Homework 8 - Deep Learning Course: Full Conversation Trace

Teaching Assistant: Gemini

gemini 2.5 fast

December 5, 2025

note: √ = correctly one-shotted

x = wrong answer

Conversation Trace

User Prompt (Initial Request)

You are a Teaching Assistant and for a Deep Learning Course, who will rigorously and completely solve a homework assignment.

initial prompting with PDF given

I will upload a PDF of the homework assignment. First, please analyze the document and index the question numbers. We will solve one problem at a time. DO NOT solve the entire homework until prompted with "Solve question [number]". Please use LaTex formatting so that your solutions are easily exportable. After solving each problem, please double check your work to check for hallucinations. Please keep account of your entire trace of this conversation, including my prompts. I will ask for the full trace in Latex form. Do you understand? Please respond yes / no and ask any clarifying questions if the prompt is unclear.

Agent Response (Initial Setup)

Yes, I understand. I will analyze the document, index the question numbers, and wait for the prompt **"Solve question [number]"** before providing a solution for each problem. I will use **LaTeX** formatting for the solutions and double-check my work. I will also keep an account of our entire conversation trace.

Homework Question Index

Based on the provided PDF, the questions are indexed as follows:

- **Question 1:** SSM Convolution Kernel
- **Question 2:** Coding SSM Forward
- **Question 3:** Self-Supervised Linear Purification
- **Question 4:** Ridge-Attention
- **Question 5:** Homework Process and Study Group

Which question would you like to start with?

User Prompt (Solve Question 1)

Solve question 1

Agent Response (Solution to Question 1: SSM Convolution Kernel)

(a) Convolution Kernel and the Output Equation

The discrete-time State-Space Model (SSM) equations are:

$$
\begin{aligned}
x_{k+1} &= Ax_k + Bu_k && (1)\\
y_k &= Cx_k + Du_k && (2)
\end{aligned}
$$

state the problem

Assuming $x_0 = 0$, the state x_k can be unrolled:

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

quickly jumps to expression, doesn't necessarily show smaller examples

Substituting into y_k (Eq. 2):

$$y_k = C \left(\sum_{l=0}^{k-1} A^{k-1-l} B u_l \right) + D u_k = D u_k + \sum_{l=1}^{k} C A^{l-1} B u_{k-l}$$

By comparing this to the convolution form $y_k = \sum_{l=0}^{L} K_l u_{k-l}$, we find the kernel K:

$$\boxed{K_l = \begin{cases} D & \text{if } l = 0 \\ CA^{l-1}B & \text{if } 1 \le l \le L \\ 0 & \text{if } l > L \end{cases}}$$

✓ written more concisely than solutions which is nice to see

(b) Concrete Examples

i. Scalar Case ($A = 0.8, B = 1, C = 1.5$) Assuming $D = 0$:

not explicitly set in problem, good assumption made

$$K_0 = D = 0$$
$$K_l = CA^{l-1}B = 1.5 \cdot (0.8)^{l-1} \cdot 1, \quad \text{for } l \ge 1$$

For $L = 4$:

$$K_1 = 1.5 \cdot (0.8)^0 = 1.5$$
$$K_2 = 1.5 \cdot (0.8)^1 = 1.2$$
$$K_3 = 1.5 \cdot (0.8)^2 = 0.96$$
$$K_4 = 1.5 \cdot (0.8)^3 = 0.768$$

notes: answer is thorough and correct

The kernel up to $L = 4$ is $K = \{0, 1.5, 1.2, 0.96, 0.768\}$. ✓

ii. 2D Case

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

$$K_0 = D = 0$$
$$K_1 = CA^0B = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 1$$
$$K_2 = CA^1B = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 0.7 \\ 0.2 \end{pmatrix} = 0.7$$
$$K_3 = CA^2B = C(A(AB)) = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 0.51 \\ 0.26 \end{pmatrix} = 0.51$$

The kernel up to $L = 3$ is $K = \{0, 1, 0.7, 0.51\}$. ✓

(c) Efficient Computation with Convolutions

- **Convolution Output (y_k):** The computation $y_k = \sum_{l=0}^{L} K_l u_{k-l}$ is independent for each k. The summation can be parallelized using a binary tree reduction (parallel prefix sum).
- **Minimum Critical Path (Convolution):** $O(\log L)$ (dominated by the summation depth).

✗ forgot to factor in mat mul critical path

- **Naive Recurrence Unrolling:** The computation of x_{k+1} depends directly on x_k, creating a sequential dependency chain: $x_0 \to x_1 \to \cdots \to x_L$.
- **Minimum Critical Path (Recurrence):** $O(L)$.

✗ again forgot to factor in mat mul

The convolution-based approach achieves a shorter critical path ($O(\log L)$ vs $O(L)$), enabling significantly greater parallelization.

(d) Efficient Kernel Computation

To compute all L kernel elements $K_l = CA^{l-1}B$, the dominant task is computing A^{l-1}. The most parallel approach is to use a **Parallel Prefix Product (or Blelloch Scan)** on the sequence of matrices $[A, A, \ldots, A]$.

right idea of log structure - tree / divide and conquer

- **Method:** Compute $A^1, A^2, \ldots, A^L$ by applying a binary tree reduction/scan structure.
- **Critical Path Length:** This involves $O(\log L)$ sequential matrix-matrix multiplications. Since matrix multiplication is $O(n^3)$, the critical path is $\mathbf{O}(\log \mathbf{L} \cdot \mathbf{n^3})$.

× wrong max depth for mat mul

(e) Diagonal Matrix A

should compute in O(L) time

If $A = \text{diag}(\lambda_1, \ldots, \lambda_n)$, the powers simplify: $A^{l-1} = \text{diag}(\lambda_1^{l-1}, \ldots, \lambda_n^{l-1})$. The kernel term becomes a sum of n scalar terms:

$$K_l = CA^{l-1}B = \sum_{i=1}^{n} c_i \lambda_i^{l-1} b_i$$

highlights diagonal form power

- **Efficiency Gain:** Computing A^{l-1} is $O(n)$. Computing K_l is $O(n)$.
- **Total Complexity (all L kernels): $\mathbf{O(Ln)}$.** This is much faster than the $O(L \log L \cdot n^3)$ work for the general case.

(f) DPLR Structure $(A = I_n + pp^\top)$

Using the rank-1 update identity: $A^{l-1} = (I_n + pp^\top)^{l-1} = I_n + f_{l-1}pp^\top$, where $f_{l-1} = \frac{(1+p^\top p)^{l-1}-1}{p^\top p}$ (a scalar). The kernel computation simplifies:

$$K_l = CA^{l-1}B = C(I_n + f_{l-1}pp^\top)B = \underbrace{CB}_{\text{scalar/vector}} + f_{l-1} \underbrace{(Cp)}_{\text{scalar}} \underbrace{(p^\top B)}_{\text{scalar}}$$

correct realization

- **Efficiency Gain:** The terms CB, Cp, and $p^\top B$ are pre-computed in $O(n)$.
- **Kernel Computation Cost:** For each l, computing the scalar f_{l-1} involves $O(\log L)$ exponentiation, and the rest is $O(1)$ scalar arithmetic.

computing L kernels instead of 1 kernel of length L

- **Total Complexity (all L kernels): $\mathbf{O(n + L \log L)}$.** This is extremely efficient when n is large.

User Prompt (Solve Question 3)

```
Solve question 3
```

Agent Response (Solution to Question 3: Self-Supervised Linear Purification)

The objective function is $\mathcal{L}_2(W; X, \lambda) = ||X - WX||_F^2 + \lambda ||W||_F^2$.

(a) Concrete Examples

i. Compute Losses and Fill Table For $W^{(\alpha)} = I$:

- Reconstruction Loss $||X - W^{(\alpha)}X||_F^2 = ||X - IX||_F^2 = \mathbf{0}$.
- Regularization Loss $||W^{(\alpha)}||_F^2 = ||I||_F^2 = 1^2 + 1^2 = \mathbf{2}$ (for $m = 2$).

Encoder	Reconstruction Loss ✓ $\|\|X - WX\|\|_F^2$	Regularization Loss ✓ $\|\|W\|\|_F^2$
α	**0** ✓	**2** ✓
β	0.001	× R_β (Unknown)

unable to solve

ii. Value of λ where $\mathcal{L}_2(W^{(\alpha)}) > \mathcal{L}_2(W^{(\beta)})$ We assume $W^{(\beta)}$ is a better regularized solution, so $R_\beta < 2$. We require:

$$\mathcal{L}_2(W^{(\alpha)}) > \mathcal{L}_2(W^{(\beta)})$$
$$0 + \lambda \cdot 2 > 0.001 + \lambda \cdot R_\beta$$
$$\lambda(2 - R_\beta) > 0.001$$
$$\lambda > \frac{0.001}{2 - R_\beta}$$

appears unable to calc regularization loss of $W^{(\beta)}$ before

The identity matrix $W^{(\alpha)}$ is penalized more than $W^{(\beta)}$ when λ is above the threshold $\frac{0.001}{2-R_\beta}$.

(b) Derivation of Optimal Weight Matrix $\hat{W}$

i. Fill in the Expression

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

✓

no work shown, simply following the instructions to 'fill'

ii. Derive the Expression $\hat{W}$ Using the orthogonal invariance of the Frobenius norm, we transform the problem using $X = U\Sigma V^\top$ and $\tilde{W} = U^\top W U$.

$$\min_{\tilde{W}} \quad \mathcal{L}_2 = ||\Sigma - \tilde{W}\Sigma||_F^2 + \lambda||\tilde{W}||_F^2$$

Since Σ is diagonal, the loss decouples, forcing $\tilde{W}$ to be diagonal ($\tilde{W}_{ij} = 0$ for $i \neq j$). For the diagonal entries $\tilde{w}_i$:

$$\min_{\tilde{w}_i} \quad \mathcal{L}_i = \sigma_i^2(1 - \tilde{w}_i)^2 + \lambda\tilde{w}_i^2$$

Setting the derivative w.r.t. $\tilde{w}_i$ to zero:

decoupled approach

$$\frac{\partial \mathcal{L}_i}{\partial \tilde{w}_i} = -2\sigma_i^2(1 - \tilde{w}_i) + 2\lambda\tilde{w}_i = 0$$

$$\implies \hat{\tilde{w}}_i = \frac{\sigma_i^2}{\sigma_i^2 + \lambda}$$

minimizing individual wi

Substituting back $\hat{W} = U\hat{\tilde{W}}U^\top$ yields the result.

never explicitly defined

(c) Guaranteeing Attenuation and Preservation

The shrinkage factor is $s_i = \frac{\sigma_i^2}{\sigma_i^2+\lambda}$. We use the critical (worst-case) singular values for each constraint.

1. Preservation (First 3 $\geq 80\%$): Critical $\sigma_3 = 4$.

$$\frac{4^2}{16 + \lambda} \geq 0.8 \implies 16 \geq 12.8 + 0.8\lambda \implies 3.2 \geq 0.8\lambda \implies \lambda \leq 4$$ ✓

2. Attenuation (Remaining $\leq 50\%$): Critical $\sigma_4 = 1$.

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

Conclusion: To satisfy both constraints, the hyperparameter λ must be in the range:

$$\mathbf{1 \leq \lambda \leq 4}$$ ✓

follows solution exactly

User Prompt (Solve Question 4)

`Solve question 4`

Agent Response (Solution to Question 4: Ridge-Attention)

(a) Efficient Update of Simple Averaging

The new average m' of $n+1$ points in terms of the old average m of n points is:

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

no rigorous work shown

(b) Ridge-Regression Components

The optimal weight vector is $w_* = (A^\top A + \lambda I)^{-1} A^\top y$.

1. $(A^\top A + \lambda I) = \boldsymbol{\lambda \mathbf{I} + \sum_{i=1}^{n} x_i x_i^\top}$ ✓
2. $A^\top y = \boldsymbol{\sum_{i=1}^{n} x_i y_i}$

again, no work shown

(c) Computational Cost of Non-Causal Ridge Self-Attention

The output Z is computed as $Z = QW$, where $W = (K^\top K + \lambda I)^{-1} K^\top V$.

- $K^\top K$ and $K^\top V$: $O(nd^2)$ each.
- Matrix inversion $(K^\top K + \lambda I)^{-1}$: $O(d^3)$.
- W: $O(d^3)$ matrix multiplication.
- $Z = QW$: $O(nd^2)$ matrix multiplication.

all correct, very compact

Assuming $d < n$, the total cost is dominated by $O(nd^2)$:

$$\mathbf{O(nd^2)}$$ ✓

(d) Backpropagation Gradients

The backpropagation successfully passes gradients to the parameters that define the layer's output function $Z(Q, K, V)$:

- **The queries (Q):** Q appears directly in $Z = QW$.
- **The keys (K):** K appears in W (in $K^\top K$ and $K^\top V$).
- **The values (V):** V appears in W (in $K^\top V$).

✗ does not choose λ

Gradients pass to **The keys**, **The values**, and **The queries**.

(e) Computational Cost of Sherman-Morrison Formula

$$(M + uv^\top)^{-1} = M^{-1} - \frac{1}{1 + v^\top M^{-1} u}(M^{-1}u)(v^\top M^{-1})$$

The cost is dominated by the two matrix-vector multiplications ($M^{-1}u$ and $v^\top M^{-1}$), both $O(d^2)$.

$$\mathbf{O(d^2)}$$ ✓

correctly notes both are $O(d^2)$

(f) Efficient Causal Ridge-Self-Attention

For a causal implementation, we compute the output z_t at step t using data up to t:

$$M_t = M_{t-1} + k_t k_t^\top \quad \text{and} \quad R_t = R_{t-1} + k_t v_t^\top$$

reference from above

1. **Update M_t^{-1}:** Use the Sherman-Morrison formula: $O(d^2)$ per step t.
2. **Update R_t:** Compute outer product $k_t v_t^\top$ and add: $O(d^2)$ per step t.
3. **Compute Output z_t:** $z_t = W_t q_t = (M_t^{-1} R_t) q_t$. The product $M_t^{-1} R_t$ is $O(d^3)$.

The total complexity over n steps is $n \times O(d^3) = \mathbf{O(nd^3)}$. ✗

doesn't realize matrix vector products can be used

(g) Computing Weights for Ridge-Attention

The output z for a query q is a linear combination of the value vectors v_i: $z = \sum_{i=1}^n v_i \alpha_i$. Using the dual form of ridge regression, the vector of weights $\alpha \in \mathbb{R}^{n\times 1}$ is:

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

using the K matrix instead of A

where $K \in \mathbb{R}^{n\times d}$ is the key matrix, $q \in \mathbb{R}^{d\times 1}$ is the query vector, and $I \in \mathbb{R}^{n\times n}$ is the identity matrix. The final output is $z = V^\top \alpha$ (if V rows are $v_i^\top$) or $z = V\alpha$ (if V is $d \times n$, and v_i are $d \times 1$). The i-th element of α is the weight α_i applied to the i-th value vector v_i.