Training Deep Nets with Sublinear Memory Cost

Tianqi Chen, Bing Xu, Chiyuan Zhang, Carlos Guestrin

15-849 Presentation
Senyu Tong 2/28/2022

Overview

Task: Reduce the memory consumption to train a DL model.

Main Challenge: Cost to store intermediate results and gradients.

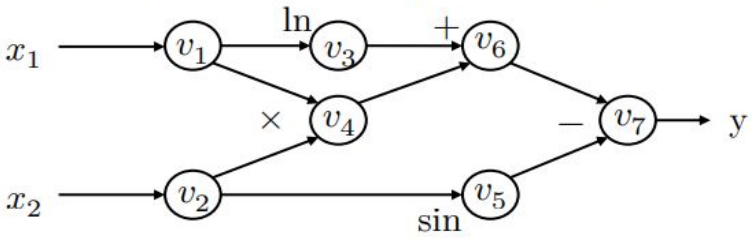
Background: Reverse Mode Auto Differentiation

Key Ideas:

- Computation Graph Memory Optimization (In-place Storing, Memory Sharing)
- Save Memory by Re-computing (Gradient Checkpointing)

Background: Reverse Mode AD

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin x_2$$



Forward evaluation trace

$$\begin{aligned}
v_1 &= x_1 = 2 \\
v_2 &= x_2 = 5 \\
v_3 &= \ln v_1 = \ln 2 = 0.693 \\
v_4 &= v_1 \times v_2 = 10 \\
v_5 &= \sin v_2 = \sin 5 = -0.959 \\
v_6 &= v_3 + v_4 = 10.693 \\
v_7 &= v_6 - v_5 = 10.693 + 0.959 = 11.652 \\
y &= v_7 = 11.652
\end{aligned}$$

Define adjoint $\overline{v_i} = \frac{\partial y}{\partial v_i}$

We can then compute the $\overline{v_i}$ iteratively in the **reverse** topological order of the computational graph

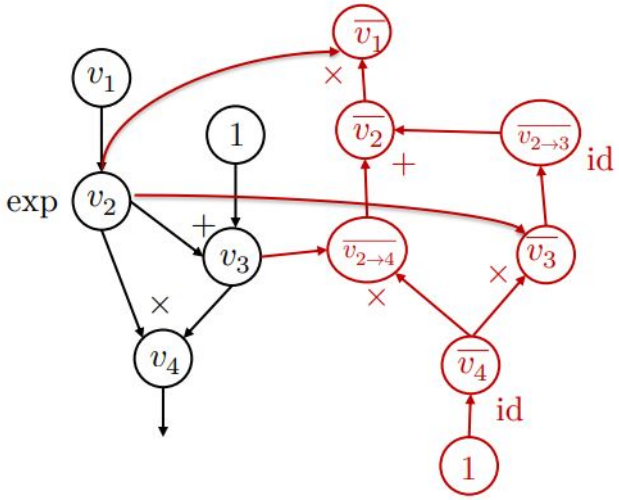
Reverse AD evaluation trace

$$\begin{aligned}
\overline{v_7} &= \frac{\partial y}{\partial v_7} = 1 \\
\overline{v_6} &= \overline{v_7}\frac{\partial v_7}{\partial v_6} = \overline{v_7} \times 1 = 1 \\
\overline{v_5} &= \overline{v_7}\frac{\partial v_7}{\partial v_5} = \overline{v_7} \times (-1) = -1 \\
\overline{v_4} &= \overline{v_6}\frac{\partial v_6}{\partial v_4} = \overline{v_6} \times 1 = 1 \\
\overline{v_3} &= \overline{v_6}\frac{\partial v_6}{\partial v_3} = \overline{v_6} \times 1 = 1 \\
\overline{v_2} &= \overline{v_5}\frac{\partial v_5}{\partial v_2} + \overline{v_4}\frac{\partial v_4}{\partial v_2} = \overline{v_5} \times \cos v_2 + \overline{v_4} \times v_1 = -0.284 + 2 = 1.716 \\
\overline{v_1} &= \overline{v_4}\frac{\partial v_4}{\partial v_1} + \overline{v_3}\frac{\partial v_3}{\partial v_1} = \overline{v_4} \times v_2 + \overline{v_3}\,\frac{1}{v_1} = 5 + \frac{1}{2} = 5.5
\end{aligned}$$

Background: Extending Computation Graph

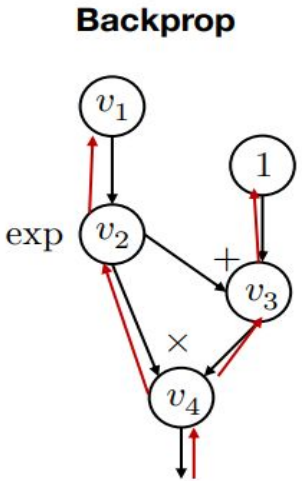
def gradient(out):
  node_to_grad = {out: [1]}
  for i in reverse_topo_order(out):
    $\overline{v_i} = \sum_j \overline{v_{i \to j}}$ = sum(node_to_grad[i])
    for $k \in inputs(i)$:
      compute $\overline{v_{k \to i}} = \overline{v_i} \frac{\partial v_i}{\partial v_k}$
      append $\overline{v_{k \to i}}$ to node_to_grad[k]
  return adjoint of input $\overline{v_{input}}$

$i = 2$
node_to_grad: {
  1: $[\overline{v_1}]$
  2: $[\overline{v_{2 \to 4}}, \overline{v_{2 \to 3}}]$
  3: $[\overline{v_3}]$
  4: $[\overline{v_4}]$
}



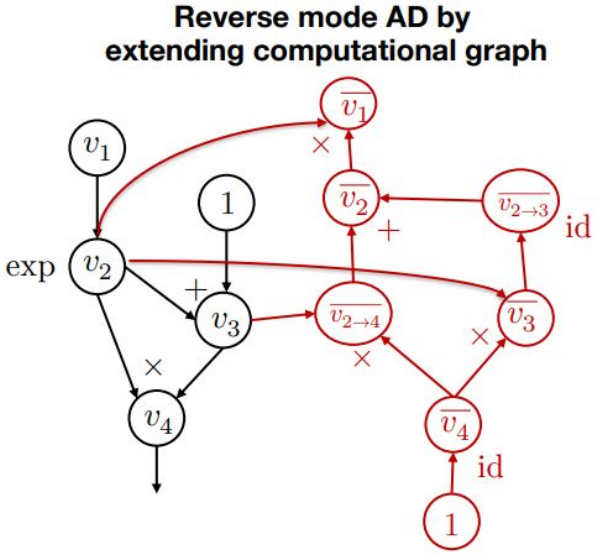
NOTE: id is identity function

Background: Reverse Mode AD vs Backprop



- Run Backward operations on the same forward graph.

- Used in first generation deep learning frameworks (caffe, cuda-convnet) and torch



- Construct separate graph nodes for Adjoints

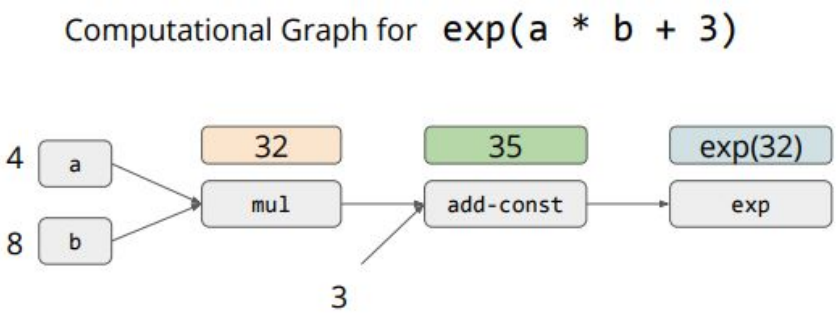
- Used by modern DL frameworks (CGT, MXNets)

Advantages of Using **Explicit Gradient Path**:

- Clearly describes computation dependency for better memory managements;
- Ability to have a different backward path (multiple out-path, gradient aggregation instead of introducing an explicit split layer.
- Ability to calculate higher order gradients.

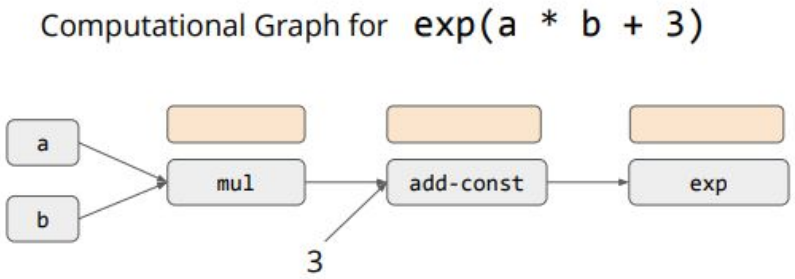
Memory Allocation: Graph Executor

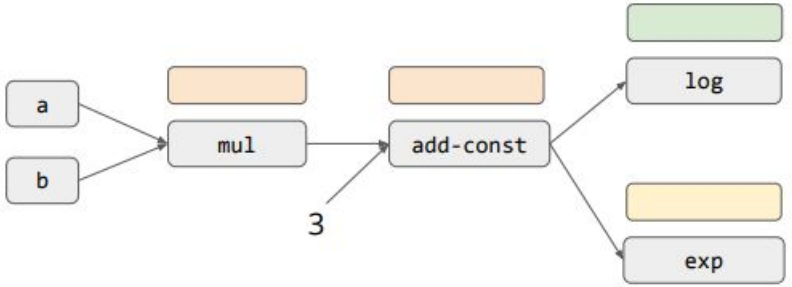
- allocate temp memory for intermediate computation
- traverse and execute the graph by topological order.
- Temporary space Linear to number of ops)



Memory Optimization: Inplace Optimization

- Store the result in the input;
- Works only if the result operation is the only consumer of the current value.

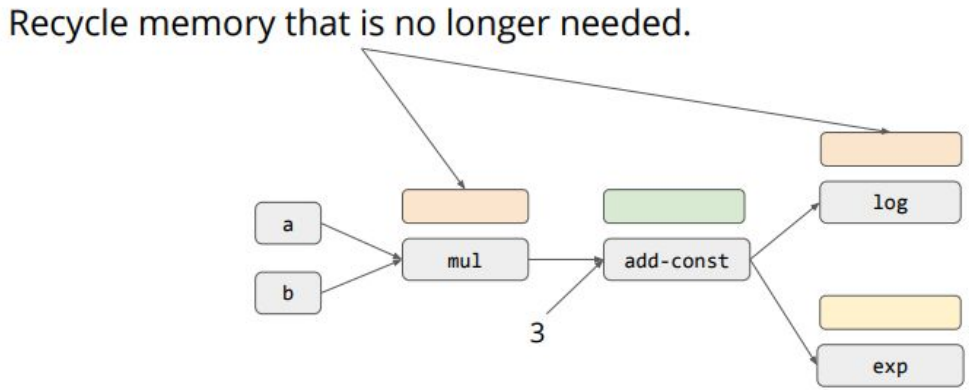




Nodes with same color share memory.

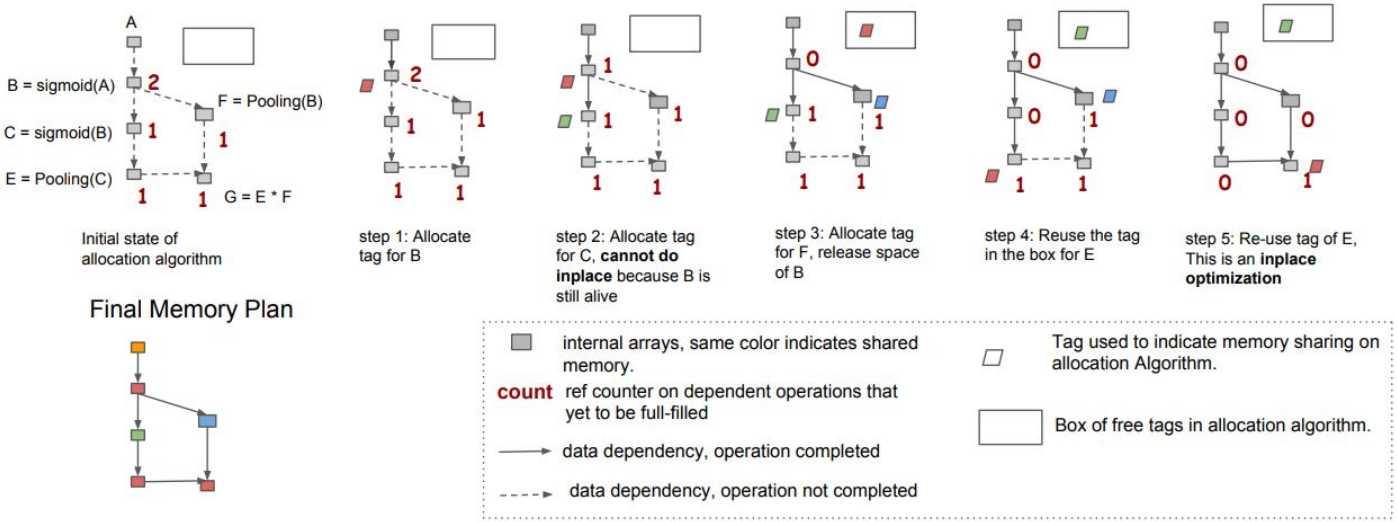
Memory Optimization: Memory Sharing

- Memory used by intermediate results that are no longer needed can be recycled and used in another node;
- Share memory between the nodes whose lifetime do not overlap.



Memory Allocation Algorithm

- Static Memory Planning: plan for reuse **ahead of time**.
- Usage of liveness **counters** counting operations to be full-filled.
- Important to declare minimum dependencies.



Gradient Checkpointing

We can save memory usage by a **constant factor** using memory sharing, how can we do better?

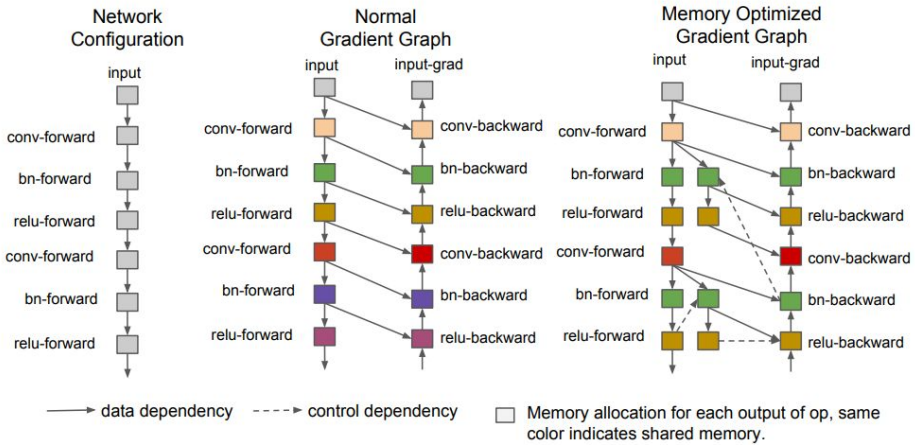
Idea: Drop some of the intermediate results and recover them from an extra forward computation (from the closest recorded results) when needed.

Algorithm:

1. Divide the network into several segments;
2. store the output of each segment and drops all the intermediate results within each segment;
3. recompute dropped results at segment-level during back-propagation.

Gradient Graph Construction Algorithm

- Mirror Count: how many times a result can be recomputed $m : \mathcal{V} \to \mathbb{N}$
- $\forall v, m(v) = 0$, normal gradient graph (O(n) memory, O(n) computation)
- $\forall v, m(v) = 1$, memory poor, O(1) memory O(n^2) computation
- Set “checkpoints” m = 1, and others m= 0.



Sublinear Memory Cost

- Drop the results of low cost operations. E.g. In a Conv-BatchNorm-Activation, we drop result of the batch norm, activation and pooling but keep the result of convolution.
- We can divide the n network into k segments so we have

$$\text{Cost-total} = \max_i \text{cost-of-segment}(i) + O(k) = O(\tfrac{n}{k}) + O(k)$$

Setting $k = \sqrt{n}$ we get $O(2\sqrt{n})$ with only an additional forward pass

Using a Memory Budget

Problem: the memory cost of each layer is not the same so we can't simply set $k = \sqrt{n}$

Solution: Greedy allocation with a given memory budget.

Intuition: Static memory allocation gives an exact memory cost, perform a heuristic search over the bucket to find optimal memory plan.

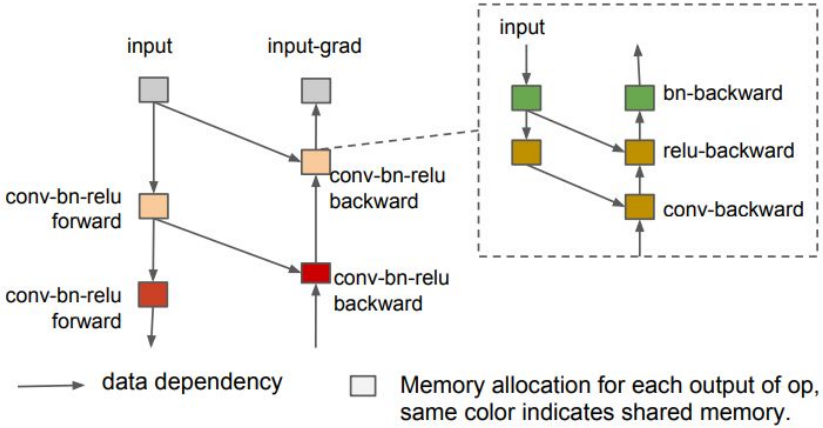
Algorithm 3: Memory Planning with Budget

Input: $G = (V, \text{pred})$, input computation graph.
Input: $C \subset V$, candidate stage splitting points, we will search splitting points over $v \subset C$
Input: B, approximate memory budget. We can search over B to optimize the memory allocation.

$temp \leftarrow 0, x \leftarrow 0, y \leftarrow 0$
for v *in topological-order*(V) **do**
  $temp \leftarrow temp +$ size-of-output(v)
  if $v \in C$ *and* $temp > B$ **then**
    $x \leftarrow x +$ size-of-output(v), $y \leftarrow max(y, temp)$
    $m(v) = 0, temp \leftarrow 0$
  else
    $m(v) = 1$
  end
end
Output: x approximate cost to store inter-stage feature maps
Output: y approximate memory cost for each sub stage
Output: m the mirror plan to feed to Alg. 2

More General View: Recursion and Subroutine

- View each segment as a **subroutine** that combines all the operations inside the segment;
- We can recursively apply memory optimization scheme to each subgraph.



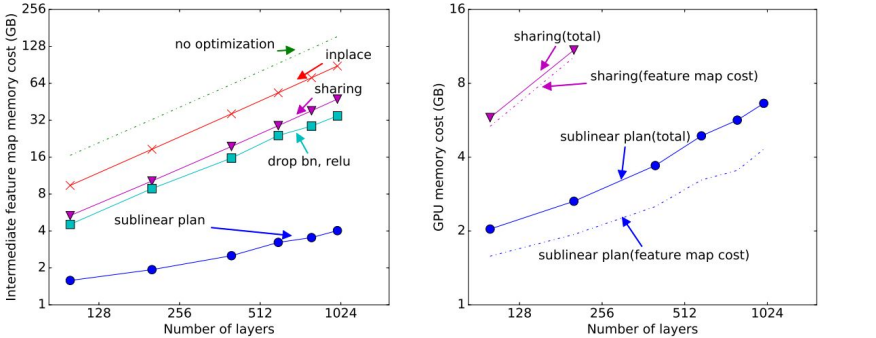
Let g(n) be the memory cost, we have

$$g(n) = k + g\left(n/(k+1)\right)$$

$$g(n) = k \log_{k+1}(n)$$

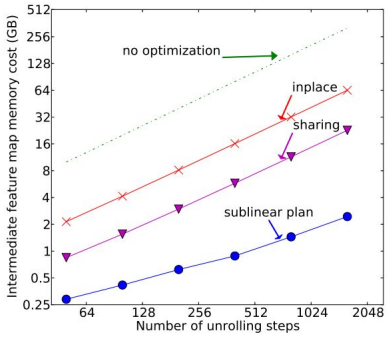
Results: Memory Cost

Running different schemes on top of MXNet on ResNet and LSTM.

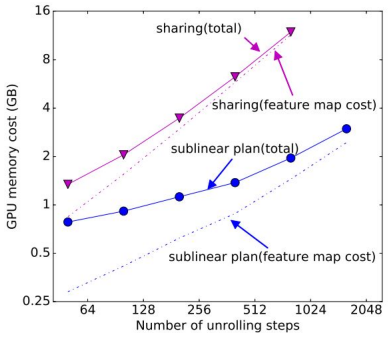


(a) Feature map memory cost estimation

(b) Runtime total memory cost



(a) Feature map memory cost estimation

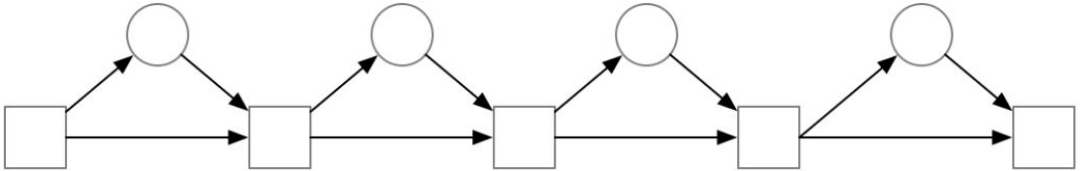


(b) Runtime total memory cost

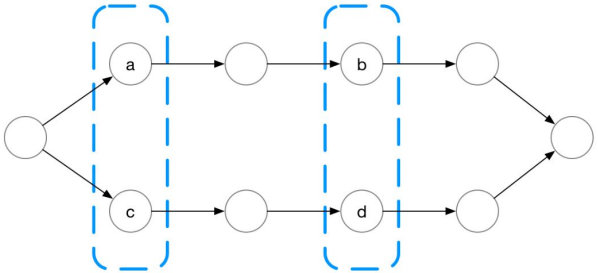
Training time roughly increase 30%.

Problem remains: How to select checkpoints

In a resnet architecture as below, it’s better to select square nodes than circle nodes.



In a multi-tower architecture, can’t select a single node as the checkpoint.



Conclusion

- Reverse Mode AD: explicit gradient path provides space for memory optimization;
- Inplace storage and Memory sharing in allocation;
- Gradient Checkings: Reduce memory cost to sublinear by re-computing;

Discussion Questions

1. When there are parallel operations, how can we trade **resource sharing and parallelization** when we allocate memory?
2. When the model is too large to fit on a single device even with sublinear cost, we have to split the model to different devices. How to share memory when there are communications and dependencies **across devices**?
3. In gradient checkpointing, we have to **manually select the checkpoints.** How can we make this automated?

References

Some slides are adapted from Tianqi Chen's lectures

CMU 10-714, https://dlsyscourse.org/slides/4-automatic-differentiation.pdf

UW CSE599W, https://dlsys.cs.washington.edu/pdf/lecture9.pdf

Medium post:
https://medium.com/tensorflow/fitting-larger-networks-into-memory-583e3c758ff9

MXNet tutorial, https://mxnet.apache.org/versions/1.4.1/architecture/note_memory.html

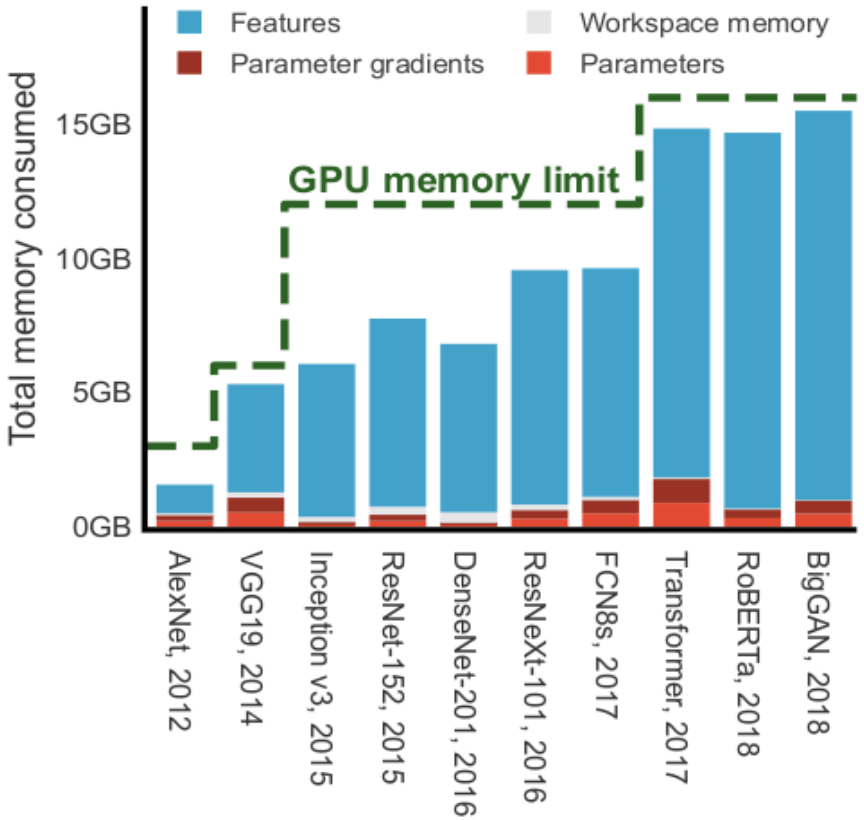
PITTSCI
School of Computing and Information

Checkmate: Breaking the Memory Wall with Optimal Tensor Rematerialization

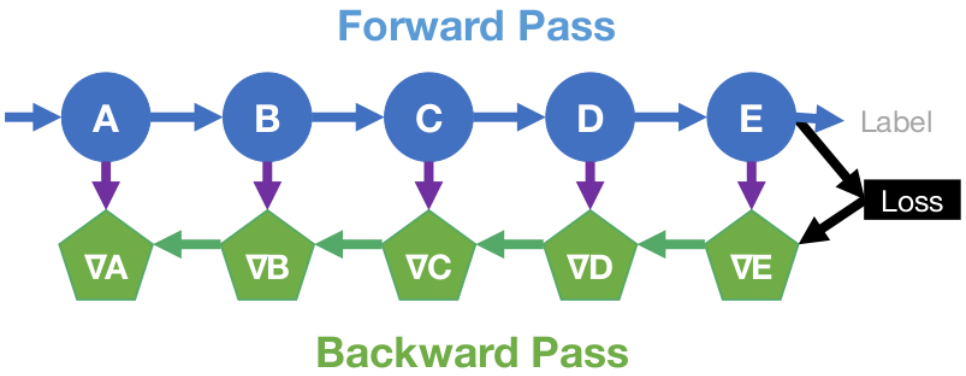
Erhu He
University of Pittsburgh
February 28, 2022

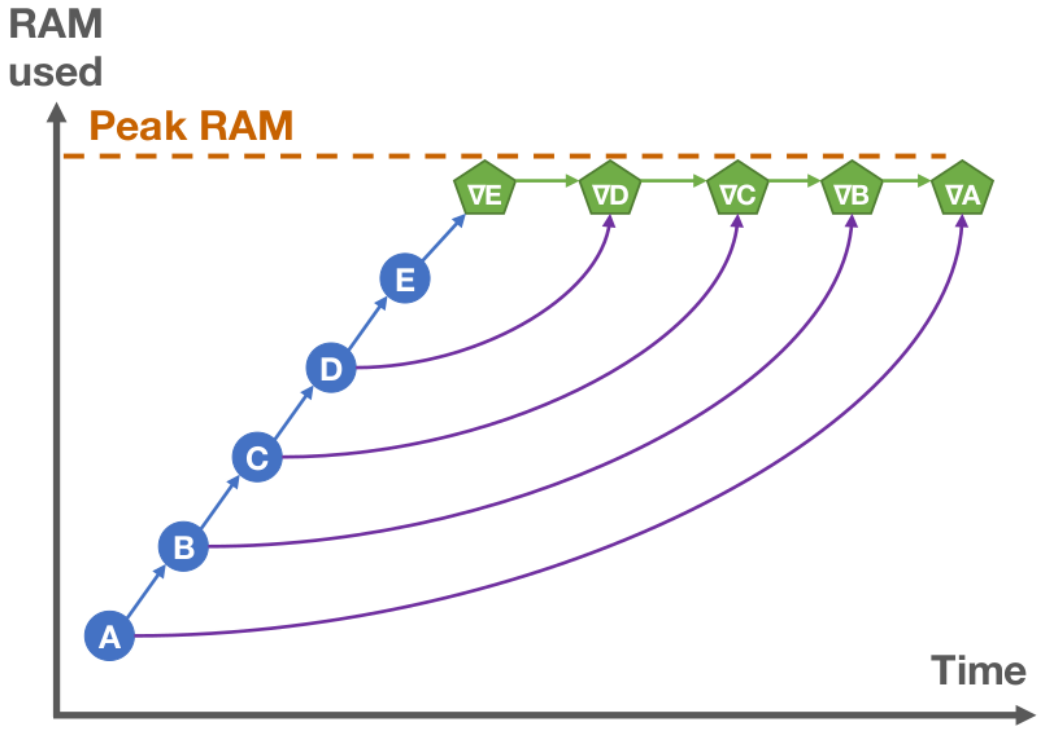
Motivation

- **Emerging trend:**
  - Rapid growth in model size.
- **Problem:**
  - Limited GPU memory is slowing progress in new deep learning models.
  - Most memory is used by activations, not graph parameters.
- Method:
  - Tensor rematerialization.



Traditional backpropagation policy

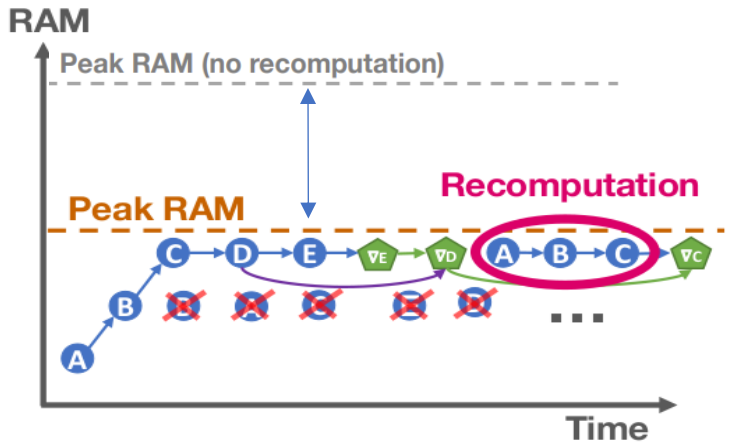




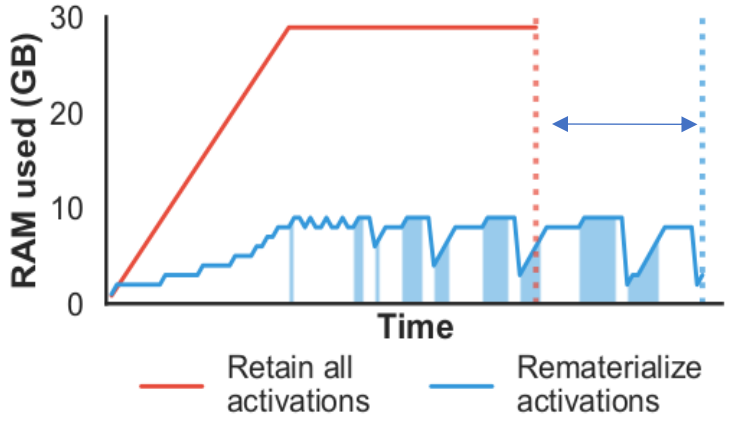
Paras Jain(2020). Checkmate: Breaking the Memory Wall with Optimal Tensor Rematerialization [PowerPoint slides]. Department of EECS, UC Berkeley. https://www.parasjain.com/projects/20checkmate/checkmate_mlsys_slides.pdf

Rematerialization

- **Recomputing intermediate values rather than retaining them in memory**
  - introduce extra computation cost.

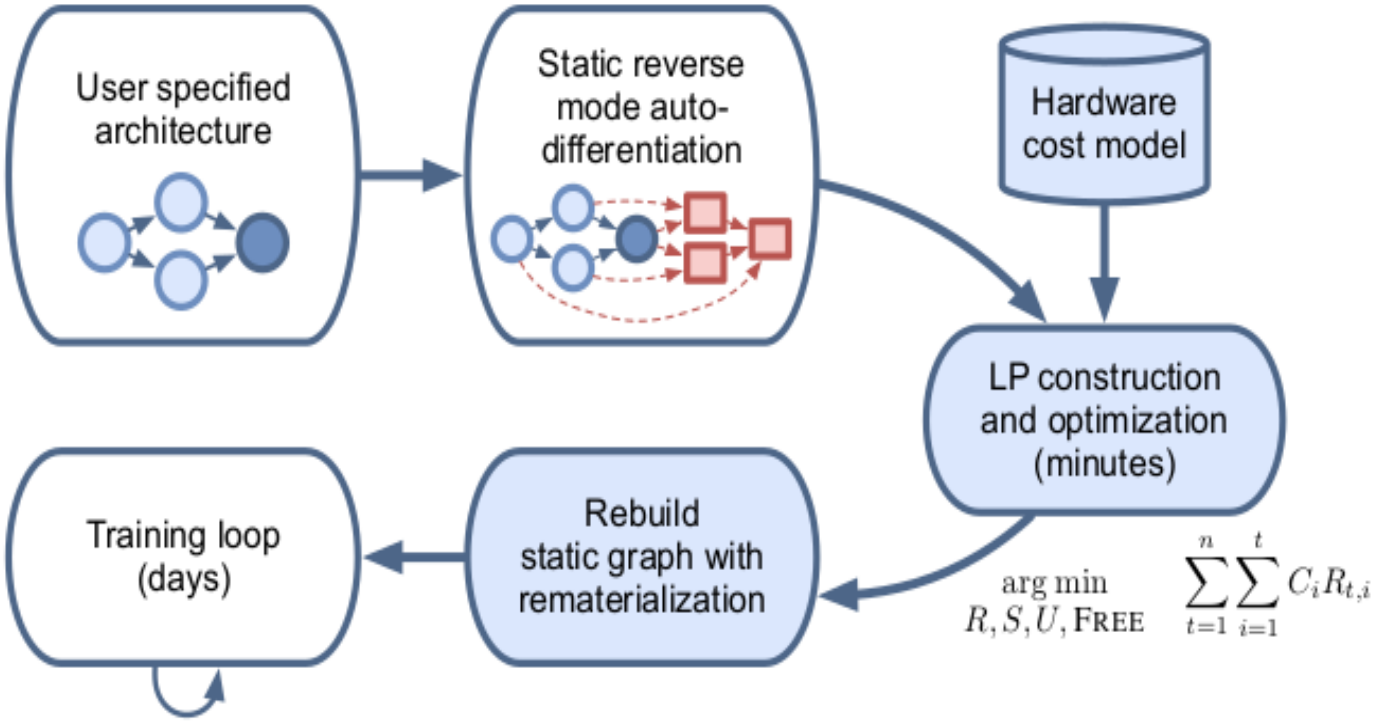


- **Goal:**
  - minimizing computation or execution time while guaranteeing that the rematerialization schedule doesn't exceed device memory limitation.



Overview

Checkmate is a system for optimal tensor rematerialization



Optimal solver - Integer linear program

Minimize the forward and backward cost

$$\underset{R,S,U}{\operatorname{argmin}} \sum \sum C_i R_{t,i}$$

Use the matrix R to create linear objective

Decision variables

$S_{t,i} \in \{0,1\}$	Layer i stored for stage t
$R_{t,i} \in \{0,1\}$	Layer i (re)computed in stage t

Optimal solver - Integer linear program

$$\underset{R,S,U}{\operatorname{argmin}} \sum \sum C_i R_{t,i}$$

Decision variables

$S_{t,i} \in \{0,1\}$ Layer i stored for stage t

$R_{t,i} \in \{0,1\}$ Layer i (re)computed in stage t

Dependency constraints

$$R_{t,j} \leq R_{t,i} + S_{t,i} \qquad \forall t \, \forall (v_i, v_j) \in E,$$

A layer's dependencies must be computed before it can be computed

$$S_{t,i} \leq R_{t-1,i} + S_{t-1,i} \qquad \forall t \geq 2 \, \forall i,$$

A layer must be computed before it can be stored in RAM

$$\sum_i S_{1,i} = 0,$$

No values are initially in memory

$$\sum_t R_{t,n} \geq 1,$$

Model is training

Memory constraints

$$U_{t,i} \leq \text{budget}$$

Peak memory consumption under memory limit at all execution steps

$$U_{t,0} = \underbrace{M_{\text{input}} + 2M_{\text{param}}}_{\text{Constant overhead}} + \sum_{i=1}^{n} \underbrace{M_i S_{t,i}}_{\text{Checkpoints}}$$

memory consumption at the beginning of a stage

$$U_{t,k+1} = U_{t,k} - \text{GC Deps}[v_k] + M_{k+1} R_{t,k+1}$$

memory consumption after evaluating operator k+1

Optimal solver - Integer linear program

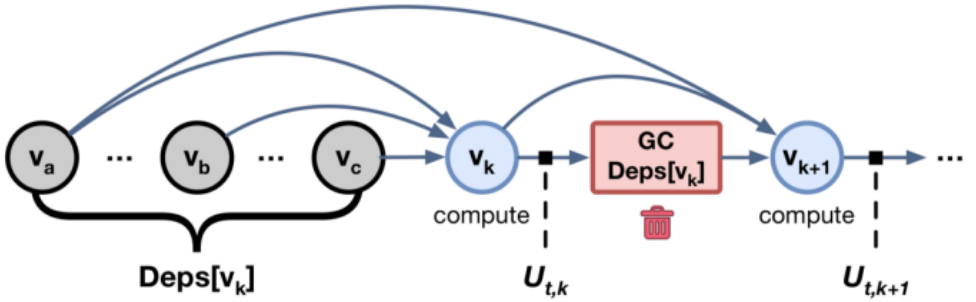
$$\underset{R,S,U}{\operatorname{argmin}} \sum \sum C_i R_{t,i}$$

$$U_{t,i} \leq \text{budget}$$

$$U_{t,0} = \underbrace{M_{\text{input}} + 2M_{\text{param}}}_{\text{Constant overhead}} + \sum_{i=1}^{n} \underbrace{M_i S_{t,i}}_{\text{Checkpoints}}$$

$$U_{t,k+1} = U_{t,k} - \text{GC Deps}[v_k] + M_{k+1} R_{t,k+1}$$

$$U_{t,k+1} = U_{t,k} - \text{GC Deps}[v_k] + M_{k+1} R_{t,k+1}$$



$$\text{GC Deps}[v_k] = \sum_{i \in \text{DEPS}[k]} M_i * \text{FREE}_{t,i,k}$$

Non-linear!

$$\text{FREE}_{t,i,k} = R_{t,k} * \underbrace{(1 - S_{t+1,i})}_{\text{Not checkpoint}} \prod_{\substack{j \in \text{USERS}[i] \\ j > k}} \underbrace{(1 - R_{t,j})}_{\text{Not dep.}}$$

Optimal solver - Integer linear program

$$\arg\min_{R,S,U,\text{FREE}} \sum_{t=1}^{n}\sum_{i=1}^{t} C_i R_{t,i}$$

$$U_{t,i} \leq \text{budget}$$

$$U_{t,0} = \underbrace{M_{\text{input}} + 2M_{\text{param}}}_{\text{Constant overhead}} + \sum_{i=1}^{n} \underbrace{M_i S_{t,i}}_{\text{Checkpoints}}$$

$$U_{t,k+1} = U_{t,k} - \text{GC Deps}[v_k] + M_{k+1} R_{t,k+1}$$

Non-linear!

$$\text{FREE}_{t,i,k} = R_{t,k} * \underbrace{(1 - S_{t+1,i})}_{\text{Not checkpoint}} \prod_{\substack{j \in \text{USERS}[i] \\ j>k}} \underbrace{(1 - R_{t,j})}_{\text{Not dep.}}$$

Lemma 4.1 (Linear Reformulation of Binary Polynomial). *If* $x_1, \ldots, x_n \in \{0,1\}$, *then*

$$\prod_{i=1}^{n} x_i = \begin{cases} 1 & \sum_{i=1}^{n}(1-x_i) = 0 \\ 0 & \textit{otherwise} \end{cases}$$

Lemma 4.2 (Linear Reformulation of Indicator Constraints). *Given* $0 \leq y \leq \kappa$ *where* y *is integral and* κ *is a constant upper bound on* y, *then*

$$x = \begin{cases} 1 & y = 0 \\ 0 & \textit{otherwise} \end{cases}$$

if and only if $x \in \{0,1\}$ *and* $(1-x) \leq y \leq \kappa(1-x)$.

reformulate

$$\begin{aligned} \text{FREE}_{t,i,k} &\in \{0,1\} \\ 1 - \text{FREE}_{t,i,k} &\leq \text{num\_hazards}(t,i,k) \\ \kappa(1 - \text{FREE}_{t,i,k}) &\geq \text{num\_hazards}(t,i,k) \end{aligned}$$

where $$\text{num\_hazards}(t,i,k) = (1 - R_{t,k}) + S_{t+1,i} + \sum_{\substack{j \in \text{USERS}[i] \\ j>k}} R_{t,j}$$

Optimal solver - Integer linear program

$$\underset{R,S,U}{\operatorname{argmin}} \sum \sum C_i R_{t,i}$$

$$R_{t,j} \leq R_{t,i} + S_{t,i} \qquad \forall t \, \forall (v_i, v_j) \in E,$$
$$S_{t,i} \leq R_{t-1,i} + S_{t-1,i} \qquad \forall t \geq 2 \, \forall i,$$
$$\sum_i S_{1,i} = 0,$$
$$\sum_t R_{t,n} \geq 1,$$

$$U_{t,i} \leq \text{budget}$$

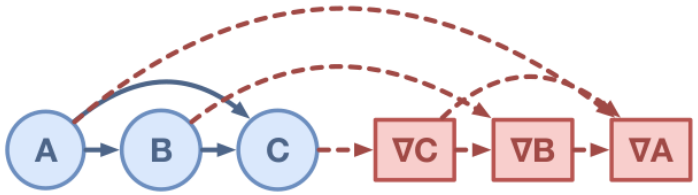
$$U_{t,0} = \underbrace{M_{\text{input}} + 2M_{\text{param}}}_{\text{Constant overhead}} + \sum_{i=1}^{n} \underbrace{M_i S_{t,i}}_{\text{Checkpoints}}$$

$$U_{t,k+1} = U_{t,k} - \text{GC Deps}[v_k] + M_{k+1} R_{t,k+1}$$

- Gurobi optimizer
- Given 8-layer graph neural network, how long is the solve time? 9 hours!
- Optimizing Search space: partition schedule into frontier-advancing stages
  - Time: 9 hours → 0.2 seconds

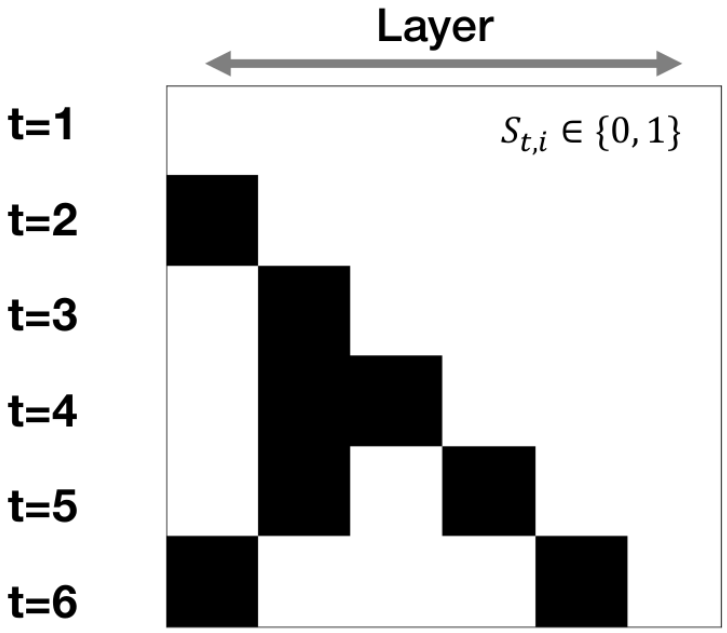
Node v_i is evaluated for the first time in stage i.

Optimal solver - Integer linear program

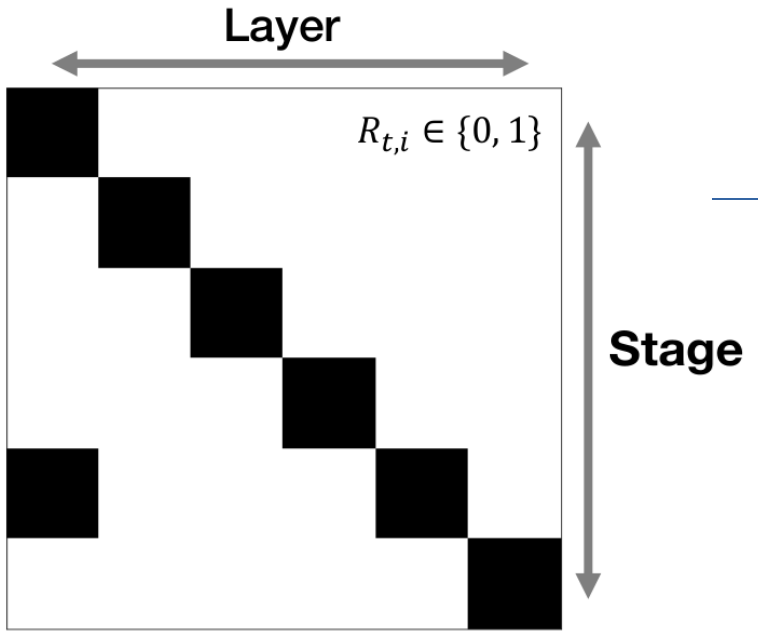


Free and recompute operator: node A

What is in memory?



What is computed?



$$R_{i,i} = 1 \ \forall i \quad \text{(frontier-advancing partitions)}$$
$$\sum_{i \geq t} S_{t,i} = 0 \quad \text{(lower tri., no initial checkpoints)}$$
$$\sum_{i > t} R_{t,i} = 0 \quad \text{(lower triangular)}$$

Approximation

- **ILP optimization is NP-hard**
- **Polynomial runtime approximation:**
  - Relax boolean constraints to real value.
  - Solve LP:
    - Two-Phase Rounding: Round S, solve other variables (e.g., R) optimally

Experiments

- What is the trade-off between memory usage and computational overhead when using rematerialization?
- Are large inputs practical with rematerialization?
- How well can we approximate the optimal rematerialization policy?

Q1: memory vs computation

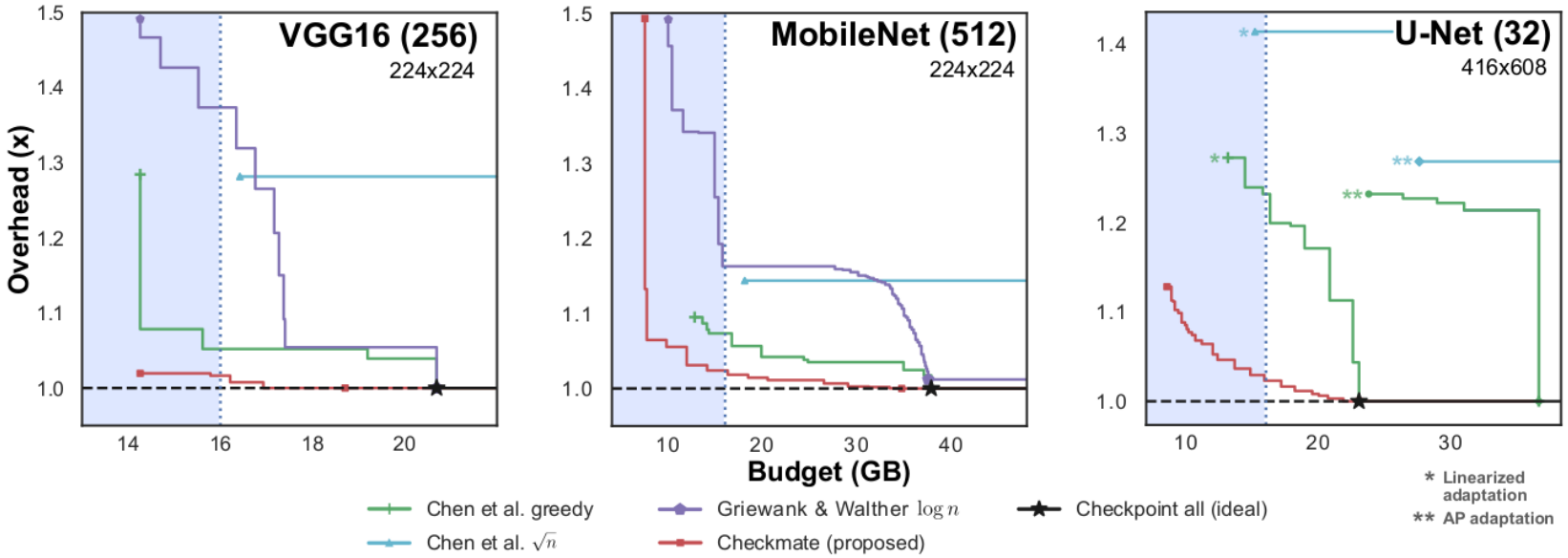


Figure 5. Computational overhead versus memory budget for (a) VGG16 image classification NN (Simonyan & Zisserman, 2014), (b) MobileNet image classification NN, and (c) the U-Net semantic segmentation NN (Ronneberger et al., 2015). Overhead is with respect to the best possible strategy without a memory restriction based on a profile-based cost model of a single NVIDIA V100 GPU. For U-Net (c), at the 16 GB V100 memory budget, we achieve a $1.20\times$ speedup over the best baseline—linearized greedy—and a $1.38\times$ speedup over the next best—linearized $\sqrt{n}$. **Takeaway:** our model- and hardware-aware solver produces in-budget solutions with the lowest overhead on linear networks (a-b), and dramatically lowers memory consumption *and* overhead on complex architectures (c).

Q2: batch size

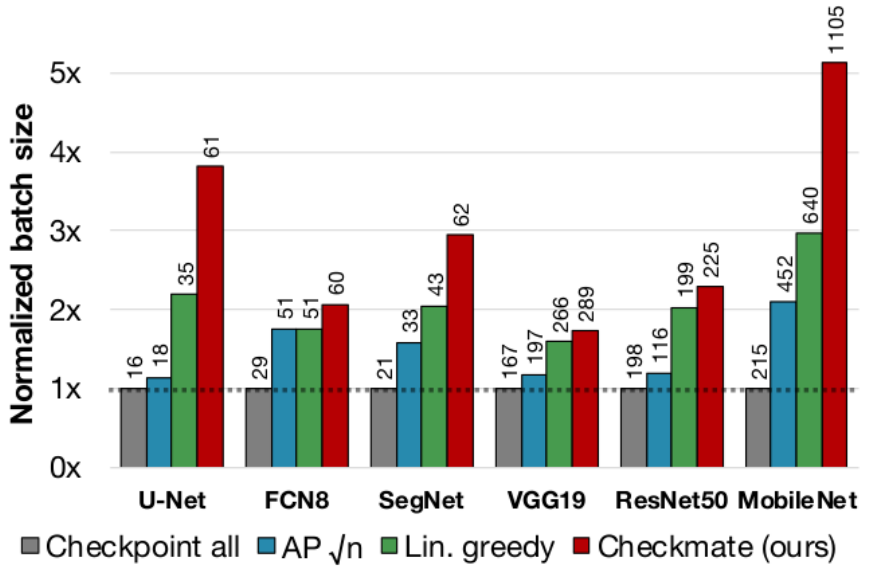


Figure 6. Maximum batch size possible on a single NVIDIA V100 GPU when using different generalized rematerialization strategies with at most a single extra forward pass. We enable increasing batch size by up to 5.1× over the current practice of caching all activations (on MobileNet), and up to 1.73× over the best checkpointing scheme (on U-Net).

Q3: two-phase LP rounding

	Chen $\sqrt{n}$	Chen greedy	Griewank $\log n$	Two-phase LP rounding
MobileNet	1.14×	1.07×	7.07×	**1.06**×
VGG16	1.28×	1.06×	1.44×	**1.01**×
VGG19	1.54×	1.39×	1.75×	**1.00**×
U-Net	1.27×	1.23×	-	**1.03**×
ResNet50	1.20×	1.25×	-	**1.05**×

Table 2. Approximation ratios for baseline heuristics and our LP rounding strategy. Results are given as the geometric mean speedup of the optimal ILP across feasible budgets.

Discussion

- Checkmate needs a static graph to optimize tensor rematerialization. Is it possible to design and integrate a reinforcement learning model to automatically decide which variables to be recomputed without the knowledge of static computation graph to reduce the memory cost as well as minimize computation cost?

- Andrew Ng: Unbiggen AI. It's time for smart-sized, "data-centric" solutions to big issues. Giant datasets simply don't exist, the focus has to shift from big data to good data. What's your opinion about it?