# Memory saving for neural network trainng

Xunyi Zhao

01

## Introduction

## Work progress

- Rotor: Rematerializing Optimally with pyTORch
  - Efficient with sequential models

## Work progress

- Rotor: Rematerializing Optimally with pyTORch
  - Efficient with sequential models
- Weight Offloading
  - Non-optimal but efficient

## Work progress

- Rotor: Rematerializing Optimally with pyTORch
  - Efficient with sequential models
- Weight Offloading
  - Non-optimal but efficient
- Rotor+checkmate(Ongoing)
  - Combined ideas to be general and efficient

02

## Weight offloading

## Training iteration

Possibility of training large model in a single GPU

Only keep what is necessary

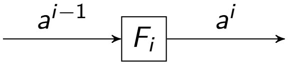
## Training iteration

Possibility of training large model in a single GPU

Only keep what is necessary

**Forward $i$**

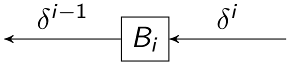
- $a^i \leftarrow a^{i-1}, w_i$



$$\begin{tikzcd} {} \arrow[r, "a^{i-1}"] & \boxed{F_i} \arrow[r, "a^i"] & {} \end{tikzcd}$$

## Backward $i$

- $\delta w_i \leftarrow a^{i-1}, \delta^i$;
- $\delta^{i-1} \leftarrow \delta^i, w_i$;
- $w_i \leftarrow w_i, \delta w_i$



## Selection

Offload layers that cover most memory needs

### Greedy selection

- Assume infinite bandwidth
- Start: every layer in GPU
- Choose $w_i$ that covers the most memory overflow
- Iterate until memory fits
- A coefficient is introduced for single offloading

## Greedy schedule

- Insert prefetch and offload operations into $F$'s and $B$'s
- Data required earlier is prefetched first
- Data produced earlier is offloaded first
- One particular case: $w_i$ is sent to CPU after $B_i$, but only deleted after $F_i$

## Simulation results

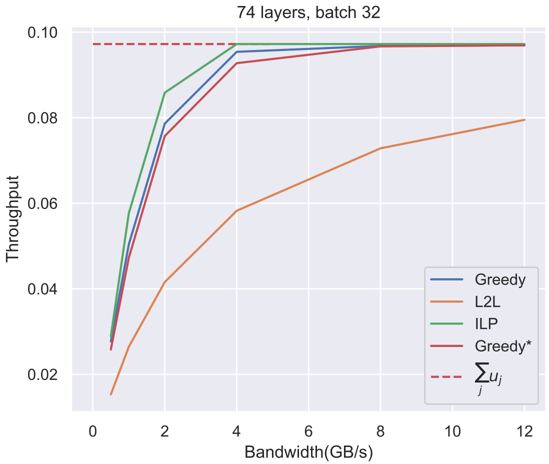


Figure: Greedy* represents the Greedy algorithm while weights are not only offloaded once.

03

## Rotor+checkmate

## Checkmate

Checkmate provides an optimal solution for rematerializations.

**Application**

- Can deal with any graphs
- ILP with $O(n^2)$ variables
- Implementations based on TensorFlow

## Checkmate+Rotor

Rotor is efficient, checkmate is general

## Checkmate+Rotor

Rotor is efficient, checkmate is general

### Combined ideas

- One network can be divided into several blocks: e.g. GPT-2 $\rightarrow$ Decoder
- Each block is (probably) small enough for checkmate
- Rotor can solve the schedule based on the solutions from checkmate

## Progress

- Checkmate on PyTorch: obtain the computation graph from any Torch Module

  - We can now open the box and generate the forward&backward graph

  - Each node in the graph can be executed by a torch function/python code