vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design

Zhuo Xu
Introduction

- Memory usage bottleneck
- Virtualized DNN
- Methodology
- Results
Trend of GPU memory usage for training DNN

- DRAM capacity of the GPU(s) limits the size of the DNN that can be trained.
- Trend in deep learning is to move towards larger and deeper network designs.
Breakdown of GPU memory usage
Virtualized DNN

vDNN adopts a sliding-window based, layer-wise memory management strategy.
Virtualized DNN

- evaluating the feasibility of offloading a layer
- multiple layers can be the consumers of a previously computed layer’s output
Core Operations And Its Design

- Memory Allocation/Release
  - asynchronous memory allocation/release API library
- Memory Offload
  - Offloading input feature maps
- Memory Prefetch
  - vDNN determines the best layer to prefetch among the preceding layers
Core Operations And Its Design

- **Static vDNN**
  - vDNNall
  - vDNNconv

- **Dynamic vDNN**
  - For Large and deep networks
Methodology

- vDNN Memory Manager
  - host-side memory manager that interacts with cuDNN

- Baseline
  - Torch's memory management policy

- GPU Node Topology
  - NVIDIA's Titan X with Intel i7-5930K
Results

- vDNNall(m)
  - 73% and 93% reduction on maximum and average memory usage
  - 58% and 55% performance loss

- vDNNdyn
  - reducing the maximum and average memory consumption by 49% and 69%
  - Achieving 97% of baseline's throughput

![Graph showing memory usage and savings](image.png)

Fig. 11: Average and maximum memory usage (left axis). Right axis corresponds to the savings in average memory usage.
Checkmate: Breaking the Memory Wall with Optimal Tensor Rematerialization
State-of-the-art models have hit a memory capacity wall

- Emerging trend:
  - Rapid growth in model size

- Limited GPU memory is slowing progress in new deep learning models
How do we efficiently train large models beyond memory limits?

- Compute is outstripping DRAM capacity growth
- Backprop is optimized for compute efficiency, not RAM usage
Checkmate: optimal tensor rematerialization system

- RAM-hungry backprop policy
  - Keep all layers in RAM
- RAM-optimized backpropagation policy
  - Recompute all layers as needed
  - Free early & recompute
- Which layers to recompute?
  - One possible answer could be heuristics
Checkmate: optimal tensor rematerialization system

Forward Pass

Backward Pass

**Compute:** $O(n)$ additional overhead

**RAM:** $O(\sqrt{n})$ RAM usage
Checkmate: optimal tensor rematerialization system

S = What is in memory

R = What is computed
Checkmate: optimal tensor rematerialization system

- **Optimal Solver**
  - Integer Linear Program
  - Takes 9 hour to solve
  - Partition schedule into frontier-advancing stages
    - Takes 0.23s

- **Near-optimal Solver**
  - Two phase rounding
    - Round S, solve other variables optimally

\[
\begin{align*}
\text{arg min}_{R, S, U, \text{FREE}} & \quad \sum_{t=1}^{n} \sum_{i=1}^{i} C_{i}R_{t,i} \\
R_{t,j} & \leq R_{t,i} + S_{t,i} \\
S_{t,i} & \leq R_{t-1,i} + S_{t-1,i} \\
R_{t,i}, S_{t,i} & \in \{0, 1\} \\
U_{t,k} & \leq M_{\text{budget}}
\end{align*}
\]
Evaluation: What is the memory vs compute trade-off?
Evaluation: How much can we increase batch size?
Evaluation: How much can we increase batch size?
Thank you!