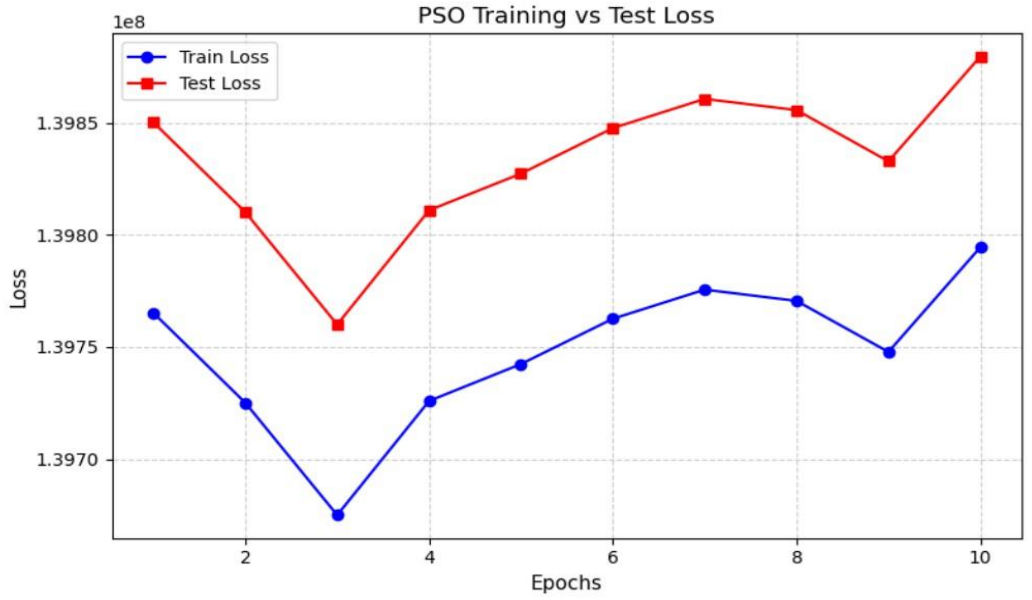
$$v_{i(t+1)} = \omega\, v_{i(t)} + \alpha\left(pbest_i - x_{i(t)}\right) + \beta\left(gbest - x_{i(t)}\right) - \eta\, \nabla f(x_i(t))$$

$$\eta \rightarrow Learning\ Rate$$

PSO Approaches

Approach1:
PSO (One Particle per Node)

- Each particle = one full set of NN weights.
- Evaluate MSE loss for each particle.
- Update positions and velocities using PSO.
- Repeat for multiple iterations.

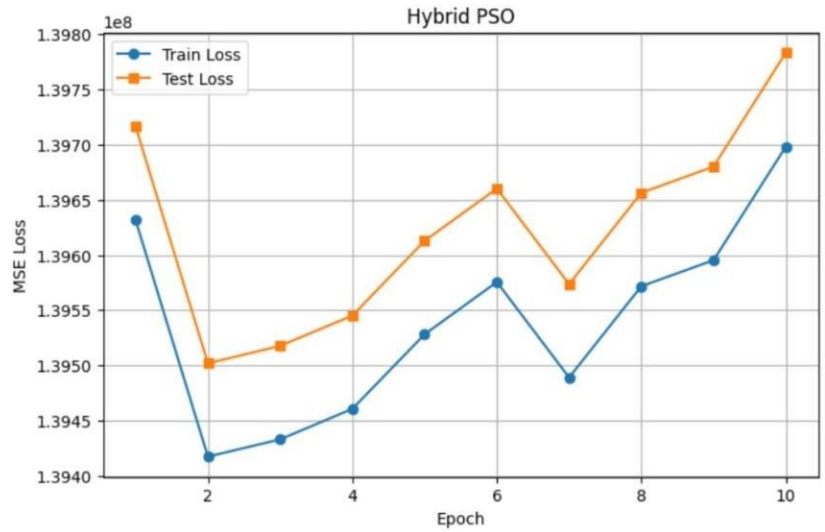


SHIV NADAR UNIVERSITY CHENNAI

Approach 2

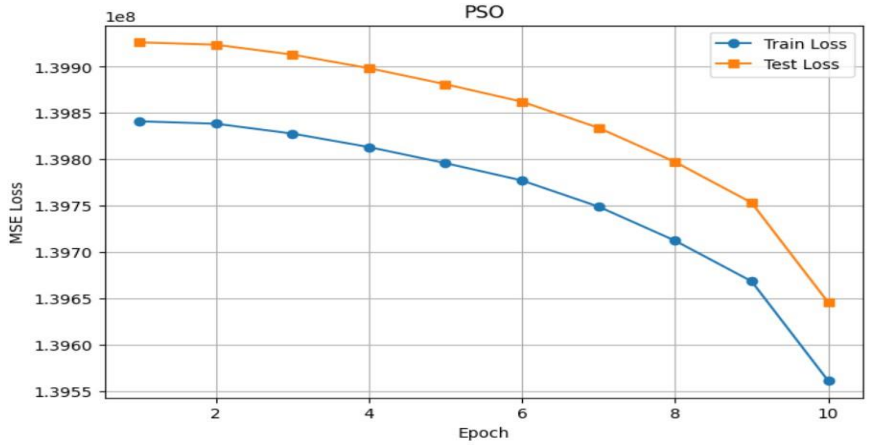
Hybrid PSO (One Particle per Node):

- Each node = one particle.
- Train locally for 10 epochs.
- after local training, perform PSO update to align global best.



Approach 3
Hybrid PSO (Multiple Particles per Node):

- Each node contains multiple particles.
- Run local PSO + short local gradient refinement.
- Share global best across all nodes after each epoch.



SHIV NADAR UNIVERSITY CHENNAI

CONCLUSION

- Built and trained a baseline neural network
- Completed three different training approaches:
  - Base Model Approach
  - Data Split Approach
  - Epoch Split Approach
- Analyzed how each approach learns from the dense OHLCV dataset.
- Compared performance trends across all three methods.
- Implemented and analyzed multiple PSO-based optimization methods.

FUTURE WORK

•Implement a master–slave architecture where the master coordinates and slaves perform parallel optimization.
•Distribute data across slave nodes and let each slave run PSO iterations independently.
•After completing local iterations, each slave sends its current local best (pbest) to the master.
•The master collects all local best solutions and determines the global best among them.
•The master applies the Adam optimizer to further optimize the global best solution for improved convergence.
•The updated global best is broadcast back to all slave nodes for futher optimization.
•This process repeats until the termination condition is reached.
•Dataset characteristics and domain logic are used to tune PSO parameters on slave nodes for better performance.

References

1 Moreno-Alvarez, S., Paoletti, M.E., Cavallaro, G. and Haut, J.M., 2023. Enhancing distributed neural network training through node-based communications. *IEEE transactions on neural networks and learning systems*.

2 Wei, Y., Hu, T., Liang, C. and Cui, Y., 2024. Communication optimization for distributed training: architecture, advances, and opportunities. *IEEE Network*.

3 Bazizi, T., Maghenem, M., Frasca, P., Loria, A. and Panteley, E., 2025. On the Perturbed Projection-Based Distributed Gradient-Descent Algorithm: A Fully-Distributed Adaptive Redesign. *arXiv preprint arXiv:2509.03443*.

4 Yu, Z., Fan, J., Shi, Z. and Zhou, D.X., 2024. Distributed gradient descent for functional learning. *IEEE Transactions on Information Theory*.

5 Wang, S., Zheng, H., Wen, X. and Fu, S., 2024. Distributed high-performance computing methods for accelerating deep learning training. *Journal of Knowledge Learning and Science Technology ISSN: 2959-6386 (online)*, *3*(3), pp.108-126.

6 Cavalca, D.L. and Fernandes, R.A., 2018, July. Gradient-based mechanism for PSO algorithm: A comparative study on numerical benchmarks. In 2018 IEEE Congress on Evolutionary Computation (CEC) (pp. 1-7). IEEE.

7 7]Ruder, S., 2016. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.