Weekly Report (April 21,2025 - April 27,2025)

1st Xiaosong Yuan
AISIG
Shanghai Jiao Tong University
Shanghai, China
yuanxiaosong1999@gmail.com

***Abstract*—This report documents my learning progress during April 21-27, 2025. It covers three main aspects: (1) Completion of three chapters from the deep learning textbook by Mu Li, focusing on computational performance, computer vision, and recurrent neural networks; (2) A detailed study of the PartialLoading paper about parameter-sharing edge inference optimization; (3) Identification of current challenges and specific improvement plans.**

***Index Terms*—deep learning, edge computing, model parallelism, parameter sharing, RNN, computer vision**

I. Knowledge Learning

This week I mainly completed the study of three chapters from Mu Li's "Dive into Deep Learning": computational performance, computer vision, and recurrent neural networks. Only one chapter on attention mechanisms remains, which I plan to finish next week.

The computational performance chapter primarily discussed the characteristics and application scenarios of different parallel strategies, such as data parallelism, model parallelism (including inter-layer and operator parallelism), and pipeline parallelism. It then introduced single-machine multi-GPU training and distributed GPU cluster training based on data parallelism, covering relevant concepts and workflows.

The computer vision chapter introduced classical tasks in the visual domain along with their corresponding processing methods and model implementations, including image augmentation techniques, transfer learning (fine-tuning techniques, which form the basis for problems discussed in the PartialLoading paper), object detection tasks, semantic segmentation, and style transfer.

Through the recurrent neural networks chapter, I began learning about NLP. Unlike the visual domain, deep neural networks in NLP are all autoregressive sequence models. This chapter lays the foundation for understanding and mastering LLM technologies later.

Through this week's study, I have gained a relatively clear understanding of neural networks and basic knowledge about the technical background involved in related papers. The next step is to continue learning other knowledge while emphasizing review and consolidation of deep learning.

II. Paper Study

PartialLoading: User Scheduling and Bandwidth Allocation for Parameter-sharing Edge Inference

The authors argue that the latency of edge devices consists of two parts: model loading latency and inference computation latency. As shown in the figure below, model loading latency accounts for 90% of the total, so reducing model loading latency can significantly decrease the overall edge device latency.

1. **What is model loading latency?**

In edge computing scenarios, **model loading latency** mainly refers to the time required to load the AI model from storage devices (such as hard disks or memory) **into the GPU/TPU memory**.

Since model loading latency refers to the time to load model parameters into memory (or GPU memory), if part of the model weights (parameters) are already in memory, there's no need for swapping, which can significantly reduce model loading latency. Therefore, the authors proposed using parameter-sharing methods, keeping some parameters in memory, and directly reusing them for the next model.

2. **When can parameter sharing occur?**

The **parameter sharing** in this paper mainly occurs between **different downstream models deployed on the same edge server**:

- **Downstream models derived from the same pre-trained model**:
  - For example, multiple task-specific models (e.g., Task 1 for cat/dog classification, Task 2 for vehicle classification) obtained through **layer freezing** or **parameter-efficient fine-tuning (such as LoRA)** based on a ResNet-50 pre-trained model.
  - The **low-level parameters (e.g., convolutional layers)** of these downstream models directly share weights from the pre-trained model (as shown in Figure 2).
- **Structurally similar heterogeneous models**:
  - Different architecture models may have functionally similar layers (e.g., shallow feature extraction layers in CNNs), enabling parameter sharing through alignment.

The authors designed PartialLoading to maximize parameter sharing, thereby reducing latency and maximizing model throughput.

3. **How does PartialLoading combine user scheduling and bandwidth allocation to maximize throughput?**

- **User scheduling**: Determines the optimal user scheduling order through DP algorithm or greedy algorithm.
  Concrete applications
  Application of Dynamic Programming (DP) Algorithm

Applicable scenarios

For the **"bottom-layer sharing"** special case (where models only share parameters at the bottom layers, like ResNet fine-tuned models in Figure 2):

- **Problem characteristics**: Shared parameters are concentrated at the model bottom, and scheduling order impact can be structurally decomposed.
- **DP design approach**:
  - **State definition**: Uses "already loaded shared layers" and "remaining users to schedule" as states
  - **State transition**: Each time select a user whose model shares the most layers with currently loaded models
  - **Optimal substructure**: Global optimal solution can be derived from optimal solutions of subproblems (scheduling subsets of users)
- **Complexity**: Polynomial time (e.g., O(n^2)), suitable for real-time edge server computation.

Example

Assume there are 3 tasks (Model A, B, C) sharing bottom-layer parameter blocks:

DP will prioritize scheduling models sharing more layers (e.g., A→B→C) to avoid repeated loading of shared parts.

Application of Greedy Algorithm

Applicable scenarios

For **general cases** (shared parameter blocks appear at arbitrary layers, like middle CNN layers or LLM attention modules):

- **Problem characteristics**: Irregular parameter sharing locations make DP state space explode (cannot be efficiently decomposed).
- **Greedy strategy**:
  - **Local optimization at each step**: Always selects the unscheduled model sharing the most parameters with currently loaded models
  - **Heuristic rule**: Prioritizes reusing the largest possible parameter blocks to reduce immediate loading time
- **Advantages**: Computationally efficient (e.g., O($n \log n$)), suitable for dynamic edge environments
- **Cost**: May yield suboptimal solutions, but experiments show actual performance approaches optimal

Example

If Model A shares 50% parameters with currently loaded models while Model B shares 30%, prioritize scheduling A.

Assume an edge server needs to process 3 tasks:

- **Model A** (Task 1): Shares bottom-layer parameter block α
- **Model B** (Task 2): Shares α + middle-layer parameter block β
- **Model C** (Task 3): No shared parameters

Traditional scheduling: Random order (e.g., A→C→B) → Each time loads full parameters, total loading time = T(A) + T(C) + T(B).

PartialLoading scheduling: A→B→C → When loading B, only needs to load non-shared parts (β), saving loading time for α.

- **Bandwidth allocation problem**

  Objective: After determining the scheduling order, allocate spectrum bandwidth to each user to ensure their data transmission matches model loading time. Has **closed-form optimal solution** (can be directly solved via convex optimization).

III. Current Challenges and Next Steps

1) **Insufficient paper reading**: Need to further study the four papers from next week while reading more related papers to deepen understanding. **Lack comprehensive survey-level understanding of current research field.**
2) **Foundational knowledge learning progresses slower than expected**: Didn't complete the goal of finishing deep learning this week. About 3 more days are needed next week, while weak points in relevant knowledge still require further consolidation.
3) **Hands-on practice not yet started** for the `edge-llm-serving` project. Need to make progress on this project next week.