Lecture 9: Memory Optimization

CSE599W: Spring 2018
Where are we

**High level Packages**
- Programming API
- Gradient Calculation (Differentiation API)

**System Components**
- Computational Graph Optimization and Execution
- Runtime Parallel Scheduling

**Architecture**
- GPU Kernels, Optimizing Device Code
- Accelerators and Hardwares
Where are we

- Programming API
- Gradient Calculation (Differentiation API)
- Computational Graph Optimization and Execution
- Runtime Parallel Scheduling
- GPU Kernels, Optimizing Device Code
- Accelerators and Hardwares
Recap: Computation Graph
Recap: Automatic Differentiation

Backprop in Graph

Autodiff by Extending the Graph: assignment 1
Questions for this Lecture

Why do we need automatic differentiation that extends the graph instead of backprop in graph?
Memory Problem of Deep Nets

Deep nets are becoming deeper

LeNet

Inception
State-of-Art Models can be Resource Bound

- Examples of recent state of art neural nets
  - Convnet: ResNet-1k on CIFAR-10, ResNet-200 on ImageNet
  - Recurrent models: LSTM on long sequences like speech

- The maximum size of the model we can try is bounded by total RAM available of a Titan X card (12G)

We need to be frugal
Q: How to build an Executor for a Given Graph

Computational Graph for $\exp(a \times b + 3)$
Build an Executor for a Given Graph

1. **Allocate** temp memory for intermediate computation

   Computational Graph for $\exp(a \times b + 3)$

   Same color represent same piece of memory

   $a$, $b$ $\rightarrow$ $\text{mul}$ $\rightarrow$ $\text{add-const}$ $\rightarrow$ $\exp$
Build an Executor for a Given Graph

1. **Allocate** temp memory for intermediate computation

2. **Traverse and execute** the graph by topo order.

Computational Graph for $\exp(a \times b + 3)$
Build an Executor for a Given Graph

1. **Allocate** temp memory for intermediate computation

2. **Traverse and execute** the graph by topo order.

**Temporary space linear to number of ops**

Computational Graph for \( \exp(a \times b + 3) \)
Dynamic Memory Allocation

1. **Allocate** when needed

2. **Recycle** when a memory is not needed.

3. Useful for both declarative and imperative executions

**Memory Pool**
Dynamic Memory Allocation

1. **Allocate** when needed
2. **Recycle** when a memory is not needed.
3. Useful for both declarative and imperative executions

**Memory Pool**
Dynamic Memory Allocation

1. **Allocate** when needed

2. **Recycle** when a memory is not needed.

3. Useful for both declarative and imperative executions

**Memory Pool**
Static Memory Planning

1. Plan for reuse **ahead of time**

2. Analog: register allocation algorithm in compiler

Same color represents same piece of memory
Common Patterns of Memory Planning

- **Inplace** store the result in the input
- **Normal Sharing** reuse memory that are no longer needed.
Inplace Optimization

- Store the result in the input
- Works if we only care about the final result
- Question: what operation cannot be done inplace?

Computational Graph for $\exp(a \times b + 3)$
Inplace Pitfalls

We can only do inplace if result op is the only consumer of the current value
Normal Memory Sharing

Recycle memory that is no longer needed.
Memory Planning Algorithm

Memory Planning Algorithm

Initial state of allocation algorithm

step 1: Allocate tag for B

step 2: Allocate tag for C, **cannot do inplace** because B is still alive

step 3: Allocate tag for F, release space of B

step 4: Reuse the tag in the box for E

step 5: Re-use tag of E, This is an **inplace optimization**

Final Memory Plan

- **count**: ref counter on dependent operations that yet to be full-filled
- **internal arrays, same color indicates shared memory**
- **Tag used to indicate memory sharing on allocation Algorithm.**
- **Box of free tags in allocation algorithm.**

Data dependency, operation not completed

Data dependency, operation completed
Concurrency vs Memory Optimization

Cannot Run in Parallel

\[
\]

- internal arrays
- Memory allocation for result, same color indicates shared memory.

Enables two Parallel Path

\[
\]

- data dependency
- implicit dependency introduced due to allocation
Concurrency aware Heuristics

First the Longest Path

Reset the Reward of visited Node to 0. Find the next longest Path

The final node Color

Restrict memory reuse in the same colored path
Memory Allocation and Scheduling

Introduces implicit control flow dependencies between ops

Solutions:
- Explicitly add the control flow dependencies
  - Needed in TensorFlow
- Enable mutation in the scheduler, no extra job needed
  - Both operation “mutate” the same memory
  - Supported in MXNet
Memory Plan with Gradient Calculation

Back to the Question: Why do we need automatic differentiation that extends the graph instead of backprop in graph?
Memory Plan with Gradient Calculation

Back to the Question: Why do we need automatic differentiation that extends the graph instead of backprop in graph?
Memory Optimization on a Two Layer MLP

Network Configuration:
- input
- fullc-forward
- sigmoid-forward
- fullc-forward
- softmax-forward

Gradient Calculation Graph:
- input
- input-grad
- fullc-forward
- sigmoid-forward
- fullc-forward
- softmax-forward
- log-loss
- label
- fullc-backward
- sigmoid-backward
- fullc-backward
- softmax-backward

A Possible Allocation Plan:
- input
- input-grad
- fullc-forward
- sigmoid-forward
- fullc-forward
- softmax-forward
- log-loss
- label
- fullc-backward
- sigmoid-backward
- fullc-backward
- softmax-backward

Data dependency:
- Memory allocation for each output of op, same color indicates shared memory.
Impact of Memory Optimization in MXNet

![Memory Optimization Graphs](image-url)
We are still Starved

- For training, cost is still linear to the number of layers
- Need to book-keep results for the gradient calculation
Trade Computation with Memory

- Only store a few of the intermediate result
- Recompute the value needed during gradient calculation

Forward

Backward1

Backward2

- Data to be checkpointed for backprop
- Data to be dropped
Computation Graph View of the Algorithm

Network Configuration
- input
- conv-forward
- bn-forward
- relu-forward
- conv-forward
- bn-forward
- relu-forward

Normal Gradient Graph
- input
- input-grad
- conv-forward
- bn-forward
- relu-forward
- conv-forward
- bn-forward
- relu-forward

Memory Optimized Gradient Graph
- input
- input-grad
- conv-forward
- bn-forward
- relu-forward
- conv-forward
- bn-forward
- relu-forward

→ data dependency  --- control dependency

Memory allocation for each output of op, same color indicates shared memory.
Sublinear Memory Complexity

- If we check point every $K$ steps on a $N$ layer network
- The memory cost = $O(K) + O(N/K)$
- We can get $\sqrt{N}$ memory cost plan
- With one additional forward pass (25% overhead)
Alternative View: Recursion

More memory can be saved by multiple level of recursion

Memory allocation for each output of op, same color indicates shared memory.
Comparison of Allocation Algorithm on ResNet

Chen et.al 2016
Comparison of Allocation Algorithm on LSTM

(a) Feature map memory cost estimation

(b) Runtime total memory cost

Chen et.al 2016
Execution Overhead

(a) ResNet  
![Graph showing execution overhead for ResNet with sublinear plan and sharing]

(b) LSTM  
![Graph showing execution overhead for LSTM with sublinear plan and sharing]
Take-aways

- Computation graph is a useful tool for tracking dependencies
- Memory allocation affects concurrency
- We can trade computation for memory to get sublinear memory plan
Assignment 2

- Assignment 1 implements computation graph and autodiff
- Assignment 2 implements the rest of DL system stack (Graph Executor) to run on hardware
  - Shape Inference
  - Memory management
  - TVM-based operator implementation

- **Deadline in two weeks: 5/8/2018**
- **Post questions to #dlsys slack channel so course staff can help**