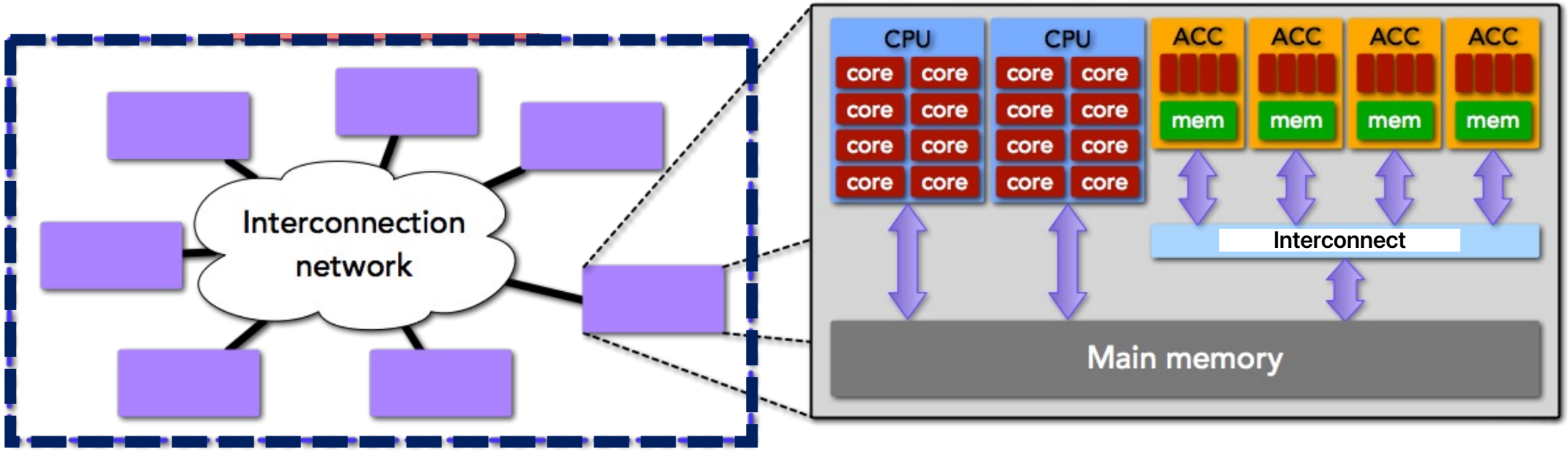
Lecture 22 – Parallelism (AI Models)

References:
ZeRO: Memory Optimizations Toward Training Trillion Parameter Models (https://arxiv.org/abs/1910.02054)
ZeRO-Offload: Democratizing Billion-Scale Model Training (https://arxiv.org/abs/2101.06840)
ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning (https://arxiv.org/abs/2104.07857)

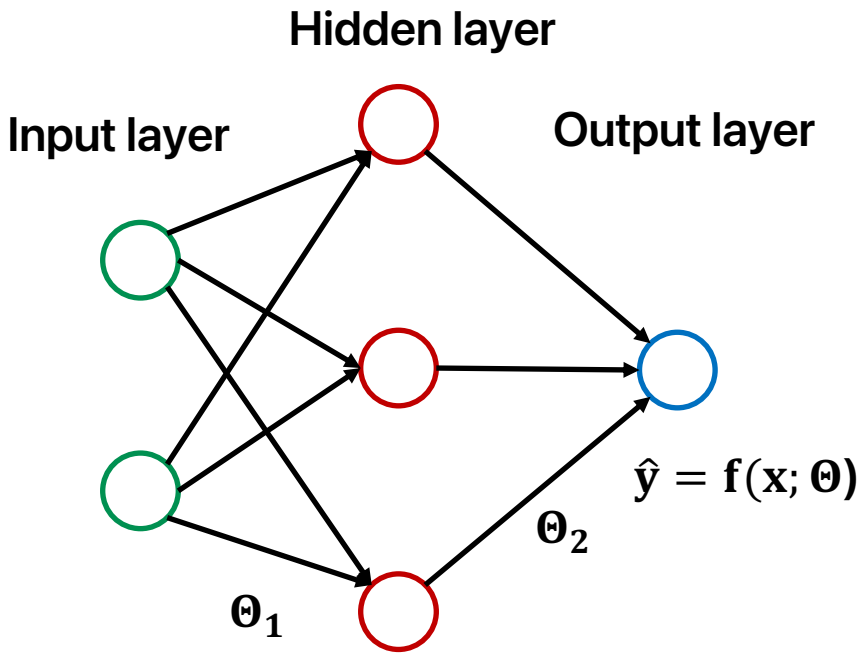
Jaejin Lee
Dept. of Computer Science and Engineering, College of Engineering
Dept. of Data Science, Graduate School of Data Science
Seoul National University
http://aces.snu.ac.kr

CHAMP
SNU THUNDER Research Group

Target Architectures

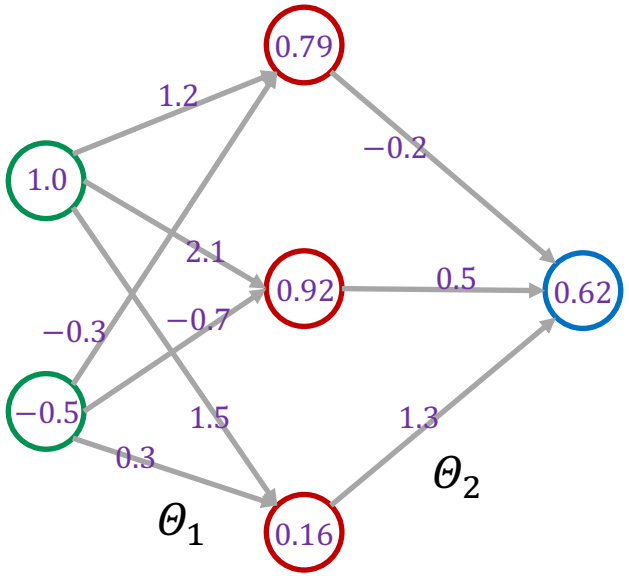


AI Models


Hidden layer
Input layer
Output layer
$\hat{y} = f(x; \Theta)$
Θ_2
Θ_1

Forward Propagation

- $\boldsymbol{x}_i = \begin{bmatrix} 1.0 \\ -0.5 \end{bmatrix}, \Theta_1 = \begin{bmatrix} 1.2 & 2.1 & 1.5 \\ -0.3 & -0.7 & 0.3 \end{bmatrix}, \Theta_2 = \begin{bmatrix} -0.2 \\ 0.5 \\ 1.3 \end{bmatrix}$
- $\Theta_1^T \boldsymbol{x}_i = \begin{bmatrix} 1.2 & -0.3 \\ 2.1 & -0.7 \\ -1.5 & 0.3 \end{bmatrix} \begin{bmatrix} 1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 1.35 \\ 2.45 \\ -1.65 \end{bmatrix}$
- $\boldsymbol{h}_1 = \sigma(\Theta_1^T \boldsymbol{x}_i) = \begin{bmatrix} \sigma(1.35) \\ \sigma(2.45) \\ \sigma(-1.65) \end{bmatrix} = \begin{bmatrix} 0.79 \\ 0.92 \\ 0.16 \end{bmatrix}$
- $\Theta_2^T \boldsymbol{h}_1 = \begin{bmatrix} -0.2 & 0.5 & 1.3 \end{bmatrix} \begin{bmatrix} 0.79 \\ 0.92 \\ 0.16 \end{bmatrix} = 0.51$
- $\hat{y}_i = \sigma(\Theta_2^T \boldsymbol{h}_1) = \sigma(0.51) = 0.62$



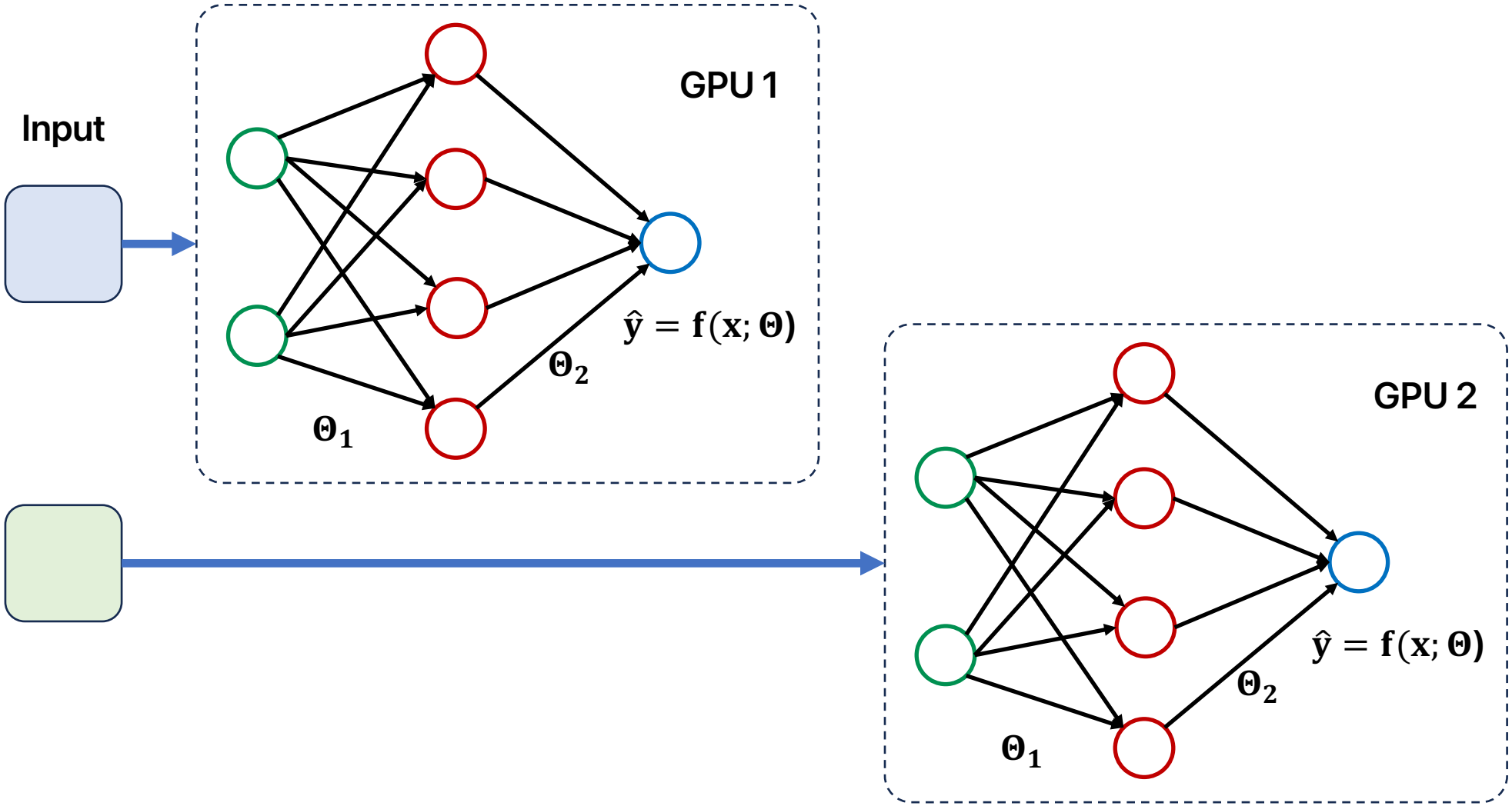
Backward Propagation

- Find parameters that minimize the error
  - $\text{argmin}_\Theta \frac{1}{N} \sum_{i=1}^{N} E(f(\boldsymbol{x}_i; \Theta), \mathbf{y}_i)$
  - $E(\cdot,\cdot)$ is a loss function, e.g., mean square error (MSE) or cross entropy
- Parameter updates
  - $w_{ji}^{t+1} \longleftarrow w_{ji}^{t} - \eta \frac{\partial E}{\partial w_{ji}^{t}}$
  - To make the model converges better, we have different optimizers
    - E.g., Adam optimizer

Memory Consumption by AI Models

- The majority of large language models' memory consumption
- Model states
  - Optimizer states
    - E.g., momentum and variances in Adam
  - Gradients
  - Parameters
- Others
  - Activations
  - Temporary buffers
  - Unusable fragmented memory spaces

Data Parallelism on AI Models

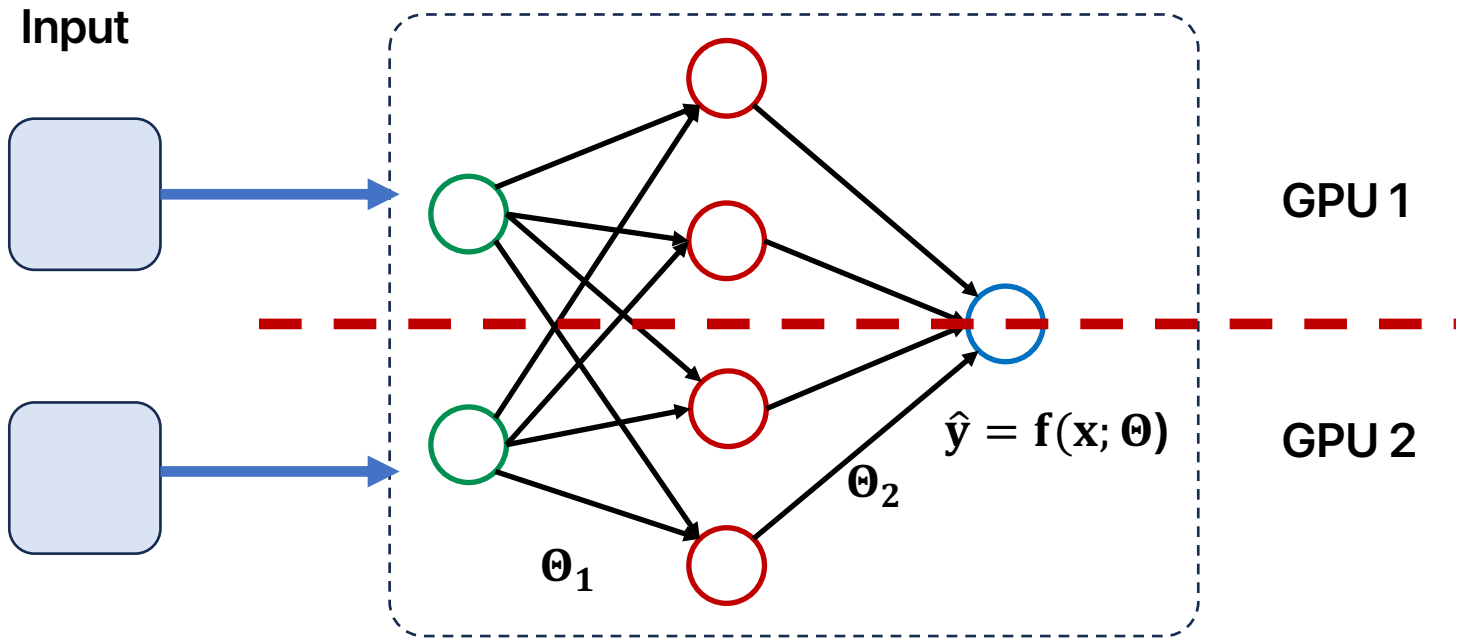


Data Parallelism on AI Models (cont'd)

- A.k.a. Distributed Data Parallelism (DDP)
- Procedure
  - The dataset is divided into mini-batches
  - Each mini-batch is divided into micro-batches, and each micro-batch is assigned to a different GPU
  - Each GPU independently computes the gradients for its assigned micro-batch using its own copied model
  - The gradients from all GPUs are then aggregated (all reduce)
  - The aggregated gradients are used to update the model parameters on all GPUs
- Memory inefficient as model states are stored redundantly across all GPUs

Model Parallelism on AI Models

- A kind of task parallelism



Model Parallelism on AI Models (cont'd)

- The computational workload of a neural network is distributed across multiple GPUs
  - Splits a single neural network across many GPUs, each responsible for computing a portion of the model's operations
- Needs to synchronize model parameters across devices after each training step
  - Updates the weights of the entire model based on the aggregated gradients
- Reduces the granularity of the computation and increases the communication overhead

Tensor Parallelism and Pipeline Parallelism

- Tensor Parallelism (TP)
  - A kind of task parallelism
  - Each tensor is split up into multiple chunks, so instead of having the whole tensor reside on a single GPU, each shard of the tensor resides on its designated GPU
  - During processing, each shard gets processed separately and in parallel on different GPUs, and the results are synchronized at the end of the step
- Pipeline Parallelism (PP)
  - A kind of task parallelism
  - The model is split up layer-wise across multiple GPUs
  - Only one or several layers of the model are placed on a single GPU
  - Each GPU processes in parallel different stages of the pipeline and works on a small chunk of the batch

ZeRO (Zero Redundancy Optimizer)

- Decreases the memory consumption per GPU by removing redundant data in the GPU memory while retaining the training efficiency
- ZeRO-DP removes the memory state redundancies across data-parallel processes by partitioning the model states instead of replicating them
  - It performs sharding of the tensors somewhat similar to TP, except the whole tensor gets reconstructed in time for a forward or backward computation
- ZeRO also supports various offloading techniques to compensate for limited GPU memory
  - There are many different techniques of ZeRO
- Sharded Distributed Data Parallelism (DDP)
  - A.K.A. Fully Sharded Data Parallelism (FSDP)
  - Another name for the foundational ZeRO concept that is used by various other implementations of ZeRO