Introduction to Parallel Computing (CMSC416 / CMSC616)

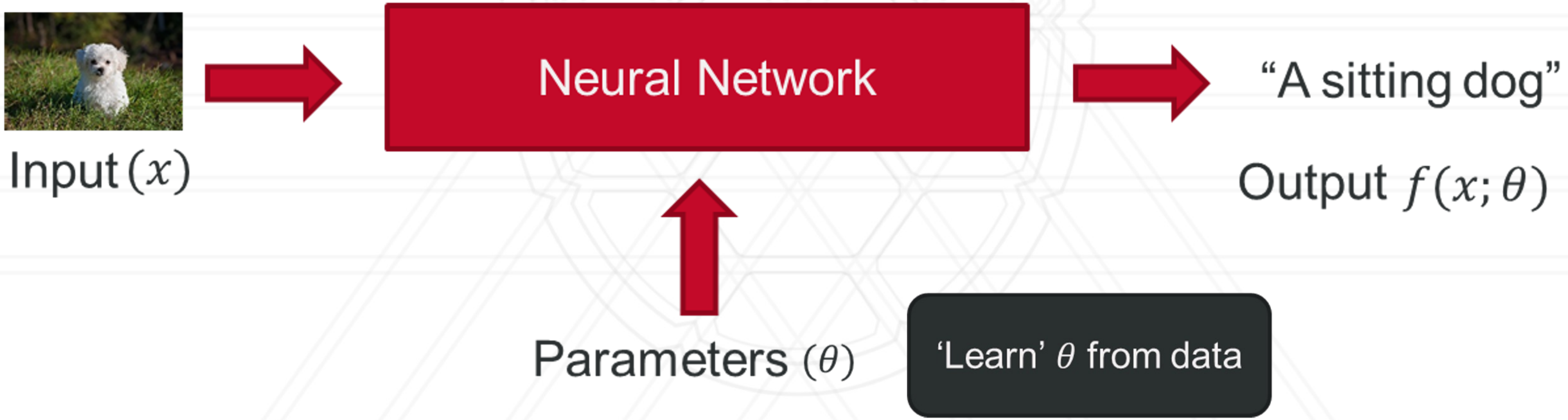
Deep Learning in Parallel

Siddharth Singh, Department of Computer Science

UNIVERSITY OF MARYLAND

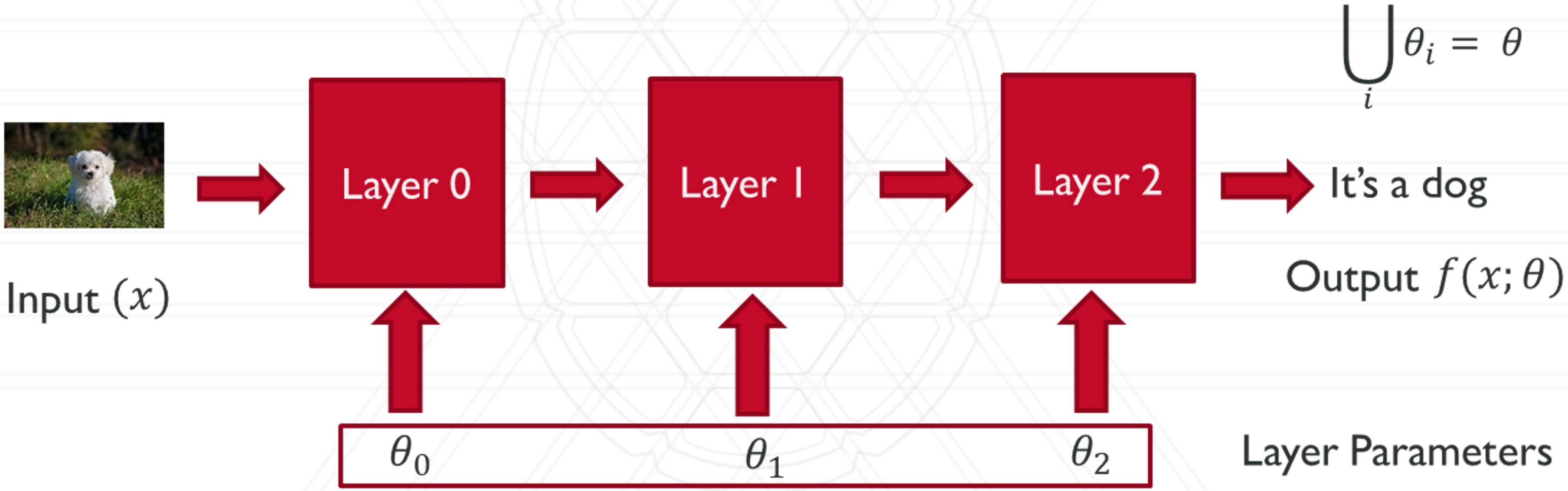
Deep neural networks

- Neural Networks (NN): Parameterized function approximators
- Can work with very high dimensional data (text, videos, audio)



Neural Networks have a Layered Structure

- Computation organized in a sequence of layers with linear dependencies.

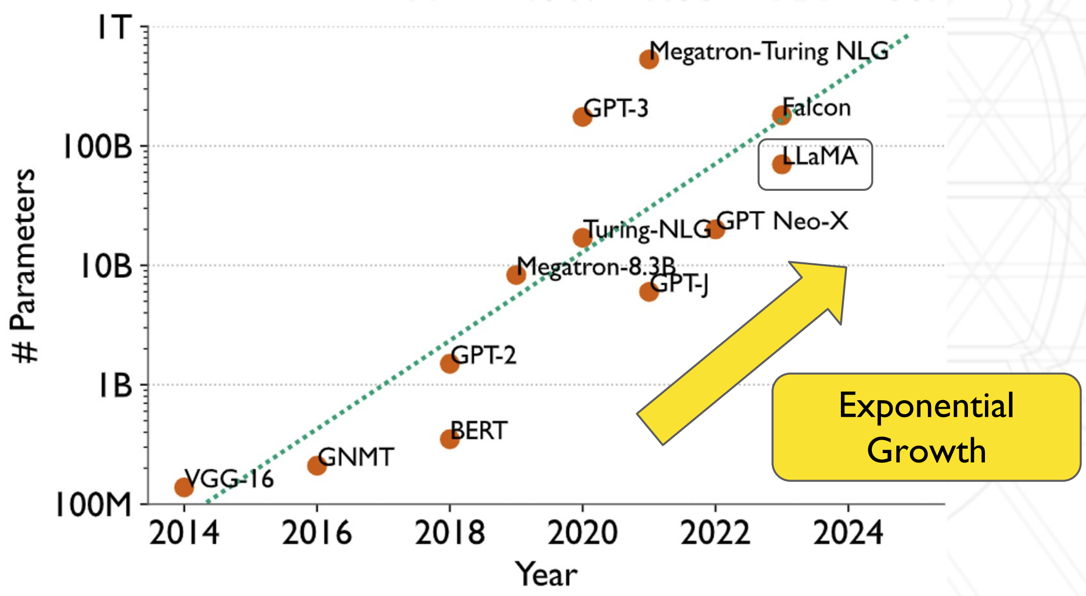


Other definitions

- Learning/training: task of selecting weights that lead to an accurate function
- Loss: a scalar proxy that when minimized leads to higher accuracy
- Gradient descent: process of updating the weights using gradients (derivatives) of the loss weighted by a learning rate
- Batch: Small subsets of the dataset processed iteratively
- Epoch: One pass over all the mini-batches

Why Parallel Deep Learning?

- Parallel Deep Learning - Training on multiple GPUs.



Time to train on a single A100 GPU on Zaratan?

172 years!!

Networks are trained on 1000s of GPUs!

TABLE I

COMPARISON OF RECENT LARGE-SCALE LLM TRAINING STUDIES, COVERING DIVERSE FRAMEWORKS AND HARDWARE. THE TABLE DISPLAYS EACH STUDY'S MAX CHIP SCALE, CORRESPONDING MODEL & BATCH SIZE, PERCENTAGE OF PEAK FLOP/S, ALONG WITH ACTUAL PFLOP/S ACHIEVED.

Study	Framework	Model Size	Batch Size	Hardware	Scale	% Peak Flop/s	Total Pflop/s
FORGE [1]	GPT-NeoX	1.44B	16.8M	AMD MI250X	2,048 GCDs	~29%*	~112.6*
Dash et al. [2]	Megatron-DeepSpeed	1000B	19.7M	AMD MI250X	3,072 GCDs	31.9%†	188.0†
SUPER [3]	LBANN	3B‡	0.5M‡	NVIDIA V100	1,024 GPUs	-	-
KARMA [4]	KARMA	17B	2.0M‡	NVIDIA V100	2,048 GPUs	-	-
Narayan et al. [5]	Megatron-LM	1000B	6.3M	NVIDIA A100	3,072 GPUs	52%	502.0
MT-NLG [6]	Megatron-DeepSpeed	530B	4.0M	NVIDIA A100	3,360 GPUs§	36%	379.7
MegaScale [7]	MegaScale	175B	12.5M	NVIDIA A100	12,288 GPUs	55%	2166.3
Google [8]	Google Cloud TPU Multislice Training	32B	417M	TPUv5e	55,094 TPUs	44.67%	4480.0
This Work	AxoNN	40B	16.8M	NVIDIA A100	4,096 GPUs	49%	620.1
		160B	16.8M	AMD MI250X	16,384 GCDs	26%	815.7

* Estimated from plots in the paper as exact numbers not mentioned

† Calculated from flop/s at lower GPU/GCD count and weak scaling efficiency

‡ Estimated from description in pape ... t men

§ Authors claim training using 4480 ... t scali

Billions of parameters

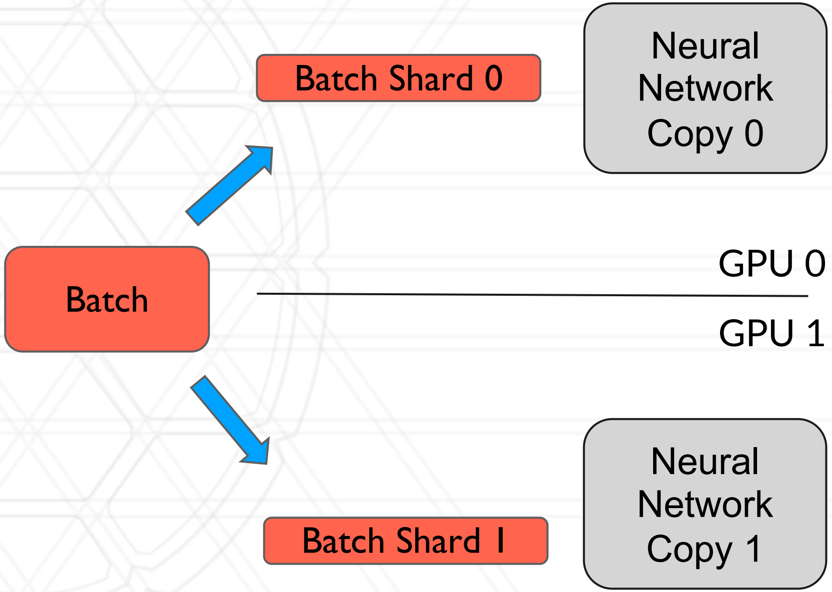
1000s of GPUs

Parallel/distributed training

- Many opportunities for exploiting parallelism
- Iterative process of training (epochs)
- Many iterations per epoch (mini-batches)
- Many layers in DNNs

Data parallelism

- Divide training data among workers (GPUs)
- Each worker has a full copy of the entire NN.
- All reduce operation to synchronize gradients.



Pros and Cons of Data Parallelism

Pros

1. Embarrassingly parallel
2. Easy to implement and use

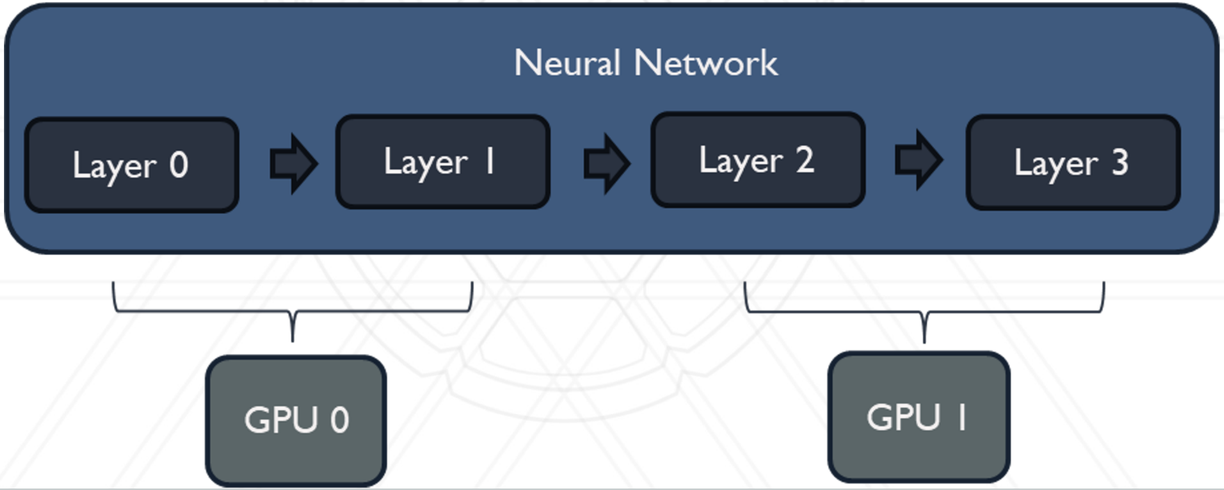
Cons

1. Cannot train models that exceed memory capacity of a single GPU.

How to train models that do not fit on a single GPU?

Inter-layer Parallelism

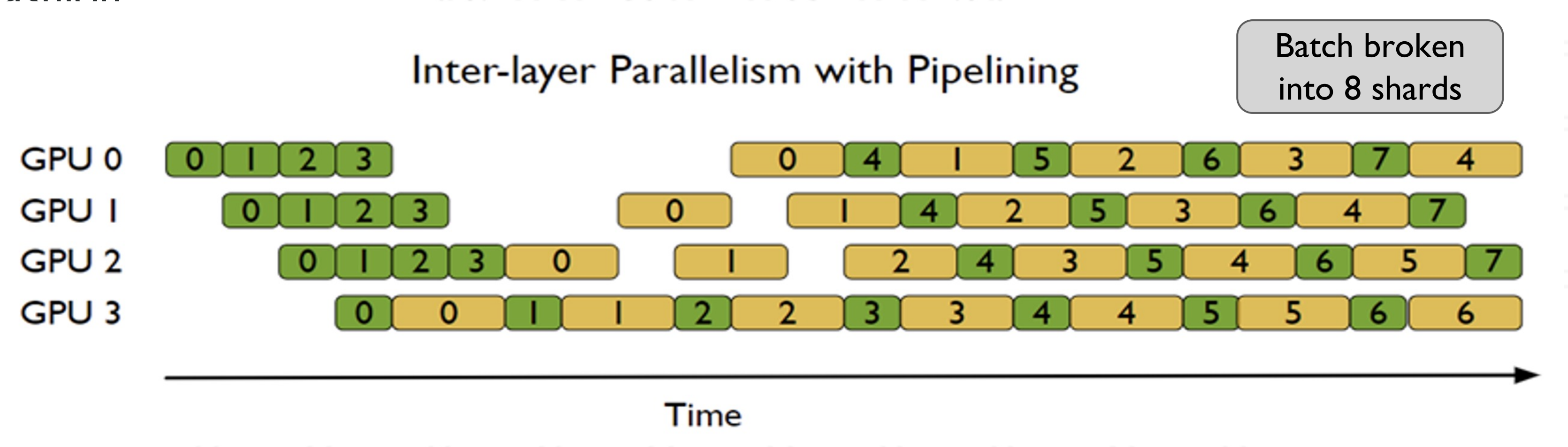
- Distribute entire layers to different processes/GPUs
- Point-to-point communication (activations and gradients) between processes/GPUs managing different layers



Pipelining in Inter-Layer Parallelism

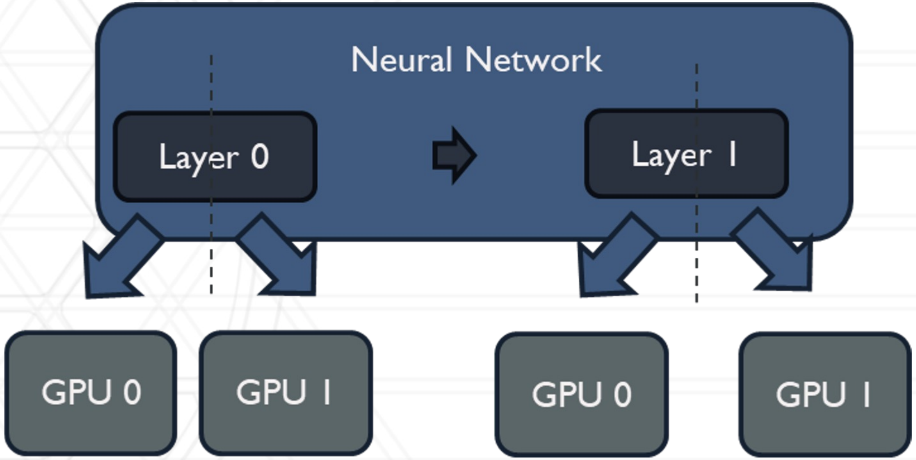
- Layers have sequential dependencies, so only one GPU would be active at a time.

- Break batch into multiple shards (microbatches) and process them in a pipelined fashion



Intra-layer Parallelism

- Divide the work of each individual layer across multiple GPUs.
- Compute intensive layers involve large matrix multiplications.
- Intra-layer parallelism = Parallel Matrix multiplication.



Hybrid parallelism

- Using two or more approaches together in the same parallel framework
- 3D parallelism: use all three
- Popular serial frameworks: pytorch, tensorflow
- Popular parallel frameworks: DDP, FSDP, ZeRO, Megatron-LM

Parallel Deep Learning @ PSSG

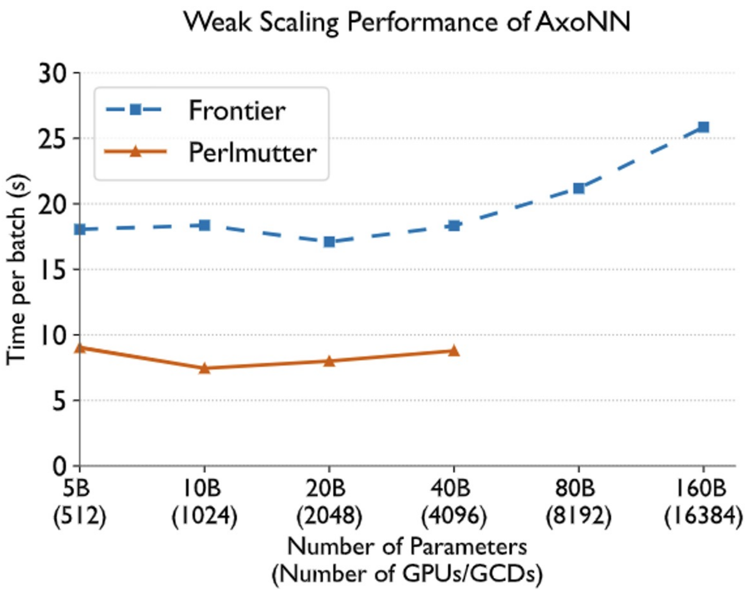


Fig. 5. Weak scaling performance (time per batch or iteration) of AxoNN on Perlmutter and Frontier.

Thrust 1: Designing Communication Efficient Algorithms for Parallel Training on 1000s of GPUs!

Parallel Deep Learning @ PSSG

Thrust 2: Designing User-Friendly Parallel DL Algorithms for Non-HPC Experts

```
net = MyFavModel()
```

Serial Model Declaration

```
with axonn.auto_parallelize:
    net = MyFavModel()
```

Tensor Parallel Model Declaration

Questions?

UNIVERSITY OF MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu