Large Scale Models and Out-of-core DNNs

CSE 5194

Shashank Gugnani

*Slides taken from https://goo.gl/3W4M34 and https://goo.gl/kB9JKW
Selected Papers

- vDNN: Virtualized Deep Neural Networks for Scalable, Memory Efficient Neural Network Design, NVIDIA, IEEE MICRO’16

- Outrageously Large Neural Networks: The Sparsely Gated Mixture-Of-Experts Layer, Google Brain, ICLR’17
Selected Papers

- **vDNN: Virtualized Deep Neural Networks for Scalable, Memory Efficient Neural Network Design**, NVIDIA, IEEE MICRO’16

- **Outrageously Large Neural Networks: The Sparsely Gated Mixture-Of-Experts Layer**, Google Brain, ICLR’17
Introduction

Titan X: 12 GB memory capacity

The trend in deep learning is to move towards larger and deeper network designs.
- The physical memory limitations of GPUs is becoming increasingly important.
Introduction

In this paper, the authors propose \textit{vDNN}

\textbf{vDNN (virtualized Deep Neural Network)}

\begin{itemize}
\item A runtime memory management solution that virtualizes the memory usage of DNN across both GPU and CPU memories
\item vDNN allows to train larger and deeper networks beyond the capacity of GPU
\end{itemize}
Background and Motivation

*Forward propagation* is performed from the first layer to the last layer. *Backward propagation* is performed in the opposite direction. Both propagations traverse the network layer-wise.
Background and Motivation

Per layer memory allocations required are determined by the layer’s input-output relationships and its mathematical function.

e.g) Convolutional layer
- Forward: input/output feature maps (X and Y), weights of the layer (W)
- Backward: input/output gradient maps (dY and dX), weight’s gradient (dW), X and W
- If FFT based convolution algorithm is used, it needs an additional workspace (WS) buffer
Background and Motivation

Because of the layer-wise gradient update rule of the backward propagation algorithm, each layer’s feature maps \( (X) \) are later reused during its own backward propagation pass:

- All \( Xs \) must still be available in GPU memory until backward computation is completed.

As the number of layers increases, the fraction of memory allocated for feature maps grows.
Background and Motivation

VGG network

Per layers memory usage of VGG-16 (batch size: 256) during forward
Background and Motivation

There are the following key observations about memory usage:

◦ The intermediate feature maps (X) and workspace (WS) incur higher memory usage compared to the weights (W) of each layer
◦ Most of these X are concentrated on the feature extraction layers
◦ Most of these W are concentrated on the later classifier layers
◦ The per layer memory usage is much smaller than memory usage of the entire network
The design objective of vDNN memory manager is to virtualize the memory usage of DNNs, using both GPU and CPU memory, while minimizing its impact on performance.

There is overhead due to communication between GPU and CPU.
vDNN

vDNN is based on the three key observations:

1. DNNs are via SGD are designed and structured with multiple layers.
2. The training of these neural networks involves a series of layer-wise computations.
3. The GPU only processes a single layer’s computation at any given time.

vDNN adopts a sliding-window based, layer-wise memory management strategy:

- The runtime memory manager allocates memory for the immediate usage of the layer that is currently being processed by the GPU.
vDNN

Forward propagation:
- vDNN allocates current layer’s X on GPU
- Other layer’s Xs are offloaded to CPU memory
vDNN

Backward propagation:

- Similar to forward propagation, vDNN aggressively releases data that are not needed for current layer’s backward computation
vDNN


- cuDNN is a GPU-accelerated library for DNN
  - Various frameworks (including Caffe, Tensorflow, chainer) use cuDNN

- cuDNN provide some algorithms for each layer’s operation, and can find the best suited algorithm
  
  e.g) convolutionForward: IMPLICIT_GEMM, GEMM, FFT, etc
vDNN uses CUDA streams (https://docs.nvidia.com/cuda/cuda-c-programming-guide/)

- A stream is a sequence of operations that execute in order on GPU
- Different streams may execute their operations out of order with respect to one another or concurrently

```c
cudaMemcpyAsync(HtoD, stream1);
Convolution(stream1);
cudaMemcpyAsync(DtoH, stream1);
```
vDNN

vDNN employs two streams, $stream_{compute}$ and $stream_{memory}$

- $stream_{compute}$: all the layer’s forward and backward computation
- $stream_{memory}$: the memory allocation/release, offload, and prefetch

Memory Allocation/Release

- When the program launches, the vDNN allocates memory pool
- Whenever vDNN allocates (and releases) data structure, the memory is allocated (released) from memory pool without cudaMalloc() and cudaFree()
vDNN

Memory Offload
- Input feature maps (Xs) are offloaded from GPU to CPU
- vDNN overlaps offloading with the same layer’s forward computation
vDNN

Memory Prefetch

- Offloaded Xs are prefetched back from CPU to GPU
- vDNN overlaps other layer’s prefetching with the current layer’s backward computation
vDNN

Static vDNN
- $vDNN_{\text{all}}$
  - offload all layers’ $X$ from GPU
  - most memory-efficient solution

- $vDNN_{\text{conv}}$
  - only offload CONV layers’ $X$ from GPU
  - This policy is based on the observation that CONV layers have long computation latency to hide the latency of offload/prefetch
vDNN

Static vDNN

- Convolutional algorithm is determined with memory-optimal or performance-optimal

- While static vDNN is simple and easy to implement, it does not account for the system architectural components that determine the trainability and performance of a DNN
vDNN

**Dynamic vDNN**

- $vDNN_{dyn}$
  - automatically determine the offloading layers and the convolutional algorithms at runtime
  - balance the trainability and performance of a DNN

- The dynamic vDNN launches profiling to optimization before training iterations
  - This profiling is based on a greedy algorithm that tries to locally optimize layer’s memory usage and performance, seeking a global optimum state in terms of trainability and performance
Methodology: GPU Node Topology

NVIDIA’s Titan X
- Single precision throughput: 7 TFLOPS
- Memory bandwidth: 336 GB/sec
- Memory capacity: 12 GB

The GPU communicates with an Intel i7-5930K via a PCIe switch
- Bandwidth of communication with CPU: 16 GB/sec
Results

all: static $vDNN_{all}$
conv: static $vDNN_{conv}$
dyn: dynamic $vDNN_{dyn}$
base: without vDNN (all memory is allocated on GPU)

$\diamond vDNN_{all}, vDNN_{conv}$ and base are evaluated with both memory-optimal (m) and performance-optimal (p) algorithms
Results: GPU Memory Usage

Performance-optimal vDNN tend to allocate more memory on GPU to improve performance
- Performance-efficient algorithms requires larger workspace
- The total number of offload layers is reduced

Average and maximum memory usage (left axis). Right axis corresponds to the savings in average memory usage.
Results: Performance

The $vDNN_{conv}$ throughput reaches an average 97% of baseline throughput.

The dynamic vDNN does much better in terms of balancing memory efficiency and overall throughput.
Results: Training Very Deep Networks

$vDNN$ allocates most of memory in CPU memory
- Very deep networks can be trained

$vDNN_{dyn}$ did not incur any noticeable performance degradation
- Because the offload/prefetch latency is completely hidden
Conclusion

Existing ML frameworks require users to carefully manage their GPU memory usage
- vDNN solution improves the memory-efficiency of DNN

We also study the scalability of vDNN to extremely deep network
- vDNN can train networks with hundreds of layers without any performance loss
Selected Papers

- **vDNN**: Virtualized Deep Neural Networks for Scalable, Memory Efficient Neural Network Design, NVIDIA, IEEE MICRO’16

- **Outrageously Large Neural Networks**: The Sparsely Gated Mixture-Of-Experts Layer, Google Brain, ICLR’17
Motivation

- Key contributors in performance of deep networks:
  - Model size
  - Training data size
- “When datasets are sufficiently large, increasing the capacity (number of parameters) of neural networks can give much better prediction accuracy”
- Current computational infrastructure fall short of providing the computing demands
- Roughly quadratic growth in training cost as both the model size and the number of training examples increase
Previous Solutions

- *Conditional computation* schemes - parts of a network are used depending on the example
- Gating decisions could be binary or sparse and continuous, stochastic or deterministic
- Various forms of reinforcement learning and back-propagation for training the gating decisions
- None has demonstrated massive improvements
Challenges

• Most computing devices are much faster at arithmetic than branching
• Conditional computing reduces the batch sizes due to the conditionally active chunk
• Network bandwidth speed is slower than computation speed
• Loose information to achieve the desired level of sparsity
• Small model capacity for acceptable datasets
Mixture of Experts (MoE) Layer

- Consists of a number of experts, each a simple feed-forward neural network
- A trainable gating network which selects a combination of the experts to process each input
- All parts are trained jointly by back-propagation
Mixture of Experts (MoE) Layer

\[ y = \sum_{i=1}^{n} G(x)_i E_i(x) \]

- If \( G(x)_i \) is 0, \( E_i(x) \) need not be computed
- In each round, only a handful of experts are active
Large Number of Experts

• Reduce branching factor by creating a two-level hierarchy of experts
• *Hierarchical MoE*: Each expert is itself a MoE
Gating Network

- Desired characteristics
  - Sparsity
  - Load balancing

\[
G(x) = \text{Softmax}(\text{KeepTopK}(H(x), k))
\]

\[
H(x)_i = (x \ast W_g)_i + \text{StandardNormal()} \ast \text{Softplus}((x \ast W_{\text{noise}})_i)
\]

\[
\text{KeepTopK}(v, k)_i = \begin{cases} 
  v_i, & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\
  -\infty, & \text{otherwise.}
\end{cases}
\]
Shrinking Batch Size Problem

• Large batch sizes are necessary to achieve high throughput
  • Amortizing the overhead of data transfer
• If the gating chooses $k$ out of $n$ experts for a batch of $b$ examples, each expert receives a batch of approximately $kb/n \ll b$ examples
• Extremely large batch sizes limited by memory
Mixed Data and Model Parallelism

• Solution: run the standard layers in parallel with different batches of data
• Feed into only 1 shared MoE layer
• Each expert receives a combined batch from all the parallel inputs
• If there are $d$ parallel devices, each expert receives $kbd/n$ examples
• Factor of $d$ improvement
Load Balancing

• Gating network selects a certain number of experts and the process is self-reinforcing

• It is required to define an additional loss term to discourage such a behavior for a given batch $X$

\[
\text{Importance}(X) = \sum_{x \in X} G(x)
\]

\[
L_{\text{importance}}(X) = w_{\text{importance}} \times CV(\text{Importance}(X))^2
\]

• As the gating favors a few experts, the overall loss increases
Performance Evaluation

• Training using TensorFlow
• Cluster containing 16-32 K40 GPUs
• Focus on language modeling and machine translation
• MoE inserted between stacked LSTM layers
• Benchmarks
  • 1 Billion Word Language Modeling Benchmark
  • 100 Billion Word Google News Corpus
  • Machine Translation (Single Language Pair)
Performance Measurement: Perplexity

• In machine learning, perplexity is a measure of prediction error

\[ 2^H(p) = 2^{-\sum_x p(x) \log_2 p(x)} \]

• A measure to determine how strongly results are predicted
Performance Measurement: Capacity

![Graphs showing performance measurement for different models.](image)
Performance Measurement: Capacity

100 Billion Word Language Modeling Benchmark
## Billion words language modeling benchmark

<table>
<thead>
<tr>
<th></th>
<th>Perplexity</th>
<th>#Parameters</th>
<th>Training Time</th>
<th>TFLOPS/GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best public results [7]</td>
<td>34.7</td>
<td>151 Millions</td>
<td>59 Hours – 32 K40</td>
<td>1.09</td>
</tr>
<tr>
<td>Proposed</td>
<td>28.0</td>
<td>4.4 Billions</td>
<td>47 Hours – 32 K40</td>
<td>1.56</td>
</tr>
</tbody>
</table>
## Performance Comparison: Translation

<table>
<thead>
<tr>
<th></th>
<th>GNMT-Multi</th>
<th>MoE-Multi</th>
<th>MoE-Multi vs. GNMT-Multi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>278M</td>
<td>8.7B</td>
<td></td>
</tr>
<tr>
<td>ops/timestep</td>
<td>212M</td>
<td>102M</td>
<td></td>
</tr>
<tr>
<td>training time, hardware</td>
<td>21 days, 96 k20s</td>
<td>12 days, 64 k40s</td>
<td></td>
</tr>
<tr>
<td>Perplexity (dev)</td>
<td>4.14</td>
<td><strong>3.35</strong></td>
<td>-19%</td>
</tr>
<tr>
<td>French → English Test BLEU</td>
<td>34.40</td>
<td><strong>37.46</strong></td>
<td>+3.06</td>
</tr>
<tr>
<td>German → English Test BLEU</td>
<td>31.17</td>
<td><strong>34.80</strong></td>
<td>+3.63</td>
</tr>
<tr>
<td>Japanese → English Test BLEU</td>
<td>21.62</td>
<td><strong>25.91</strong></td>
<td>+4.29</td>
</tr>
<tr>
<td>Korean → English Test BLEU</td>
<td>22.87</td>
<td><strong>28.71</strong></td>
<td>+5.84</td>
</tr>
<tr>
<td>Portuguese → English Test BLEU</td>
<td>42.53</td>
<td><strong>46.13</strong></td>
<td>+3.60</td>
</tr>
<tr>
<td>Spanish → English Test BLEU</td>
<td>36.04</td>
<td><strong>39.39</strong></td>
<td>+3.35</td>
</tr>
<tr>
<td>English → French Test BLEU</td>
<td>34.00</td>
<td><strong>36.59</strong></td>
<td>+2.59</td>
</tr>
<tr>
<td>English → German Test BLEU</td>
<td>23.15</td>
<td>24.53</td>
<td>+1.38</td>
</tr>
<tr>
<td>English → Japanese Test BLEU</td>
<td>21.10</td>
<td>22.78</td>
<td>+1.68</td>
</tr>
<tr>
<td>English → Korean Test BLEU</td>
<td>18.41</td>
<td>16.62</td>
<td>-1.79</td>
</tr>
<tr>
<td>English → Portuguese Test BLEU</td>
<td>37.35</td>
<td>37.90</td>
<td>+0.55</td>
</tr>
<tr>
<td>English → Spanish Test BLEU</td>
<td>34.25</td>
<td><strong>36.21</strong></td>
<td>+1.96</td>
</tr>
</tbody>
</table>
Conclusion

• Algorithmic and engineering solution
• Significantly increase in model capacity with minimal impact on training cost
• Focused on text experiments but can be applied for other situations
References


Backup
Background and Motivation

DNNs are designed using a combination of multiple types of layers

- Convolutional layer
- Activation layer
- Pooling layer
- Fully-connected layer

\[ \text{feature extraction layers} \]

\[ \text{classification layers} \]

Convolutional neural networks are one of the most popular ML algorithms for image recognition

- These DNNs are trained using a backward propagation algorithm
vDNN

Non-linear feedforward network still involves a series of layer-wise computations.

• vDNN can also handle non-linear
vDNN

cudaMemcpyAsync(HtoD, stream1);
Convolution(stream2);
cudaMemcpyAsync(DtoH, stream1);

stream1: HtoD DtoH
stream2: CONV

time
vDNN

cudaMemcpyAsync(HtoD, stream1);
Convolution(stream2);
cudaStreamSynchronize(stream2);
cudaMemcpyAsync(DtoH, stream1);
vDNN

Determining the best layers to offload their X is a multi-dimensional optimization problem that must consider

1. GPU memory capacity
2. The convolutional algorithms used and the overall layer-wise memory usage
   ◦ “memory-optimal implicit GEMM” VS “performance-optimal convolutional algorithm”
3. The network-wide performance
   ◦ The additional latency possibly incurred due to offload/prefetch
Results: Impact on Memory System

vDNN does come at the cost of adding read/write traffic to the GPU memory subsystem

- Potentially interfering with the normal cuDNN operations
- The feature extraction layers rarely saturate the 336 GB/sec of peak memory bandwidth
MoE: Background and Motivation
MoE: Network Bandwidth

• Problem: overhead cost for communicating inputs and outputs
• Use a larger hidden layer or more hidden layers within memory limit
## Performance Comparison: Translation

<table>
<thead>
<tr>
<th>Expert 381</th>
<th>Expert 752</th>
<th>Expert 2004</th>
</tr>
</thead>
<tbody>
<tr>
<td>... with <em>researchers</em> , ...</td>
<td>... plays a core ...</td>
<td>... with <em>rapidly</em> growing ...</td>
</tr>
<tr>
<td>... to <em>innovation</em> .</td>
<td>... plays a critical ...</td>
<td>... under <em>static</em> conditions ...</td>
</tr>
<tr>
<td>... tics <em>researchers</em> .</td>
<td>... provides a legislative ...</td>
<td>... to <em>swiftly</em> ...</td>
</tr>
<tr>
<td>... the <em>generation</em> of ...</td>
<td>... play a leading ...</td>
<td>... to <em>drastically</em> ...</td>
</tr>
<tr>
<td>... technology <em>innovations</em> is ...</td>
<td>... assume a leadership ...</td>
<td>... the <em>rapid</em> and ...</td>
</tr>
<tr>
<td>... technological <em>innovations</em>, ...</td>
<td>... plays a central ...</td>
<td>... the <em>fastest</em> ...</td>
</tr>
<tr>
<td>... support <em>innovation</em> throughout ...</td>
<td>... taken a leading ...</td>
<td>... the Quick Method ...</td>
</tr>
<tr>
<td>... role <em>innovation</em> will ...</td>
<td>... established a reconciliation ...</td>
<td>... rec <em>urrent</em> ) ...</td>
</tr>
<tr>
<td>... research <em>scientist</em> ...</td>
<td>... played a vital ...</td>
<td>... provides quick access ...</td>
</tr>
<tr>
<td>... promoting <em>innovation</em> where ...</td>
<td>... have a central ...</td>
<td>... of <em>volatile</em> organic ...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>