Efficient Combination of Rematerialization and Offloading for Training DNNs Appendix with Missing Proofs

Anonymous Author(s)
Affiliation
Address
email

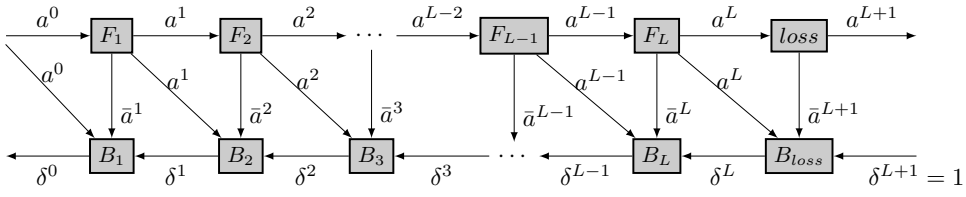

Figure 1: Data dependencies induced the training phase of Sequential Deep Neural Networks.

	Operation	Input	Output	Time	Memory overhead
F^ℓ_{all}	Forward and save all	$\{a^{\ell-1}\}$ $\{\bar{a}^{\ell-1}\}$	$\{a^{\ell-1}, \bar{a}^\ell\}$ $\{\bar{a}^{\ell-1}, \bar{a}^\ell\}$	u_{F_ℓ}	o^f_ℓ
F^ℓ_{ck}	Forward and materialize input	$\{a^{\ell-1}\}$ $\{\bar{a}^{\ell-1}\}$	$\{a^{\ell-1}, a^\ell\}$ $\{\bar{a}^{\ell-1}, a^\ell\}$	u_{F_ℓ}	o^f_ℓ
$F^\ell_\varnothing$	Forward without saving	$\{a^{\ell-1}\}$ $\{\bar{a}^{\ell-1}\}$	$\{a^\ell\}$ $\{a^\ell\}$	u_{F_ℓ}	o^f_ℓ
B^l	Backward step	$\{\delta^\ell, \bar{a}^\ell, a^{\ell-1}\}$ $\{\delta^\ell, \bar{a}^\ell, \bar{a}^{\ell-1}\}$	$\{\delta^{\ell-1}\}$ $\{\delta^{\ell-1}, \bar{a}^{\ell-1}\}$	u_{B_ℓ}	o^b_ℓ

Table 1: Generic operations available in DL frameworks.

A Proof of Lemma 1, Section 4.2

Lemma 1 *Let us consider a fixed sequence of operations, for which the available memory increases (resp. decreases) during execution because of data unloading (resp. prefetching), with its minimum at m_{min}. Let us denote by $\mathcal{M}_o^{\mathcal{S}}$ the memory required to process operation $o \in \mathcal{S}$, and d_o the distance between o and the end (beginning) of sequence $\mathcal{S}$,i.e. the cumulative duration of operations taking place before (after) operation o. Then, the execution of $\mathcal{S}$ needs to be delayed by some idle time*

$$\epsilon = \max\left(\frac{\max_{o\in\mathcal{S}}(\mathcal{M}_o^{\mathcal{S}} - \beta d_o) - m_{min}}{\beta}, 0\right).$$

Let us concentrate on prefetching, while the case with offloading is done analogously. The idle time is not zero when it is not possible to overlap entirely communications and computations because of the memory. During prefetching the available memory (excluding the memory needed for operations in $\mathcal{S}$) is decreasing at speed β until it reaches m_{min}, following the rule $m(d) = m_{min} + \beta d$,

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

where d denotes the distance to the end of prefetching. Therefore, for each operation o, d must satisfy $m_{min} + \beta d \geq \mathcal{M}_o^{\mathcal{S}}$ and by construction $d \geq d_o$. Therefore, for each o, some idle time $d - d_o \geq \frac{\mathcal{M}_o^{\mathcal{S}}}{\beta} - d_o - \frac{m_{min}}{\beta}$ must take place before o. Since this constraint must be satisfied for all o, then

$$\epsilon = \max\left(\frac{\max_{o\in\mathcal{S}}(\mathcal{M}_o^{\mathcal{S}} - \beta d_o) - m_{min}}{\beta}, 0\right).$$

B Proof of Theorem 1, Section 4.2. Detailed dynamic program to compute the optimal sequence

Theorem 1 *Under assumptions of Section 3.2, the problem of finding the minimal processing time for the chain from Figure 1, using operations from Table 1 together with offloading O_ℓ and prefetching P_ℓ, under memory limit M_{GPU} discretized with N_{GPU} values and bandwidth β, can be solved optimally with a dynamic programming algorithm with a complexity of $O(L^2 N_{GPU}^3 + L^3 N_{GPU}^2)$.*

In this section, we detail the dynamic programming equations whose intuition has been given in Section 4.

In order to proceed, one should know the minimal memory requirements for each operation. Given the chain between i and j, taking into account the data dependencies from Table 1 and the fact that the global input of the chain $\mathcal{I}_i^x = (1-x)a^{i-1} + x\bar{a}^{i-1}$ (it is equal to a^{i-1} when $x = 0$ and $\bar{a}^{i-1}$ when $x = 1$) is already stored in the memory (and counted separately in all equations) and the current gradient value should be stored at any time, then

- $\mathcal{M}_{F_\varnothing^i}^{i,j} = \delta^j + a^i + o_i^f$;
- $\forall k \neq i : \mathcal{M}_{F_\varnothing^k}^{i,j} = \delta^j + a^{k-1} + a^k + o_k^f$;
- $\mathcal{M}_{F_{all}^i}^{i,j} = \delta^j + \bar{a}^i + o_i^f$;
- $\forall k \neq i : \mathcal{M}_{F_{all}^k}^{i,j} = \delta^j + a^{k-1} + \bar{a}^k + o_k^f$;
- $\mathcal{M}_{B_i}^{i,j} = \bar{a}^i + \delta^i + \delta^{i-1} + o_i^b$;
- $\forall k \neq i : \mathcal{M}_{B_k}^{i,j} = a^{k-1} + \bar{a}^k + \delta^k + \delta^{k-1} + o_k^b$.

B.1 Forward phase

In this section, we provide the analysis of the forward phase,*i.e.* for all operations that take place before the computation of the loss. We consider the subchain that starts from layer i and assume that its input $\mathcal{I}_i^x$ is saved by F_{ck}^i or F_{all}^i. To denote the total duration of the execution from F_i to B_i, we use $\mathrm{OC}_x(i, A_i, \Delta_{F_i}, \Delta_{B_i})$. We also distinguish between $\mathrm{OC}_x^{F_{all}}$ (resp. $\mathrm{OC}_x^{F_{ck}}$) that represents the duration of the chain execution when the first operation is constrained to be F_{all}^i (resp. F_{ck}^i). By construction, the first operation cannot be $F_\varnothing^i$ since we assume that the input of the chain must not be discarded.

Case 1: F_{all}^i is the first operation

If the first operation is F_{all}^i, then in the optimal schedule the next operation cannot be $F_\varnothing^{i+1}$. Indeed, as F_{all}^i stores $\bar{a}^i$ together with the input $\mathcal{I}_i^x$, then this $\bar{a}^i$ should stay in the memory until the backward operation B_i. At the same time, as $F_\varnothing^{i+1}$ discards its input after its execution, then each time when we need to recompute a^i we need to re-execute F_i. Therefore, if the next operation is $F_\varnothing^{i+1}$, the overall duration does not change if the first instance of F_{all}^i is replaced by F_{ck}^i, while F_{all}^i replaces the last F_i that computes a^i in the sequence. Simultaneously, this transformed sequence keeps less data in memory, which contradicts the optimality of F_{all}^i being followed by $F_\varnothing^{i+1}$.

Since we assume memory persistency,*i.e.* that $\bar{a}^i$ stays in the memory until the backward operation B_i, the sequence after F_{all}^i and before B_i can be computed with a recursive call to the dynamic

	offload input $v=1$	no offload $v=2$
new A^v	A_i	$A_i + \mathcal{I}_i^x$
new Δ_F^v	$\max\{\Delta_{F_i} + \mathcal{I}_i^x - D_F, 0\}$	$\max\{\Delta_{F_i} - D_F, 0\}$
new Δ_B^v	$\max\{\Delta_{B_i} - D_B, 0\} + \mathcal{I}_i^x$	$\max\{\Delta_{B_i} - D_B, 0\}$

Table 2: Values for the new state

programming. After this sequence, B_i can be directly executed, having all the necessary input already stored in device memory. This shows that $\mathrm{OC}_x^{F_{all}}(i, A_i, \Delta_{F_i}, \Delta_{B_i})$ can be computed as

$$\mathrm{OC}_x^{F_{all}}(i, A_i, \Delta_{F_i}, \Delta_{B_i}) = u_{F_i} + u_{B_i} + \min_{v=1,2} \mathrm{OC}_{x=1}(i+1, A_{i+1}^v, \Delta_{F_{i+1}}^v, \Delta_{B_{i+1}}^v) + \epsilon^F + \epsilon^B,$$

where the values ϵ^F and ϵ^B represent the idle time required for communications in the forward and backward respectively, and can be determined using Lemma 1. Taking into account that the maximal memory occupation for F_{all}^i and B_i are given by $M_{F_i}^x$ and $M_{B_i}^x$ respectively, we obtain

$$\epsilon^F = \max\left(\frac{\mathcal{M}_{F_{all}^i}^{i,L} - M_{\text{GPU}} + M_{F_i}^x}{\beta}, 0\right) \text{ and } \epsilon^B = \max\left(\frac{\mathcal{M}_{B_i}^{i,L} - M_{\text{GPU}} + M_{B_i}^x}{\beta}, 0\right).$$

We also need to compute the new state variables $A_{i+1}^v, \Delta_{F_{i+1}}^v, \Delta_{B_{i+1}}^v$, where the value of v indicates whether the input $\mathcal{I}_i^x$ is offloaded or not. During the forward step, the communication link is able to offload a quantity of data given by $D_F = (u_{F_i} + \epsilon^F)\beta$; similarly, the data that can be prefetched during the backward step is given by $D_B = (u_{B_i} + \epsilon^B)\beta$. These values can be used to obtain the new state variables, as described in Table 2.

The final sequence is valid if memory limit is not violated. So if $\mathcal{M}_{F_{all}^i}^{i,L} + \mathcal{I}_i^x > M_{\text{GPU}} - A_i$ or $\mathcal{M}_{B_i}^{i,L} + \mathcal{I}_i^x > M_{\text{GPU}} - A_i$ then we set $\mathrm{OC}_x^{F_{all}}(i, A_i, \Delta_{F_i}, \Delta_{B_i}) = \infty$.

Case 2: F_{ck}^i is the first operation

After an F_{ck}^i operation, any forward operation or offload operation is possible. Let us assume that the next saved activation is a^j for some $j \geq i$, which implies that after F_{ck}^i there is a sequence of $F_\emptyset^k$ for $i < k \leq j$. Due to memory persistency, it also implies that the sequence processing layers from $j+1$ till L can be obtained recursively, looking at $\mathrm{OC}_{x=0}(j+1, A_{j+1}^v, \Delta_{F_{j+1}}^v, \Delta_{B_{j+1}}^v)$ (once again, v denotes whether the input $\mathcal{I}_i^x$ is offloaded or not). Afterwards, processing B_k for $i < k \leq j$ requires recomputing forward operations from a^{i-1} (or $\bar{a}^{i-1}$). Since no offloading operation can be performed after the computation of the loss, this corresponds to computing a rematerialization sequence between layers i and j that should have Δ_{B_i} prefetched by the end. $C_\Delta(i, j, A_i + \mathcal{I}_i^x, \Delta_{B_i})$ is used to compute the optimal duration to process layers from i to j, having $A_i + \mathcal{I}_i^x$ as a cumulative storage of all activations at the GPU, and Δ_{B_i} that corresponds to the data that has to be prefetched. We show in Section B.3 how to compute $C_\Delta(i, j, A, \Delta)$. This yields the formula for $\mathrm{OC}_x^{F_{ck}}(i, A_i, \Delta_{F_i}, \Delta_{B_i})$

$$\mathrm{OC}_x^{F_{ck}}(i, A_i, \Delta_{F_i}, \Delta_{B_i}) = \min_{i \leq j \leq L-1} \Big(\sum_{k=i}^{j} u_{F_k} + \quad \min_{v=1,2} \mathrm{OC}_{x=0}(j+1, A_{j+1}^v, \Delta_{F_{j+1}}^v, \Delta_{B_{j+1}}^v) + C_\Delta(i, j, A_i + \mathcal{I}_i^x, \Delta_{B_i}) + \epsilon^F\Big),$$

where ϵ^F is computed with the help of Lemma 1

$$\epsilon^F = \max\left(\frac{\max_{k=i,\dots,j}(\mathcal{M}_{F_\emptyset^k}^{i,L} - \beta \sum_{h=i}^{k-1} u_{F_h}) - M_{\text{GPU}} + M_{F_i}^x}{\beta}, 0\right).$$

Similarly to the Case 1, the values of the new state variables A^v, Δ_F^v and Δ_B^v can be found using the formulas in Table 2. The only difference is the possible amount of offloaded data and prefetched data: in this case $D_F = (\sum_{k=i}^{j} u_{F_k} + \epsilon^F)\beta$ and $D_B = C_\Delta(i, j, A, \Delta_{B_i})\beta$.

The final sequence is valid if memory limit is not violated. Thus, if for some $k \geq i$ we have $\mathcal{M}^{i,L}_{F^k_\varnothing} + \mathcal{I}^x_i > M_{\text{GPU}} - A_i$ when $\mathcal{I}^x_i$ is not offloaded (or when $k = i$) or $\mathcal{M}^{i,L}_{F^k_\varnothing} > M_{\text{GPU}} - A_i$ when $\mathcal{I}^x_i$ is offloaded, then we set $\text{OC}^{F_{ck}}_x(i, A_i, \Delta_{F_i}, \Delta_{B_i}) = \infty$.

Combining everything together

Therefore, $\text{OC}_x(i, A_i, \Delta_{F_i}, \Delta_{B_i})$ can be computed as

$$\text{OC}_x(i, A_i, \Delta_{F_i}, \Delta_{B_i}) = \min \begin{cases} \text{OC}^{F_{all}}_x(i, A_i, \Delta_{F_i}, \Delta_{B_i}) \\ \text{OC}^{F_{ck}}_x(i, A_i, \Delta_{F_i}, \Delta_{B_i}) \end{cases} \tag{1}$$

B.2 Loss: how to concatenate forward and backward phases

In the previous section, we have shown how to compute the optimal duration of the sequence, using dynamic programming with $\text{OC}_x(i, A_i, \Delta_{F_i}, \Delta_{B_i})$. This dynamic program finds the solution through recursive calls to smaller sub-chains, until reaching the subchain consisting of only loss computation. We further represent the loss computation with F_{L+1} and B_{L+1} operations. The loss should be computed with F^{L+1}_{all} and B_{L+1}, so that this case is similar to $\text{OC}^{F_{all}}_x$ and

$$\text{OC}_x(L+1, A_i, \Delta_{F_i}, \Delta_{B_i}) = u_{F_{L+1}} + u_{B_{L+1}} + \epsilon^F + \epsilon^B + \epsilon^G,$$

where ϵ^F and ϵ^B are found with the same expressions as the idle times for Case 1 of Forward phase (see Section B.1).

As no offloading is possible during backward propagation and no prefetching is possible during forward propagation, the idle time between the phases comes from the completion of both communication tasks, *i.e.*

$$\epsilon^G = (\Delta_{F_{L+2}} + \Delta_{B_{L+2}})/\beta.$$

B.3 Backward phase

The situation in the case of backward is a bit more complex. We must indeed perform all the operations $B_j, \ldots, B_{i+1}$, with only $\mathcal{I}^x_i$ and δ^j into memory. This is nevertheless enough since we can execute the whole forward chain $F_{i+1}, \ldots, F_j$ from a^i and then the whole chain $B_j, \ldots, B_{i+1}$ using the values computed during the processing of the forward chain.

In general, due to memory constraints, it is not possible to perform $F_{i+1}, \ldots, F_j B_j, \ldots, B_{i+1}$ in sequence. Since we assume that offloading cannot take place in the backward phase, we rely on rematerialization in order to save memory if needed. Our goal, given Δ_{F_i} and M_{F_i}, is to find a valid schedule, that satisfies memory constraints and whose duration is minimal. One additional difficulty comes from prefetched data. Indeed, several valid rematerialization sequences for computing $B_j, \ldots, B_{i+1}$ will differ both by their duration and by the amount of data that can be prefetched during the sequence. Indeed, for a given Δ_{B_i}, the sequences that enable to prefetch a lot of data during the computation of $B_j, \ldots, B_{i+1}$ are preferable since they will induce a smaller value of Δ_{B_j} and therefore less memory pressure to perform $B_j, \ldots B_L$.

Let $C_\Delta(i, j, A, \Delta)$ denotes the optimal duration to execute the chain between layers i and j with A denoting the cumulative size of all activations already stored in GPU, including input $\mathcal{I}^x_i$, and given minimal available memory $M_{\text{GPU}} - A - \Delta$. In the beginning of the execution $\mathcal{I}^x_i$ and δ^j are stored in the device memory. Moreover, $\mathcal{I}^x_i$ must stay in the memory until the end of the execution. Therefore, the first operation should be F^i_{all} or F^i_{ck}. Depending on the first operation, the optimal duration can be read in $C^{F_{all}}_\Delta(i, j, A, \Delta)$ or $C^{F_{ck}}_\Delta(i, j, A, \Delta)$ respectively.

Case 1: F^i_{all} is the first operation

Let us first notice that after F^i_{all} the activation $\bar{a}^i$ must be kept in the memory until its associated backward operation (due to memory persistency). We must solve the subproblem of finding the optimal duration for the chain between layers $i+1$ and j, which can be scheduled optimally with $C_\Delta(i+1, j, A + \bar{a}^i, \Delta_1)$, according to our assumption. Once B_{i+1} is performed, then B_i can be directly executed, as all necessary input are already in the device memory. Therefore, $C^{F_{all}}_\Delta(i, j, A, \Delta)$ is given by

$$C^{F_{all}}_\Delta(i, j, A, \Delta) = u_{F_i} + C_\Delta(i+1, j, A + \bar{a}^i, \Delta_1) + u_{B_i} + \epsilon^{F^i_{all}} + \epsilon^{B_i}.$$

The parameters for the recursive call of the dynamic programming are given by

$$\Delta_1 = \max\{\Delta - (u_{B_i} + \epsilon^{B_i})\beta, 0\}.$$

We can also estimate the minimal available memory when F^i_{all} is executed, which is given by $M_{\text{GPU}} - A - \Delta_2$ where

$$\Delta_2 = \max\{\Delta_1 - C_\Delta(i+1, j, A + \bar{a}^i, \Delta_1)\beta, 0\}.$$

The idle times caused by prefetching can in turn be computed with the help of Lemma 1 as

$$\epsilon^{B_i} = \max\left(\frac{\mathcal{M}^{i,j}_{B_i} - M_{\text{GPU}} + A + \Delta}{\beta}, 0\right) \text{ and } \epsilon^{F^i_{all}} = \max\left(\frac{\mathcal{M}^{i,j}_{F^i_{all}} - M_{\text{GPU}} + A + \Delta_2}{\beta}, 0\right).$$

$\mathcal{M}^{i,i}_{F^i_{all}} > M_{\text{GPU}} - A$ or $\mathcal{M}^{i,i}_{B_i} > M_{\text{GPU}} - A$ means that there is not enough memory to perform this sequence, and as previously the dynamic programming cost should be set to ∞.

Case 2: F^i_{ck} is the first operation

Let us suppose that F^i_{ck} is used to save $\mathcal{I}^x_i$ and consider the next value $a^{i'}$ to be kept in memory. To compute this value, a sequence of $F_\varnothing$ operations from layer $i+1$ till layer i' is performed. Due to memory persistency, after checkpointing $a^{i'}$ we can find the optimal sequence from $i'+1$ to j by reading $C_\Delta(i'+1, j, A + a^{i'}, \Delta_2)$. Once $B_{i'+1}$ is performed, the chain from i to i' can be scheduled using $C_\Delta(i, i', A, \Delta_1)$. Thus, $C^{F_{ck}}_\Delta(i, j, A, \Delta)$ is given by

$$C^{F_{ck}}_\Delta(i, j, A, \Delta) = \min_{i \leq i' < j} \sum_{k=i}^{i'} u_{F_k} + C_\Delta(i'+1, j, A + a^{i'}, \Delta_2) + C_\Delta(i, i', A, \Delta_1) + \epsilon^F$$

The parameters for the recursive call of the dynamic programming can be found using

$$\Delta_1 = \Delta \text{ and } \Delta_2 = \max\{\Delta_1 - C_\Delta(i, i', A, \Delta_1)\beta, 0\}.$$

Similarly to the previous case, we can estimate the minimal available memory when forwards from i to i' are performed for the first time. This minimal memory is given by $M_{\text{GPU}} - A - \Delta_3$, where

$$\Delta_3 = \max\{\Delta_2 - C_\Delta(i'+1, j, A + a^{i'}, \Delta_2)\beta, 0\}.$$

The idle time caused by prefetching can be computed with the help of Lemma 1 as

$$\epsilon^F = \max\left(\frac{\max_{k=i,\ldots,i'}(\mathcal{M}^{i,j}_{F^k_\varnothing} - \beta(\sum_{h=k+1}^{i'} u_{F_h})) - M_{\text{GPU}} + A + \Delta_3}{\beta}, 0\right).$$

If $\mathcal{M}^{i,j}_{F^k_\varnothing} > M_{\text{GPU}} - A$ for some $k,\ i \leq k < j$, then this means that there is not enough memory to perform this sequence, and as previously the dynamic programming cost should be set to ∞.

Initialization: solution for a single layer Each recursive call to the dynamic programming is solving the problem for smaller sub-chains. The recursion stops when reaching the sub-chain consisting of only one layer. The schedule for one layer i is straightforward: perform F^i_{all} and B_i, *i.e.*

$$C_\Delta(i, i, A, \Delta) = u_{F_i} + u_{B_i} + \epsilon$$

where, according to Lemma 1

$$\epsilon = \max\left(\frac{\mathcal{M}^{i,i}_{F^i_{all}} - \beta u_{B_i} - M_{\text{GPU}} + A + \Delta}{\beta}, \frac{\mathcal{M}^{i,i}_{B_i} - M_{\text{GPU}} + A + \Delta}{\beta}, 0\right).$$

Again, this solution will be infeasible if there is not enough memory to perform any operation, *i.e.* if $\mathcal{M}^{i,i}_{F^i_{all}} > M_{\text{GPU}} - A$ or $\mathcal{M}^{i,i}_{B_i} > M_{\text{GPU}} - A$. In this case, the dynamic programming cost should be set ∞.

Combining everything together:

Therefore, $C_\Delta(i, j, A, \Delta)$ can be computed as

$$C_\Delta(i, j, A, \Delta) = \min \begin{cases} C^{F_{all}}_\Delta(i, j, A, \Delta) \\ C^{F_{ck}}_\Delta(i, j, A, \Delta) \end{cases} \tag{2}$$

B.4 Complexity

Finally, finding the optimal duration schedule for a chain of length L corresponds to computing $\mathrm{OC}_{x=0}(1,0,0,0)$ using the dynamic programming presented above. The analysis of the complexity of the above dynamic programming can be decomposed in two parts. Let N_{GPU} denote the number of discretized values for the memory M_{GPU}. During the backward phase, the size of the state space is $O(L^2 N_{\text{GPU}}^2)$, and for case 2, computing a new value requires $O(L)$ operations, which leads to $O(L^3 N_{\text{GPU}}^2)$ operations. In the forward phase, the size of the state space is $O(L N_{\text{GPU}}^3)$, and again computing a new value requires $O(L)$ operations. This results in $O(L^2 N_{\text{GPU}}^3)$ operations for the forward phase. Therefore, the overall complexity is given by $O(L^2 N_{\text{GPU}}^3 + L^3 N_{\text{GPU}}^2)$. In practice, we observe that for all the experiments presented in Section 5, discretizing the memory with $N_{\text{GPU}} = 50$ values is enough since considering a finer discretization does not lead to any practical improvement.