Neural Networks

- Neural Networks (NNs): ‘Parameterized’ function approximators
- Can work with very high dimensional data.

\[
\begin{align*}
\text{Input} (x) & \quad \rightarrow \quad \text{Neural Network} \\
& \quad \rightarrow \quad \text{Parameters} (\theta) \quad \text{‘Learn’ } \theta \text{ from data} \\
& \quad \rightarrow \quad \text{Output } f(x; \theta) \quad \text{“A sitting dog”}
\end{align*}
\]

Mostly composed of matrix multiplications.
Stochastic Gradient Descent

- Repeat until loss (L) has been minimized sufficiently:
  - Read in a batch of training data
  - Forward Pass: Calculate output $f(x; \theta)$ and the loss (L) on the batch.
  - Backward Pass: Calculate gradients of the loss wrt the parameters $\left( \frac{\partial L}{\partial \theta} \right)$.
  - Optimizer Step: Use $\frac{\partial L}{\partial \theta}$ to update $\theta$. 
Neural Network Pruning

- Zeroing parameters with small magnitudes permanently mid-training.

- DL pruning algorithms can prune as many as 80-90% of the parameters without affecting model quality.

Validation perplexity for GPT-3 1.3B on Wikitext-103

GPT-3 1.3B pruned to 90% sparsity using [1].
Can we exploit pruning in large models to improve performance of parallel training on multi-GPU clusters?
Sparse matrix multiplication?

- Most compute in a pruned NNs is sparse matrix multiplication.

- Can we use optimized implementations of SpMM?

Instead, we focus on optimizing communication volume in parallel training of NNs.
Background on AxoNN

- In this work we used AxoNN [2] as our parallel DL framework of choice.

- AxoNN implements a hybrid parallel algorithm of data and inter-layer parallelism.

A two layer neural network on 4 GPUs.
Distribution of Compute in AxoNN

Organize GPUs in a 2D grid
Distribution of Compute in AxoNN

Partition layers equally across columns

Layer 0

\[ g_{00} \]

\[ g_{10} \]

Layer 1

\[ g_{01} \]

\[ g_{11} \]

\[ G_{\text{inter}} = 2 \]

Inter-Layer Parallelism
Distribution of Compute in AxoNN

Partition batch equally across rows

Batch Shard 0
Batch Shard 1

Layer 0
Layer 1

$g_{00}$ $g_{01}$
$g_{10}$ $g_{11}$

$G_{\text{data}} = 2$

$G_{\text{inter}} = 2$
Communication in Inter-Layer Parallelism

Point-to-point communication of activations (FW pass) and their gradients (BW pass)

$G_{\text{inter}} = 2$

$G_{\text{data}} = 2$
Communication in Data Parallelism

Batch Shard 0

Batch Shard 1

Layer 0

Layer 1

$g_{00}$

$g_{01}$

$g_{10}$

$g_{11}$

$G_{\text{data}} = 2$

$G_{\text{inter}} = 2$

All-reduce to synchronize gradients after BW pass.
Optimizing Communication in Pruned NNs

• Data Parallelism
  • Communication – All-reduce on gradients.

• Volume $\propto |\theta|$  

• Simple! – Only communicate gradients of unpruned parameter
Optimizing Communication in Pruned NNs

- Inter-Layer Parallelism
  - Communication – P2P comm. of activations and their gradients
- Messages aren’t sparse
- Volume $\propto G_{\text{inter}}$ (proof in paper)
- Decrease $G_{\text{inter}}$ → More layers per GPU

Need to optimize memory consumption by exploiting pruning.
Sparsity Aware Memory Optimization (SAMO)

- Selective compression of model states after pruning.
  - \( \theta \) Parameters
  - \( \frac{\partial L}{\partial \theta} \) Gradients
  - \( S_{\text{opt}} \) Optimizer data

  Do not compress

  - Store in dense with 0s explicitly filled out.
  - Invoke efficient dense CuBLAS kernels for matrix mult.

  Compress

  - Store in a 1D sparse COO format.
  - Common index vector of non-zero elements.
Overheads in SAMO

- Backward pass – Compute gradients with **dense computation kernels and then compress**

![Diagram](https://via.placeholder.com/150)

- **Input** ($x$)
- **Parameters** ($\theta$)
- **Gradients** w.r.t weight ($\frac{\partial L}{\partial \theta}$)
- **Indices of non-zero values**
- **Compressed Gradients**
- **Gradient** w.r.t output ($\frac{\partial L}{\partial f}$)
Sparsity Aware Memory Optimization (SAMO)

- Assuming mixed precision and the Adam Optimizer, we prove that our method saves 66-78% memory for an 80-90% pruned NN.

- Exploit the saved memory to decrease $G_{\text{inter}}$ and decrease point-to-point communication.
Results

Strong Scaling of GPT3-13B on Summit. We prune to 90% sparsity using [1]. We annotate AxoNN+SAMO’s line with its percentage speedup over AxoNN.

31% of peak flop/s
Conclusion

• Developed a novel method that exploits neural network pruning algorithm in large models to improve performance of parallel training.

• Presented Sparsity-Aware Memory Optimization (SAMO) to significantly reduce memory consumption while not sacrificing performance.

• Demonstrated how the memory saved can be used to optimize communication in data and inter-layer parallelism.
Future Work

- Training pruned large language models.
  - Imagine a ChatGPT like model that fits on your laptop.
- Accelerating inference tasks via pruning.
- Experimenting with other forms of parallelism like tensor parallelism.

Bibliography

