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

High level Packages

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

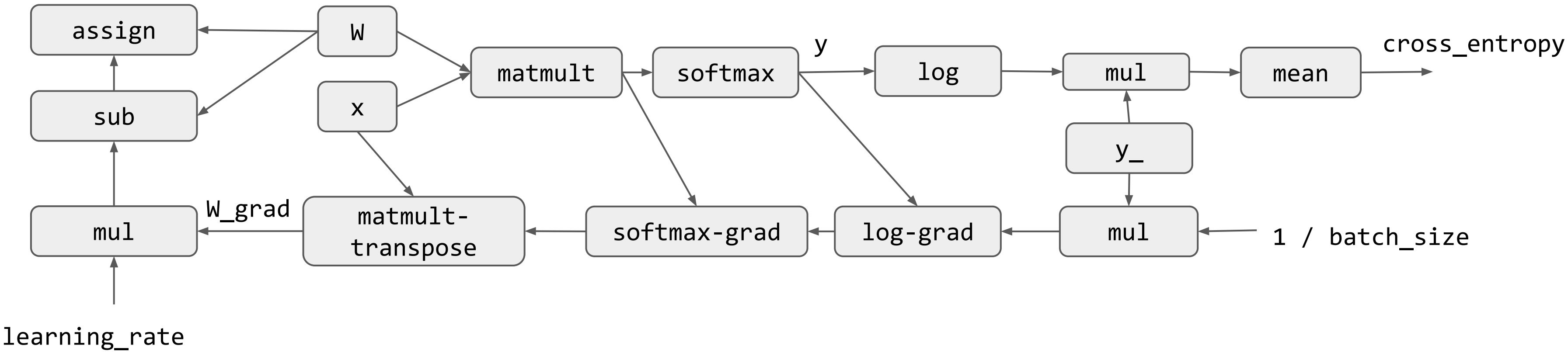
Where are we

- Programming API
- Gradient Calculation (Differentiation API)

- Computational Graph Optimization and Execution
- Runtime Parallel Scheduling

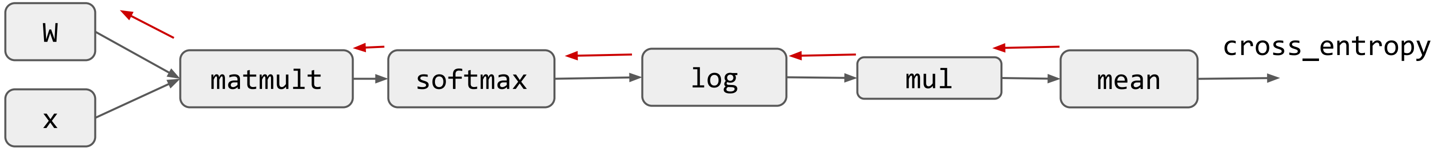
- GPU Kernels, Optimizing Device Code
- Accelerators and Hardwares

Recap: Computation Graph

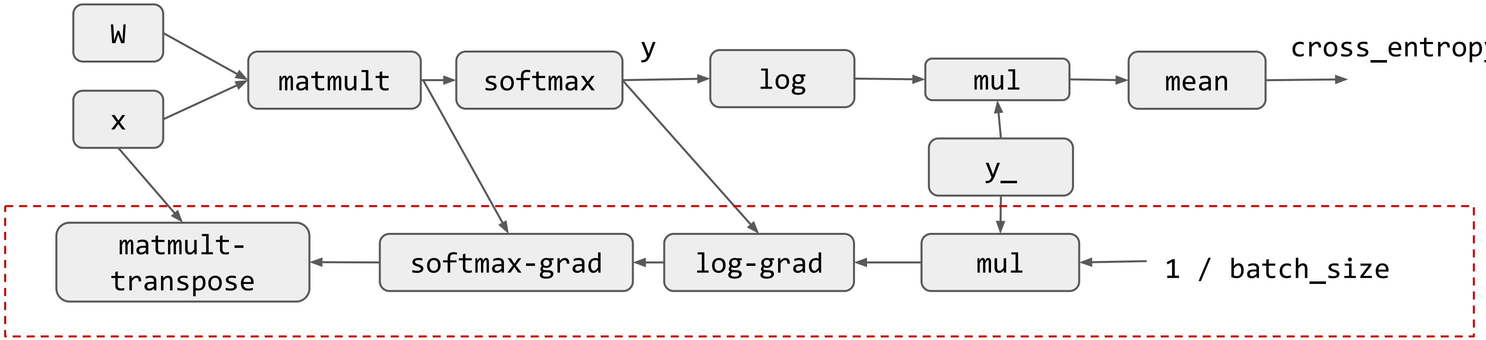


Recap: Automatic Differentiation

Backprop in Graph

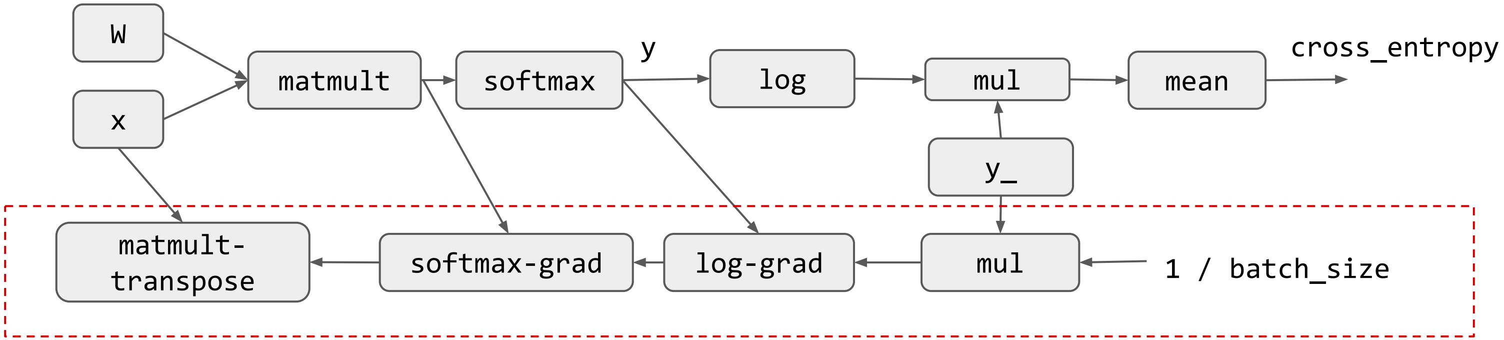


Autodiff by Extending the Graph: assignment 1



Questions for this Lecture

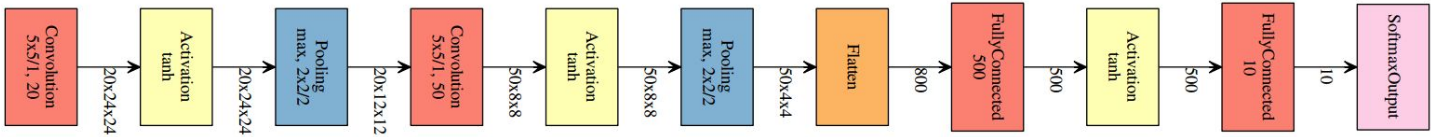
Why do we need automatic differentiation that extends the graph instead of backprop in graph?



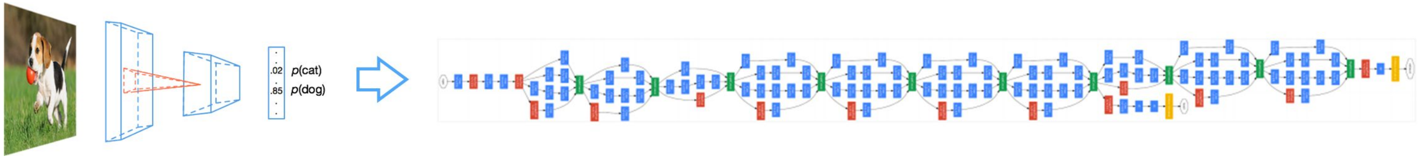
Memory Problem of Deep Nets

Deep nets are becoming deeper

LeNet



Inception



State-of-Art Models can be Resource Bound

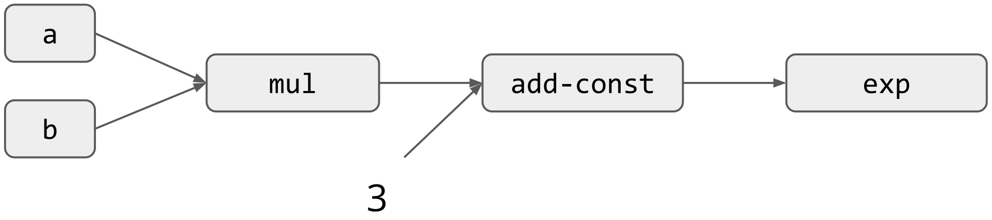
- Examples of recent state of art neural nets
  - Convnet: ResNet-1k on CIFAR-10, ResNet-200 on ImageNet
  - Recurrent models: LSTM on long sequences like speech

- 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 * b + 3)`

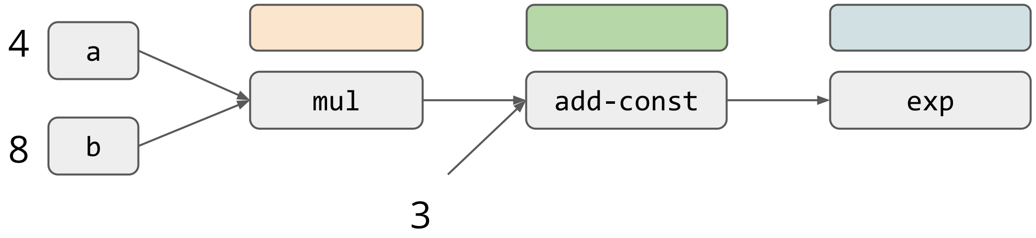


Build an Executor for a Given Graph

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

Computational Graph for `exp(a * b + 3)`

Same color represent same piece of memory



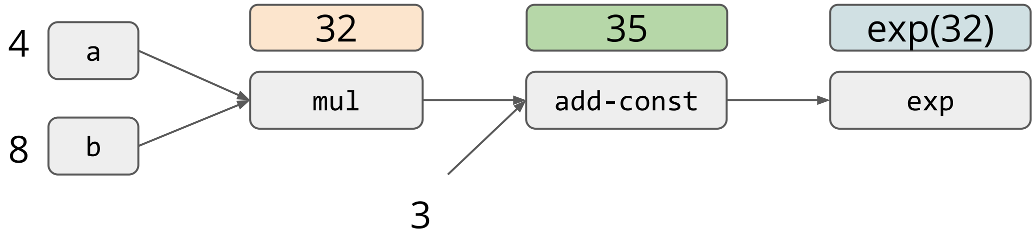
PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

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 * b + 3)`



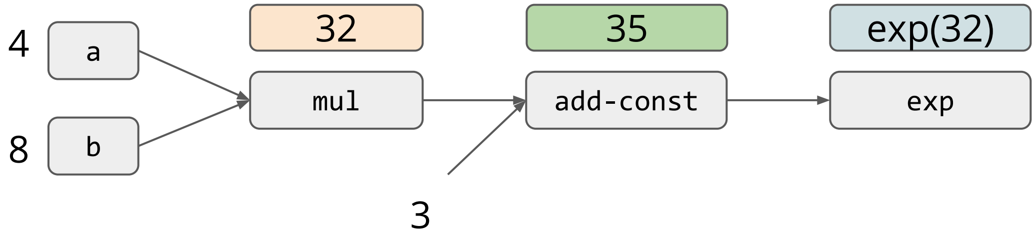
Build an Executor for a Given Graph

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 * b + 3)`



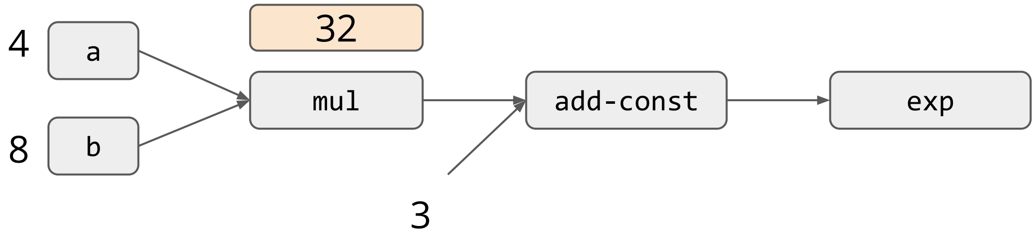
Dynamic Memory Allocation

1. **Allocate** when needed

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

3. Useful for both declarative and imperative executions

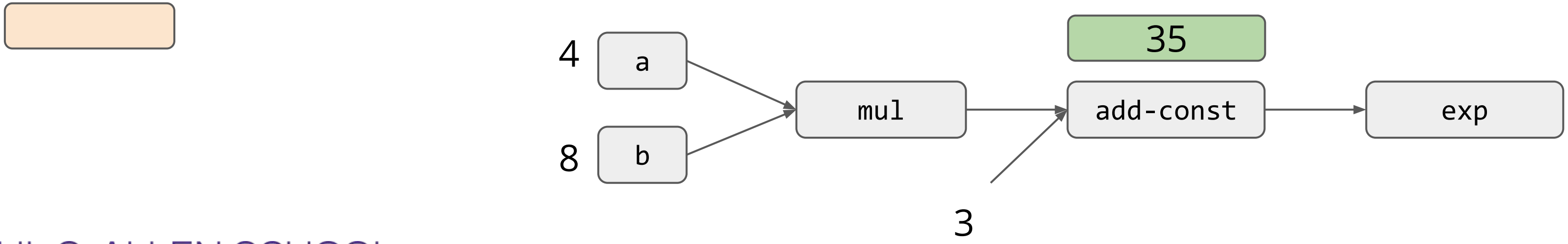
Memory Pool



Dynamic Memory Allocation

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

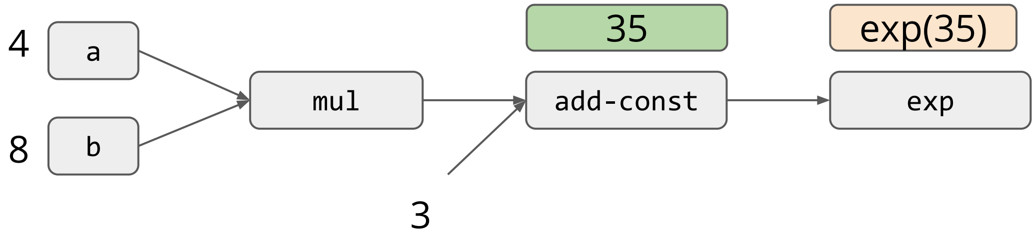
Memory Pool



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

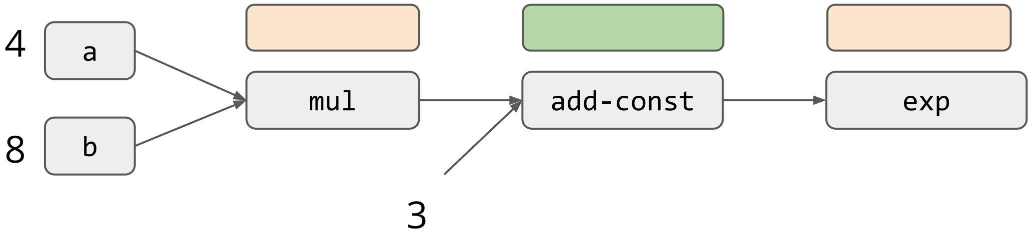


Static Memory Planning

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

2. Analog: register allocation algorithm in compiler

Same color represent same piece of memory



PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Common Patterns of Memory Planning

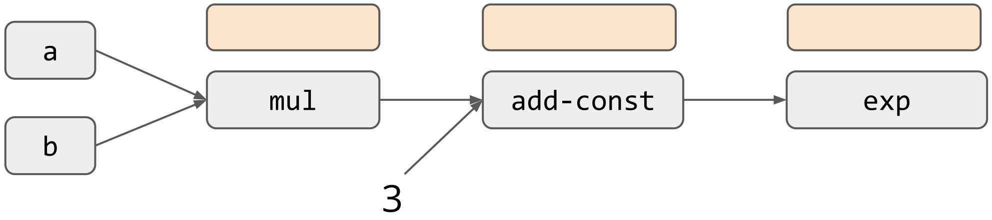
- **Inplace** store the result in the input
- **Normal Sharing** reuse memory that are no longer needed.

Inplace Optimization

- Store the result in the input
- Works if we only care about the final result

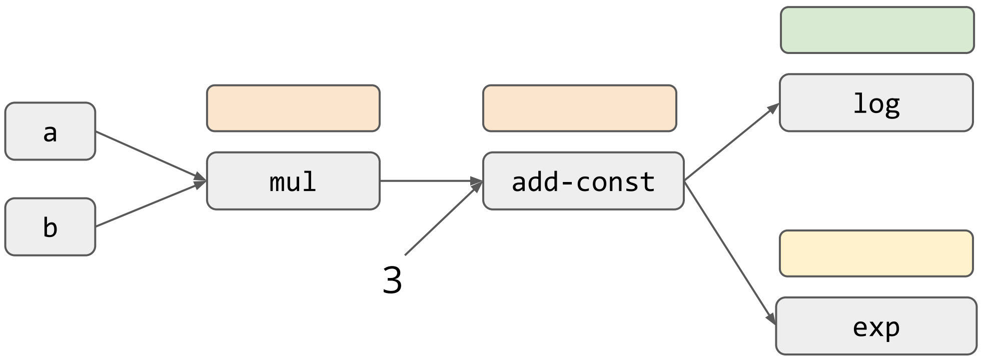
- Question: what operation cannot be done inplace ?

Computational Graph for `exp(a * b + 3)`



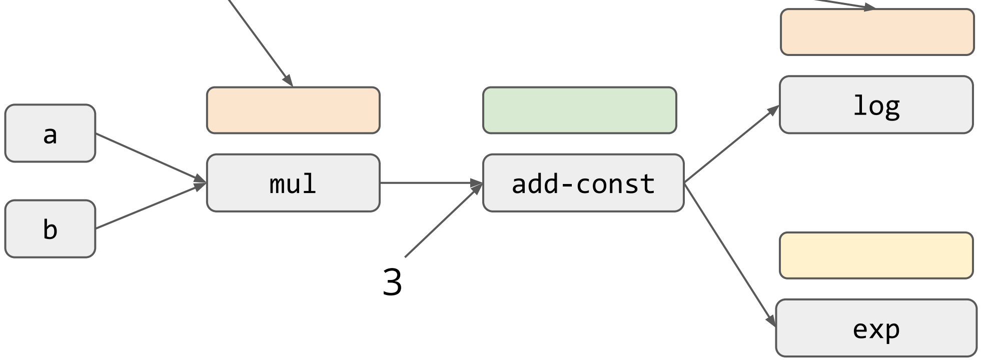
Inplace Pitfalls

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

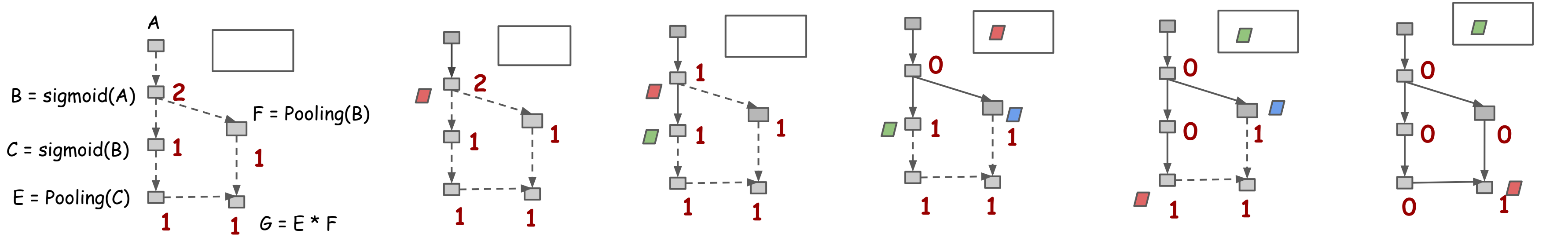


Normal Memory Sharing

Recycle memory that is no longer needed.



Memory Planning Algorithm



Initial state of allocation algorithm

step 1: Allocate tag for B

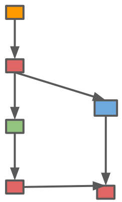
step 2: Allocate tag for C, **cannot do inplace** because B is still alive

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



internal arrays, same color indicates shared memory.

count ref counter on dependent operations that yet to be full-filled

data dependency, operation completed

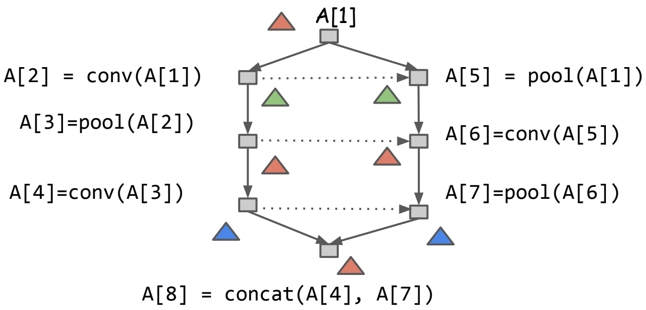
data dependency, operation not completed

Tag used to indicate memory sharing on allocation Algorithm.

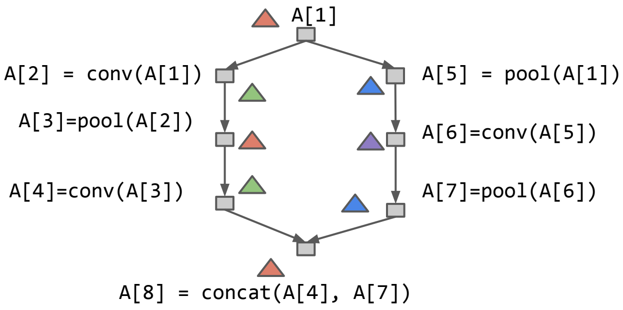
Box of free tags in allocation algorithm.

Concurrency vs Memory Optimization

Cannot Run in Parallel



Enables two Parallel Path



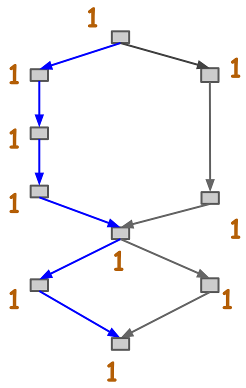
internal arrays

Memory allocation for result, same color indicates shared memory.

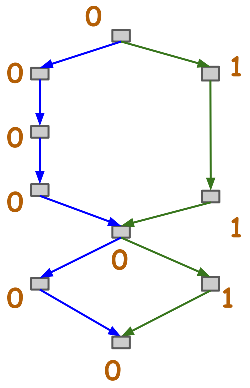
data dependency

implicit dependency introduced due to allocation

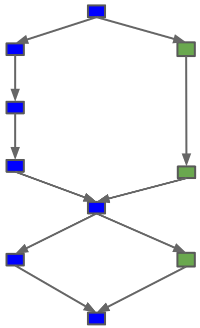
Concurrency aware Heuristics



First the
Longest Path



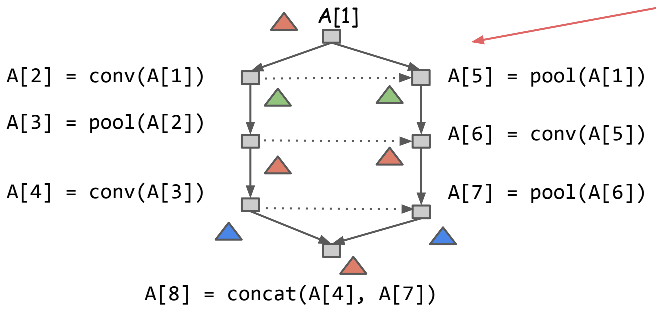
Reset the Reward of
visited Node to 0. Find
the next longest Path



The final node Color

Restrict memory reuse in the same colored path

Memory Allocation and Scheduling



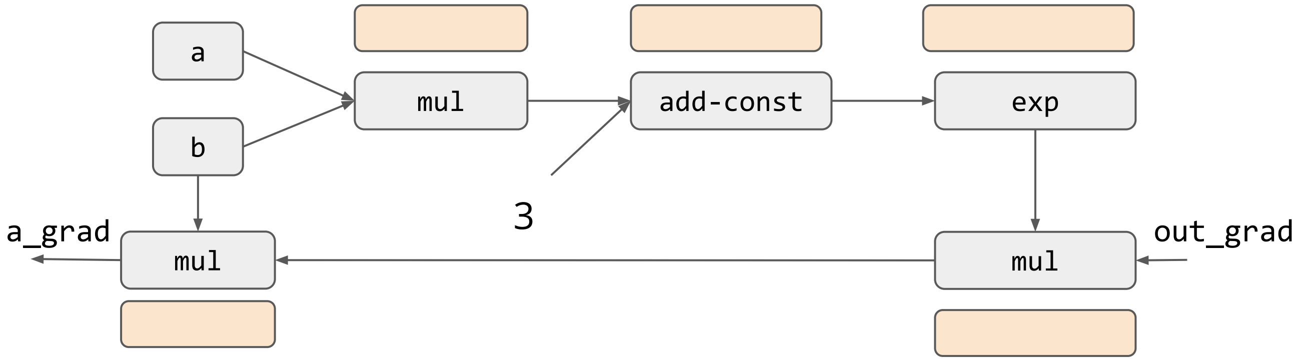
Introduces implicit control flow dependencies between ops

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

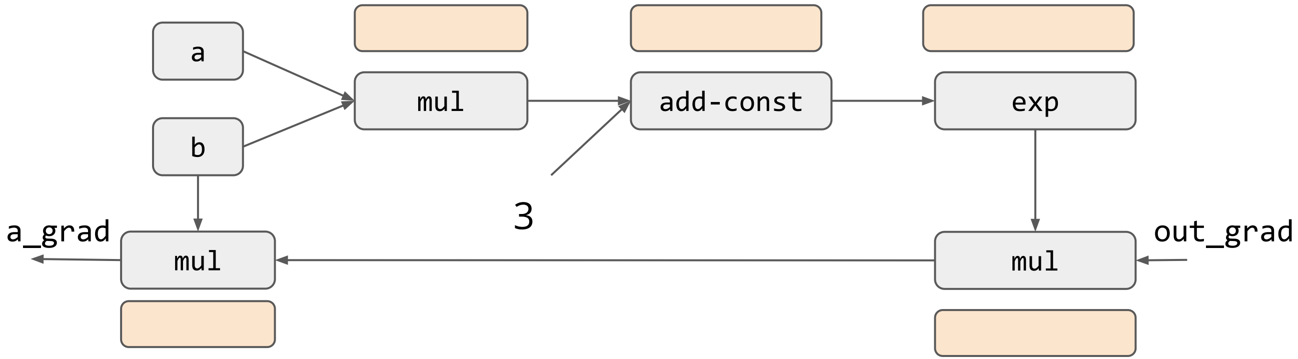
Memory Plan with Gradient Calculation

Back to the Quesion: Why do we need automatic differentiation that extends the graph instead of backprop in graph?

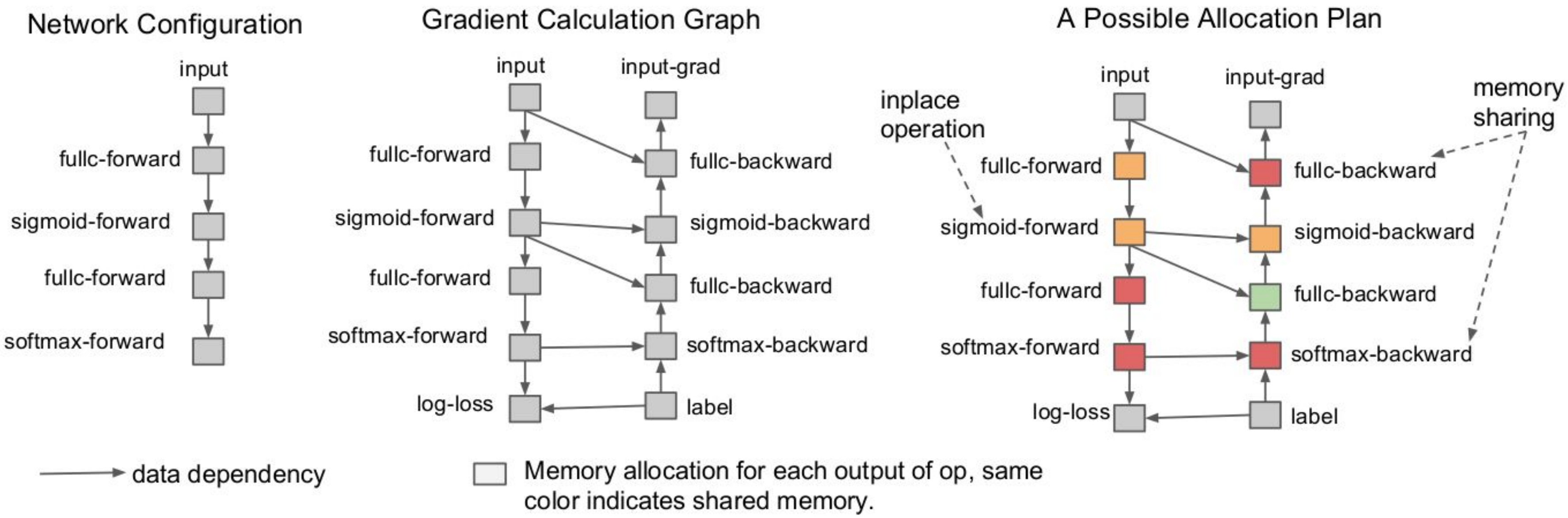


Memory Plan with Gradient Calculation

Back to the Question: Why do we need automatic differentiation that extends the graph instead of backprop in graph?

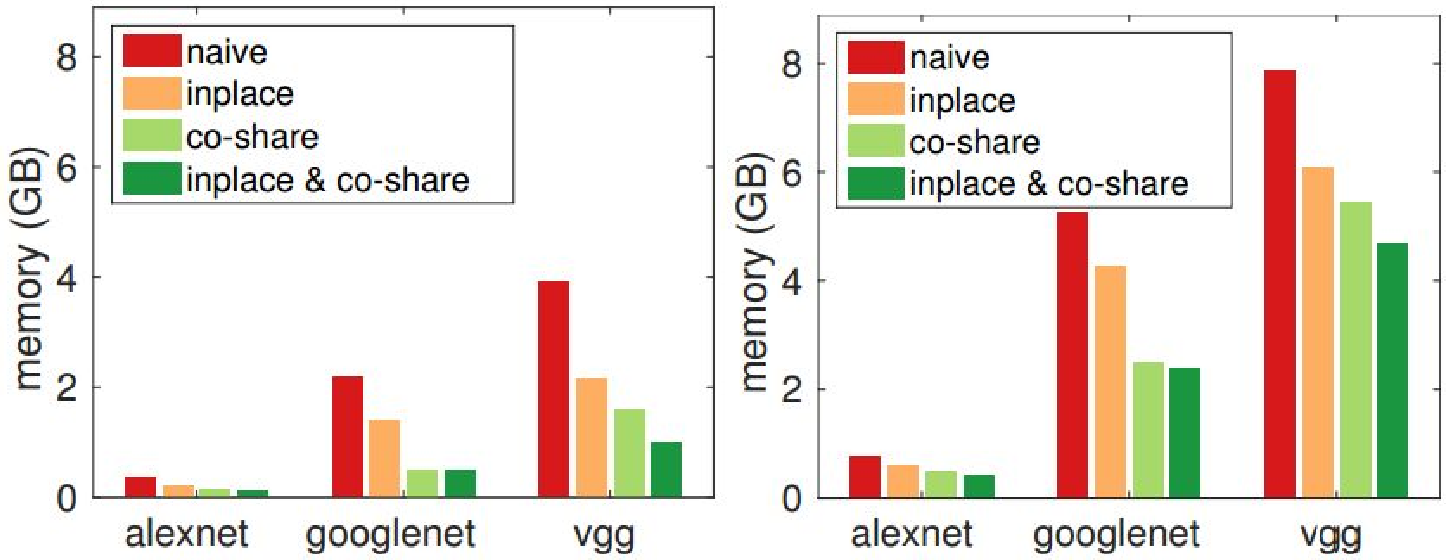


Memory Optimization on a Two Layer MLP



PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

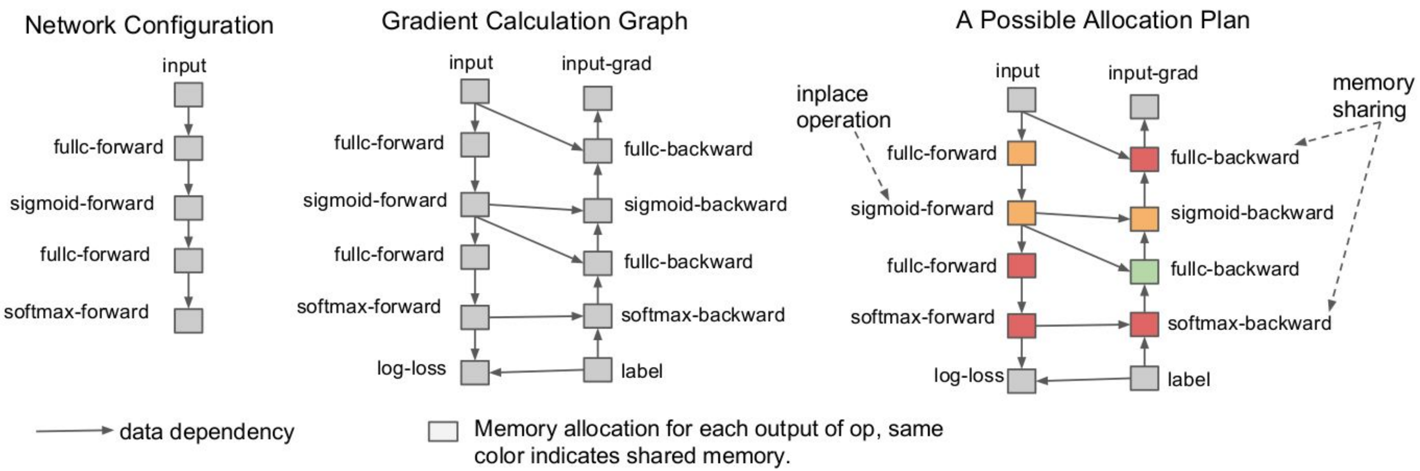
Impact of Memory Optimization in MXNet



PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

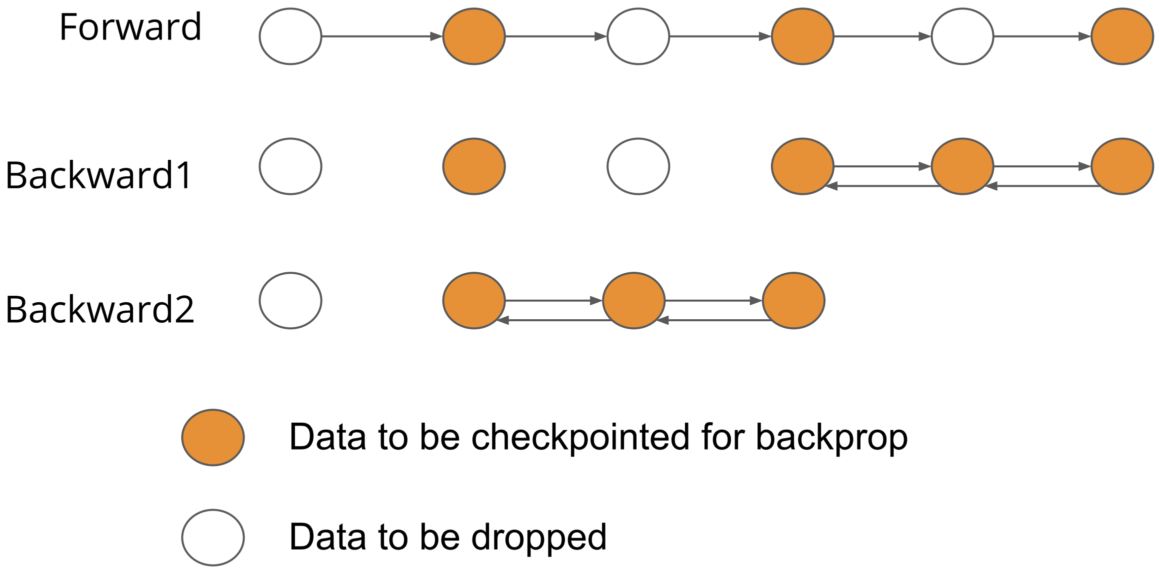
We are still Starved

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

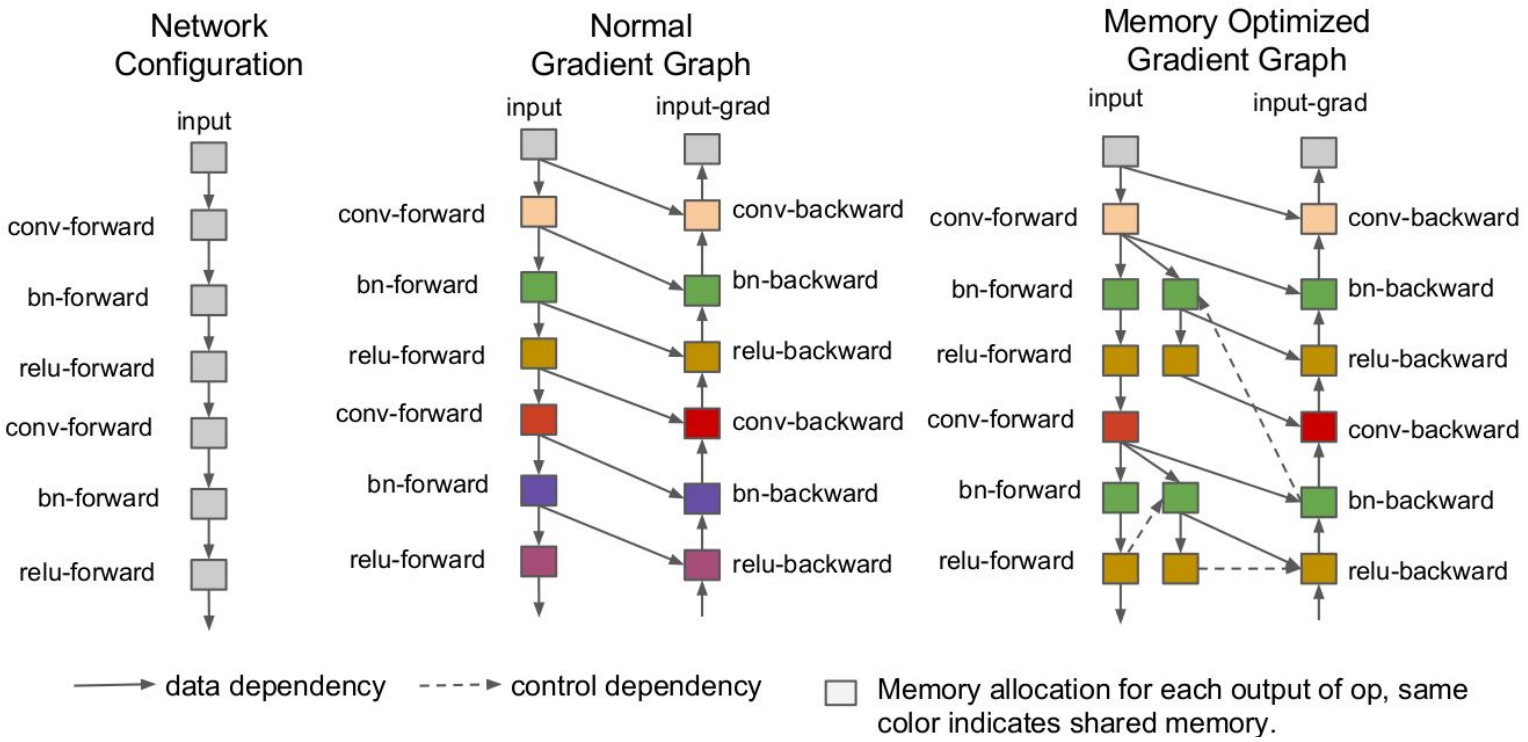


Trade Computation with Memory

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



Computation Graph View of the Algorithm



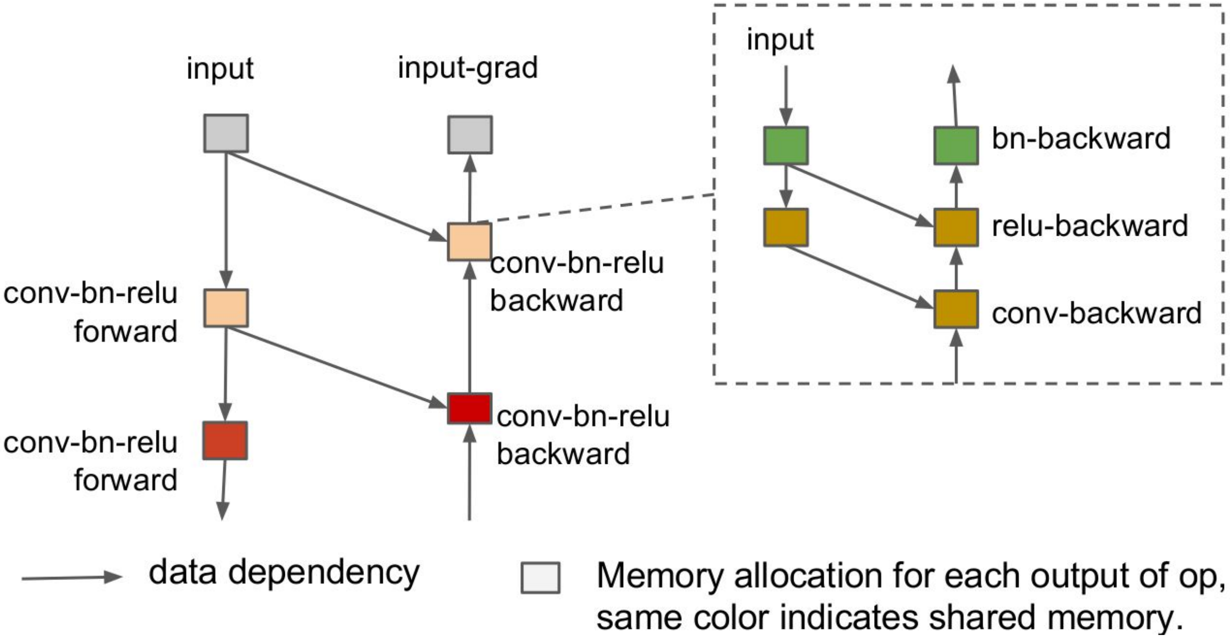
Sublinear Memory Complexity

- If we check point every K steps on a N layer network
- The memory cost = O(K) + O(N/K)

Cost per segment **Cost to store results**

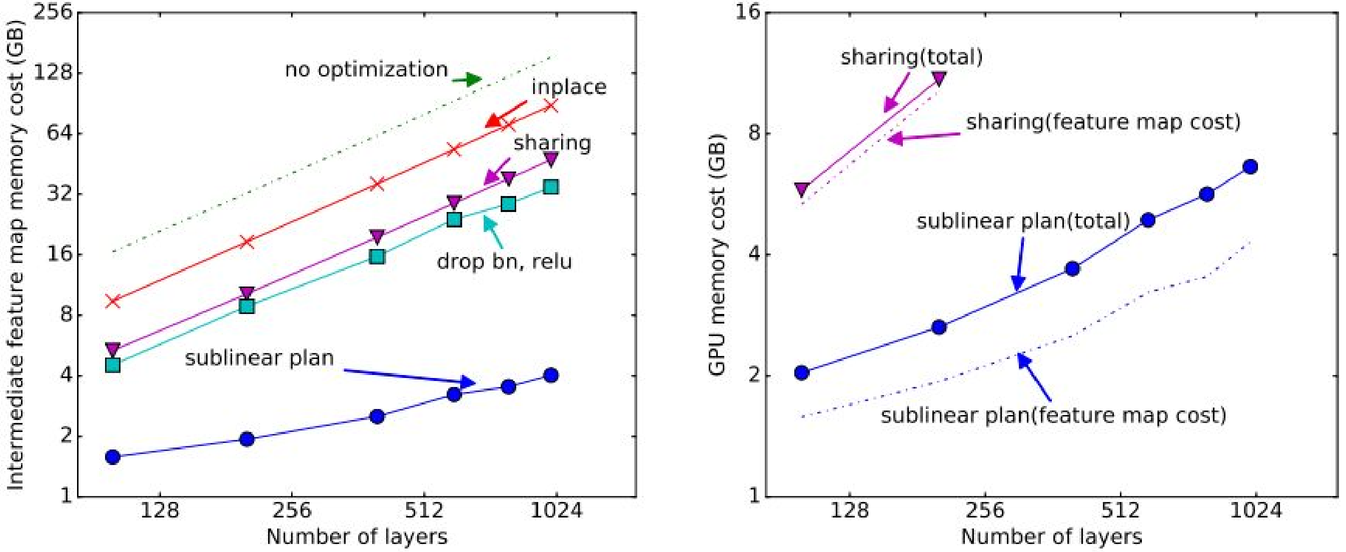
- We can get sqrt(N) memory cost plan
- With one additional forward pass(25% overhead)

Alternative View: Recursion



More memory can be saved by multiple level of recursion

Comparison of Allocation Algorithm on ResNet

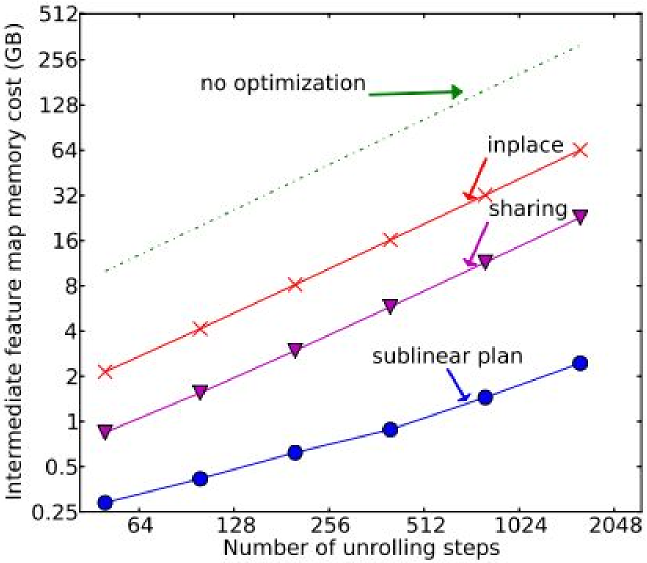


(a) Feature map memory cost estimation

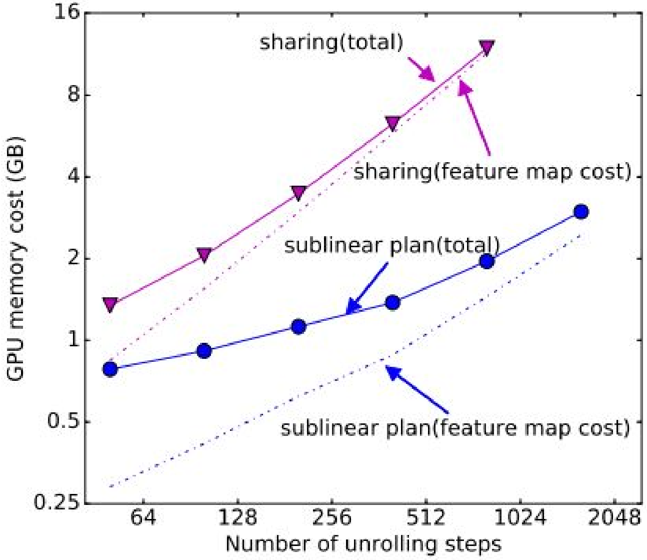
(b) Runtime total memory cost

Chen et.al 2016

Comparison of Allocation Algorithm on LSTM



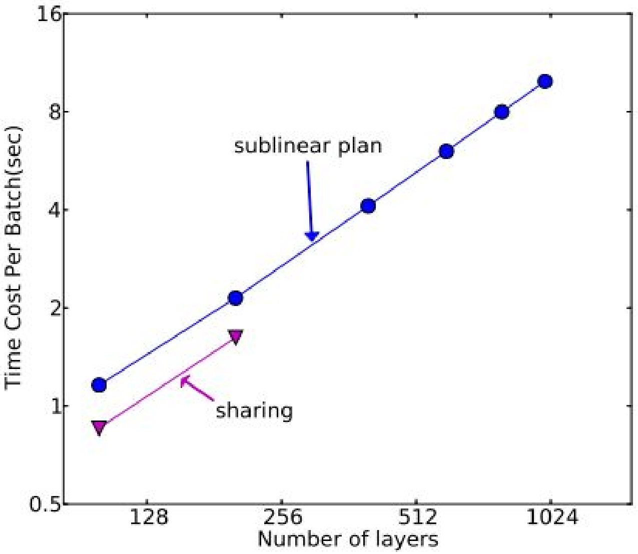
(a) Feature map memory cost estimation



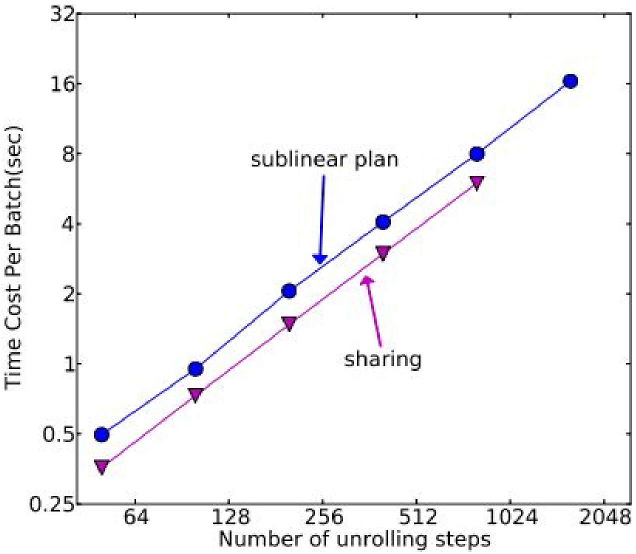
(b) Runtime total memory cost

Chen et.al 2016

Execution Overhead



(a) ResNet



(b) LSTM

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