Deep Learning Inference as a Service

Mohammad Babaeizadeh
Hadi Hashemi
Chris Cai

Advisor: Prof Roy H. Campbell
<table>
<thead>
<tr>
<th>mite</th>
<th>container ship</th>
<th>motor scooter</th>
<th>leopard</th>
</tr>
</thead>
<tbody>
<tr>
<td>black widow</td>
<td>container ship</td>
<td>motor scooter</td>
<td>leopard</td>
</tr>
<tr>
<td>cockroach</td>
<td>lifeboat</td>
<td>go-kart</td>
<td>jaguar</td>
</tr>
<tr>
<td>tick</td>
<td>amphibian</td>
<td>moped</td>
<td>cheetah</td>
</tr>
<tr>
<td>starfish</td>
<td>fireboat</td>
<td>bumper car</td>
<td>snow leopard</td>
</tr>
<tr>
<td></td>
<td>drilling platform</td>
<td>golfcart</td>
<td>Egyptian cat</td>
</tr>
<tr>
<td>grille</td>
<td>mushroom</td>
<td>cherry</td>
<td>Madagascar cat</td>
</tr>
<tr>
<td>convertible</td>
<td>agaric</td>
<td>dalmatian</td>
<td>squirrel monkey</td>
</tr>
<tr>
<td>grille</td>
<td>mushroom</td>
<td>grape</td>
<td>spider monkey</td>
</tr>
<tr>
<td>pickup</td>
<td>jelly fungus</td>
<td>elderberry</td>
<td>titi</td>
</tr>
<tr>
<td>beach wagon</td>
<td>gill fungus</td>
<td>currant</td>
<td>indri</td>
</tr>
<tr>
<td>fire engine</td>
<td>dead-man's-fingers</td>
<td></td>
<td>howler monkey</td>
</tr>
</tbody>
</table>
ILSVRC top-5 error on ImageNet
Organizations Engaged with NVIDIA on Deep Learning

- Higher Education
- Development Tools
- Internet
- Automotive
- Finance
- Government
- Life Science
- Other

2014: 1,549
2016: 19,439

NVIDIA
Use case 1: Model Developer
Use case 1: Model Developer
Use case 2: Application Developer
Use case 2: Application Developer
Use case 2: Application Developer
Problem Formulation

Models

Queries

Nodes
Characteristics of DNNs
VGG 16

Inception
Constant Runtime

For TensorFlow + CPU:
- ResNet: 150 ms
- VGG16: 250 ms
- VGG19: 275 ms
- Inception: 180 ms

For TensorFlow + GPU:
- ResNet: 45 ms
- VGG16: 20 ms
- VGG19: 30 ms
- Inception: 45 ms
Batch Size (Vectorized Computation)
Stateless

• Embarrassingly parallel workload
  • No centralized data storage

• Load/unload models as necessary
  • No data synchronization is needed for load/unload
  • Load/unload is as expensive as running a process

• Light-weight fault-tolerance
  • Bookkeeping only for queries
Stateless

• Embarrassingly parallel workload
  • No centralized data storage

• Load/unload models as necessary
  • No data synchronization is needed for load/unload
  • Load/unload is as expensive as running a process

• Light-weight fault-tolerance
  • Bookkeeping only for queries
Stateless

• Embarrassingly parallel workload
  • No centralized data storage

• Load/unload models as necessary
  • No data synchronization is needed for load/unload
  • Load/unload is as expensive as running a process

• Light-weight fault-tolerance
  • Bookkeeping only for queries
Deterministic

• Output is always the same for the same input

• An effective caching mechanism
Query Characteristics
Query Patterns

• Offline queries
  • Batch queries with high latency (hours) SLA

• Online stateless queries
  • Single queries with low latency (100s ms) SLA

• Online stateful queries
  • Session with a sequence of queries, each with low latency (100s ms) SLA
  • Session lifetime is of minutes
Inference as a Service
Problem Statement

• To serve arbitrary number of models with different service-level objections on minimum resource.

• Number of models >> Number of nodes (in contrast with previous works)

• Isolating overall performance of the system from model’s. (vectorization is optional)

• Low overhead on request response time.
Use case: Registering a new model

1. Model should use our lightweight API
2. Host a model in a container (docker)
3. Benchmark the model (load/runtime) for different batches
4. Reject a model if impossible to server (with respect to SLO)
5. Stored the model on distributed file system
Use case: Submitting a query

1. Client asynchronously sends request(s) to a Master Server and gets request ID(s) as response
2. Master may respond from the cache
3. Otherwise passes the query to scheduler
4. Scheduler assigns each query to a worker, and sends the commands to the pub/sub
   - May unload a loaded model
   - May load a new model
   - May duplicate a running model
   - May wait for more requests to come (for batching)
5. Compute nodes follow the commands to load/unload models off the distributed file system
6. Workers fetch all the requests, server them in batch, and puts the results back in pub/sub
7. Master fetches the results and waits for client to request a responses
Client API

- `Send_Requests(data[], model)`
- `Get_Responses(request_id[])`
- `Start_Session(model)`
- `Send_Requests(data[], session_id)`
- `End_Session(session_id)`
Demo
In progress / Open Problems

- **Scheduler**
  - Elasticity
  - Analytical model

- **API Expansion**
  - More languages: Currently Python
  - Pipelines
  - Ensembles

- **Model Efficiency**
  - Load/Unload
  - Stateful Models

- **Model Isolation**
  - Currently limited to computation, Outsourced to Docker
  - Memory Bandwidth, PCIE

- **Fault Tolerance**
  - Currently outsourced to Redis
  - Approximated response
Related Problems

- Model Compression
- DNN specific hardware
- Realtime on mobile devices