# 10-414/714 – Deep Learning Systems: Algorithms and Implementation

# **Training Large Models**

Fall 2024 J. Zico Kolter and Tianqi Chen (this time) Carnegie Mellon University

# Outline

Techniques for memory saving

Parallel and distributed training

# Outline

#### Techniques for memory saving

Parallel and distributed Training

# Elements of machine learning systems



Bigger dataset requires larger model capacity. Which in turn puts demands on computing devices. The success of machine learning is a combination of all the three elements. Many recent advances requires us to push all three to their limits.

Today we will study two topics:

- How to reduce the memory consumption, so we can fit bigger models into a single device.
- How to scale up the training process

# Recap: GPU memory hierarchy



# Sources of memory consumption

A simplified view of a typical computational graph for training, weights are omitted and implied in the grad steps.



Sources of memory consumption

- Model weights
- Optimizer states
- Intermediate activation values

**Optimizer states** 



# Techniques for memory saving inference only



We only need O(1) memory for computing the final output of a N layer deep network by cycling through two buffers

# Activation memory cost for training



Because the need to keep intermediate value around (checkpoint) for the gradient steps. Training a N-layer neural network would require O(N) memory.

We will use the following simplified view to combine gradient and forward computation



# Checkpointing techniques in AD



- Only checkpoint colored nodes (step 0)
- Recompute the missing intermediate nodes in small segments (step 1, 2)

## Sublinear memory cost



For a *N* layer neural network, if we checkpoint every *K* layers



# Outline

Programming abstractions

Parallel and distributed training

# Parallel training problem

Leverage multiple (GPU) devices that are possibly distributed over several worker nodes to train a model.



# Data parallel training

Loss function

$$\theta \coloneqq \theta - \frac{\alpha}{B} \sum_{i=1}^{B} \nabla_{\theta} \ell(h_{\theta}(x^{(i)}), y^{(i)})$$

Let each worker access  $\frac{B}{K}$  fraction of the minibatch, and run gradient computation then sum up all gradients together.

Every worker runs the same replica of the model

#### Allreduce abstraction

Interface result = allreduce(float buffer[size])

```
Running Example
```

```
Worker 0
```

#### Worker 1

```
comm = communicator.create()
a = [1, 2, 3]
b = comm.allreduce(a, op=sum)
assert b == [2, 2, 4]
comm = communicator.create()
a = [1, 0, 1]
b = comm.allreduce(a, op=sum)
assert b == [2, 2, 4]
```

#### Data parallel training via allreduce

Many replicas of the same graph run in parallel



#### Parameter server abstraction

Interface

ps.push(index, gradient)

ps.pull(index)

Performs weight update on the server(key value store)



# Data parallel training via parameter server

Many replicas of the same graph run in parallel



# Communication computation overlap



Many opportunities to continue computation while sending data over the network

# Model parallel training

Maps parts of the computation graph to workers



# Breaking up the computation for model parallelism



# Breaking up the computation for model parallelism



Partition the graph, put send/recv pairs in the boundary



Partitions tensor data across devices How to feed to the next layer?

### Allgather abstraction

Interface result = allgather(float buffer[size])

Running Example

Worker 0

#### Worker 1

comm = communicator.create()
a = [1, 2]
b = comm.allgather(a)
assert b == [1, 2, 3, 4]
comm = communicator.create()
a = [3, 4]
b = comm.allgather(a)
assert b == [1, 2, 3, 4]



Allgather turns sharded to replicated view

How to avoid allgather?



Start with sharded X, generate partial result(Y1, Y2), then sum them together



A typical tensor parallel pipeline involves two matmuls

# **Parallelization summary**

Model parallel training partition by parts in the computational graph.

Data parallel training partition by data.

In all cases, leverage the opportunities to overlap compute with communication.

# Advanced parallelization methods

There are more ways to parallelize a computational graph.

Some optional reference readings:

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models.

Beyond Data and Model Parallelism for Deep Neural Networks.

GSPMD: General and Scalable Parallelization for ML Computation Graphs

FSDP: Fully Sharded Data Parallel

# Outline

Techniques for memory saving

Parallel and distributed training