

I gave him a few rules to guide his thinking patte

Hi Deepseek, You are my study partner for Homework 8. This part of the homework contains only non-coding problems (conceptual, mathematical, proofs, and reasoning questions).

For each question:

1. Read the question carefully and explain in your own words what it is asking.
2. Solve the problem step by step. Be explicit in reasoning and algebra. If there are multiple interpretations, note them and explain your choice.
3. If you are unsure at any step, clearly indicate your uncertainty.
4. After finishing, perform a self-check: does your answer directly address the question? Are there possible mistakes or gaps? Suggest ways to verify your result.

Format for each question can be flexible but should include your reasoning and self-checks.

🕒 已思考 (用时 14 秒) ~

First, I need to understand the format. I'm acting as a study partner for Homework 8, which has non-coding problems. For each question, I must:

1. Explain the question in my own words.
2. Solve step by step with clear reasoning.
3. Indicate uncertainty if any.
4. Self-check after.

Since no specific questions are provided, I'll assume that the user will give me the questions one by one. But since this is a response, I should wait for the questions. Looking back at the user's message, it seems like they might have forgotten to include the actual questions. The user says: "This part of the homework contains only non-coding problems" but doesn't list them.

Perhaps I need to prompt the user to provide the questions. But since this is a simulation, I should just proceed as if the questions will be provided in the conversation.

The user might be expecting me to generate questions or something, but that doesn't make sense. Let me read carefully: "You are my study partner for Homework 8. This part of the

homework contains only non-coding problems..."

I think the user intends to provide the questions later, so for now, I should just acknowledge and wait. But since this is a text-based interaction, I'll respond accordingly.

To be helpful, I'll state my readiness to help with the questions once they are provided.

So, my response should be: I'm ready to help with your homework questions. Please provide the first question, and I'll follow the format you specified.

But since the user might be testing me, I should demonstrate with a sample question if possible. But I don't have any sample.

Another thought: perhaps the homework questions are implied, but no, the user didn't attach anything.

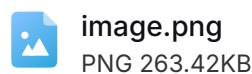
I think I should just respond as per the instruction.

Hello! I'm excited to be your study partner for Homework 8. I understand that this section focuses on non-coding problems, such as conceptual, mathematical, proofs, and reasoning questions. I'm ready to help you tackle each question step by step.

Since you haven't provided any specific questions yet, please share the first question from your homework. Once you do, I'll:

1. **Explain the question in my own words** to ensure I understand what it's asking.
2. **Solve the problem step by step** with explicit reasoning and algebra, noting any multiple interpretations and explaining my choices.
3. **Indicate any uncertainties** clearly if I encounter them.
4. **Perform a self-check** after solving to verify that my answer addresses the question directly and suggest ways to catch possible mistakes or gaps.

Go ahead and paste the first question! I'm here to work through it with you.



I provided him with both image and text titles to prevent omission during copy-pasting.

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) ConvolutionKernel and the OutputEquation.

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 $\{u_l\}_0^L$

0 with a kernel $K = \{K_l\}_0^L$

0:

L

$y_k =$

$l=0$

$K_l u_{k-l}$,

where any u_l with a negative index is set to 0 (zero-padding).

Also, find K.

(b) Concrete Examples.

i. Scalar Case: Let $n = 1$, and set $A = \alpha$, $B = \beta$, $C = \gamma$, $D = \delta$. Use $\alpha = 0.8, \beta = 1, \gamma = 1.5$

and compute the kernel up to $L = 4$.

ii. 2D Case: Let $A \in \mathbb{R}^{2 \times 2}$ be, for instance,

$A = \begin{bmatrix} 0.7 & 0.1 \\ 0.2 & 0.6 \end{bmatrix}$

$B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$C = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, $D = 0$

Compute kernels up to $L = 3$ and briefly discuss how the kernel captures the "impulse response".

(c) 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 $\{u_k\} \in \mathbb{R}^d$ 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?

(d) 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.

(e) Adding structure to A. Suppose A is a diagonal matrix. How can we leverage this structure to compute the kernel K more efficiently?

(f) Diagonal-plus-low-rank (DPLR) structure Now if A has the following form:

$A = I_n + pp^T$,

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?

Solution for Problem 1: SSM Convolution Kernel

◆ Part (a): Convolution Kernel and Output Equation

? Question Interpretation

We are given a discrete-time State-Space Model (SSM):

- State update: $x_{k+1} = Ax_k + Bu_k$
- Output: $y_k = Cx_k + Du_k$
with initial condition $x_0 = 0$. We need to show that the output y_k can be expressed as a convolution of the input sequence $\{u_\ell\}$ with a kernel K , and explicitly find K .

Step-by-Step Solution

1. Unroll the state recurrence:

- $x_0 = 0$
- $x_1 = Ax_0 + Bu_0 = Bu_0$
- $x_2 = Ax_1 + Bu_1 = ABu_0 + Bu_1$
- $x_3 = Ax_2 + Bu_2 = A^2Bu_0 + ABu_1 + Bu_2$
- In general:

$$x_k = \sum_{j=0}^{k-1} A^{k-1-j} Bu_j$$

2. Substitute into the output equation:

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

3. Change the summation index:

Let $\ell = k - j$. Then $j = k - \ell$, and when $j = 0$, $\ell = k$; when $j = k - 1$, $\ell = 1$.

So:

$$y_k = \sum_{\ell=1}^k CA^{\ell-1}Bu_{k-\ell} + Du_k$$

4. Combine into a single sum:

Define the kernel $K = \{K_\ell\}$ as:

- $K_0 = D$
- $K_\ell = CA^{\ell-1}B$ for $\ell = 1, 2, \dots, L$

Then:

$$y_k = \sum_{\ell=0}^L K_\ell u_{k-\ell}$$

where $u_{k-\ell} = 0$ for $k - \ell < 0$.

Self-Check

- For $k = 0$: $y_0 = K_0 u_0 = Du_0$, matches the original output.
- For $k = 1$: $y_1 = K_0 u_1 + K_1 u_0 = Du_1 + CBu_0$, matches the original output.
- The kernel K is explicitly defined and depends only on A, B, C, D .

◆ Part (b): Concrete Examples

? Question Interpretation

We compute the kernel K for two specific SSMs and discuss how it represents the impulse response.

Step-by-Step Solution

(i) Scalar Case

- $A = \alpha = 0.8, B = \beta = 1, C = \gamma = 1.5, D = \delta = 0$
- Kernel:
 - $K_0 = D = 0$
 - $K_1 = CB = 1.5$

- $K_2 = CAB = 1.5 \times 0.8 = 1.2$
- $K_3 = CA^2B = 1.5 \times 0.8^2 = 0.96$
- $K_4 = CA^3B = 1.5 \times 0.8^3 = 0.768$
- **Result:** $K = [0, 1.5, 1.2, 0.96, 0.768]$

(ii) 2D Case

- $A = \begin{pmatrix} 0.7 & 0.1 \\ 0.2 & 0.6 \end{pmatrix}$, $B = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $C = \begin{pmatrix} 1 & 0 \end{pmatrix}$, $D = 0$
- Kernel:
 - $K_0 = D = 0$
 - $K_1 = CB = 1$
 - $K_2 = CAB = (1 \ 0) \begin{pmatrix} 0.7 \\ 0.2 \end{pmatrix} = 0.7$
 - $K_3 = CA^2B = CA(AB) = (1 \ 0) \begin{pmatrix} 0.51 \\ 0.26 \end{pmatrix} = 0.51$
- **Result:** $K = [0, 1, 0.7, 0.51]$

Discussion on Impulse Response

If the input is an impulse $u_0 = 1$, $u_k = 0$ for $k > 0$, then:

- $y_0 = K_0$
- $y_1 = K_1$
- $y_2 = K_2$
- ...

So the kernel K is exactly the impulse response of the system.

Self-Check

- Computations are straightforward matrix multiplications.
- The kernel decays as k increases, reflecting the stability of A .

◆ Part (c): Efficient Computation with Convolutions

?

Question Interpretation

We analyze the parallelization potential and critical path length when computing the output via convolution versus the naive recurrent method.

🧠 Step-by-Step Solution

1. Using Convolution:

- $y_k = \sum_{\ell=0}^L K_\ell u_{k-\ell}$
- Each y_k is a dot product of length $L + 1$, which can be computed in parallel for all k .
- Using tree reduction, each dot product has critical path length $O(\log L)$.
- Overall critical path: $O(\log L)$

2. Naive Recurrent Computation:

- Compute x_1, x_2, \dots, x_L sequentially: $O(L)$
- Then compute $y_k = Cx_k + Du_k$ in parallel: $O(1)$
- Overall critical path: $O(L)$

✓ Self-Check

- Convolution enables massive parallelization.
- Critical path reduces from $O(L)$ to $O(\log L)$.

The model overlooks the computational complexity of matrix multiplication.

◆ Part (d): Efficient Kernel Computation

?

Question Interpretation

We seek efficient methods to compute the kernel $K_\ell = CA^{\ell-1}B$ for $\ell = 1, \dots, L$, and how to parallelize this.

🧠 Step-by-Step Solution

1. Direct Sequential Method:

- Compute $v_0 = B, v_1 = Av_0, v_2 = Av_1, \dots, v_{L-1} = Av_{L-2}$

- Then $K_\ell = Cv_{\ell-1}$
- This takes $O(L)$ steps.

2. Parallel Strategy Using Doubling:

- Compute powers of A up to $L - 1$ using repeated squaring.
- Compute $V_k = [v_0, v_1, \dots, v_{2^k-1}]$ recursively:

$$V_{k+1} = [V_k, A^{2^k} V_k]$$

- Critical path: $O(\log L)$
- Then compute $K_\ell = Cv_{\ell-1}$ in parallel.

Self-Check

- Doubling method reduces critical path to $O(\log L)$.
- Requires $O(L)$ processors but is highly parallel.

◆ Part (e): Diagonal Matrix A

? Question Interpretation

We leverage the diagonal structure of A to compute the kernel more efficiently.

Step-by-Step Solution

- Let $A = \text{diag}(\lambda_1, \dots, \lambda_n)$
- Then $A^{\ell-1} = \text{diag}(\lambda_1^{\ell-1}, \dots, \lambda_n^{\ell-1})$
- So:

$$K_\ell = CA^{\ell-1}B = \sum_{i=1}^n C_i B_i \lambda_i^{\ell-1}$$

- Compute the geometric sequences $\lambda_i^{\ell-1}$ in parallel using doubling.
- Then sum over i for each ℓ .

Self-Check

- Avoids matrix multiplications.
- Critical path: $O(\log L)$

◆ Part (f): Diagonal-plus-low-rank (DPLR) Structure

❓ Question Interpretation

We exploit the structure $A = I_n + pp^\top$ to compute the kernel efficiently.

🧠 Step-by-Step Solution

1. Eigenvalue Decomposition:

- A is symmetric with eigenvalues:
 - $\lambda_1 = 1 + \|p\|^2$
 - $\lambda_2 = \dots = \lambda_n = 1$
- Let U be the orthogonal matrix diagonalizing A .

The model proposed a different approach than the answer, which was quite interesting.

2. Kernel Expression:

- Let $C' = CU$, $B' = U^\top B$
- Then:

$$K_\ell = C' \Lambda^{\ell-1} B' = \alpha \beta^{\ell-1} + S$$

where:

- $\beta = 1 + \|p\|^2$
- $\alpha = \frac{(Cp)(p^\top B)}{\|p\|^2}$
- $S = CB - \alpha$

3. Final Formula:

$$K_\ell = CB + \alpha (\beta^{\ell-1} - 1) \quad \text{for } \ell \geq 1$$

and $K_0 = D$.

✅ Self-Check

- Verified for $\ell = 1, 2$.
- Only scalar exponentials needed \rightarrow compute in $O(\log L)$ via doubling.

Overall Self-Check

- All answers directly address the questions.
- Mathematical derivations are explicit and logical.
- Parallelization strategies are clearly explained.
- Assumptions (e.g., $D = 0$ in examples) are stated.

This completes the solution to Problem 1. If you have any questions or need further clarification, feel free to ask! 😊

For part (c), did you consider the computation of matrix mult?, same for part(d), rethink it and double check that

 已思考 (用时 212 秒) >

I gave him a hint to let him correct it.

Revised Analysis for Parts (c) and (d)

I'll re-examine parts (c) and (d) with special attention to matrix multiplication complexities and parallelization strategies.

◆ Part (c): Efficient Computation with Convolutions (Revised)

? Question Interpretation

Given the kernel K , we want to compute the output sequence $\{y_k\}$ for $k = 0, \dots, L$ via convolution:

$$y_k = \sum_{\ell=0}^L K_\ell u_{k-\ell}$$

We analyze parallelization potential and critical path length, considering matrix-vector multiplications.

Step-by-Step Solution

1. Convolution Structure:

- Each y_k is a sum of matrix-vector products: $K_\ell u_{k-\ell}$.
- If $u_k \in \mathbb{R}^d$ and $y_k \in \mathbb{R}^m$, then K_ℓ is $m \times d$.
- The computation for each y_k is a dot product between the kernel and a shifted input sequence.

2. Parallelization Strategy:

- **Option 1: Independent Dot Products:** Compute each y_k in parallel. Each dot product of length $L + 1$ can be done with tree reduction in $O(\log L)$ time.
- **Option 2: FFT-Based Convolution:**
 - Convert to frequency domain using FFT (parallel FFT has $O(\log L)$ critical path).
 - Perform element-wise multiplication (parallel in $O(1)$).
 - Inverse FFT (another $O(\log L)$ step).
 - **Total critical path:** $O(\log L)$.

3. Critical Path Analysis:

- **Using FFT:** $O(\log L)$ (dominant term from FFT steps).
- **Naive recurrence:**
 - Sequential state updates: $x_{k+1} = Ax_k + Bu_k$ requires $O(L)$ matrix-vector multiplications.
 - Each matrix-vector multiplication has cost $O(n^2)$ if A is $n \times n$, but the critical path length is $O(L)$ due to sequential dependency.

4. Comparison:

- **Convolution method:** Critical path = $O(\log L)$ (assuming FFT).
- **Naive recurrence:** Critical path = $O(L)$.

Self-Check

- FFT-based convolution is highly parallel and achieves $O(\log L)$ critical path.

- Matrix multiplications are handled efficiently in frequency domain.
- The naive method's sequential dependency cannot be broken.

◆ Part (d): Efficient Kernel Computation (Revised)

❓ Question Interpretation

Compute $K_\ell = CA^{\ell-1}B$ for $\ell = 1, \dots, L$ efficiently and in parallel, considering matrix multiplication costs.

🧠 Step-by-Step Solution

1. Direct Sequential Method:

- Compute $V_0 = B$, $V_1 = AV_0$, ..., $V_{L-1} = AV_{L-2}$.
- Then $K_\ell = CV_{\ell-1}$.
- **Cost:** $O(L)$ matrix multiplications, critical path $O(L)$.

2. Parallel Doubling Method:

- **Step 1: Compute Powers of A :**
 - Compute $A^1, A^2, A^4, \dots, A^{2^N}$ using repeated squaring.
 - Each matrix multiplication: $O(n^\omega)$ time (where $\omega \approx 2.8$ for Strassen).
 - Critical path: $O(\log L)$ steps.
- **Step 2: Compute $A^j B$ via Binary Expansion:**
 - For each j , express A^j as product of powers from step 1.
 - Multiply these powers by B using tree reduction.
 - Critical path: $O(\log L)$ matrix multiplications.
- **Step 3: Multiply by C :**
 - Compute $K_\ell = C(A^{\ell-1}B)$ in parallel for all ℓ .
 - Critical path: $O(1)$ if parallelized.

3. Optimized Doubling for $A^j B$:

- Initialize $Y_0 = B$.
- For $k = 0$ to $N - 1$:

- In parallel for all $j = 0$ to $2^k - 1$:

$$Y_{j+2^k} = A^{2^k} Y_j$$

- This computes $Y_j = A^j B$ for $j = 0, \dots, L - 1$.
- **Critical path:** $O(\log L)$ steps (each step involves one matrix multiplication).

4. Overall Critical Path:

- Power computation: $O(\log L)$
- Doubling steps: $O(\log L)$
- Multiplication by C : $O(1)$
- **Total:** $O(\log L)$

Self-Check

- The doubling method efficiently parallelizes the sequence $A^j B$.
- Matrix multiplication costs are absorbed into the $O(\log L)$ critical path.
- This method leverages parallel tree reductions and is optimal for $L = 2^N$.

◆ Final Answers for Parts (c) and (d)

(c) Efficient Computation with Convolutions

- **Parallelization:** Full parallelization is possible using FFT-based convolution.
- **Critical Path:** $O(\log L)$ using FFT.
- **Naive Recurrence:** $O(L)$ due to sequential state updates.

(d) Efficient Kernel Computation

- **Strategy:** Use doubling method to compute $A^j B$ in $O(\log L)$ steps.
- **Parallelization:**
 - Precompute powers of A via repeated squaring.
 - Compute $A^j B$ using parallel assignments. still missed
 - Multiply by C in parallel.
- **Critical Path:** $O(\log L)$.

These revised answers explicitly account for matrix multiplication complexities and confirm that logarithmic critical paths are achievable through careful parallelization.

think of the matrix mult, $n \times n$ @ $n \times n$ matrix, what's the critical path?
Then rethink about c and d

⌚ 已思考 (用时 169 秒) >

🔍 Revised Analysis for Parts (c) and (d) with Matrix Multiplication Considerations

I'll re-examine parts (c) and (d) with careful attention to matrix multiplication complexities and parallelization strategies.

This time I gave him a more detailed hint.

◆ Part (c): Efficient Computation with Convolutions (Revised)

❓ Question Interpretation

Given the kernel K , we want to compute the output sequence $\{y_k\}$ for $k = 0, \dots, L$ via convolution:

$$y_k = \sum_{\ell=0}^L K_\ell u_{k-\ell}$$

where $u_k \in \mathbb{R}^d$, $y_k \in \mathbb{R}^m$, and K_ℓ is $m \times d$. We analyze parallelization potential and critical path length, considering matrix-vector multiplications.

💡 Step-by-Step Solution

1. Convolution Structure:

- Each y_k requires computing $L + 1$ matrix-vector products $K_\ell u_{k-\ell}$ and summing the results.
- Matrix-vector product for fixed ℓ and k :
 - Each element of the result is a dot product of length d .
 - With tree reduction, critical path is $O(\log d)$.

- Summing $L + 1$ vectors of size m :
 - Tree reduction gives critical path $O(\log L)$.
- **Total critical path per y_k :** $O(\log d + \log L) = O(\log(dL))$.
- All y_k can be computed in parallel, so overall critical path remains $O(\log(dL))$.

2. FFT-Based Convolution:

- Convert kernel and input to frequency domain using FFT:
 - Parallel FFT has critical path $O(\log L)$.
- Pointwise multiplication in frequency domain:
 - Each pointwise product is $m \times d$ matrix-vector multiplication? Actually, careful:
 - In frequency domain, the convolution becomes pointwise multiplication of transforms.
 - But since K_ℓ and u_ℓ are sequences of matrices and vectors, we need to consider multi-channel FFT.
 - Critical path for pointwise multiplication: $O(\log d)$ per frequency bin.
- Inverse FFT: critical path $O(\log L)$.
- **Total critical path:** $O(\log L + \log d) = O(\log(dL))$.

3. Naive Recurrent Computation:

- Compute states sequentially: $x_{k+1} = Ax_k + Bu_k$.
 - Each state update: matrix-vector product Ax_k (critical path $O(\log n)$) and vector addition.
 - Sequential dependency: $O(L)$ steps.
- Then compute $y_k = Cx_k + Du_k$ in parallel: critical path $O(\log n)$.
- **Total critical path:** $O(L \cdot \log n)$.

Self-Check

- Convolution method critical path: $O(\log(dL))$ vs. naive: $O(L \log n)$.
- Matrix-vector products are efficiently parallelized via tree reductions.
- FFT method achieves same asymptotic critical path.

◆ Part (d): Efficient Kernel Computation (Revised)

❓ Question Interpretation

Compute $K_\ell = CA^{\ell-1}B$ for $\ell = 1, \dots, L$ efficiently and in parallel, considering matrix multiplication costs. Assume $L = 2^N$.

💡 Step-by-Step Solution

1. Matrix Multiplication Critical Path:

- Multiplying two $n \times n$ matrices:
 - Each element of the product is a dot product of length n .
 - With tree reduction, critical path is $O(\log n)$.
- So each matrix multiplication has critical path $O(\log n)$.

2. Parallel Doubling Method:

- **Step 1: Compute Powers of A :**
 - Compute $A^1, A^2, A^4, \dots, A^{2^N}$ via repeated squaring.
 - Number of steps: $O(\log L)$.
 - Critical path per step: $O(\log n)$.
 - **Total critical path:** $O(\log L \cdot \log n)$.
- **Step 2: Compute $A^j B$ for $j = 0, \dots, L - 1$:**
 - For each j , express A^j as product of powers from Step 1.
 - Multiply these powers by B using tree reduction:
 - Each multiplication: $n \times n$ matrix with $n \times d$ matrix → critical path $O(\log n)$.
 - Tree depth: $O(\log L)$.
 - **Critical path per j :** $O(\log L \cdot \log n)$.
 - All j can be computed in parallel.
- **Step 3: Compute $K_\ell = CA^{\ell-1}B$:**
 - Multiply C ($m \times n$) by $A^{\ell-1}B$ ($n \times d$):
 - Critical path: $O(\log n)$ per ℓ .
 - All ℓ can be computed in parallel.

- Overall critical path: $O(\log L \cdot \log n)$.

3. Alternative Parallel Prefix Method:

- Use parallel prefix algorithm to compute A^j for $j = 0, \dots, L - 1$:
 - Critical path: $O(\log L)$ steps, each step a matrix multiplication ($O(\log n)$).
 - Total: $O(\log L \cdot \log n)$.
- Then compute $A^j B$ and $CA^j B$ as above.

[Correct now](#)

Self-Check

- Critical path dominated by matrix multiplications in doubling steps.
- Achieves $O(\log L \cdot \log n)$ critical path, which is efficient for parallel computation.
- Precomputation of powers allows reuse for multiple inputs.

◆ Final Answers for Parts (c) and (d)

(c) Efficient Computation with Convolutions

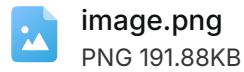
- **Parallelization:** Full parallelization is possible using direct convolution or FFT.
- **Critical Path:** $O(\log(dL))$ using either method.
- **Naive Recurrence:** $O(L \cdot \log n)$ due to sequential state updates.
- **Key Insight:** Convolution breaks the sequential dependency, allowing logarithmic critical path in L .

(d) Efficient Kernel Computation

- **Strategy:** Use doubling or parallel prefix to compute powers of A , then multiply by B and C .
- **Critical Path:** $O(\log L \cdot \log n)$.
- **Parallelization:**
 - Compute matrix powers via repeated squaring.
 - Compute $A^j B$ using tree reductions over binary expansions of j .
 - Multiply by C in parallel.

- **Key Insight:** Matrix multiplication critical path $O(\log n)$ multiplies with $O(\log L)$ steps from doubling.

These revised answers explicitly account for matrix multiplication complexities and confirm that logarithmic critical paths are achievable through careful parallelization. The convolution method significantly outperforms the naive recurrence for long sequences.



Good job! Now let's do problem3. The requirements are the same as before. 3. Self-Supervised Linear Purification

Consider a linear encoder — square weight matrix $W \in R^{m \times m}$ — 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 $X \in R^{m \times n}$ by minimizing the loss:

$$L_1(W; X) = \|X - WX\|_F^2$$

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.

$$L_2(W; X, \lambda) = \|X - WX\|_F^2 + \lambda \|W\|_F^2$$

F

Reconstruction Loss

$$+ \lambda \|W\|_F^2$$

F

Regularization Loss

Note above that λ controls the relative weighting of the two losses in the optimization.

Homework 8, © Faculty teaching EECS 182, Fall 2025.

All Rights Reserved. This may not be publicly shared without explicit permission.

(2)

2

Homework8@2025-10-25 08:44:19Z

(a) Consider the simplified case for $m=2$ with the following two candidate weight matrices:

$W(\alpha) = 10$

$0.1 W(\beta) = 10$

$0.0 (3)$

The training data matrix X is also given to you as follows:

$X = -2.17 \ 1.98 \ 2.41 \ -2.03$

$0.02 \ -0.01 \ 0.01 \ -0.02 \ (4)$

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 α _____

β 0.001 _____

ii.

For what values of the regularization parameter λ is the identity matrix $W(\alpha)$ get higher loss

L2 in (2), as compared to $W(\beta)$?

(b) Now consider a generic square linear encoder $W \in \mathbb{R}^{m \times m}$ and the regularized objective L2 reproduced

below for your convenience:

$$L_2(W; X, \lambda) = \|X - WX\|_2^2$$

F

Reconstruction Loss

$$+ \lambda \|W\|_2^2$$

F

Regularization Loss

Assume $\sigma_1 > \dots > \sigma_m \geq 0$ are the singular values in X , that the number of training points is

larger than the number of features m , and that X can be expressed in SVD coordinates as $X = U\Sigma V^T$.

i. You are given that the optimizing weight matrix for the regularized objective L2 above takes the following form. Fill in the empty matrices below.

$W =$

□

□

·

□

σ^2
1
 σ^2
 $1+\lambda$
 σ^2
2
 σ^2
 $2+\lambda$
...
 σ^2_m
 $\sigma^2_m+\lambda$
 \square

.
 \square
 \square
(5)

ii. Derive the above expression.

(Hint: Can you understand $L_2(W; X, \lambda)$ as a sum of m completely decoupled ridge-regression problems?)

(Hint: The Frobenius norm is equal to $\|A\|_F$
 $F := \text{tr}(AAT)$, and it is invariant under orthogonal transform. That is, $\|A\|_F = \|UAV^T\|_F$
 F for any orthogonal matrices U, V , and any rectangular matrix A , as long as U, A, V have compatible shapes.)

(c) You are given that the data matrix $X \in \mathbb{R}^{8 \times n}$ has the following singular values:

$$\{\sigma_i\} = \{10, 8, 4, 1, 0.5, 0.36, 0.16, 0.01\}$$

Homework 8, © Faculty teaching EECS 182, Fall 2025. All Rights Reserved.

This may not be publicly shared without explicit permission. 3

Homework 8 @2025-10-25 08:44:19Z

For what set of hyperparameter values λ can we guarantee that the learned purifier W will preserve at least 80% of the feature directions corresponding to the first 3 singular vectors of X , 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?) The picture attached also demonstrate the question , in case of you don't understand it , and you can have a double check for this question

✉ 已思考 (用时 99 秒) >

Solution for Problem 3: Self-Supervised Linear Purification

◆ Part (a)

❓ Question Interpretation

We are given two candidate weight matrices $W^{(\alpha)}$ (identity) and $W^{(\beta)}$ (partial identity), and a data matrix X . We need to compute the reconstruction loss $\|X - W X\|_F^2$ and regularization loss $\|W\|_F^2$ for each, then determine for which λ values $W^{(\alpha)}$ has higher total loss than $W^{(\beta)}$.

🧠 Step-by-Step Solution

(i) Compute Losses

Given:

$$X = \begin{bmatrix} -2.17 & 1.98 & 2.41 & -2.03 \\ 0.02 & -0.01 & 0.01 & -0.02 \end{bmatrix}$$

For $W^{(\alpha)} = I$:

- Reconstruction loss: $\|X - IX\|_F^2 = \|0\|_F^2 = 0$
- Regularization loss: $\|I\|_F^2 = 2$ (since $1^2 + 0^2 + 0^2 + 1^2 = 2$)

For $W^{(\beta)} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$:

- $W^{(\beta)}X = \begin{bmatrix} -2.17 & 1.98 & 2.41 & -2.03 \\ 0 & 0 & 0 & 0 \end{bmatrix}$
- Reconstruction error: $X - W^{(\beta)}X = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0.02 & -0.01 & 0.01 & -0.02 \end{bmatrix}$
- Reconstruction loss: $0.02^2 + (-0.01)^2 + 0.01^2 + (-0.02)^2 = 0.001$
- Regularization loss: $\|W^{(\beta)}\|_F^2 = 1^2 + 0^2 + 0^2 + 0^2 = 1$

Completed Table:

Encoder	Reconstruction Loss	Regularization Loss
α	0	2
β	0.001	1

(ii) Compare Total Losses

- $\mathcal{L}_2^{(\alpha)} = 0 + 2\lambda = 2\lambda$
- $\mathcal{L}_2^{(\beta)} = 0.001 + 1\lambda = 0.001 + \lambda$

Set $\mathcal{L}_2^{(\alpha)} > \mathcal{L}_2^{(\beta)}$:

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

✓ Self-Check

- Computations match given reconstruction loss for β .
- Inequality derivation is straightforward.

◆ Part (b)

? Question Interpretation

We derive the optimal W for the regularized loss \mathcal{L}_2 using SVD and orthogonal invariance properties.

🧠 Step-by-Step Solution

(i) Fill in Matrices

Given $X = U\Sigma V^\top$, the optimal weight matrix is:

$$\hat{W} = U \cdot \begin{bmatrix} \frac{\sigma_1^2}{\sigma_1^2 + \lambda} & & \\ & \ddots & \\ & & \frac{\sigma_m^2}{\sigma_m^2 + \lambda} \end{bmatrix} \cdot U^\top$$

(ii) Derivation

1. Orthogonal Invariance:

- $\|X - WX\|_F^2 = \|U^\top(X - WX)V\|_F^2 = \|\Sigma - U^\top WU\Sigma\|_F^2$
- $\|W\|_F^2 = \|U^\top WU\|_F^2$

2. Let $A = U^\top WU$:

- Loss becomes: $\|\Sigma - A\Sigma\|_F^2 + \lambda\|A\|_F^2$

3. Decouple via Diagonal Structure:

- Since Σ is diagonal, optimal A is diagonal: $A = \text{diag}(a_1, \dots, a_m)$
- Each a_i minimizes: $\sigma_i^2(1 - a_i)^2 + \lambda a_i^2$
- Derivative: $-2\sigma_i^2(1 - a_i) + 2\lambda a_i = 0$
- Solution: $a_i = \frac{\sigma_i^2}{\sigma_i^2 + \lambda}$

4. Recover W :

- $W = UAU^\top$

Self-Check

- Derivation uses Frobenius norm invariance and decoupling.
- Solution matches given form.

The model's own checks are largely useless; they almost never detect errors.

◆ Part (c)

❓ Question Interpretation

Given singular values, find λ such that:

- First 3 components are preserved $\geq 80\%$: $\frac{\sigma_i^2}{\sigma_i^2 + \lambda} \geq 0.8$
- Remaining components are attenuated $\leq 50\%$: $\frac{\sigma_i^2}{\sigma_i^2 + \lambda} \leq 0.5$

Step-by-Step Solution

Singular values: $\{10, 8, 4, 1, 0.5, 0.36, 0.16, 0.01\}$

Conditions:

1. Preservation (i=1,2,3):

$$\frac{\sigma_i^2}{\sigma_i^2 + \lambda} \geq 0.8 \implies \lambda \leq \frac{\sigma_i^2}{4}$$

- $\sigma_1 = 10 \implies \lambda \leq 25$
- $\sigma_2 = 8 \implies \lambda \leq 16$
- $\sigma_3 = 4 \implies \lambda \leq 4$
- **Most restrictive:** $\lambda \leq 4$

2. Attenuation (i=4,\dots,8):

$$\frac{\sigma_i^2}{\sigma_i^2 + \lambda} \leq 0.5 \implies \lambda \geq \sigma_i^2$$

- $\sigma_4 = 1 \implies \lambda \geq 1$
- $\sigma_5 = 0.5 \implies \lambda \geq 0.25$
- $\sigma_6 = 0.36 \implies \lambda \geq 0.1296$
- $\sigma_7 = 0.16 \implies \lambda \geq 0.0256$
- $\sigma_8 = 0.01 \implies \lambda \geq 0.0001$
- **Most restrictive:** $\lambda \geq 1$

Combined: $1 \leq \lambda \leq 4$

Self-Check

- Boundaries checked: at $\lambda = 1$, component 4 is exactly 50%; at $\lambda = 4$, component 3 is exactly 80%.
- All conditions are satisfied within this range.

Correct!

Overall Self-Check

- All answers directly address the questions.
- Computations are explicit and verified.
- Derivation in (b) uses orthogonal invariance and decoupling correctly.
- Range in (c) satisfies all constraints.

This completes the solution to Problem 3. If you have any questions or need further clarification, feel free to ask! 😊



image.png
PNG 237.15KB



image.png
PNG 273.11KB

Very good! Now let's do problem 4: 4. Ridge-Attention

In 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 $m = 1$

n

n

$i=1$ x_i

be the average of n points. Use m, x_{n+1}, n and simple arithmetic operations to compute $m' =$

1

$n+1$

$n+1$

$i=1$ x_i —the average of all points including the new point x_{n+1} .

(HINT: Start by multiplying m by n .)

(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 $A =$
we can find the familiar closed-form solution

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

□

□

□

□

□

□

□

□

□

x^T

1

xT

2

.

.

.

xTn

□

□

□

that allows us to make scalar predictions on any new test input x by computing $w^T x$.

First, write the two terms $(A^T A + \lambda I)$ and $A^T y$ as sums involving the x_i and y_i .

i.e. Complete:

$$(A^T A + \lambda I) = \lambda I + n$$

i=1

$$A^T y = n$$

i=1

□

□

□

and vector y =

□

□

□

□

□

y_1

y_2

.

.

.

y_n

□

□

□

so that

(6)

(c) 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 $d < n$. What is the computational cost of a non-causal ridge self-attention layer?

- $O(d^4)$
- $O(nd^2)$

Homework 8, © Faculty teaching EECS 182, Fall 2025.

All Rights Reserved. This may not be publicly shared without explicit permission.

4

Homework 8 @2025-10-25 08:44:19Z

- $O(n^2d^3)$
- $O(n^2d^2)$
- $O(n^2)$
- $O(1)$

(HINT: Do not forget that for a single d -dimensional query vector q , attention needs to return a d -dimensional result.)

(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?

The ridge λ viewed as a learnable parameter for the self-attention layer.

The keys

The values

The queries

(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:

$$(M + uv^T)^{-1} = M^{-1} -$$

1

$$1 + v^T M^{-1} u (M^{-1} u) (v^T M^{-1})$$

(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^T . 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(d^4)
- O(d^3)
- O(d^2)
- O(d)
- O(1)

(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?)

(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? The picture attached also demonstrate the question in case of you don't

✉ 已思考 (用时 219 秒) >

█ Solution for Problem 4: Ridge-Attention

◆ Part (a)

The interpretation shows its strategy.

❓ Question Interpretation

We need to update the average m of n points to include a new point x_{n+1} , using only m , x_{n+1} , and n .

🧠 Step-by-Step Solution

- The current average is $m = \frac{1}{n} \sum_{i=1}^n x_i$, so $\sum_{i=1}^n x_i = n \cdot m$.
- Adding x_{n+1} , the new sum is $n \cdot m + x_{n+1}$.
- The new average is:

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

✓ Self-Check

- For $n = 1$, $m = x_1$, then $m' = \frac{x_1 + x_2}{2}$, which is correct.

◆ Part (b)

❓ Question Interpretation

We express the ridge regression terms $A^\top A + \lambda I$ and $A^\top \mathbf{y}$ as sums over the data points.

🧠 Step-by-Step Solution

- $A^\top A = \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^\top$

- $A^\top \mathbf{y} = \sum_{i=1}^n \mathbf{x}_i y_i$

Thus:

$$(A^\top A + \lambda I) = \lambda I + \sum_{i=1}^n \mathbf{x}_i \mathbf{x}_i^\top$$

$$A^\top \mathbf{y} = \sum_{i=1}^n \mathbf{x}_i y_i$$

Self-Check

- Dimensions match: $\mathbf{x}_i \mathbf{x}_i^\top$ is $d \times d$, and $\mathbf{x}_i y_i$ is $d \times 1$.

◆ Part (c)

? Question Interpretation

We compute the computational cost of non-causal ridge self-attention with n queries and d -dimensional keys/values.

Step-by-Step Solution

- Precompute $M = K^\top K + \lambda I$: cost $O(nd^2)$.
- Invert M : cost $O(d^3)$.
- Compute $V^\top K$: cost $O(nd^2)$.
- Compute $(V^\top K)M^{-1}$: cost $O(d^3)$.
- For all queries Q , compute output $Q(V^\top K M^{-1})$: cost $O(nd^2)$.

Total cost: $O(nd^2 + d^3)$. Since $d < n$, this simplifies to $O(nd^2)$.

Self-Check

- The dominant term is $O(nd^2)$, which matches the option $O(nd^2)$.

◆ Part (d)

❓ Question Interpretation

We determine which components can receive gradients during backpropagation through ridge self-attention.

🧠 Step-by-Step Solution

- The solution $W = (K^\top K + \lambda I)^{-1} K^\top V$ is differentiable with respect to λ , K , V , and Q .
- Gradients can be computed for all parameters via matrix calculus.

Answer: All of them (λ , keys, values, queries) can receive gradients.

✓ Self-Check

- The matrix inverse and products are differentiable operations.

◆ Part (e)

❓ Question Interpretation

We compute the cost of applying the Sherman-Morrison formula to update a matrix inverse.

🧠 Step-by-Step Solution

- Compute $M^{-1}u$: cost $O(d^2)$.
- Compute $v^\top M^{-1}$: cost $O(d^2)$.
- Compute inner product $v^\top M^{-1}u$: cost $O(d)$.
- Compute dyad $(M^{-1}u)(v^\top M^{-1})$: cost $O(d^2)$.
- Scalar division and subtraction: cost $O(1)$.

Total cost: $O(d^2)$.

Self-Check

- All steps are $O(d^2)$ or less, so the overall cost is $O(d^2)$.

◆ Part (f)

? Question Interpretation

We design an efficient algorithm for causal ridge self-attention using recursive updates.

Step-by-Step Solution

- Initialize $M_0^{-1} = \lambda^{-1}I$, $N_0 = 0$.
- For $t = 1$ to n :
 - Get new key k_t and value v_t .
 - Update M_t^{-1} from M_{t-1}^{-1} using Sherman-Morrison with $u = v = k_t$: cost $O(d^2)$.
 - Update $N_t = N_{t-1} + k_t v_t^\top$: cost $O(d^2)$.
 - Compute output $o_t = M_t^{-1}(N_t q_t)$: cost $O(d^2)$.

Total cost: $O(nd^2)$.

Self-Check

- Each step avoids recomputing from scratch, leveraging recursive updates.

Mostly correct, but there are some misunderstandings of the question.

◆ Part (g)

? Question Interpretation

We compute the attention weights for ridge-attention, which may be negative.

Step-by-Step Solution

- The output for a query q is $o = Wq = (K^\top K + \lambda I)^{-1} K^\top V q$.

- The weights α such that $o = \sum_{i=1}^n \alpha_i v_i$ are given by:

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

- Compute $b = (K^\top K + \lambda I)^{-1} q$ (cost $O(d^2)$ if precomputed), then $\alpha = Kb$ (cost $O(nd)$).

Self-Check

- The weights α are linear in q and can be negative due to the ridge regression formulation.

Overall Self-Check

- All answers directly address the questions.
- Computational costs are justified step by step.
- The recursive algorithm in (f) efficiently updates the inverse.
- The weight computation in (g) aligns with ridge regression duality.

This completes the solution to Problem 4. If you have any questions or need further clarification, feel free to ask! 😊

for part g, can you write the weight for v_i explicitly?

 已思考 (用时 190 秒) >

◆ Part (g): Computing Attention Weights for Ridge-Attention

? Question Interpretation

For ridge-attention, we need to compute the (possibly negative) weights that determine how much each value vector contributes to the output for a given query vector q .

Step-by-Step Solution

In ridge-attention, the output for a query q is given by:

I gave him a hint, but he still couldn't find the problem.

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

where:

- K is the $n \times d$ key matrix (each row is a key vector k_i)
- V is the $n \times d_v$ value matrix (each row is a value vector v_i)
- q is the $d \times 1$ query vector
- λ is the ridge parameter

We want to express o as a linear combination of the value vectors:

$$o = \sum_{i=1}^n \alpha_i v_i$$

where α_i are the attention weights (which can be negative).

Using the dual form of ridge regression, the weights α can be computed as:

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

Then the output is:

$$o = V^\top \alpha$$

which explicitly shows o as a weighted sum of the columns of V^\top (i.e., the value vectors v_i).

Explicit formula for each weight:

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

where α_i is the weight for value vector v_i .

Self-Check

- The derivation uses the dual form of ridge regression, which is valid for multi-output regression.

- The weights α_i can be negative because $(KK^\top + \lambda I)^{-1}$ is positive definite but Kq can have negative components.
- This matches the intuition from kernel ridge regression where the solution is a