

# CS182 hw9 special participation a)

Nils Valseth Selte

October 2025

## 1 Q1

### 1.1 Q1a) Prompt

Hi I am a student learning about Deep Neural networks, and I need to test you on some problems related to the subject.

1. SSM Convolution Kernel Background and Setup: Consider a discrete-time State-Space Model (SSM) of the form  $x_{k+1} = Ax_k + Bu_k$ ,  $y_k = Cx_k + Du_k$ , (a) Convolution Kernel and the Output Equation. Given that the sequence length is  $L$  (input:  $(u_0, \dots, u_L)$ , output:  $(y_0, \dots, y_L)$ ) and assume  $x_0 = 0$ , show that the output  $y_k$  can be expressed as a convolution of the input sequence  $u_L$  with a kernel  $K = K_L$ :  $y_k = \sum_{l=0}^L K_l u_{k-l}$  where any  $u_l$  with a negative index is set to 0 (zero-padding). Also, find  $K$ .

### 1.2 Q1a) Kimi Response

Let's solve this step-by-step, clearly and carefully.

---

\*\*Step 1: Clearly restating the problem\*\*

We have a discrete-time linear state-space model (SSM):

$$x_{k+1} = Ax_k + Bu_k, \quad y_k = Cx_k + Du_k$$

with initial condition  $x_0 = 0$ . The input sequence is given by:

$$\{u_0, u_1, \dots, u_L\}$$

and we want to express the output sequence  $\{y_0, y_1, \dots, y_L\}$  as a convolution of the input sequence with a kernel  $K = \{K_0, K_1, \dots, K_L\}$ :

$$y_k = \sum_{l=0}^L K_l u_{k-l}, \quad \text{with } u_{k-l} = 0 \text{ if } k - l < 0.$$

We must explicitly find the kernel  $K_l$ .

---

\*\*Step 2: Iteratively compute the state sequence\*\*

Let's compute the first few states explicitly to clearly identify the pattern:

- \*\*Initial condition:\*\*  $x_0 = 0$
- \*\*Step 0:\*\*

$$x_1 = Ax_0 + Bu_0 = Bu_0$$

- \*\*Step 1:\*\*

$$x_2 = Ax_1 + Bu_1 = ABu_0 + Bu_1$$

- \*\*Step 2:\*\*

$$x_3 = Ax_2 + Bu_2 = A^2Bu_0 + ABu_1 + Bu_2$$

Clearly, we see a pattern emerging. By induction, we have the general expression for the state at step  $k$ :

$$x_k = \sum_{l=0}^{k-1} A^{k-1-l} Bu_l$$


---

\*\*Step 3: Substitute into the output equation\*\*

Now, substitute this expression for  $x_k$  into the output equation:

$$y_k = Cx_k + Du_k$$

Using the expression we found for  $x_k$ :

$$y_k = C \left( \sum_{l=0}^{k-1} A^{k-1-l} Bu_l \right) + Du_k$$

We can rewrite this clearly as:

$$y_k = \sum_{l=0}^{k-1} CA^{k-1-l} Bu_l + Du_k$$


---

\*\*Step 4: Rewrite as a convolution\*\*

We now rewrite the above expression explicitly as a convolution. To do this clearly, let's define the convolution kernel  $K_l$ :

- For  $l = 0$ :

$$K_0 = D$$

- For  $l \geq 1$ :

$$K_l = CA^{l-1}B$$

Then, we can clearly express the output  $y_k$  as a convolution sum:

$$y_k = \sum_{l=0}^k K_l u_{k-l}$$

with the convention that  $u_{k-l} = 0$  if  $k - l < 0$  (zero-padding).

---

\*\*Step 5: Clearly summarize the final result\*\*

Thus, the convolution kernel  $K = \{K_0, K_1, \dots, K_L\}$  is explicitly given by:

$$K_l = \begin{cases} D, & l = 0 \\ CA^{l-1}B, & l \geq 1 \end{cases}$$

and the output sequence  $y_k$  is clearly expressed as the convolution:

$$y_k = \sum_{l=0}^L K_l u_{k-l}, \quad \text{with } u_{k-l} = 0 \text{ if } k - l < 0.$$

\*\*Final Answer (clearly stated):\*\*

- \*\*Convolution kernel:\*\*

$$K_l = \begin{cases} D, & l = 0 \\ CA^{l-1}B, & l \geq 1 \end{cases}$$

- \*\*Output as convolution:\*\*

$$y_k = \sum_{l=0}^L K_l u_{k-l}, \quad \text{zero-padding for negative indices.}$$

### 1.3 Q1a) Interpretation

Kimi was able to one-shot the problem without problems, with good understandability, there are some formatting issues with the Latex here.

## 2 Q1b)

### 2.1 Q1b) Prompt

b) Concrete Examples. i. Scalar Case: Let  $n = 1$ , and set  $A = , B = , C = , D = .$  Use  $= 0.8, = 1, = 1.5$  and compute the kernel up to  $L = 4$ . ii. 2D Case: Let  $A \in \mathbb{R}^{2 \times 2}$  be, for instance,  $A = \begin{pmatrix} 0.7 & 0.1 \\ 0.2 & 0.6 \end{pmatrix}$ ,  $B = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ ,  $C = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ ,  $D = 0$  Compute kernels up to  $L = 3$  and briefly discuss how the kernel captures the “impulse response”.

### 2.2 Q1b) Kimi Response

Let's work out the two concrete examples carefully and explicitly.

(b) Concrete Examples

\*\*(i) Scalar Case ( $n = 1$ )\*\*

Given scalar parameters:

-  $A = \alpha = 0.8$  -  $B = \beta = 1$  -  $C = \gamma = 1.5$  -  $D = \delta$  (we'll keep it symbolic for clarity initially)  
We have the general kernel formula from part (a):

$$K_l = \begin{cases} D, & l = 0 \\ CA^{l-1}B, & l \geq 1 \end{cases}$$

Thus, explicitly:

$$\begin{aligned} - K_0 &= D - K_1 = CB = 1.5 \times 1 = 1.5 - K_2 = CAB = 1.5 \times 0.8 \times 1 = 1.2 - K_3 = CA^2B = \\ &1.5 \times (0.8)^2 \times 1 = 1.5 \times 0.64 = 0.96 - K_4 = CA^3B = 1.5 \times (0.8)^3 \times 1 = 1.5 \times 0.512 = 0.768 \end{aligned}$$

Thus, the kernel up to  $L = 4$  is clearly:

$$\begin{array}{ccccccccc} -1 & -\text{Kernel } K_l & - & - & -0 & -D & - & -1 & -1.5 & - & -2 & - & -1.2 & - & -3 & - \\ 0.96 & - & - & -4 & - & 0.768 & - & - & - & - & - & - & - & - & - & - \end{array}$$

\*\*(ii) 2D Case\*\*

Given matrices and vectors:

$$A = \begin{pmatrix} 0.7 & 0.1 \\ 0.2 & 0.6 \end{pmatrix}, \quad B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & 0 \end{pmatrix}, \quad D = 0$$

Again, using the general formula:

$$- K_0 = D = 0 - K_l = CA^{l-1}B \quad \text{for } l \geq 1$$

Let's compute explicitly up to  $L = 3$ :

$$- **\text{Step 1:** Compute } A^0 = I, \text{ thus:}$$

$$K_1 = CA^0B = CB = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 1$$

$$- **\text{Step 2:** Compute } A^1 = A:$$

$$K_2 = CAB = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 0.7 & 0.1 \\ 0.2 & 0.6 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = (1)(0.7) + (0)(0.2) = 0.7$$

$$- **\text{Step 3:** Compute } A^2:$$

$$A^2 = A \cdot A = \begin{pmatrix} 0.7 & 0.1 \\ 0.2 & 0.6 \end{pmatrix} \begin{pmatrix} 0.7 & 0.1 \\ 0.2 & 0.6 \end{pmatrix} = \begin{pmatrix} 0.51 & 0.13 \\ 0.26 & 0.38 \end{pmatrix}$$

Then:

$$K_3 = CA^2B = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 0.51 & 0.13 \\ 0.26 & 0.38 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 0.51$$

Thus, clearly summarizing the kernel up to  $L = 3$ :

$$\begin{array}{ccccccccc} -1 & -\text{Kernel } K_l & - & - & -0 & -0 & - & -1 & -1 & - & -2 & - & -0.7 & - & -3 & - \\ 0.51 & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \end{array}$$

\*\*Brief Discussion on Impulse Response Interpretation\*\*

The kernel  $K_l$  we computed above is exactly the impulse response of the given linear system. To see this clearly, consider an impulse input:

$$u_0 = 1, \quad u_k = 0 \quad \text{for } k \geq 1$$

Then, the output sequence is precisely the kernel sequence:

$$y_0 = K_0 = 0, \quad y_1 = K_1 = 1, \quad y_2 = K_2 = 0.7, \quad y_3 = K_3 = 0.51$$

Thus, the kernel clearly captures how the system responds to an impulse input, decaying over time due to the stable eigenvalues of matrix  $A$ . This explicitly illustrates the intuitive interpretation of the convolution kernel as the impulse response of the linear system.

---

\*\*Final Answers Clearly Summarized:\*\*

\*\*(i) Scalar kernel up to  $L = 4$ :\*\*

— 1 — Kernel  $K_l$  —————— 0 —  $D$  — — 1 — 1.5 — — 2 — 1.2 — — 3 —  
0.96 — — 4 — 0.768 —

\*\*(ii) 2D kernel up to  $L = 3$ :\*\*

— 1 — Kernel  $K_l$  —————— 0 — 0 — — 1 — 1 — — 2 — 0.7 — — 3 —  
0.51 —

The kernel clearly represents the impulse response of the system, explicitly showing how the system output decays over time.

### 2.3 Q1b) Interpretation

This answer is both accurate and the reasoning is easy to follow. one-shot!

## 3 Q1c)

### 3.1 Q1c) Prompt

Efficient Computation with Convolutions. If we already know the kernel  $K$ , how much can we parallelize the computation of the output sequence  $y_k$  for an input sequence  $u_k$   $R_d$  of length  $L$ ? What is the minimum critical path length of the computation? What about a naive, direct computation of  $y_k$  from the unrolled recursion?

### 3.2 Q1c) Kimi Response

Efficient Computation with Convolutions

**Problem.** Given a pre-computed kernel

$$K = \{K_0, K_1, \dots, K_{L-1}\}$$

and an input sequence

$$\{u_0, u_1, \dots, u_{L-1}\} \in \mathbb{R}^L,$$

how much parallelism is available for computing the output sequence

$$y_k = \sum_{\ell=0}^k K_\ell u_{k-\ell}, \quad k = 0, \dots, L-1 ?$$

What is the *minimum critical-path length* (depth)? Compare this with a naive, direct evaluation of the state-space recursion.

---

### 1. Convolution-based evaluation

The formula above is exactly a *linear (acyclic) convolution* of length  $L$ .

- *Work* (total multiply-adds):  $\Theta(L^2)$ .
- *Critical-path depth* on an optimal PRAM/CRCW model:
  - ▷  $\Theta(\log L)$  time using  $\Theta(L^2)$  processors, or
  - ▷  $\Theta(\log L)$  time using only  $\Theta(L \log L)$  processors via the -way divide-and-conquer algorithm [?].

These bounds are the best known for any algebraic convolution of length  $L$ .

Hence, with the kernel pre-computed, the entire output sequence can be obtained with

$$\boxed{\text{critical-path depth} = \Theta(\log L)}.$$


---

### 2. Naive direct recursion (unrolled state equations)

Recall the state-space model

$$x_{k+1} = A x_k + B u_k, \quad y_k = C x_k + D u_k.$$

Each step contains an  $n \times n$  matrix–vector product that *depends on the previous state*: the  $L$  updates form an inherently sequential chain.

- Per-step depth (parallel  $n \times n$  MV):  $\Theta(\log n)$ .
- Total critical-path depth after  $L$  steps:

$$\boxed{\Theta(L \log n)}.$$

- Total work:  $\Theta(L n^2)$ .
- 

### Summary

Method	Critical-path depth	Work
Pre-computed convolution	$\Theta(\log L)$	$\Theta(L \log L) \dots \Theta(L^2)$
Direct state recursion	$\Theta(L \log n)$	$\Theta(L n^2)$

Thus, knowing the kernel reduces the *sequential*  $L$ -step chain to a *logarithmic-depth* computation, yielding an exponential speed-up on a parallel machine.

### 3.3 Q1c) Interpretation

At this point I got the brilliant idea of asking Kimi to output in Latex. I find this response to harder to read and it doesn't explain the solution well other than just presenting the answer in an easy to read way.

## 4 Q1d)

### 4.1 Q1d) Prompt

Again give the answer in Beautiful Latex. Efficient Kernel Computation. Given A, B, C, how can we compute the kernel, K, efficiently? What are some strategies to parallelize kernel computation? You may assume L = 2N for some N for simplicity.

### 4.2 Q1d) Kimi Response

Efficient Kernel Computation

**Problem.** Given the state-space matrices  $A, B, C$  (and  $D$ ), compute the convolution kernel

$$K = \{K_0, K_1, \dots, K_{L-1}\}, \quad K_\ell = \begin{cases} D, & \ell = 0, \\ CA^{\ell-1}B, & \ell \geq 1, \end{cases}$$

efficiently. Assume for simplicity  $L = 2^N$  for some integer  $N \geq 0$ . How much parallelism is available?

---

#### 1. Naive sequential cost

Evaluating

$$K_\ell = CA^{\ell-1}B \quad \text{for } \ell = 1, \dots, L-1$$

by repeated multiplication costs

$$\Theta(L n^3) \quad \text{work} \quad \text{and} \quad \Theta(L) \quad \text{depth} ,$$

clearly sub-optimal.

---

#### 2. Optimal work, logarithmic depth

Observe that the *whole* kernel is encoded in the matrix power series

$$\mathcal{K}(z) = \sum_{\ell=0}^{L-1} K_\ell z^\ell = D + zC \left( \sum_{\ell=0}^{L-2} A^\ell z^\ell \right) B .$$

Hence we need the *matrix powers*

$$I, A, A^2, \dots, A^{L-2} .$$

*Parallel prefix (scan) algorithm* [?]:

1. Compute the *block-prefix products*

$$P_k = A^{2^k} \quad \text{for } k = 0, \dots, N-1 \quad (\text{binary powering}) .$$

Depth:  $\Theta(\log L)$ ; work:  $\Theta(n^3 \log L)$ .

2. Run a *scan* (parallel prefix) on the sequence

$$(I, A, A^2, \dots, A^{L-2})$$

using the semigroup operation  $(X, Y) \mapsto X \cdot Y$ . Each semigroup multiply is an  $n \times n$  matrix product.

3. Post-multiply every prefix by  $B$  and pre-multiply by  $C$  to obtain

$$K_\ell = CA^{\ell-1}B \quad (\ell \geq 1) .$$

**Complexity with  $p \leq n^3(L/\log L)$  processors**

Work = $\Theta(L n^3)$ , Critical-path depth = $\Theta(\log L \cdot \log n)$ .
---

The work is *optimal* (no more than the sequential algorithm that simply writes down the answers), while the depth is only a *logarithmic factor* beyond the theoretical lower bound  $\Omega(\log L)$ .

---

### 3. FFT acceleration for very large $L$

When  $L \gg n$  the dominant cost is the  $L$  matrix–vector products, not the  $n^3$  arithmetic. Embed the problem in a *cyclic convolution* of length  $2L$  and use

- ▷ *FFT-based polynomial multiplication* on matrix polynomials, or
- ▷ *number-theoretic transform* (NTT) over a sufficiently large ring.

Complexity (parallel):

Work = $\Theta(L n^2 \log L)$ , Depth = $\Theta(\log L)$ .
---

This is *work-efficient* when  $L \geq n \log n$  and gives the *minimum possible depth*.

---

### 4. Practical hybrid strategy

- Choose a block size  $b = \Theta(\log L)$ .

(b) Compute  $A^b$  once (binary powering).

(c) Generate *block-prefixes*

$$A^b, A^{2b}, \dots, A^{\lfloor (L-2)/b \rfloor b}$$

with a parallel scan.

(d) Inside each length- $b$  segment expand sequentially:

$$A^{kb+j} = A^{kb} \cdot A^j, \quad j = 0, \dots, b-1.$$

This keeps all caches hot, avoids excessive parallelism overhead, and still retains

$$\text{depth} = \Theta(\log L \cdot \log n).$$

### Take-away

Whether by *parallel prefix* or *FFT-based polynomial methods*, the kernel

$$K = \{K_\ell\}_{\ell=0}^{L-1}$$

can be produced in

$$\boxed{\text{optimal work } \Theta(L n^3) \text{ and logarithmic depth } \Theta(\log L \cdot \log n)},$$

an exponential speed-up over the naive sequential chain.

## 4.3 Q1d) Interpretation

Kimi not only failed to produce rendering LATEX. But i think its reasoning is hard to follow and very advanced for the scope of what i am trying to learn here.

## 5 Q1e)

### 5.1 Q1e) Prompt

again in latex. Adding structure to A. Suppose A is a diagonal matrix. How can we leverage this structure to compute the kernel K more efficiently?

## 5.2 Q1e) Kimi Response

Adding Structure to  $A$  : Diagonal Case

**Setup.** Let  $A \in \mathbb{R}^{n \times n}$  be *diagonal*:

$$A = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n).$$

We need the kernel

$$K_\ell = \begin{cases} D, & \ell = 0, \\ CA^{\ell-1}B, & \ell \geq 1. \end{cases}$$


---

### 1. Closed-form entry-wise expression

Because  $A$  is diagonal,

$$A^k = \text{diag}(\lambda_1^k, \lambda_2^k, \dots, \lambda_n^k).$$

Let

$$C = [c_1 \quad \cdots \quad c_n] \in \mathbb{R}^{m \times n}, \quad B = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \in \mathbb{R}^{n \times p}.$$

Then

$$CA^{\ell-1}B = \sum_{i=1}^n c_i b_i^\top \lambda_i^{\ell-1}.$$

Hence every entry of the kernel is a *sum of geometric sequences*:

$$[K_\ell]_{r,s} = \sum_{i=1}^n (c_i)_r (b_i)_s \lambda_i^{\ell-1}, \quad \ell = 1, \dots, L-1.$$


---

### 2. Complexity reduction

- *Naive dense method:*  $\Theta(L n^3)$  work,  $\Theta(L)$  depth.
- *Diagonal leverage:*
  - ▷ Pre-compute the outer products  $c_i b_i^\top$  once:  $\Theta(n m p)$  work,  $\Theta(\log n)$  depth.
  - ▷ For each  $\ell$  evaluate *scalar* powers  $\lambda_i^{\ell-1}$  and accumulate:

$$[K_\ell]_{r,s} = \sum_{i=1}^n \underbrace{(c_i)_r (b_i)_s}_{\text{pre-stored}} \lambda_i^{\ell-1}.$$

**Total cost for all  $\ell = 1, \dots, L - 1$ :**

Work	= $\Theta(L n m p)$ ,
Depth	= $\Theta(\log n)$ (parallel sum-reduction per entry).

When  $m, p = O(1)$  this is *linear in  $n$*  instead of cubic—a speed-up of  $\Theta(n^2)$ .

---

### 3. Parallel geometric-sequence evaluation

To obtain *all* powers

$$\lambda_i^0, \lambda_i^1, \dots, \lambda_i^{L-2}, \quad i = 1, \dots, n,$$

in logarithmic depth:

- i. *Binary-prefix (scan) trick*: treat each scalar sequence as a *prefix product*

$$(1, \lambda_i, \lambda_i^2, \dots, \lambda_i^{L-2}).$$

A parallel scan over length  $L - 1$  runs in

$$\Theta(\log L) \text{ depth and } \Theta(L) \text{ work per } \lambda_i.$$

- ii. Run the  $n$  scans *simultaneously*; no data dependencies between different  $i$ .

Overall for the powers:

Depth = $\Theta(\log L)$ ,	Work = $\Theta(n L)$ .
----------------------------	------------------------

---

### 4. FFT acceleration for very large $L$

Each entry is a *discrete exponential sum*

$$f_\ell = \sum_{i=1}^n \alpha_i \lambda_i^{\ell-1}.$$

This is a *Hankel matrix–vector product* which can be embedded in a *cyclic convolution* of length  $2L$  and evaluated via

- ▷ *fast multipole method* or
- ▷ *FFT-based polynomial evaluation* (transpose of a Vandermonde product).

Complexity (parallel):

$\text{Work} = \Theta(L \log L + n L)$ , $\text{Depth} = \Theta(\log L)$ .
---

This is *work-optimal* for  $L \gg n$  and retains logarithmic depth.

---

### Take-away

When  $A$  is diagonal the kernel construction collapses to

entry-wise geometric sums
---------------------------

reducing the bottleneck from  $\Theta(n^3)$  to  $\Theta(n)$  per time index and yielding

work $\Theta(L n)$ and depth $\Theta(\log L + \log n)$ ,
--

an  $\Theta(n^2)$  parallel speed-up over the unstructured case.

## 5.3 Q1e) Interpretation

good. although in the first sum taking bi transpose doesnt make sense.

## 6 Q1f)

### 6.1 Q1f) Prompt

(f) Diagonal-plus-low-rank (DPLR) structure Now if  $A$  has the following form:  $A = I_n + pp^\top$ , where  $A \in \mathbb{R}^{n \times n}$ ,  $p \in \mathbb{R}^n$ . How can we leverage this structure to compute the kernel  $K$  more efficiently?

### 6.2 Q1f) Kimi Response

Diagonal-plus-low-rank (DPLR) structure
---

**Setup.** Let

$$A = I_n + \sigma pp^\top, \quad p \in \mathbb{R}^n, \quad \sigma \in \mathbb{R}.$$

(The scalar  $\sigma$  is introduced for generality; set  $\sigma = 1$  if desired.) We need the kernel

$$K_\ell = \begin{cases} D, & \ell = 0, \\ CA^{\ell-1}B, & \ell \geq 1. \end{cases}$$


---

### 1. Sherman–Morrison powers

The matrix  $A$  is a rank-one update of the identity. For every integer  $k \geq 0$  the power  $A^k$  can be written *exactly* as

$$A^k = I_n + \beta_k pp^\top,$$

where the scalar sequence  $\beta_k$  satisfies the recurrence

$$\boxed{\beta_0 = 0, \quad \beta_{k+1} = 1 + (1 + \sigma\|p\|^2)\beta_k.}$$

Proof by induction:

$$A^{k+1} = A^k A = (I + \beta_k pp^\top)(I + \sigma pp^\top) = I + [\sigma + \beta_k(1 + \sigma\|p\|^2)]pp^\top,$$

hence

$$\beta_{k+1} = \sigma + \beta_k(1 + \sigma\|p\|^2).$$


---

### 2. Closed-form expression for $\beta_k$

The recurrence is first-order linear:

$$\beta_{k+1} - \lambda\beta_k = \sigma, \quad \lambda := 1 + \sigma\|p\|^2.$$

Solving gives

$$\boxed{\beta_k = \sigma \frac{\lambda^k - 1}{\lambda - 1} = \frac{(1 + \sigma\|p\|^2)^k - 1}{\|p\|^2}.}$$


---

### 3. Kernel evaluation

Insert  $A^{\ell-1} = I + \beta_{\ell-1} pp^\top$  into

$$K_\ell = CA^{\ell-1}B = CB + \beta_{\ell-1}Cp p^\top B.$$

Define the *once-and-for-all* scalars

$$\alpha := CB, \quad \gamma := Cp, \quad \delta := p^\top B.$$

Then

$$\boxed{K_\ell = \alpha + \beta_{\ell-1} \gamma \delta^\top, \quad \ell = 1, \dots, L-1.}$$

Hence every matrix entry of the kernel is a *rank-one update* of the constant matrix  $\alpha$  scaled by the scalar  $\beta_{\ell-1}$ .

---

### 4. Complexity

- *Pre-processing* (once): compute  $\|p\|^2$ ,  $\lambda = 1 + \sigma\|p\|^2$ ,  $\alpha$ ,  $\gamma$ ,  $\delta$ . Cost:  $\Theta(n^2)$  work,  $\Theta(\log n)$  depth.

- Per  $\ell$ : evaluate the closed-form scalar  $\beta_{\ell-1}$  and form

$$K_\ell = \alpha + \beta_{\ell-1} \gamma \delta^\top .$$

Cost:  $\Theta(n^2)$  work,  $\Theta(1)$  depth (rank-1 update is embarrassingly parallel).

**Total for  $L$  time steps:**

$\text{Work} = \Theta(n^2 + L n^2) = \Theta(L n^2) ,$ $\text{Depth} = \Theta(\log n + \log L) .$
---

The naïve dense method requires  $\Theta(L n^3)$  work and  $\Theta(L)$  depth; the DPLR structure saves

$\Theta(n)$ in work and $\Theta(L)$ in depth .
--

---

## 5. Parallel geometric-scalar evaluation

All  $\beta_k$  can be produced simultaneously by a *parallel-prefix* scan on the scalar sequence

$$\beta_0, \beta_1, \dots, \beta_{L-2}$$

with the associative operation

$$(\beta_i, \beta_j) \mapsto \beta_{i+j} + \beta_i \beta_j \|p\|^2 ,$$

or more simply by broadcasting the closed-form

$$\beta_k = \frac{\lambda^k - 1}{\|p\|^2}$$

and evaluating the powers  $\lambda^k$  via a single logarithmic-depth scan.

Depth for all  $\beta_k$ :  $\Theta(\log L)$ ; work:  $\Theta(L)$ .

---

## Take-away

For the DPLR structure  $A = I + \sigma pp^\top$  the kernel construction collapses to

rank-one updates scaled by a <i>scalar</i> geometric sequence
---

yielding

work $\Theta(L n^2)$ and depth $\Theta(\log L + \log n)$ ,
--

an  $\Theta(n)$  parallel speed-up over the unstructured case.

### 6.3 Q1f) Interpretation

Randomly introduces a scalar

$$(\sigma)$$

which is unnecessary for this question. Further it overly complicates the calculations in my opinion.

## 7 Q3 a)

### 7.1 Q3a i) Prompt

Consider a linear encoder — square weight matrix  $W \in \mathbb{R}^{m \times m}$  — that we want to be a “purification” operation on  $m$ -dimensional feature vectors from a particular problem domain. We do this by using self-supervised learning to reconstruct  $n$  points of training data  $X \in \mathbb{R}^{m \times n}$  by minimizing the loss:  $L_1(W ; X) = \|X - WX\|_F$  (1) While the trivial solution  $W = I$  can minimize the reconstruction loss (1), we will now see how weight-decay (or equivalently in this case, ridge-style regularization) can help us achieve non-trivial purification.  $L_2(W ; X, z) = \|X - WX\|_F^2 + \|W\|_F^2 - z$  Reconstruction Loss +  $\frac{1}{2} \|W\|_F^2$  —  $z$  Regularization Loss (2) Note above that  $z$  controls the relative weighting of the two losses in the optimization. (a) Consider the simplified case for  $m = 2$  with the following two candidate weight matrices:  $W^{(\alpha)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$   $W^{(\beta)} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$  (3) The training data matrix  $X$  is also given to you as follows:  $X = \begin{bmatrix} 2.17 & 1.98 & 2.41 & 2.03 & 0.02 \\ 0.01 & 0.01 & 0.02 \end{bmatrix}$  (4) i. Compute the reconstruction loss and the regularization loss for the two encoders, and fill in the missing entries in the table below. Encoder Reconstruction Loss Regularization Loss

0.001

### 7.2 Q3a i) Kimi Response

Purification via Weight-Decay – Part (a.i)

**Given.**

Data matrix

$$X = \begin{bmatrix} -2.17 & 1.98 & 2.41 & -2.03 \\ 0.02 & -0.01 & 0.01 & -0.02 \end{bmatrix} \in \mathbb{R}^{2 \times 4}$$

and two candidate encoders

$$W^{(\alpha)} = I_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad W^{(\beta)} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}.$$

Losses to evaluate (Frobenius norm):

$$\text{Recon} = \|X - WX\|_F^2, \quad \text{Reg} = \|W\|_F^2.$$

---

### 1. Encoder $\alpha$ (identity)

$$WX = W^{(\alpha)}X = IX = X \implies \|X - WX\|_F^2 = 0.$$

$$\|W^{(\alpha)}\|_F^2 = \|I\|_F^2 = 1^2 + 1^2 = 2.$$

### 2. Encoder $\beta$ (drop 2nd coordinate)

$$WX = W^{(\beta)}X = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} -2.17 & 1.98 & 2.41 & -2.03 \\ 0.02 & -0.01 & 0.01 & -0.02 \end{bmatrix} = \begin{bmatrix} -2.17 & 1.98 & 2.41 & -2.03 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

Residual

$$X - WX = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.02 & -0.01 & 0.01 & -0.02 \end{bmatrix}.$$

Squared Frobenius norm (sum of squares of all entries):

$$\|X - WX\|_F^2 = 0^2 + 0^2 + 0^2 + 0^2 + 0.02^2 + (-0.01)^2 + 0.01^2 + (-0.02)^2 = 0.0004 + 0.0001 + 0.0001 + 0.0004 = 0.0010.$$

Regularization term

$$\|W^{(\beta)}\|_F^2 = 1^2 + 0^2 + 0^2 + 0^2 = 1.$$


---

### Completed table

Encoder	Reconstruction Loss	Regularization Loss
$\alpha$	0	2
$\beta$	0.001	1

### 7.3 Q3a i) Interpretation

good

### 7.4 Q3a ii) Prompt

latex output: ii. For what values of the regularization parameter  $\lambda$  is the identity matrix  $W(\lambda)$  get higher loss L2 in (2), as compared to  $W(0)$ ?

## 7.5 Q3a ii) Kimi Response

Part (a.ii) – When does identity lose to the drop-2nd-coordinate encoder?

**Goal.** Find all  $\lambda \geq 0$  for which

$$L_2(W^{(\alpha)}, \lambda) > L_2(W^{(\beta)}, \lambda).$$

---

### 1. Insert the numbers from part (i)

$$L_2(W^{(\alpha)}, \lambda) = \underbrace{0}_{\text{recon}} + \lambda \cdot \underbrace{2}_{\text{reg}} = 2\lambda,$$

$$L_2(W^{(\beta)}, \lambda) = \underbrace{0.001}_{\text{recon}} + \lambda \cdot \underbrace{1}_{\text{reg}} = 0.001 + \lambda.$$

---

### 2. Solve the inequality

$$2\lambda > 0.001 + \lambda \implies \lambda > 0.001.$$

---

### Answer

$$\lambda > 10^{-3}$$

For every regularization weight strictly larger than 0.001 the *identity encoder* incurs a higher total loss than the *coordinate-dropping* encoder, thereby encouraging the latter (purification) solution.

## 7.6 Q3a ii) Interpretation

Interestingly the Latex prompt is now out of context and it had to be mentioned again in the prompt. Other than that this is easy to understand correct and handeld in one-shot.

## 8 Q3 b)

### 8.1 Q3b i) Prompt

I used a screenshot of the question due to the fact it was hard to copy. it is also fun to test its multimodal capabilities.

## 8.2 Q3b i) Kimi Response

Optimal Purifying Encoder in SVD Coordinates

**Given:**

$$L_2(W) = \|X - WX\|_F^2 + \lambda\|W\|_F^2, \quad X = U\Sigma V^\top, \quad \Sigma = \text{diag}(\sigma_1, \dots, \sigma_m).$$

**Goal:** fill the blank matrices in the supplied formula for the minimizer  $W_\star$ .

---

### 1. Closed-form minimizer (standard ridge regression)

For every column of  $X$  (and hence for the whole matrix) the problem separates into  $m$  independent ridge problems. The well-known solution is

$$W_\star = XX^\top (XX^\top + \lambda I_m)^{-1}.$$

Insert the SVD:

$$XX^\top = U\Sigma^2 U^\top, \quad (XX^\top + \lambda I)^{-1} = U(\Sigma^2 + \lambda I)^{-1} U^\top.$$

Therefore

$$W_\star = U\Sigma^2 (\Sigma^2 + \lambda I)^{-1} U^\top = U \text{ diag}\left(\frac{\sigma_1^2}{\sigma_1^2 + \lambda}, \dots, \frac{\sigma_m^2}{\sigma_m^2 + \lambda}\right) U^\top.$$


---

### 2. Fill-in-the-blanks answer

The supplied template (5) is exactly the diagonal shrinkage expression above. Hence the empty matrix is the diagonal shrinkage operator:

$$W_\star = U \text{ diag}\left(\frac{\sigma_1^2}{\sigma_1^2 + \lambda}, \frac{\sigma_2^2}{\sigma_2^2 + \lambda}, \dots, \frac{\sigma_m^2}{\sigma_m^2 + \lambda}\right) U^\top$$

(If the template literally contains an “ $\sigma_{m+1}$ ” placeholder, simply set that entry to zero, because  $X$  has only  $m$  non-zero singular values when  $n \geq m$ .)

## 8.3 Q3b i) Interpretation

Kimi K2 does not support direct image inputs!! it is not mulitmodal but I assume it uses some image to text tool?? But still it got it right!!

## 8.4 Q3b ii) Prompt

ii. Derive the above expression. (Hint: Can you understand  $L_2(W ; X, )$  as a sum of  $m$  completely decoupled ridge-regression problems?) (Hint: The Frobenius norm is equal to  $A_2 F := \text{tr}(A A^\top)$ , and it is invariant under orthogonal transform. That is,  $A_2 F = U A V^\top 2 F$  for any orthogonal matrices  $U, V$ , and any rectangular matrix  $A$ , as long as  $U, A, V$  have compatible shapes.)

## 8.5 Q3b ii) Kimi Response

Derivation of the Optimal Encoder  $W_*$

### Step 1: Decouple the matrix problem into $m$ vector problems

Write the objective in terms of columns. Let

$$X = [x_1 \mid \dots \mid x_n] \in \mathbb{R}^{m \times n}, \quad (WX)_{\cdot j} = Wx_j.$$

Then

$$\begin{aligned} L_2(W) &= \sum_{j=1}^n \|x_j - Wx_j\|^2 + \lambda\|W\|_F^2 \\ &= \sum_{j=1}^n \left[ \|x_j - Wx_j\|^2 + \frac{\lambda}{n}\|W\|_F^2 \right] \quad (\text{same regulariser for every column}). \end{aligned}$$

Each summand depends only on the *same* matrix  $W$ , hence the minimiser of the sum is the minimiser of the identical summand. Equivalently, solve the single *ridge regression* problem

$$\min_W \|X - WX\|_F^2 + \lambda\|W\|_F^2 \iff \min_W \sum_{j=1}^n \|x_j - Wx_j\|^2 + \lambda\|W\|_F^2.$$

### Step 2: Vectorise and use the normal equations

Vectorise column-wise:

$$\text{vec}(X - WX) = (I_n \otimes I_m) \text{vec}(X) - (I_n \otimes W) \text{vec}(X).$$

The objective becomes

$$\|\text{vec}(X) - (I_n \otimes W) \text{vec}(X)\|^2 + \lambda\|\text{vec}(W)\|^2,$$

which is exactly  $n$  identical ridge problems stacked. The normal equation for each column is therefore

$$(WX)X^\top + \lambda W = XX^\top \implies W(XX^\top + \lambda I_m) = XX^\top.$$

Hence

$$W_* = XX^\top (XX^\top + \lambda I_m)^{-1}.$$

### Step 3: Insert the SVD and orthogonal invariance

Write  $X = U\Sigma V^\top$  with  $\Sigma \in \mathbb{R}^{m \times m}$  diagonal and  $U, V$  orthogonal. Then

$$XX^\top = U\Sigma^2 U^\top, \quad XX^\top + \lambda I_m = U(\Sigma^2 + \lambda I_m)U^\top.$$

Therefore

$$\begin{aligned} W_* &= U\Sigma^2 U^\top U(\Sigma^2 + \lambda I_m)^{-1} U^\top \\ &= U\Sigma^2 (\Sigma^2 + \lambda I_m)^{-1} U^\top \\ &= U \text{diag}\left(\frac{\sigma_1^2}{\sigma_1^2 + \lambda}, \dots, \frac{\sigma_m^2}{\sigma_m^2 + \lambda}\right) U^\top. \end{aligned}$$

---

### Final expression

$$W_{\star} = U \operatorname{diag}\left(\frac{\sigma_1^2}{\sigma_1^2 + \lambda}, \frac{\sigma_2^2}{\sigma_2^2 + \lambda}, \dots, \frac{\sigma_m^2}{\sigma_m^2 + \lambda}\right) U^{\top}$$

This is a *diagonal shrinkage* in the left-singular space of  $X$ : large singular values (signal directions) are preserved, small ones are down-weighted, giving the desired purification effect.

## 8.6 Q3b ii) Interpretation

Kimi is really starting to impress me. Its really good. Its not even a thinking model and has so little inaccuracy.

## 9 Q3 c)

### 9.1 Q3c) Prompt

(c) You are given that the data matrix  $X \in \mathbb{R}^{8 \times n}$  has the following singular values:  $\sigma_1 = 10, \sigma_2 = 8, \sigma_3 = 4, \sigma_4 = 1, \sigma_5 = 0.5, \sigma_6 = 0.36, \sigma_7 = 0.16, \sigma_8 = 0.01$ . For what set of hyperparameter values can we guarantee that the learned purifier  $cW$  will preserve at least 80% of the strength of the first two singular values while attenuating components in the remaining directions to at most 50% of their original strength? (Hint: What are the two critical singular values to focus on?)

### 9.2 Q3c) Kimi Response

Preservation / Attenuation Constraints on  $\lambda$

**Data.** Eight singular values (descending)

$$\sigma_1 = 10, \sigma_2 = 8, \sigma_3 = 4, \sigma_4 = 1, \sigma_5 = 0.5, \sigma_6 = 0.36, \sigma_7 = 0.16, \sigma_8 = 0.01.$$

**Optimal encoder** (from part b)

$$W_{\star} = U \operatorname{diag}(s_1, \dots, s_8) U^{\top}, \quad s_i = \frac{\sigma_i^2}{\sigma_i^2 + \lambda}.$$

Hence every component along the  $i$ -th singular vector is multiplied by the *shrinkage factor*  $s_i$ .

---

### Requirements

- i. *Preserve* the first 3 directions to at least 80% of their original strength:

$$s_1, s_2, s_3 \geq 0.8.$$

ii. *Attenuate* the remaining 5 directions to at most 50% of their original strength:

$$s_4, s_5, s_6, s_7, s_8 \leq 0.5.$$


---

### 1. Preservation inequality (tightest at smallest preserved singular value)

The function  $s(\sigma) = \sigma^2 / (\sigma^2 + \lambda)$  is increasing in  $\sigma$ , so the *most restrictive* of the three preservation constraints is the smallest singular value among the first three:

$$\sigma_3 = 4 \implies \frac{4^2}{4^2 + \lambda} \geq 0.8 \implies \frac{16}{16 + \lambda} \geq 0.8 \implies 16 \geq 0.8(16 + \lambda) \implies \lambda \leq 4.$$

### 2. Attenuation inequality (tightest at largest attenuated singular value)

Similarly, the *most restrictive* attenuation constraint is the largest singular value among the last five:

$$\sigma_4 = 1 \implies \frac{1^2}{1^2 + \lambda} \leq 0.5 \implies \frac{1}{1 + \lambda} \leq 0.5 \implies 1 \leq 0.5(1 + \lambda) \implies \lambda \geq 1.$$


---

#### Feasible set of hyper-parameters

Combining the two tightest conditions gives

$$1 \leq \lambda \leq 4.$$

For every  $\lambda$  in this interval the learned purifier preserves at least 80% of the strength along the first three singular vectors while attenuating all remaining directions to at most 50% of their original magnitude.

### 9.3 Q3c) Interpretation

good

## 10 Q4 a)

### 10.1 Q4a) Prompt

continue with latex: n lecture, you saw how the standard softmax-attention mechanism can be viewed as a softened version of something akin to a nearest-neighbor model in which the value returned for a query reflects a weighted combination of the values that correspond to the keys closest to the query. In this view, the (key, value) pairs in the memory represent a kind of in-context “training data” and the query is a test input for which we want to predict the right output given that data. (a) To start, let’s think about why it is possible to efficiently update simple averaging. Let  $m = \frac{1}{n} \sum_{i=1}^n x_i$  be the average of  $n$  points. Use  $m$ ,  $x_{n+1}$ ,  $n$  and simple arithmetic operations to compute  $m' = \frac{1}{n+1} \sum_{i=1}^{n+1} x_i$  — the average of all points including the new point  $x_{n+1}$ . (HINT: Start by multiplying  $m$  by  $n$ .)

## 10.2 Q4a) Kimi Response

Efficient Update of a Simple Average

**Given:**

$$m = \frac{1}{n} \sum_{i=1}^n x_i , \quad \text{new point } x_{n+1} .$$

**Goal:** compute

$$m' = \frac{1}{n+1} \sum_{i=1}^{n+1} x_i$$

using only  $m$ ,  $x_{n+1}$ ,  $n$  and elementary arithmetic.

---

### Derivation

Start from the definition of the current average:

$$m = \frac{1}{n} \sum_{i=1}^n x_i \implies \sum_{i=1}^n x_i = n m .$$

Add the new point:

$$\sum_{i=1}^{n+1} x_i = n m + x_{n+1} .$$

Divide by the new count:

$$m' = \frac{n m + x_{n+1}}{n+1} .$$


---

### Final update formula

$$m' = \frac{n m + x_{n+1}}{n+1}$$

This requires only *one multiplication, one addition, and one division*, i.e.  $O(1)$  time and memory regardless of how large  $n$  is.

## 10.3 Q4a) Interpretation

good

## 11 Q4 b)

### 11.1 Q4b) Prompt

(b) Let us now shift to thinking about traditional ridge-regression with  $n$  training pairs  $(x_i, y_i)$  where  $x_i$  are  $d$ -dimensional vectors and  $y_i$  are scalars. Let the matrix  $A = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}$  and vector  $y = \begin{bmatrix} y_1 & y_2 & \dots & y_n \end{bmatrix}$  so that we can find the familiar closed-form solution  $w = (AA^\top + I)^{-1}Ay$  (6) that allows us to make scalar predictions on any new test input  $x$  by computing  $w^\top x$ . First, write the two terms  $(AA^\top + I)$  and  $Ay$  as sums involving the  $x_i$  and  $y_i$ . i.e. Complete:  $(AA^\top + I) = I + \sum_{i=1}^n A x_i x_i^\top$

### 11.2 Q4b) Kimi Response

Ridge Regression – Sum Forms

**Given training pairs**  $(x_i, y_i)$ ,  $x_i \in \mathbb{R}^d$ ,  $y_i \in \mathbb{R}$ ,  $i = 1, \dots, n$ .

**Data matrix and target vector**

$$A = \begin{bmatrix} x_1^\top \\ x_2^\top \\ \vdots \\ x_n^\top \end{bmatrix} \in \mathbb{R}^{n \times d}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \in \mathbb{R}^n.$$

**Closed-form solution**

$$w^* = (A^\top A + \lambda I)^{-1} A^\top y.$$


---

#### 1. Outer-product form of $A^\top A$

$$A^\top A = \sum_{i=1}^n x_i x_i^\top.$$

Hence

$$A^\top A + \lambda I = \lambda I + \sum_{i=1}^n x_i x_i^\top.$$

#### 2. Weighted sum form of $A^\top y$

$$A^\top y = \sum_{i=1}^n y_i x_i.$$

Therefore

$$A^\top y = \sum_{i=1}^n y_i x_i.$$


---

These two identities expose the *rank-one* structure of the data covariance and the *linear* dependence on the labels, which are key for both efficient batch solvers and streaming (online) updates.

### 11.3 Q4b) Interpretation

good only i would put  $x_i^*y_i$  not other way around.

## 12 Q4 c)

### 12.1 Q4c) Prompt

Suppose we wanted to do ridge-self-attention (non-causal – “encoder-style”) with a context length of  $n$  and  $d$ -dimensional query, key, and value vectors. Recall that this is the style of attention where each of the  $n$  queries is applied to the same pool of  $n$  (key, value) pairs. The goal is to calculate multi-dimensional ridge-regression predictions, after “training” on the pool of (key, value) pairs, and given the query as a kind of “test” input. (Note: the keys are playing the role of the  $A$  matrix in ridge-regression, each query is like the  $x$  we are testing on, and in place of the scalar  $y_i$ , we have an entire value vector for a multi-dimensional ridge-regression problem so the  $y$  vector is replaced with a matrix that has a row for each value vector.) Assume that the cost of inverting a  $d \times d$  matrix is  $O(d^3)$  and the cost of multiplying two such matrices is also  $O(d^3)$ . Assume that a  $d \times d$  matrix times either a  $d$ -dimensional row or column vector costs  $d^2$  operations. You should assume  $d \leq n$ . What is the computational cost of a non-causal ridge self-attention layer?  $O(d^4)$   $O(nd^2)$   $O(n^2d^3)$   $O(n^2d^2)$   $O(n^2)$   $O(1)$  (HINT: Do not forget that for a single  $d$ -dimensional query vector  $q$ , attention needs to return a  $d$ -dimensional result.)

### 12.2 Q4c) Kimi Response

Cost of Non-Causal Ridge Self-Attention

#### Set-up

- Context length:  $n$  (same set of keys and values)
- Dimension:  $d$  (for queries, keys, values); assume  $d < n$
- Keys matrix:  $K \in \mathbb{R}^{n \times d}$  (each row is a key vector)
- Values matrix:  $V \in \mathbb{R}^{n \times d}$  (each row is a value vector)

- Query matrix:  $Q \in \mathbb{R}^{n \times d}$  (each row is a query vector)

For *one* query vector  $q \in \mathbb{R}^{1 \times d}$  the ridge-regression solution (ridge attention) is

$$\text{Att}(q) = q \underbrace{(K^\top K + \lambda I_d)^{-1} K^\top V}_{\text{shared } d \times d \text{ coefficient matrix}} \in \mathbb{R}^{1 \times d}.$$

The expensive part is computing the *shared*  $d \times d$  matrix

$$M = (K^\top K + \lambda I)^{-1} K^\top V$$

once and then multiplying every query row by  $M$ .

---

### Step 1: Build the shared matrix

- $K^\top K$ :  $d \times n$  times  $n \times d \rightarrow \Theta(nd^2)$  operations
- Add  $\lambda I$ :  $\Theta(d^2)$
- Invert  $d \times d$  matrix:  $\Theta(d^3)$
- $K^\top V$ :  $d \times n$  times  $n \times d \rightarrow \Theta(nd^2)$
- Multiply  $(K^\top K + \lambda I)^{-1}$  with  $(K^\top V)$ :  $d \times d$  times  $d \times d \rightarrow \Theta(d^3)$

Total for building  $M$ :  $\boxed{\Theta(nd^2 + d^3)}$ .

### Step 2: Apply to all $n$ queries

Each  $q_i M$  costs  $d^2$ ; for  $n$  queries:  $\Theta(nd^2)$ .

---

### Overall complexity

$$\boxed{\Theta(nd^2 + d^3)}.$$

Under the given assumption  $d < n$  the  $nd^2$  term dominates, so the layer costs

$$\boxed{\Theta(nd^2)}.$$


---

### Choice from the list

- $\Theta(d^4)$      $\Theta(nd^2)$      $\Theta(n^2d^3)$      $\Theta(n^2d^2)$      $\Theta(n^2)$      $\Theta(1)$

$$\boxed{\text{Correct tick: } \Theta(nd^2)}$$

### 12.3 Q4c) Interpretation

good

## 13 Q4 d)

### 13.1 Q4d) Prompt

(d) Assume that a ridge self-attention layer is used in a Transformer architecture and there is a downstream loss. For which of these will backprop successfully pass gradients if we use ridge self-attention? 2 The ridge viewed as a learnable parameter for the self-attention layer. 2 The keys 2 The values 2 The queries

### 13.2 Q4d) Kimi Response

Gradient Flow through Ridge Self-Attention

#### Set-up

- Forward pass (non-causal ridge attention)

$$A = (K^\top K + \lambda I)^{-1} K^\top V \in \mathbb{R}^{d \times d}, \quad \text{Out} = Q A \in \mathbb{R}^{n \times d}.$$

- Down-stream loss  $\mathcal{L}$  is scalar; back-prop computes

$$\frac{\partial \mathcal{L}}{\partial (\cdot)} \quad \text{for every tensor that is (i) differentiable and (ii) participates in the graph.}$$

---

#### Check-list for differentiability

- green!60!black ✓ **Keys**  $K$   
Appear explicitly in  $K^\top K$  and  $K^\top V$ ; matrix inverse is differentiable w.r.t. its operands. Gradients flow.
- green!60!black ✓ **Values**  $V$   
Appear linearly in  $K^\top V$ ; no non-differentiable operation. Gradients flow.
- green!60!black ✓ **Queries**  $Q$   
Appear only in the final multiplication  $Q A$ ; standard matrix product. Gradients flow.
- green!60!black ✓ **Ridge parameter**  $\lambda$   
Appears inside the inverse as  $(K^\top K + \lambda I)^{-1}$ ; the inverse is differentiable w.r.t. the scalar shift. Gradient flows.

---

## Conclusion

All four quantities are embedded in differentiable operations (matrix products and the matrix inverse, which is analytic in its argument). Therefore back-propagation will *successfully* compute gradients for

Keys, Values, Queries, and the ridge $\lambda$ .
--

No gradient blocking occurs; any of them can be treated as learnable parameters or inputs for further optimization.

### 13.3 Q4d) Interpretation

good

## 14 Q4 e)

### 14.1 Q4e) Prompt

(e) Now step back. There is a nice trick (called the Sherman–Morrison Formula) by which one can update the inverse of an invertible matrix to which you make a rank-1 update. Let  $M$  be an invertible square  $d \times d$  matrix and let  $u, v$  be two  $d$ -dimensional vectors. Then:  $(M + uv^\top)^{-1} = M^{-1} - \frac{1}{1 + v^\top M^{-1} u} (M^{-1} u)(v^\top M^{-1})$  (7) Assume that a  $d \times d$  matrix times either a  $d$ -dimensional row or column vector costs  $d^2$  operations, and so does the evaluation of a dyad  $uv$ . Assume that computing a Euclidean inner-product costs  $d$  operations. Assuming that you already had  $M^{-1}$  in hand, what is the computational cost of one application of (7)? O(d4) O(d3) O(d2) O(d) O(1)

### 14.2 Q4e) Kimi Response

Cost of one Sherman–Morrison update
-------------------------------------

Given:

$$(M + uv^\top)^{-1} = M^{-1} - \frac{1}{1 + v^\top M^{-1} u} (M^{-1} u)(v^\top M^{-1}), \quad M^{-1} \text{ already known.}$$

Operations to evaluate the right-hand side once:

- (a)  $M^{-1}u$ :  $d \times d$  matrix  $\times$   $d$ -column vector  $\rightarrow \Theta(d^2)$
- (b)  $v^\top M^{-1}$ :  $d$ -row vector  $\times$   $d \times d$  matrix  $\rightarrow \Theta(d^2)$
- (c)  $v^\top(M^{-1}u)$ : inner product of two  $d$ -vectors  $\rightarrow \Theta(d)$

- 
- (d) Dyad  $(M^{-1}u)(v^\top M^{-1})$ : outer product of two  $d$ -vectors  $\rightarrow \Theta(d^2)$
- (e) Scale the dyad by the scalar  $\frac{1}{1+v^\top M^{-1}u}$  and subtract from  $M^{-1}$ :  $\Theta(d^2)$
- 

**Total cost**

$\Theta(d^2)$ .

**Choice from the list**

- $\Theta(d^4)$      $\Theta(d^3)$      $\Theta(d^2)$      $\Theta(d)$      $\Theta(1)$

Correct tick:  $\Theta(d^2)$

### 14.3 Q4e) Interpretation

good.

## 15 Q4 f)

### 15.1 Q4f) Prompt

f) Consider implementing causal ridge-self-attention with a context length of  $n$  but where the pool of (key, value) vectors that one is querying at position  $t$  consists only of the  $t$  (key, value) pairs so far. Describe explicitly how you would compute causal ridge-self-attention in a computationally efficient manner. Leverage your decomposition of the ridge-regression formula in part (b) of this problem together with the Sherman-Morrison formula from (7) to avoid having to do  $O(n^2)$  computations while still calculating causal ridge-self-attention outputs correctly for all  $n$  positions in the context. (HINT: Think recursively. What do you need to track from one time step to the next to avoid repeating work? )

### 15.2 Q4f) Kimi Response

Efficient Causal Ridge-Self-Attention via Recursive Rank-1 Updates

**Set-up**

- Causal (autoregressive) setting: at position  $t$  only keys

$$K_t = [k_1^\top; \dots; k_t^\top] \in \mathbb{R}^{t \times d}$$

and values

$$V_t = [v_1^\top; \dots; v_t^\top] \in \mathbb{R}^{t \times d}$$

are available.

- Ridge solution for query  $q_t \in \mathbb{R}^{1 \times d}$ :

$$\text{Att}_t = q_t \underbrace{(K_t^\top K_t + \lambda I)^{-1} K_t^\top V_t}_{\text{must be computed online}} \in \mathbb{R}^{1 \times d}.$$

- Goal: compute  $\text{Att}_t$  for  $t = 1, \dots, n$  with total cost  $\boxed{O(nd^2)}$  instead of  $O(n^2d^2)$ .
- 

### Step 1: maintain sufficient statistics recursively

From part (b) we know

$$K_t^\top K_t = \sum_{i=1}^t k_i k_i^\top, \quad K_t^\top V_t = \sum_{i=1}^t k_i v_i^\top.$$

Define running statistics

$$S_t := K_t^\top K_t + \lambda I \in \mathbb{R}^{d \times d}, \quad Z_t := K_t^\top V_t \in \mathbb{R}^{d \times d}.$$

Rank-1 updates:

$$\boxed{\begin{aligned} S_t &= S_{t-1} + k_t k_t^\top & (S_0 = \lambda I), \\ Z_t &= Z_{t-1} + k_t v_t^\top & (Z_0 = 0). \end{aligned}}$$

### Step 2: recursive inverse via Sherman–Morrison

Let

$$P_t := S_t^{-1} = (S_{t-1} + k_t k_t^\top)^{-1}.$$

Sherman–Morrison formula gives

$$\boxed{P_t = P_{t-1} - \frac{1}{1 + k_t^\top P_{t-1} k_t} (P_{t-1} k_t)(k_t^\top P_{t-1})} \quad \text{with } P_0 = \lambda^{-1} I.$$

Cost per update:  $O(d^2)$  (see part e).

### Step 3: produce the attention output

Once  $P_t$  and  $Z_t$  are available, the coefficient matrix is simply

$$A_t = P_t Z_t.$$

Multiply by query:

$$\text{Att}_t = q_t A_t = q_t P_t Z_t.$$

Matrix–vector products only:  $O(d^2)$ .

---

### Complete online algorithm (per token $t$ )

(a) Receive  $(k_t, v_t, q_t)$ .

(b) Update statistics:

$$\begin{aligned} \rho_t &= 1 + k_t^\top P_{t-1} k_t && \text{(scalar, } O(d^2)\text{)} \\ P_t &= P_{t-1} - \rho_t^{-1} (P_{t-1} k_t) (k_t^\top P_{t-1}) && \text{(SM update, } O(d^2)\text{)} \\ Z_t &= Z_{t-1} + k_t v_t^\top && \text{(rank-1 add, } O(d^2)\text{)} \end{aligned}$$

(c) Compute output:

$$\text{Att}_t = q_t (P_t Z_t) \quad (O(d^2)).$$


---

### Complexity

Each token costs  $O(d^2)$ ; over  $n$  tokens:

$O(nd^2)$ total time	$O(d^2)$ memory
----------------------	-----------------

(storing  $P_t$  and  $Z_t$  only). No  $O(n^2)$  operations are ever performed, yet the causal ridge-attention outputs are computed *exactly* at every position.

### 15.3 Q4f) Interpretation

## 16 Q4 g)

### 16.1 Q4g) Prompt

final one. remember latex. (g) Many people consider important the ability to visualize the attention weights. For traditional softmax- attention, these are the outputs of the softmax for a given query vector. They tell you the exact amounts by which the attention outputs at this position are linear combinations of the values being fed in at this and other positions. For ridge-attention and a given query vector, how would you compute the (possibly negative) weights associated to each of the value vectors in the context?

## 16.2 Q4g) Kimi Response

Visualising (Signed) Attention Weights in Ridge-Attention

### Set-up

- Context keys  $K \in \mathbb{R}^{t \times d}$
- Context values  $V \in \mathbb{R}^{t \times d}$
- Query  $q \in \mathbb{R}^{1 \times d}$
- Ridge solution

$$\text{Att}(q) = q(K^\top K + \lambda I)^{-1} K^\top V \in \mathbb{R}^{1 \times d}.$$


---

### Attention weights as coefficients on *values*

Write the output explicitly as a *linear combination of the rows of  $V$* :

$$\text{Att}(q) = \sum_{i=1}^t \alpha_i(q) v_i^\top, \quad \text{where } \alpha(q) := q(K^\top K + \lambda I)^{-1} K^\top \in \mathbb{R}^{1 \times t}.$$

Hence the vector

$$\alpha(q) = q(K^\top K + \lambda I)^{-1} K^\top$$

contains the (possibly *negative*) attention weights associated with each value vector  $v_i$ .

---

### Efficient computation

- (a) Pre-compute (or reuse from the forward pass)

$$P = (K^\top K + \lambda I)^{-1} \quad (d \times d).$$

- (b) One matrix–vector product

$$\alpha(q) = q P K^\top \quad (1 \times d \text{ times } d \times d \text{ times } d \times t \rightarrow 1 \times t).$$

Cost:  $O(t d^2)$  for the first query,  $O(t d)$  for each additional query once  $P$  is cached.

---

### Interpretation

Each entry

$$\alpha_i(q) = [q(K^\top K + \lambda I)^{-1} k_i^\top]$$

tells us the *signed* contribution weight of the  $i$ -th value vector  $v_i$  to the final ridge-attention output for query  $q$ . Positive weights *enhance*, negative weights *suppress* the corresponding value direction—offering a richer, signed analogue of the traditional softmax attention map.

## 16.3 Q4g) Interpretation

good.