Assignment 2 out! Due 10/5 (Monday next week)

Bid on topics, submit group (1 sentences) – Oct 5

Project Proposal (2 pages) – Oct 16

Introduction
Related Work
Timeline (with eval plan)
EMPIRICAL RISK MINIMIZATION

\[
\min_{w \in \mathbb{R}^{d}} \sum_{i=1}^{N} f(w, z_i) + P(w)
\]

Supervised learning and labels

Given training data

Fit a model

Function

Data (Examples)

Model

Regularization
DEEP LEARNING

ResNet18

- Convolution
- ReLU
- MaxPool
- Fully Connected
- SoftMax
STOCHASTIC GRADIENT DESCENT

Initialize $w$

For many iterations:
- Loss = Forward pass $(model)$ → $\|f(w, \text{input}) - b\|_2$ (square loss)
- Gradient = backward $(model)$ Chain rule
- Update model

End

Good fit for $\sin$ "raiser" Tinhorn $\rightarrow$ ardent $\leftarrow f$ - eat $\[y\]eathers$

$w^{(k+1)} = w^{(k)} - \alpha_k \nabla f(w^{(k)})$

Update step

model is shared $\rightarrow$ how do we parallelize

every iteration depends on previous
DATA PARALLEL MODEL TRAINING

Parallelize one iteration

Parallelized model - replicate

\[ \text{model} \rightarrow \text{replicate} \]

\[ \text{loss} (\hat{b}_i) \rightarrow \text{gradient} (\hat{b}_i) \]

Add up/average update step that takes all grads into account

Input: \( B \) = data points

Forward pass:

\[ f(\text{model}, \hat{B}_i) \rightarrow f(\text{model}, \hat{B}_2) \]

\[ \text{input} \rightarrow \text{next iteration} \]

\[ \text{model} \rightarrow \text{replicate} \]

\[ \text{loss} (\hat{b}_i) \rightarrow \text{gradient} (\hat{b}_i) \]

\[ B = \text{data points} \leq 256 \]

Variables:

- \( B = \) data points
- \( B_1, B_2, B_3, B_4 \) = batches

X = n data points

Full gradient descent = \( \frac{n}{B} \)

Mini-batch grad descent = \( \hat{B} \)

Incremental grad descent = \( \hat{B} \rightarrow \hat{B} \)

Update step that takes all grads into account.
COLLECTIVE COMMUNICATION

Broadcast, Scatter  Gather, Reduce

MPI_Bcast  MPI_Gather  MPI_Scatter  MPI_Reduce

From https://mpitutorial.com/tutorials/
DISTRIBUTED DATA PARALLEL API

9    # setup model and optimizer
10   net = nn.Linear(10, 10)
11   net = par.DistributedDataParallel(net)
12   opt = optim.SGD(net.parameters(), lr=0.01)
13
14   # run forward pass
15   inp = torch.randn(20, 10)
16   exp = torch.randn(20, 10)
17   out = net(inp)
18
19   # run backward pass
20   nn.MSELoss()(out, exp).backward()
21
22   # update parameters
23   opt.step()
GRADIENT BUCKETING

Why do we need gradient bucketing?

→ Small tensor sizes lead to greater time for All Reduce

Every All Reduce = (latency cost) + how much data

→ Why not one big bucket

≡ Wait for all gradients to be ready
≡ Cannot overlap backward, AllReduce

$60\text{M parameter}$
GRADIENT BUCKETING + ALL REDUCE

As buckets become ready, we start **All Reduce** on them async.

In background, the gradient comp continues.

Bucket sizes are defined by size = 25 MB.
Gradient Accumulation

```python
ddp = DistributedDataParallel(net)
with ddp.no_sync():
    for inp, exp in zip(inputs, expected_outputs):
        # no synchronization, accumulate grads
        loss_fn(ddp(inp), exp).backward()
    # synchronize grads
    loss_fn(ddp(another_inp), another_exp).backward()
    opt.step()
```
IMPLEMENTATION

Bucket_cap_mb

Parameter-to-bucket mapping

Round-robin ProcessGroups

Parameter that is tunable:
- small: overhead
- large: no overlap

\( \sim \text{middle} = 25 \text{ MB} \)

Gradient → math function

GPUs = on a batch

CPUs: data

locked it

\( \text{AllReceives} \rightarrow \text{AllReceives} \uparrow \)

\( \text{Port} 4 \rightarrow \text{Port} 1 \leftarrow \text{Port} 4 \)

\( \text{Port} 1 \rightarrow \text{Port} 1234 \)

\( \text{port} 1234 \rightarrow 1040 \)

Port 1254

\( \text{layer 2} \rightarrow 20 \text{ MB} \)

\( \text{layer 3} \rightarrow 5 \text{ MB} \)

\( \text{layer 4} \rightarrow \text{bucket 3} \rightarrow \text{filled up} \)

\( \text{bucket 1} \rightarrow \text{bucket 2} \rightarrow 25 \text{ MB} \)

\( \text{query} \rightarrow \text{baiatal} \)

\( \text{SMB} \)
Figure 6: Per Iteration Latency Breakdown
SUMMARY

Pytorch: Framework for deep learning
DistributedDataParallel API
Gradient bucketing, AllReduce
Overlap computation and communication
DISCUSSION

https://forms.gle/6xhVBNBhdzsJ6gBE6
BERT latencies are higher. And larger buckets work well for BERT scales well!

- Time for 32 GPUs
- Time for 16 GPUs

- Optimal bucket size depends on Gloo or NCCL
- NCCL is more performant & less variance

Figure 7: Per Iteration Latency vs Bucket Size on 16 GPUs

Figure 8: Per Iteration Latency vs Bucket Size on 32 GPUs
This paper scales well!? 

Strong scaling

\( B = 256 \) num GPUs

increase

\( T \frac{1}{2} \)

\( T \)

This paper

Weak scaling

\( B = 64 \), increase num GPUs

\( T \frac{1}{2} \)

\( T \)

\( x \)

\( 1 \)

\( 1 \)

\( 2 \)

\( 2 \)
What could be some challenges in implementing similar optimizations for AllReduce in Apache Spark?

- Spark is "larger workloads"?
- Each worker node on Spark had dataset

<table>
<thead>
<tr>
<th>Overlap compute / communication</th>
<th>Not all tasks active at same time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task completes → shuffle</td>
<td></td>
</tr>
</tbody>
</table>

- Overlap compute / communication
- Task completes → shuffle

- More expensive than reduce
Next class: PipeDream
Assignment 2 is due soon!

Project Proposal
Groups by Oct 11
2 pager by Oct 16

Reduce -> brings to single machine
5 -> + 14 < beam

Alloc buckets -> layer 0 -> layer x [-] copy

NEXT STEPS