Assignment 2: Efficient model computation

42 points + 4 optional points

With solutions

You may use a result requested in an earlier part even if you have not proved it.

Notation. Use the multiply-add convention from Lecture 6: one accumulation $z \leftarrow z + xy$ costs one multiply-add, so multiplying an $a \times b$ matrix by a $b \times c$ matrix costs abc multiply-adds. Treat transposes as free. $\mathrm{diag}(a_1, \ldots, a_r)$ denotes the diagonal matrix with diagonal entries $a_1, \ldots, a_r$.

1 Conditional computation with MoE (10 points)

A mixture-of-experts (MoE) layer evaluates a few experts per token [1]. Following Lecture 4 and GShard [2], we batch tokens by expert and combine the expert outputs for each token.

There are T token vectors $h_t \in \mathbb{R}^d$ and E experts with separate parameters. Expert e is a two-layer multilayer perceptron (MLP):

$$F_e(h) = W_{e,2}\phi(W_{e,1}h), \qquad W_{e,1} \in \mathbb{R}^{m \times d}, \quad W_{e,2} \in \mathbb{R}^{d \times m},$$

where $\phi(z) = \max\{z, 0\}$ acts entrywise. The router computes $s_t = W_r h_t$, with $W_r \in \mathbb{R}^{E \times d}$, and selects the k largest scores, where $1 \le k \le E$. Let S_t be the selected expert indices, breaking ties by smaller index. The output is

$$w_{t,e} = \frac{e^{s_{t,e}}}{\sum_{j \in S_t} e^{s_{t,j}}}, \qquad y_t = \sum_{e \in S_t} w_{t,e} F_e(h_t).$$

Every selected token–expert pair is evaluated.[1]

(a) **(3 points)** Count the parameters in the router and experts. For the whole batch, compute the multiply-add counts for the router and for the selected experts. Exclude weighted output aggregation. By what factor does selecting experts *before* evaluating them reduce expert computation compared with evaluating all E experts on every token? With T, d, m, k fixed, which of these counts still grow with E?

Solution. The router has Ed parameters, and the experts have $2Edm$, giving $Ed + 2Edm$ in total. For the batch, the router costs TEd multiply-adds and the selected experts cost $2Tkdm$. Evaluating all experts would cost $2TEdm$, a factor of E/k more. This saving requires selecting experts before evaluation; discarding their outputs afterwards saves no computation. With T, d, m, k fixed, both parameter counts and the router cost grow with E, while the selected experts' cost is constant.

[1] There is no limit on the number of tokens assigned to an expert. The layer has no biases, and no residual term h_t is added to y_t.

(b) **(4 points)** The router's sets S_t and weights $w_{t,e}$ are available. Complete the following batched implementation.

(i) **(2 points)** Write pseudocode that first counts $N_e = |\{t : e \in S_t\}|$ and then fills an array of N_e assignment records for each expert. What must each record store to identify the destination token and apply its router weight? Prove that grouping the records takes $O(Tk + E)$ operations.[2]

(ii) **(1 point)** If expert e's records contain token indices $t_1, \dots, t_{N_e}$, define its input matrix H_e. Express its output matrix Z_e using two matrix products and ϕ. Give the dimensions of H_e, $W_{e,1}H_e$, and Z_e. Skip experts with $N_e = 0$.

(iii) **(1 point)** Starting from $y_t = 0$, give the update using column j of Z_e and its record. Prove that the procedure computes the stated y_t.

Solution. For (i), a record stores $(t, w_{t,e})$:

$N_1, \dots, N_E \leftarrow 0$
for $t = 1, \dots, T$ and $e \in S_t$:
  $N_e \leftarrow N_e + 1$
for $e = 1, \dots, E$:
  allocate N_e records in A_e; set $j_e \leftarrow 1$
for $t = 1, \dots, T$ and $e \in S_t$:
  $A_e[j_e] \leftarrow (t, w_{t,e})$; $j_e \leftarrow j_e + 1$

The two passes each process Tk assignments, with constant work per assignment. Initializing the counts and counters costs $O(E)$; the arrays contain Tk records in total. The time and space are therefore $O(Tk + E)$.

For (ii),

$$H_e = [h_{t_1} \ \cdots \ h_{t_{N_e}}] \in \mathbb{R}^{d \times N_e}, \qquad Z_e = W_{e,2}\phi(W_{e,1}H_e) \in \mathbb{R}^{d \times N_e}.$$

The intermediate matrix $W_{e,1}H_e$ has size $m \times N_e$. For (iii), use the stored token index and weight to update

$$y_{t_j} \leftarrow y_{t_j} + w_{t_j,e}(Z_e)_{:,j}.$$

Each column is evaluated independently, so $(Z_e)_{:,j} = F_e(h_{t_j})$. Every selected token–expert pair appears once in the records and contributes $w_{t,e}F_e(h_t)$ to y_t.

(c) **(3 points)** Each expert has its own device. All devices start together, and expert e takes τN_e time, where $\tau > 0$. Assume T is a multiple of E. Compare balanced loads $N_e = Tk/E$ for all e with concentrated loads $N_e = T$ for $e \le k$ and $N_e = 0$ otherwise. Compute the batch completion time and total expert multiply-add count for each case. Give the ratio of concentrated to balanced completion time, and prove that no assignment of k distinct experts per token can finish faster than the balanced case.

Solution. The completion time is $\tau \max_e N_e$: $\tau Tk/E$ for balanced loads and τT for concentrated loads, with ratio E/k. The times are equal when $k = E$. Both cases

[2] Each index or scalar read, write, or arithmetic operation has unit cost. This bound concerns records; it excludes copying token vectors and the computations in (ii)–(iii).

require Tk expert evaluations and $2Tkdm$ multiply-adds. Since $\sum_e N_e = Tk$, some expert receives at least Tk/E tokens, so no assignment can finish in less than $\tau Tk/E$ time.

The balanced case is attainable: numbering tokens and experts from zero, assign token t to $(t + j) \bmod E$ for $j = 0, \ldots, k-1$. Each block of E tokens gives every expert k assignments.

2 Latent attention and an indexer (12 points)

Lectures 5–6 combine multi-head latent attention (MLA) with position selection in DeepSeek Sparse Attention (DSA) [3]. We use a simplified indexer below.

Consider one decoding step with n cached positions and H heads of dimension d. Each position has a shared latent vector $c_i \in \mathbb{R}^{d_c}$. The key and value for head h are

$$k_{h,i} = U_{K,h}^\top c_i, \qquad v_{h,i} = U_{V,h}^\top c_i, \qquad U_{K,h}, U_{V,h} \in \mathbb{R}^{d_c \times d}.$$

For a query $q_h \in \mathbb{R}^d$, the full attention output is

$$s_{h,i} = \frac{q_h^\top k_{h,i}}{\sqrt{d}}, \qquad a_{h,i} = \frac{e^{s_{h,i}}}{\sum_{j=1}^n e^{s_{h,j}}}, \qquad o_h = \sum_{i=1}^n a_{h,i} v_{h,i}.$$

Count matrix products needed to compute o_h from q_h and the cached vectors specified in each part, including query transformations. The key-value (KV) cache contains vectors retained for future steps; its size excludes model matrices and temporary arrays.[3]

(a) **(2 points)** A full KV cache stores $k_{h,i}$ and $v_{h,i}$ for every head and position. Compute its size in scalars and the multiply-add counts for all scores and weighted value sums.

Solution. The cache stores $2Hnd$ scalars. The scores cost Hnd multiply-adds and the weighted value sums cost another Hnd, for a total of $2Hnd$ per step.

(b) **(4 points)** Suppose the cache stores only $c_1, \ldots, c_n$. Use associativity, as in Assignment 1, Problem 1(a), to avoid reconstructing the keys and values.

(i) **(2 points)** Find $\tilde{q}_h \in \mathbb{R}^{d_c}$ and $z_h \in \mathbb{R}^{d_c}$ such that $s_{h,i} = \tilde{q}_h^\top c_i / \sqrt{d}$ and $o_h = U_{V,h}^\top z_h$. Derive both identities. (*Hint:* Transform the query once per head; for the output, sum the weighted latent vectors before applying $U_{V,h}^\top$.)

(ii) **(1 point)** Count the multiply-adds in these four stages: query transformation, scores, weighted latent sum, and output projection. Compute the total over H heads and the cache size.

[3] Exclude computing the input queries, constructing or appending cache entries, and any projection after o_h. The problem has no positional terms. Exclude score scaling, softmax, and selection from the matrix-product counts.

(iii) **(1 point)** Compared with (a), when does this cache store strictly fewer scalars? When does the procedure use strictly more multiply-adds? Give an inequality in H, d, d_c, n for each comparison.

Solution. For (i), let $\widetilde{q}_h = U_{K,h} q_h$. Then

$$q_h^\top U_{K,h}^\top c_i = (U_{K,h} q_h)^\top c_i.$$

The scores, and hence the softmax weights, are unchanged. For the output,

$$o_h = \sum_i a_{h,i} U_{V,h}^\top c_i = U_{V,h}^\top \underbrace{\left(\sum_i a_{h,i} c_i \right)}_{z_h}.$$

The vector z_h uses head h's attention weights.

For (ii), the counts over all heads are

Stage	Multiply-adds
$U_{K,h} q_h$	Hdd_c
$\widetilde{q}_h^\top c_i$ for all positions	Hnd_c
$\sum_i a_{h,i} c_i$	Hnd_c
$U_{V,h}^\top z_h$	Hdd_c

The total is $2Hd_c(d+n)$, and the cache stores nd_c scalars. For (iii), the cache is smaller when $nd_c < 2Hnd$, and the computation costs more when $2Hd_c(d+n) > 2Hnd$. These simplify to

$$d_c < 2Hd, \qquad d_c(d+n) > nd.$$

Both inequalities can hold because the cache is shared across heads, but each head computes attention in dimension d_c.

(c) **(6 points)** An indexer selects K positions for all heads, where $1 \le K \le n$. The cache now stores c_i and a search key $k_i^I \in \mathbb{R}^{d_I}$ for every position. From an input query $q^I \in \mathbb{R}^{d_I}$, compute $I_i = (q^I)^\top k_i^I$ and select the K largest scores, breaking ties by smaller position index. Call this set S. Each head uses the procedure in (b), computing scores and applying softmax only on S, and summing latent vectors over S. Keep all n latent vectors and search keys for future steps.

(i) **(2 points)** Compute the cache size in scalars and the total multiply-add count $C(K)$, including the indexer and the products $U_{K,h} q_h$. Exclude forming q^I and the cached search keys. Does selecting fewer positions reduce this cache size?

(ii) **(3 points)** Take $H = 4$, $d = 10$, $d_c = 20$, $d_I = 2$, and $n = 100$. Under a budget of 5000 multiply-adds per step, find the largest allowed K and verify that $K+1$ exceeds the budget. Compute the cache size in scalars and cost per step for full KV, MLA over all positions, and indexed MLA at this K. Which procedures meet both this compute budget and a cache budget of 2500 scalars? Compare the indexer's score cost with the main-attention work saved by selecting K positions.

(iii) **(1 point)** For $H = d = d_c = 1$, $n = 2$, $q = 0$, $U_K = U_V = 1$, $c_1 = 0$, $c_2 = 1$, and $S = \{1\}$, compute the full output o using both positions and the output o^S obtained by applying softmax and summing values only over S.

Solution. For (i), the cache stores $n(d_c + d_I)$ scalars for every choice of K. It retains all positions, since a later query may select different ones. The indexer costs nd_I multiply-adds. The four stages in (b) now cost Hdd_c, HKd_c, HKd_c, and Hdd_c, so

$$C(K) = nd_I + 2Hdd_c + 2HKd_c.$$

For (ii), $C(K) = 1800 + 160K$. Hence

$$K_{\max} = \left\lfloor \frac{5000 - 1800}{160} \right\rfloor = 20.$$

The value $K = 20$ is at most n, and $C(21) = 5160 > 5000$. The three procedures have costs

Procedure	Cache (scalars)	Multiply-adds
Full KV	8000	8000
MLA, all positions	2000	17,600
Indexed MLA, $K = 20$	2200	5000

Both MLA procedures fit the cache budget; full KV does not. Only indexed MLA meets both budgets. With $d_c > d$, MLA over all positions uses more arithmetic than full KV attention, even though its cache is smaller. The indexer scores all positions in dimension 2 for 200 multiply-adds. Restricting the dimension-20 main attention to 20 positions saves $2H(n - K)d_c = 12{,}800$ multiply-adds relative to MLA over all positions.

For (iii), both full scores are zero, so $a = (1/2, 1/2)$ and $o = 1/2$. On $S = \{1\}$, softmax gives weight 1 to $c_1 = 0$, so $o^S = 0$.

(d) **(2 optional points)** Concentrated attention weights, as observed in H_2O (Heavy-Hitter Oracle) [4], motivate selecting a set that retains a large share of one head's attention weight. Fix one head's full weights $a_i > 0$, with $\sum_i a_i = 1$. For a size-K set S, let $m(S) = \sum_{i \in S} a_i$ and define $a_i^S = a_i / m(S)$ on S, and $a_i^S = 0$ elsewhere. Using the KL divergence from Assignment 1, Problem 3, prove

$$\mathrm{KL}(a^S \| a) = -\log m(S) \le -\log(1 - \varepsilon) \quad \text{if } m(S) \ge 1 - \varepsilon, \quad 0 \le \varepsilon < 1.$$

Here log is natural. With all a_i available, which size-K set minimizes this divergence? Prove your answer, breaking ties by smaller index. Explain why ranking the attention scores $s_{h,i}$ gives the same set without computing softmax.[4]

Solution. On S, $a_i^S / a_i = 1/m(S)$ and $\sum_{i \in S} a_i^S = 1$. Therefore

$$\mathrm{KL}(a^S \| a) = \sum_{i \in S} \frac{a_i}{m(S)} \log \frac{1}{m(S)} = -\log m(S).$$

[4] The minimizing set is for one head's attention scores $s_{h,i}$. The shared indexer may choose a different set.

Since $-\log$ is decreasing, $m(S) \geq 1 - \varepsilon$ gives the bound. For comparison, $\mathrm{KL}(a \| a^S)$ is infinite when $K < n$, since a^S vanishes at some positions where a is positive.

Minimizing the divergence is equivalent to maximizing $m(S)$. Thus the K largest a_i minimize the divergence: exchanging a smaller selected weight for a larger unselected weight increases $m(S)$. Ties give equally good choices; use the smaller index as specified. Exponentiation and division by a common positive denominator preserve order, so ranking the main scores gives the same set. To apply the error bound to the indexer's set, one would still need to check that it retains mass at least $1 - \varepsilon$.

3 Allocating precision in quantization (10 points)

Mixed-precision quantization assigns more bits to some weights than to others. We consider a row allocation problem motivated by HAWQ-V3 [5].

Let $W \in \mathbb{R}^{p \times d}$ be a linear layer's weight matrix and let $X \in \mathbb{R}^{d \times m}$ contain m calibration inputs. The error of a quantized matrix Q is

$$D(Q) = \frac{1}{m} \|(Q - W)X\|_F^2, \qquad \|A\|_F^2 = \sum_{i,j} A_{ij}^2.$$

Fix grids $\mathcal{Q}_L \subset \mathcal{Q}_H$ with 2^{b_L} and 2^{b_H} values, where $1 \leq b_L < b_H$ are integers. Round each entry of W to the nearest value in each grid, breaking ties toward the smaller value, to obtain the low-precision matrix Q^L and high-precision matrix Q^H. For a set of rows S, let $Q(S)$ use row i of Q^H when $i \in S$ and row i of Q^L otherwise. Choose S to minimize $D(Q(S))$ subject to the storage budget in (b).

(a) **(2 points)** Write w_i, q_i^L, q_i^H for the transposes of row i of W, Q^L, Q^H, and define

$$D_i^L = \frac{1}{m} \|X^\top (q_i^L - w_i)\|_2^2, \qquad D_i^H = \frac{1}{m} \|X^\top (q_i^H - w_i)\|_2^2.$$

Find Δ_i in terms of these errors so that

$$D(Q(S)) = D(Q^L) - \sum_{i \in S} \Delta_i.$$

Prove the identity by summing squared errors over output rows, without any assumption on correlations in X.

Solution. By the definition of the Frobenius norm, the total squared error is the sum of the squared errors in the output rows:

$$D(Q(S)) = \sum_{i \notin S} D_i^L + \sum_{i \in S} D_i^H = D(Q^L) - \sum_{i \in S} (D_i^L - D_i^H).$$

Thus $\Delta_i = D_i^L - D_i^H$ is the decrease in calibration error when row i changes to high precision. The identity holds for every X.

(b) **(5 points)** A low-precision weight uses b_L bits and a high-precision weight uses b_H bits. In addition to the weights, store both grids and one bit per row indicating its precision. The grids and row indicators use C bits in total, independent of S. The storage budget is B bits. Assume it covers this cost and all pd weights at low precision, i.e., $B \geq C + pdb_L$.[5]

(i) **(1 point)** Find the extra storage c for each row that is changed to high precision and the largest number $L \leq p$ of such rows allowed by the budget.

(ii) **(2 points)** Write pseudocode that starts with $S = \varnothing$ and traverses all p rows in decreasing order of Δ_i, breaking ties by smaller index. Add a row if and only if it strictly reduces D and the new set fits the budget. Express both conditions using Δ_i, $|S|$, and L, with $|S|$ measured before adding the row. Return S after the traversal.

(iii) **(2 points)** Prove that the returned set S_{alg} minimizes $D(Q(S))$ over all sets allowed by the budget. (*Hint:* Every row replacement costs the same number of bits. Use an exchange argument and the identity in (a).)

Solution. For (i), storing all rows at low precision costs $B_0 = C + pdb_L$ bits. Each row changed to high precision adds $c = d(b_H - b_L) > 0$ bits. Thus $Q(S)$ costs $B_0 + c|S|$ bits, and the maximum number of high-precision rows is

$$L = \min\left\{p, \left\lfloor \frac{B - B_0}{c} \right\rfloor\right\}.$$

For (ii), add row i exactly when $\Delta_i > 0$ and $|S| < L$:

Sort $i_1, \ldots, i_p$ by decreasing Δ_i, then increasing index.
$S \leftarrow \varnothing$
for $j = 1, \ldots, p$:
  if $\Delta_{i_j} > 0$ **and** $|S| < L$:
    $S \leftarrow S \cup \{i_j\}$
return S

The algorithm selects up to L rows with the largest positive benefits. If $L = 0$ or all benefits are nonpositive, it returns the empty set. The condition $\Delta_i > 0$ is needed: a finer grid can increase output error if rounding changes how the weight errors cancel.

For (iii), take any feasible set R. Remove rows with nonpositive benefit, then add positive-benefit rows until L rows are selected or all positive benefits are included. By (a), these changes cannot increase error. Replace any selected row with an unselected row of larger benefit; each exchange preserves storage and decreases error. Repeating these exchanges and breaking ties by index gives S_{alg}. Thus $D(Q(S_{\text{alg}})) \leq D(Q(R))$ for every feasible R. Tied or zero benefits may allow other optimal sets.

[5] Count the stored grids, row indicators, and quantized weights only. Do not round storage up to whole bytes or machine words, or count temporary memory used during calibration.

(c) **(3 points)** To examine the role of calibration data, take

$$W = \begin{pmatrix} 2/5 & 0 \\ 0 & 1/5 \end{pmatrix}, \qquad \mathcal{Q}_L = \{0, 1\}, \qquad \mathcal{Q}_H = \{0, 1/3, 2/3, 1\}, \qquad B = C + 6.$$

For $X_A = I_2$ and $X_B = \mathrm{diag}(1, 3)$, compute Q^L, Q^H, the two benefits, and the unique optimal set S under the budget. Can any deterministic rule that receives W, the grids, and the storage parameters, but not X, choose an optimal set for both calibration matrices? Justify your answer.

Solution. The quantized matrices are $Q^L = 0$ and $Q^H = \mathrm{diag}(1/3, 1/3)$. Here $p = d = m = 2$, $b_L = 1$, and $b_H = 2$. Storing both rows at low precision uses $C + 4$ bits. Changing one row to high precision costs 2 more bits, so the feasible sets are $\varnothing, \{1\}, \{2\}$. For the two calibration matrices,

Inputs	i	D_i^L	D_i^H	Δ_i
X_A	1	$2/25$	$1/450$	$7/90$
X_A	2	$1/50$	$2/225$	$1/90$
X_B	1	$2/25$	$1/450$	$7/90$
X_B	2	$9/50$	$2/25$	$1/10$

Both benefits are positive, and the budget allows one high-precision row. For X_A, row 1 has the larger benefit, so $S = \{1\}$ is the unique optimum, with error $1/45$. For X_B, row 2 has the larger benefit, so $S = \{2\}$ is the unique optimum, with error $4/25$.

Without X, a deterministic rule receives identical inputs in both cases. It must return the same set, so it cannot match both unique optima. Scaling the second input coordinate by 3 multiplies row 2's squared errors and benefit by 9, changing the preferred row.

4 Computing Muon's matrix normalization (10 points)

Muon (MomentUm Orthogonalized by Newton–Schulz, Lecture 3) approximately replaces the nonzero singular values of a momentum matrix by 1 [6]. We analyze an iteration for this normalization, starting from the momentum matrix.

Let $M \in \mathbb{R}^{m \times n}$, with $m \geq n$, have full column rank. Write its thin singular value decomposition (SVD) as

$$M = U\Sigma V^\top, \qquad \Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_n), \quad \sigma_i > 0, \qquad U^\top U = V^\top V = I_n,$$

where $U \in \mathbb{R}^{m \times n}$ and $V \in \mathbb{R}^{n \times n}$. The ideal normalized matrix is the polar factor $P = UV^\top$. Consider the classical cubic Newton–Schulz iteration

$$X_0 = \frac{M}{\|M\|_F}, \qquad X_{j+1} = \frac{1}{2} X_j (3I_n - X_j^\top X_j), \qquad j = 0, 1, \ldots.$$

We use the cubic iteration to analyze convergence. The Muon implementation in [6] uses a different polynomial and a fixed iteration count.

(a) **(2 points)** For $M = \mathrm{diag}(10, 1)$, compute $M/\|M\|_F$ and P. Let $v_1 = (1, 0)^\top$, $v_2 = (0, 1)^\top$, and $\eta > 0$. Define the updates

$$\Delta_{\text{scalar}} = -\eta M/\|M\|_F, \qquad \Delta_{\text{polar}} = -\eta P.$$

For each update, compute $\|\Delta v_1\|_2$, $\|\Delta v_2\|_2$, and their ratio $\|\Delta v_1\|_2/\|\Delta v_2\|_2$. Which normalization gives equal update magnitudes in these two directions?

Solution. Here $\|M\|_F = \sqrt{101}$, so

$$\frac{M}{\|M\|_F} = \mathrm{diag}\left(\frac{10}{\sqrt{101}}, \frac{1}{\sqrt{101}}\right), \qquad P = I_2.$$

Scalar normalization gives magnitudes $10\eta/\sqrt{101}$ and $\eta/\sqrt{101}$ along v_1 and v_2, with ratio 10. Polar normalization gives magnitude η in each direction, with ratio 1. It makes the two singular values equal; dividing by $\|M\|_F$ preserves their ratio. The updates have Frobenius norms η and $\eta\sqrt{2}$, respectively, despite using the same η.

(b) **(3 points)** Show that the iterates can be written using the same matrices U, V as

$$X_j = U\,\mathrm{diag}(s_{1,j}, \ldots, s_{n,j})V^\top.$$

Derive the scalar recurrence for $s_{i,j}$ and its initial value. Show that every initial value lies in $(0, 1]$.

Solution. Initially,

$$s_{i,0} = \frac{\sigma_i}{\|M\|_F} = \frac{\sigma_i}{\sqrt{\sum_{\ell=1}^n \sigma_\ell^2}} \in (0, 1].$$

Suppose $X_j = UD_jV^\top$ for diagonal D_j. Orthonormality gives $X_j^\top X_j = VD_j^2V^\top$, and therefore

$$X_{j+1} = U\frac{3D_j - D_j^3}{2}V^\top.$$

This proves the representation by induction, with scalar recurrence

$$s_{i,j+1} = f(s_{i,j}), \qquad f(s) = \frac{1}{2}s(3 - s^2).$$

Part (c) shows that the coefficients stay positive, so U and V remain singular-vector matrices for every iterate. The SVD is used only in the proof; the algorithm computes each iterate by matrix multiplication.

(c) **(3 points)** Prove that a scalar sequence following the recurrence in (b), starting in $(0, 1]$, stays in $(0, 1]$, is nondecreasing, and converges to 1. Deduce that $X_j \to P$. (*Hint:* Let f be the scalar update function derived in (b). For $0 < s \leq 1$, examine $f(s) - s$ and $1 - f(s)$, then identify possible limits.)

Solution. For $0 < s \leq 1$,

$$f(s) - s = \frac{s(1 - s^2)}{2} \geq 0, \qquad 1 - f(s) = \frac{(1 - s)^2(s + 2)}{2} \geq 0.$$

Thus $0 < s \leq f(s) \leq 1$. By induction, the sequence stays in $(0, 1]$ and is nondecreasing. It is bounded above by 1, so it converges to some $\ell \geq s_0 > 0$. Continuity gives $\ell = f(\ell)$, hence $\ell(1 - \ell^2) = 0$. Since $\ell > 0$, we have $\ell = 1$.

Each of the n singular values therefore converges to 1, giving

$$\|X_j - P\|_F^2 = \sum_{i=1}^{n} (s_{i,j} - 1)^2 \longrightarrow 0,$$

which proves convergence of the matrix iteration in exact arithmetic.

(d) **(2 points)** Now let $M \neq 0$ have rank $r < n$, with the same thin SVD notation but $\sigma_1, \ldots, \sigma_r > 0$ and $\sigma_{r+1} = \cdots = \sigma_n = 0$.[6] Determine the limit of X_j. Does it have orthonormal columns? Explain why an initially zero singular value remains zero.

Solution. Since $f(0) = 0$, a zero singular value stays zero. The r positive singular values converge to 1 by part (c), so

$$X_j \longrightarrow P_r = U \operatorname{diag}(\underbrace{1, \ldots, 1}_{r}, 0, \ldots, 0) V^\top = \sum_{i=1}^{r} u_i v_i^\top.$$

The limit P_r is the partial polar factor. Its columns are not orthonormal, since $P_r^\top P_r = \sum_{i=1}^{r} v_i v_i^\top$ has rank $r < n$ and therefore differs from I_n. Every iterate, and the limit, sends the nullspace of M to zero.

(e) **(2 optional points)** For a positive singular-value sequence in (b), let $e_j = 1 - s_j$. Derive a recurrence for e_j. Show that, once $e_j \leq 1/2$, the next error is at most $\frac{3}{2} e_j^2$. For $0 < s_0 < 1$, compute $\lim_{j \to \infty} e_{j+1}/e_j^2$ to establish quadratic convergence. If $s_0 = 1$, give closed-form expressions for s_j and e_j for every $j \geq 0$.

Solution. Substituting $s_j = 1 - e_j$ gives

$$e_{j+1} = \frac{1}{2} e_j^2 (3 - e_j).$$

For $0 \leq e_j \leq 1/2$, $\frac{5}{4} e_j^2 \leq e_{j+1} \leq \frac{3}{2} e_j^2$. If $0 < s_0 < 1$, then $0 < s_j < 1$ for every finite j, since $1 - f(s) > 0$ for $0 < s < 1$. Hence $e_j > 0$, and

$$\lim_{j \to \infty} \frac{e_{j+1}}{e_j^2} = \lim_{j \to \infty} \frac{3 - e_j}{2} = \frac{3}{2}.$$

[6] For $M = 0$, define the normalized matrix and optimizer update to be zero.

The convergence is therefore quadratic. If $s_0 = 1$, then $s_j = 1$ and $e_j = 0$ for every j, so the ratio is undefined. The asymptotic rate does not give a uniform iteration bound over all initial values $s_0 \in (0, 1]$.

Sources

[1] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *ICLR*, 2017. Sections 2–4.

[2] Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding. *ICLR*, 2021. Section 2.

[3] DeepSeek-AI. DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models. 2025.

[4] Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, Zhangyang Wang, and Beidi Chen. H_2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. *NeurIPS*, 2023. Section 3.

[5] Zhewei Yao, Zhen Dong, Zhangcheng Zheng, Amir Gholami, Jiali Yu, Eric Tan, Leyuan Wang, Qijing Huang, Yida Wang, Michael Mahoney, and Kurt Keutzer. HAWQ-V3: Dyadic Neural Network Quantization. *ICML*, 2021. Section 3.4.

[6] Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks. Technical blog post, 2024.