Checkmate: Breaking the Memory Wall with Optimal Tensor Rematerialization

Paras Jain, Ajay Jain, Ani Nrusimha, Amir Gholami, Pieter Abbeel, Kurt Keutzer, Joseph Gonzalez, Ion Stoica

Overview

- **Problem**: Limited memory prevents the development of new deep learning models, but compute is growing quickly.
- We tradeoff memory and compute with an **optimal strategy for arbitrary DNN memory checkpointing**.
- Formulation supports arbitrary DAGs and is both hardware-aware and memory-aware.
- Up to 5x higher batch sizes, 1.2x speedups.
- Integration with just one line of code.

Backprop space-time tradeoff

- Most memory is used by activations, not parameters.
- Can reduce memory usage by deleting & recomputing activations.

**This work**: How to minimize recomputation while using less than the GPU memory budget?

Why are heuristics suboptimal?

1. **Layer runtimes vary**
   - In VGG, 10x difference in early and late layer FLOPS.

2. **Layer RAM usages vary**
   - Layers significantly differ in memory usage.

3. **Real DNNs are non-linear**
   - What to checkpoint with skip connections, multi-tower architectures etc?

Checkmate optimizes the evaluation plan using a per-operation cost model, profiled on the target GPU.

Our linear program accounts for & constrains peak memory usage at all points in time, using statically known memory consumptions.

Checkmate traces fwd & bwd graph and constructs optimization problem using graph structure + flexible search space.

Representing a schedule

- **For flexibility**, unroll schedule into stages.
  - Separately model checkpoints (S) and computations (R).
  - Checkpoint matrix \( S_{t,i} \in \{0,1\} \)
    - Is operation \( i \) stored between stage \( t-1 \) and \( t \)?

- **Computation matrix**: Is operation \( i \) computed in stage \( t \)?
- **Space-time schedule repr. generalizes checkpointing**.
  - Fine-grained control of evaluation + GC.

Rematerialization ILP

\[
\text{arg min}_{R,S,U,FREE} \sum_{i} C_i R_{ij} \quad \text{subject to}\]
\[
U_{ij} \leq \text{budget}
\]
\[
R_{ij} \leq U_{ij} + S_{ij} \quad \text{and has constrained memory usage.}
\]
\[
R_{ij} = 1
\]
\[
S_{ij} \leq R_{ij} + S_{ij}
\]
\[
\text{For tractability, each stage is frontier-advancing:}
\]
\[
\rightarrow \text{Op} \ i \text{ evaluated in stage} \ i \text{ for the first time.}
\]
\[
\rightarrow \text{From 9 hr to 1.18 sec for certifiable optimality.}
\]

Model memory usage in each stage with recurrence.

- **Start of stage**: Checkpoints use memory \( U_{i,0} = \sum \{ M_i S_{ij} \} \)
- **Temporary value**: \( U_{i+1} = U_{ij} + \sum \{ M_i + \text{FREE}_{ij} \} - M_{i+1} R_{i+1,j} \)
- **Garbage collection**: Optimal \( R, U, \text{and FREE easy to compute given} \ S \)
  - “Two-phase” rounding approximation works well.

Creating new applications with Checkmate

- **One line of code for memory-efficient deep learning!**
- **Checkmate achieves up to 1.2x speedup** over our best baseline heuristic and finds schedules with the lowest memory usages.

Evaluation

- **TF 2.0 / Keras** Image classification & semantic segmentation architectures.
- **Maximize batch size as proxy for resolution, model depth etc.**
- **With +1x overhead cap**, Checkmate supports up to 5.1x larger batch sizes.