FLEET: Flexible Efficient Ensemble Training for Heterogenous Deep Neural Networks

Hui Guan, Laxmikant Kishor Mokadam, Xipeng Shen, Seung-Hwan Lim, Robert Patton
Build an image classifier?  

Deep Neural Network (DNN)

Hyperparameters tuning:
- # layers
- # parameters in each layer
- Learning rate scheduling
- ...

CPU

GPU

Multi-Stage Process:
1. Storage
2. Pre-processing
3. Decoding
   - Rotation
   - Cropping
   ...
4. Training
Ensemble Training
• concurrently train a set of DNNs on a cluster of nodes.
Preprocessing is redundant across the pipelines.

Train model 1

Train model 2

\vdots

Train model N
Ensemble training with *data sharing*

Train model 1

Train model 2

...  

Train model N
With data sharing, the training goes even slower!

Train model 1

Train model 2

Train model N
Heterogenous Ensemble

A set of DNNs with different architectures and configurations.

Varying training rate
Varying convergence speed
Heterogenous Ensemble

Varying training rate

Training rate: compute throughput of processing units used for training the DNN.
Heterogenous Ensemble

Varying training rate

If a DNN consumes data slower, other DNNs will have to wait for it before evicting current set of cached batches.
Heterogenous Ensemble

Varying training rate

Varying convergence speed

Due to differences in architectures and hyperparameters, some DNNs converge slower than others.
Heterogenous Ensemble

Varying training rate
Varying convergence speed

A subset of DNNs have already converged while the shared preprocessing have to keep working for the remaining ones.
A flexible ensemble training framework for efficiently training a heterogenous set of DNNs.

Varying training rate
Varying convergence speed

1.12 – 1.92X speedup
Our solution: FLEET

A *flexible* ensemble training framework for *efficiently* training a heterogenous set of DNNs.

**Contributions:**

1. Optimal resource allocation

**heterogenous ensemble**

Varying training rate

Varying convergence speed

1.12 – 1.92X speedup
Our solution: FLEET

A *flexible* ensemble training framework for *efficiently* training a heterogenous set of DNNs.

Varying training rate
Varying convergence speed

Data-parallel distributed training
Checkpointing

Contributions:
1. Optimal resource allocation
2. Greedy allocation algorithm
Our solution: FLEET

A flexible ensemble training framework for efficiently training a heterogenous set of DNNs.

- Varying training rate
- Varying convergence speed

Contributions:
1. Optimal resource allocation
2. Greedy allocation algorithm
3. A set of techniques to solve challenges in implementing FLEET
A flexible ensemble training framework for efficiently training a heterogenous set of DNNs.

Focus of This Talk

Contributions:
1. Optimal resource allocation
2. Greedy allocation algorithm
3. A set of techniques to solve challenges in implementing FLEET
**Resource Allocation Problem**

**Optimal CPU allocation:**
Set \#processes for preprocessing to be the one that just meet the computing requirements of training DNNs.

What is an optimal GPU allocation?
GPU Allocation

Node 1
- GPU
- GPU

Node 2
- GPU
- GPU

DNN 1
DNN 2
DNN 3
DNN 4
With data sharing, the slowest DNN determines the training rate of the ensemble training pipeline.
Another way to allocate GPUs: only DNN 1 and DNN 4 are trained together with data sharing.
GPU Allocation: Different GPUs to Different DNNs

Flotilla

A set of DNNs to be trained together with data sharing (e.g., DNN1 and DNN4).

Node 1
- GPU
- GPU
- DNN 1
- DNN 4

100 images/sec

Node 2
- GPU
- GPU
- DNN 4
- DNN 4

105 images/sec

Training rate
We need to create a list of flotillas to train all DNNs to converge.
Optimal Resource Allocation

Given a set of DNN to train and a cluster of nodes, find:

(1) the list of flotillas and
(2) GPU assignments within each flotilla
such that the end-to-end ensemble training time is minimized.

NP-hard
Greedy Allocation Algorithm

Dynamically determine the list of flotillas:

(1) whether a DNN is converged or not,
(2) the training rate of each DNN.

Once a flotilla is created,
derive an optimal GPU assignment
Create a new flotilla

Assign GPUs for DNNs in the flotilla

Train DNNs in the flotilla with *data sharing*

Converged DNNs

Profile training rates of each DNN on *m* GPUs

DNN ensemble

Greedy Allocation Algorithm
Create a new flotilla

Assign GPUs for DNNs in the flotilla

Train DNNs in the flotilla with data sharing

Converged DNNs

Greedy Allocation Algorithm: profiling

Training rates (images/sec) of DNNs on GPUs.

<table>
<thead>
<tr>
<th># GPU</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNN 1</td>
<td>100</td>
<td>190</td>
<td>270</td>
<td>350</td>
</tr>
<tr>
<td>DNN 2</td>
<td>80</td>
<td>150</td>
<td>220</td>
<td>280</td>
</tr>
<tr>
<td>DNN 3</td>
<td>80</td>
<td>150</td>
<td>200</td>
<td>240</td>
</tr>
<tr>
<td>DNN 4</td>
<td>40</td>
<td>75</td>
<td>105</td>
<td>120</td>
</tr>
</tbody>
</table>
Greedy Allocation Algorithm

Step 1: Flotilla Creation

Step 2: GPU Assignment

Step 3: Model training
**Step 1: Flotilla Creation**

1. DNNs in the same flotilla should be able to reach a similar training rate if a proper number of GPUs is assigned to each DNN.

   - Reduce GPU waiting time

2. Pack into one flotilla as many DNNs as possible.

   - Avoid inefficiency due to sublinear scaling
   - Allow more DNNs to share preprocessing
DNN ensemble

Profile training rates of each DNN on \( m \) GPUs

Create a new flotilla

Assign GPUs for DNNs in the flotilla

Train DNNs in the flotilla with \textit{data sharing}

Converged DNNs

\textbf{Step 1: Flotilla Creation}

\textbf{# GPUs available:} 4-1 \rightarrow 3-3 \rightarrow 0

\begin{tabular}{|c|c|c|c|c|}
\hline
\# GPU & 1 & 2 & 3 & 4 \\
\hline
DNN 1 & 100 & 190 & 270 & 350 \\
DNN 2 & 80  & 150 & 220 & 280 \\
DNN 3 & 80  & 150 & 200 & 240 \\
DNN 4 & 40  & 75  & 105 & 120 \\
\hline
\end{tabular}
Create a new flotilla

Profile training rates of each DNN on \( m \) GPUs

Assign GPUs for DNNs in the flotilla

Train DNNs in the flotilla with *data sharing*

Converged DNNs

**Step 2: GPU Assignment**

#1: When assigning multiple GPUs to a DNN, try to use GPUs in the same node.

#2: Try to assign DNNs that need a smaller number of GPUs to the same node.

Reduce the variation in communication latency

**Node 1**
- GPU
- GPU
- DNN 1
- DNN 4

**Node 2**
- GPU
- GPU
- DNN 4
- DNN 4
Step 1: Flotilla Creation
Step 2: GPU Assignment

DNN ensemble

Profile training rates of each DNN on \( m \) GPUs

Create a new flotilla

Assign GPUs for DNNs in the flotilla

Train DNNs in the flotilla with *data sharing*

Converged DNNs

Data-parallel distributed training

Varying training rate
Create a new flotilla

Assign GPUs for DNNs in the flotilla

Train DNNs in the flotilla with data sharing

Profile training rates of each DNN on $m$ GPUs

Converged DNNs

Checkpointing

Varying convergence speed

Step 3: Model Training

DNN ensemble
Step 3: Model Training

Once converged, mark as complete and release GPUs.

<table>
<thead>
<tr>
<th># GPU</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNN 1</td>
<td>100</td>
<td>190</td>
<td>270</td>
<td>350</td>
</tr>
<tr>
<td>DNN 2</td>
<td>80</td>
<td>150</td>
<td>220</td>
<td>280</td>
</tr>
<tr>
<td>DNN 3</td>
<td>80</td>
<td>150</td>
<td>200</td>
<td>240</td>
</tr>
<tr>
<td>DNN 4</td>
<td>40</td>
<td>75</td>
<td>100</td>
<td>120</td>
</tr>
</tbody>
</table>

Stop training the flotilla once less than 80% of GPUs remain active for training.
**Step 3: Model Training**

- Create a new flotilla
- Assign GPUs for DNNs in the flotilla
- Train DNNs in the flotilla with *data sharing*
- Profile training rates of each DNN on *m* GPUs

**DNN ensemble**

<table>
<thead>
<tr>
<th># GPU</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNN 1</td>
<td>100</td>
<td>190</td>
<td>270</td>
<td>350</td>
</tr>
<tr>
<td>DNN 2</td>
<td>80</td>
<td>150</td>
<td>220</td>
<td>280</td>
</tr>
<tr>
<td>DNN 3</td>
<td>80</td>
<td>150</td>
<td>200</td>
<td>240</td>
</tr>
<tr>
<td>DNN 4</td>
<td>40</td>
<td>75</td>
<td>100</td>
<td>120</td>
</tr>
</tbody>
</table>

Consider only un-converged DNNs when create the next flotilla.
Experiment Settings

• Heterogenous ensemble
  • 100 DNNs derived from DenseNets and ResNets
  • Training rate on a single GPU: 21~176 images/sec.

• Summit-Dev@ORNL
  • 2 IBM POWER8 CPUs with 256GB DRAM
  • 4 NVIDIA Tesla P100 GPUs

• Dataset
  • Caltech256: 30K training images (240 minutes limit)
Counterparts for Comparisons

- **Baseline**
  - Train each DNN on one GPU *independently*
  - Randomly picks one yet-to-be-trained DNN whenever a GPU is free

- **Homogeneous Training** (Pittman et al., 2018)
  - Train each DNN on one GPU with *data sharing*
  - When \#GPUs < \#DNNs, randomly picks a subset of DNNs to train after the previous subset is done

- **FLEET-G** (*global* paradigm)
- **FLEET-L** (*local* paradigm)
  - Train remaining DNNs once some GPUs are released
  - Pick the DNNs to train by the greedy algorithm in FLEET-G
End-to-End Speedups

**Homogeneous:** slowdowns are due to the waiting of other GPUs for the slowest DNN to finish.
End-to-End Speedups

**FLEET-G:** the best overall performance, **1.12-1.92X speedups** over the baseline.

**FLEET-L:** notable but smaller speedups for less favorable allocation decisions.

The overhead of scheduling and checkpointing is at most 0.1% and 6.3% of the end-to-end training time in all the settings.
Conclusions and Future Work

• Systematically explore the strategies for flexible ensemble training for a heterogenous set of DNNs.
  • Optimal resource allocation & Greedy allocation algorithm
  • Software implementation
    • Data-parallel distributed training, dynamic GPU-DNN mappings, checkpointing, data sharing

• Future work: apply FLEET to real hyperparameter tuning and neural architecture search workloads.