Lecture 7: Memory Optimization

CSE599G1: Spring 2017
This Thursday

- Assignment 1 Due

- Project proposal pitch, good chance to talk to other folks in class.

- Think about system related perspectives in your projects, talk to us
Where are we

User API
- 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

- **assign**: W
- **sub**: x
- **mul**: W_grad
- **matmul**: softmax
- **softmax**: log
- **mul**: 1 / batch_size
- **mean**: cross_entropy

**Path**:
- assign → W → matmul → softmax → log → mul → mean
- sub → x → matmul → softmax-grad → log-grad → mul
- mul → W_grad → matmul-transpose → softmax-grad → log-grad
- learning_rate → mul
Recap: Automatic Differentiation

Backprop in Graph

![Graph Diagram]

Autodiff by Extending the Graph: assignment 1

![Graph Diagram with Additional Nodes]
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
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 \( \exp(a \times b + 3) \)

- Allocate temp memory for intermediate computation
- Traverse and execute the graph by topo order.
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**

![Memory Pool Diagram]

- Node 'a' with value 4
- Node 'b' with value 8
- Node 'mul' with value 3
- Node 'add-const' with value 35
- Node 'exp' with value exp(35)
Static Memory Planning

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

2. Analog: register allocation algorithm in compiler

Same color represents the 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

\[ B = \text{sigmoid}(A) \]
\[ C = \text{sigmoid}(B) \]
\[ E = \text{Pooling}(C) \]
\[ F = \text{Pooling}(B) \]
\[ G = E \times F \]

**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**

---

**Initial state of allocation algorithm**

**Final Memory Plan**

- **Tag** used to indicate memory sharing on allocation Algorithm.
- **Box** of free tags in allocation algorithm.
- **internal arrays**, same color indicates shared memory.
- **count** ref counter on dependent operations that yet to be full-filled.
- **data dependency**, operation completed.
- **data dependency**, operation not completed.
Concurrency vs Memory Optimization

**Cannot Run in Parallel**


**Enables two Parallel Paths**


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

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
Impact of Memory Optimization in MXNet
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

![Diagram]

- 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
Comparison of Allocation Algorithm on ResNet

(a) Feature map memory cost estimation

(b) Runtime total memory cost

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

(b) LSTM

sublinear plan
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