

Annotated log of conversation with Deepseek Chat interface for solving HW3

first prompt:

You are an expert in solving deep learning problems. You will be given some problems, and output the answer in markdown format.

output:

Of course. I am ready to function as your deep learning expert.

Please provide the problems you would like me to solve. I will structure my answers in a clear, comprehensive, and well-formatted README style, breaking down the concepts, solutions, and reasoning step-by-step.

For example, if you were to ask me:

Problem: "Explain the vanishing gradient problem in Recurrent Neural Networks (RNNs) and how LSTM networks solve it."

My output would be structured as follows:

(omit the example output)

solve problem 1

link: <https://chat.deepseek.com/share/ckg99svm571xnkvb57>

1(a)

prompt:

solve problem 1(a).

output:

Of course. Here is the solution to Problem 1(a) in a clear, README format.

1. Problem Restatement

We consider a dense layer:

- Input: $x \in \mathbb{R}^{d_1}$, with $x_i \sim \mathcal{N}(0, 1)$ i.i.d.
- Weights: $W \in \mathbb{R}^{d_2 \times d_1}$, with $W_{ij} \sim \mathcal{N}(0, 1)$ i.i.d.
- Output: $y = Wx \in \mathbb{R}^{d_2}$

We are asked to find:

1. The expected squared RMS norm of y .
2. How this scales with d_1 and d_2 .

3. The constant c to multiply W by such that the expected squared RMS norm of $(cW)x$ is 1.

The RMS (Root Mean Square) norm for a vector $v \in \mathbb{R}^n$ is defined as:

$$\text{RMS}(v) = \sqrt{\frac{1}{n} \sum_{i=1}^n v_i^2} = \frac{\|v\|_2}{\sqrt{n}}$$

Therefore, the **squared RMS norm** is:

$$\text{RMS}(v)^2 = \frac{1}{n} \sum_{i=1}^n v_i^2 = \frac{\|v\|_2^2}{n}$$

2. Step-by-Step Solution

Step 1: Analyze a Single Output Component

Let's first find the distribution of a single component of the output vector, y_j , which is the dot product of the j -th row of W (denoted W_j) and the input x .

$$y_j = \sum_{k=1}^{d_1} W_{jk} x_k$$

Since W_{jk} and x_k are independent, zero-mean, unit-variance Gaussians, the product $W_{jk}x_k$ has a mean of 0. The variance of the sum is the sum of the variances because all terms are independent and mean-zero.

The variance of each term $W_{jk}x_k$ is:

$$\text{Var}(W_{jk}x_k) = \mathbb{E}[W_{jk}^2] \mathbb{E}[x_k^2] = (1)(1) = 1$$

Therefore, the variance of the sum y_j is:

$$\text{Var}(y_j) = \sum_{k=1}^{d_1} \text{Var}(W_{jk}x_k) = \sum_{k=1}^{d_1} 1 = d_1$$

Thus, each output component y_j is distributed as:

$$y_j \sim \mathcal{N}(0, d_1)$$

Step 2: Find the Expected Squared Norm of the Output Vector

The squared L2-norm of the output vector y is:

$$\|y\|_2^2 = \sum_{j=1}^{d_2} y_j^2$$

We take the expectation of this quantity:

$$\mathbb{E}[\|y\|_2^2] = \mathbb{E}\left[\sum_{j=1}^{d_2} y_j^2\right] = \sum_{j=1}^{d_2} \mathbb{E}[y_j^2]$$

Since $\mathbb{E}[y_j] = 0$, $\mathbb{E}[y_j^2] = \text{Var}(y_j) = d_1$.

$$\mathbb{E}[\|y\|_2^2] = \sum_{j=1}^{d_2} d_1 = d_1 d_2$$

Step 3: Find the Expected Squared RMS Norm

The squared RMS norm is defined as:

$$\text{RMS}(y)^2 = \frac{\|y\|_2^2}{d_2}$$

Taking the expectation:

$$\mathbb{E}[\text{RMS}(y)^2] = \mathbb{E}\left[\frac{\|y\|_2^2}{d_2}\right] = \frac{1}{d_2} \mathbb{E}[\|y\|_2^2] = \frac{1}{d_2} (d_1 d_2) = d_1$$

Scaling: The expected squared RMS norm of y is d_1 . It scales **linearly with the input dimension d_1** and is **independent of the output dimension d_2** .

Step 4: Find the Scaling Constant

We want to find a constant c such that for the scaled weight matrix cW , the output $y_{\text{scaled}} = (cW)x$ has an expected squared RMS norm of 1.

The output is $y_{\text{scaled}} = c(Wx) = cy$.

The squared RMS norm of the scaled output is:

$$\text{RMS}(y_{\text{scaled}})^2 = \frac{\|cy\|_2^2}{d_2} = c^2 \frac{\|y\|_2^2}{d_2} = c^2 \cdot \text{RMS}(y)^2$$

Taking the expectation:

$$\mathbb{E}[\text{RMS}(y_{\text{scaled}})^2] = c^2 \cdot \mathbb{E}[\text{RMS}(y)^2] = c^2 \cdot d_1$$

We set this equal to 1:

$$c^2 \cdot d_1 = 1$$

$$c = \frac{1}{\sqrt{d_1}}$$

3. Final Answer

1. The **expected squared RMS norm** of the output features $y = Wx$ is d_1 .
2. It **scales linearly with the input dimension d_1** and is **independent of the output dimension d_2** .
3. To ensure the expected squared RMS norm is 1, we should multiply the weight matrix W by the constant $\frac{1}{\sqrt{d_1}}$. This is the famous "Xavier" or "Kaiming" initialization scaling factor for linear layers, which ensures the signal variance is preserved through the network at initialization.

comment:

Deepseek's answer is correct, and the reasoning is clear and correct. It has also followed the 'Hint' provided in the problem statement.

1(b)

prompt:

solve problem 1(b). there is a typo. it should be $(g_i x_i^T)$ answer it again.

output:

You are absolutely right. Thank you for the correction. The gradient for a dense layer is indeed $\nabla_W f(W) = g_i x_i^T$, not $x_i g_i^T$. This changes the dimensions and the subsequent analysis. Let's solve it correctly.

1. Problem Restatement (Corrected)

We are considering the training update step using the SignGD optimizer. The setup is:

- Input: $x_i \in \mathbb{R}^{d_1}$, with $x_i \sim \mathcal{N}(0, I)$ i.i.d.
- Backpropagated vector: $g_i \in \mathbb{R}^{d_2}$, assumed to be a constant vector for this analysis.
- The raw gradient is an outer product: $\nabla_W f(W) = g_i x_i^T \in \mathbb{R}^{d_2 \times d_1}$
- Weight Update: $\Delta W = \eta \cdot \text{sign}(g_i x_i^T)$
- The corresponding change in the output features y for a *new* input x_i (from the same distribution) is:

$$y' = \Delta W \cdot x_i = \text{sign}(g_i x_i^T) x_i$$

(We ignore the learning rate η for now as we are analyzing the *unscaled* change).

We are asked to find:

1. The expected squared RMS norm of this unscaled feature change y' .
2. How this scales with d_1 and d_2 .
3. The constant c to multiply the update by to ensure the expected RMS norm of y' is 1.

2. Step-by-Step Solution

Step 1: Understand the `sign` operation and the matrix structure

The term $g_i x_i^T$ is an outer product, resulting in a matrix $M \in \mathbb{R}^{d_2 \times d_1}$, where the element in the j -th row and k -th column is:

$$M_{jk} = (g_i)_j \cdot (x_i)_k$$

The `sign` function is applied element-wise to this matrix. Therefore, the element $[\text{sign}(g_i x_i^T)]_{jk}$ is:

$$[\text{sign}(g_i x_i^T)]_{jk} = \text{sign}((g_i)_j \cdot (x_i)_k)$$

Since $(g_i)_j$ is a constant for a given j and minibatch, and $(x_i)_k$ is a random variable, the sign of their product depends on both.

$$\text{sign}((g_i)_j \cdot (x_i)_k) = \text{sign}((g_i)_j) \cdot \text{sign}((x_i)_k)$$

This is a crucial simplification. It means the sign matrix factors into a function of g_i and a function of x_i .

Let's define:

- $s_x = \text{sign}(x_i) \in \mathbb{R}^{d_1}$, a random vector with i.i.d. entries of ± 1 with probability 1/2 each.
- $s_g = \text{sign}(g_i) \in \mathbb{R}^{d_2}$, a constant vector of ± 1 determined by the fixed vector g_i .

The update matrix can now be written as an outer product:

$$\text{sign}(g_i x_i^T) = s_g \cdot s_x^T$$

Step 2: Compute the feature change vector y'

The feature change is:

$$y' = (\text{sign}(g_i x_i^T)) x_i = (s_g s_x^T) x_i$$

Let's compute this product. The matrix-vector multiplication $(s_g s_x^T) x_i$ results in a vector in \mathbb{R}^{d_2} . The j -th component is:

$$y'_j = \sum_{k=1}^{d_1} [s_g s_x^T]_{jk} \cdot (x_i)_k = \sum_{k=1}^{d_1} (s_g)_j (s_x)_k (x_i)_k = (s_g)_j \sum_{k=1}^{d_1} (s_x)_k (x_i)_k$$

Notice that the sum $\sum_{k=1}^{d_1} (s_x)_k (x_i)_k$ is common to all output components j . Let's denote this scalar as S :

$$S = \sum_{k=1}^{d_1} (s_x)_k (x_i)_k$$

Therefore, the entire output vector y' can be written as:

$$y' = s_g \cdot S$$

Step 3: Find the expected squared L2-norm of y'

The squared L2-norm is:

$$\|y'\|_2^2 = \sum_{j=1}^{d_2} (y'_j)^2 = \sum_{j=1}^{d_2} [(s_g)_j \cdot S]^2 = \sum_{j=1}^{d_2} (s_g)_j^2 S^2$$

Since $(s_g)_j^2 = 1$ for all j , this simplifies to:

$$\|y'\|_2^2 = \sum_{j=1}^{d_2} S^2 = d_2 \cdot S^2$$

Now we take the expectation:

$$\mathbb{E}[\|y'\|_2^2] = \mathbb{E}[d_2 \cdot S^2] = d_2 \cdot \mathbb{E}[S^2]$$

We need to find $\mathbb{E}[S^2]$, where $S = \sum_{k=1}^{d_1} (s_x)_k (x_i)_k$.

- The random variables $(s_x)_k$ and $(x_i)_k$ are *not* independent. However, $(s_x)_k = \text{sign}((x_i)_k)$.
- The product $(s_x)_k (x_i)_k = \text{sign}((x_i)_k) \cdot (x_i)_k = |(x_i)_k|$.
- Since $(x_i)_k \sim \mathcal{N}(0, 1)$, $|(x_i)_k|$ follows a half-normal distribution.

The expectation of $|(x_i)_k|$ is $\sqrt{\frac{2}{\pi}}$, but we need the second moment.

$$\mathbb{E}[(s_x)_k (x_i)_k]^2 = \mathbb{E}[|(x_i)_k|^2] = \mathbb{E}[(x_i)_k^2] = 1$$

Furthermore, for $k \neq l$, the terms $|(x_i)_k|$ and $|(x_i)_l|$ are independent because the x_i components are i.i.d. Thus:

$$\mathbb{E}[S^2] = \mathbb{E}\left[\left(\sum_{k=1}^{d_1} |(x_i)_k|\right)^2\right] = \sum_{k=1}^{d_1} \mathbb{E}[|(x_i)_k|^2] + \sum_{k \neq l} \mathbb{E}[|(x_i)_k|] \mathbb{E}[|(x_i)_l|]$$

$$\mathbb{E}[S^2] = \sum_{k=1}^{d_1} 1 + \sum_{k \neq l} \left(\sqrt{\frac{2}{\pi}}\right)^2 = d_1 + d_1(d_1 - 1) \cdot \frac{2}{\pi}$$

$$\mathbb{E}[S^2] = d_1 + \frac{2}{\pi} d_1(d_1 - 1) = \frac{2}{\pi} d_1^2 + d_1(1 - \frac{2}{\pi})$$

For large d_1 , the dominant term is $\frac{2}{\pi} d_1^2$. We will keep the exact expression.

Substituting back:

$$\mathbb{E}[\|y'\|_2^2] = d_2 \cdot \left(\frac{2}{\pi} d_1^2 + d_1(1 - \frac{2}{\pi})\right)$$

Step 4: Find the Expected Squared RMS Norm

The squared RMS norm is:

$$\text{RMS}(y')^2 = \frac{\|y'\|_2^2}{d_2}$$

Taking the expectation:

$$\mathbb{E}[\text{RMS}(y')^2] = \frac{1}{d_2} \mathbb{E}[\|y'\|_2^2] = \frac{1}{d_2} \cdot d_2 \cdot \left(\frac{2}{\pi} d_1^2 + d_1 \left(1 - \frac{2}{\pi} \right) \right)$$

$$\mathbb{E}[\text{RMS}(y')^2] = \frac{2}{\pi} d_1^2 + d_1 \left(1 - \frac{2}{\pi} \right)$$

Scaling: For large d_1 , the expected squared RMS norm scales as $\Theta(d_1^2)$. It depends **quadratically on the input dimension d_1** and is **independent of the output dimension d_2** .

Step 5: Find the Scaling Constant

We want to find a constant c such that for the scaled update $c \cdot \text{sign}(g_i x_i^T) x_i$, the expected RMS norm is 1.

The scaled feature change is $y'_{\text{scaled}} = c y'$.

$$\mathbb{E}[\text{RMS}(y'_{\text{scaled}})^2] = c^2 \cdot \mathbb{E}[\text{RMS}(y')^2] = c^2 \cdot \left(\frac{2}{\pi} d_1^2 + d_1 \left(1 - \frac{2}{\pi} \right) \right)$$

Set this equal to 1:

$$c^2 \cdot \left(\frac{2}{\pi} d_1^2 + d_1 \left(1 - \frac{2}{\pi} \right) \right) = 1$$

$$c = \frac{1}{\sqrt{\frac{2}{\pi} d_1^2 + d_1 \left(1 - \frac{2}{\pi} \right)}}$$

For large d_1 , the d_1^2 term dominates, so the scaling constant is approximately:

$$c \approx \frac{1}{d_1} \sqrt{\frac{\pi}{2}}$$

The key takeaway is the $1/d_1$ scaling.

3. Final Answer

1. The **expected squared RMS norm** of the unscaled feature change $y' = \text{sign}(g_i x_i^T) x_i$ is $\frac{2}{\pi} d_1^2 + d_1 \left(1 - \frac{2}{\pi} \right)$.
2. It **scales quadratically with the input dimension d_1** (as $\Theta(d_1^2)$) and is **independent of the output dimension d_2** .
3. To ensure the expected RMS norm of y' is 1, we should multiply the update by the constant $\frac{1}{\sqrt{\frac{2}{\pi} d_1^2 + d_1 \left(1 - \frac{2}{\pi} \right)}}$. For large d_1 , this scales as $\Theta\left(\frac{1}{d_1}\right)$.

comment:

The reasoning and answer is correct. There is a typo. It should be $(g_i x_i^T)$. The solution posted on Ed is not so clear.

$$\mathbb{E} \left[\left(\sum_{k=1}^{d_1} |(x_i)_k| \right)^2 \right] = \sum_{k=1}^{d_1} \mathbb{E}[|(x_i)_k|^2] + \sum_{k \neq l} \mathbb{E}[|(x_i)_k|] \mathbb{E}[|(x_i)_l|]$$

$\sum_{k \neq l} \mathbb{E}[|(x_i)_k|] \mathbb{E}[|(x_i)_l|]$ can't be missed in the calculation of $\mathbb{E}[S^2]$.

1(c)

prompt:

solve problem 1(c).

output:

Of course. Here is the solution to Problem 1(c).

1. Problem Restatement

In parts (a) and (b), we calculated scaling constants to normalize the expected squared RMS norm of the output features. We observed that:

- In **(a)** (Initialization), the scaling constant was $\frac{1}{\sqrt{d_1}}$.
- In **(b)** (Update), the scaling constant was $\frac{1}{\sqrt{\frac{2}{n} d_1^2 + d_1(1 - \frac{2}{n})}} \approx \Theta(\frac{1}{d_1})$.

The problem asks: **Why do these scaling factors depend only on the input dimension d_1 , and not on the output dimension d_2 ?**

2. Intuitive Explanation

The RMS norm is defined as $\text{RMS}(y) = \frac{\|y\|_2}{\sqrt{d_2}}$. It measures the **average contribution per output feature**. The scaling is designed to control the *intensity* of the signal flowing through the network, on a *per-neuron* basis, regardless of how many neurons there are in the layer.

The input dimension d_1 determines how many independent random variables are being summed together to create each output feature. This summation is the fundamental source of variance (at initialization) and the amplification of the update (during training). The output dimension d_2 merely determines how many of these independently computed features we have; it averages out in the RMS calculation.

3. Detailed Mathematical Reason

Let's trace the origin of the d_1 -dependence and the d_2 -independence through the calculations.

For Part (a): Initialization

1. **Single Output Component:** A single output y_j is a sum of d_1 independent terms: $y_j = \sum_{k=1}^{d_1} W_{jk} x_k$. Its variance scales with d_1 .
2. **Full Output Vector:** The squared L2-norm $\|y\|_2^2$ is the sum of d_2 such components: $\|y\|_2^2 = \sum_{j=1}^{d_2} y_j^2$. Its expectation is $\mathbb{E}[\|y\|_2^2] = d_2 \cdot d_1$.
3. **RMS Norm:** The RMS norm divides this total squared norm by d_2 : $\text{RMS}(y)^2 = \frac{\|y\|_2^2}{d_2}$. The expectation is $\mathbb{E}[\text{RMS}(y)^2] = \frac{1}{d_2} \cdot (d_1 d_2) = d_1$.

The factor of d_2 from summing over the output features is **exactly canceled** by the division by d_2 in the RMS definition. The remaining scaling comes entirely from the sum over d_1 input dimensions within each feature.

For Part (b): Update

1. **Single Output Component:** The analysis revealed that every output component of the update y'_j is proportional to the same scalar $S = \sum_{k=1}^{d_1} |(x_i)_k|$. The magnitude of S , specifically S^2 , has an expectation that scales with d_1^2 .
2. **Full Output Vector:** The squared L2-norm is $\|y'\|_2^2 = \sum_{j=1}^{d_2} (y'_j)^2 = d_2 \cdot S^2$. Its expectation scales with $d_2 \cdot d_1^2$.
3. **RMS Norm:** The RMS norm is $\text{RMS}(y')^2 = \frac{\|y'\|_2^2}{d_2} = S^2$. Its expectation $\mathbb{E}[\text{RMS}(y')^2] = \mathbb{E}[S^2]$ scales with d_1^2 .

Again, the factor of d_2 from the sum over output features is **canceled** by the RMS normalization. The scaling is determined solely by the sum S over the d_1 input dimensions.

4. Final Answer

The update scaling depends only on d_1 because the RMS norm calculation **averages over the output dimension** d_2 . The RMS norm $\left(\frac{\|y\|_2}{\sqrt{d_2}}\right)$ is a *per-feature* measure of signal strength. The amplification of the signal, both at initialization and during an update, originates from the **summation over the d_1 input connections** that define each individual output feature. The number of output features d_2 does not affect the intrinsic strength of each one, and is therefore normalized out by the RMS calculation.

In essence:

- d_1 controls the **depth of the summation** per feature, which amplifies the signal.
- d_2 controls the **breadth of features**, which is averaged over by the RMS norm.

comment:

The reasoning and answer is correct. The explanation is clear and intuitive.

Solve problem 3.

link <https://chat.deepseek.com/share/0ns61vklfokc4yds8>

3(a) & 3(b)

prompt:

You are an expert in solving deep learning problems. You will be given some problems, and output the answer in markdown format. Wait for me to give you the problem

output:

Of course. I am ready. Please provide the deep learning problem, and I will deliver a detailed, step-by-step solution in markdown format.

prompt:

read the problem set and the paper. solve problem 3(a) and 3(b)
(attach the problem set and the paper file)

output:

Of course. Having reviewed the homework problem and the referenced paper ("Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer"), I will now provide the solutions for problems 3(a) and 3(b).

Solution to Problem 3: Maximal Update Parameterization Research

3(a) Figure 1 Observation

The two side-by-side figures in Figure 1 of the paper plot the **training loss against the learning rate** for Transformers of varying widths (d_{model}), trained with the Adam optimizer.

- **Left Plot (Standard Parametrization - SP):** This plot shows that the optimal learning rate (the LR that gives the lowest loss) is highly unstable and changes significantly as the model's width increases. Furthermore, the performance of wider models is not consistently better; sometimes a wider model underperforms a narrower one even after tuning the learning rate (as indicated by the dashed line).
- **Right Plot (Maximal Update Parametrization - μP):** This plot shows that the optimal learning rate remains stable across different model widths. The curve for each width is almost a vertical translation of the others. Crucially, the performance monotonically improves with width for any fixed learning rate.

Observation about the width of a matrix: The key observation is that in the Standard Parametrization (SP), the optimal hyperparameters, like the learning rate, are **not stable** with respect to changes in the width of the model's layers. In contrast, under the Maximal Update Parametrization (μP), the optimal hyperparameters **become stable** across width, enabling zero-shot hyperparameter transfer from a small model to a large one.

3(b) Standard Parameterization Scaling in Table 3

The paper's Table 3 summarizes the scaling rules for both μP and the Standard Parametrization (SP) for general neural networks.

In the lecture, an elementary argument was given to support the $1/fan_in$ scaling for the Adam learning rate (LR) for hidden layers. This corresponds to the rightmost entry in the bottom row of Table 3 for "Hidden weights" under "Adam LR", which is $1/fan_in$ for μP .

The table states that the corresponding standard parametrization (SP) for the Adam learning rate of hidden weights is 1 . The SP values are indicated by the gray text in parentheses in the table.

Answer from Table 3: For hidden weights, the Standard Parametrization uses a constant learning rate for Adam that does not scale with fan_in (i.e., it is (1)).

comment:

The answer is correct. The observation about the width of a matrix in 3(a) is accurate, and the answer from Table 3 in 3(b) is also correct.

prompt:

Why do you think μP on the right figure 1 has performance improve monotonically with width and that is not the case on the left?

output:

Of course. This is a central insight of the paper, and the difference stems from the fundamental stability of the training dynamics under μP versus the instability under Standard Parametrization (SP).

Here is a breakdown of why performance improves monotonically with width in μP but not in SP:

1. The Core Problem: Standard Parametrization (SP) is Unstable

In SP, the initialization is designed to keep activations at a stable scale *at initialization* (e.g., using He or LeCun initialization with variance $\sim 1/fan_in$). However, this parametrization does not account for how the *updates* to the weights scale with width.

- **Some Layers Update Too Fast:** As shown in Figure 5 of the paper and explained in Section 5, certain internal quantities like attention logits and layer logits can **blow up** in wide SP networks after just a few training steps. This is because the product of the gradient and the learning rate for these layers results in an update that is too large relative to the network's width.
- **Some Layers Update Too Slow:** Conversely, other layers, like input embeddings, receive updates that are too small and effectively stop learning in very wide models.

This imbalance means that in a wide SP network:

- To prevent the blow-up of fast-updating layers, you are forced to use a very small global learning rate.
- But this small learning rate further cripples the slow-updating layers, which then fail to learn meaningful features.

Result in Figure 1 (Left): For a given learning rate on the x-axis:

- At a small width, the imbalance is manageable, and the model can learn.
- As width increases, the imbalance worsens. The chosen learning rate might be too large for the fast layers (causing instability/high loss) or too small for the slow layers (causing stagnation/high loss).
- This is why the curves cross and wider models can underperform narrower ones. There is no single learning rate that works well for all components of a wide SP network.

2. The Solution: Maximal Update Parametrization (μP) is Stable

μP is specifically designed so that the *effect of a single training step* on all the network's internal activations and outputs is roughly independent of width.

- **Balanced Updates:** As derived in the paper, μP scales the initialization and the per-layer learning rates (for both SGD and Adam) such that every layer's contribution to the network's output is updated by an amount of **order $\Theta(1)$** , regardless of the layer's width.
- **No Blow-up, No Vanishing:** This ensures that no part of the network blows up or stops learning as width increases. All components of the network learn features at a consistent, maximal rate.

Result in Figure 1 (Right): For a given learning rate on the x-axis:

- A wider model is strictly a more powerful function approximator than a narrower one (it has more parameters and representational capacity).
- Since the training dynamics are stable and well-behaved in μP , this increased capacity is harnessed effectively. The wider model can simply do a better job of minimizing the training loss.
- Therefore, for any fixed learning rate, the performance (training loss) **monotonically improves** with width. The curves do not cross; they simply shift downward.

Summary Analogy

- **SP** is like trying to tune a team where one member's amplifier is turned up to 11 (blowing out the speakers) while another member's microphone is off. Making the team bigger (increasing width) without fixing the individual volumes just makes this problem worse. You have to compromise on a global volume (learning rate) that leaves everyone either too loud or too quiet.
- **μP** is like giving each team member their own correctly calibrated volume knob. Now, when you add more members to the team (increasing width), the overall performance can only improve because each new member is perfectly integrated and contributes effectively.

3(c)

prompt:

solve the problem 3(c) from hw3, based on the paper

output:

Based on the paper "A Spectral Condition for Feature Learning," here is the solution to problem 3(c):

Why Desideratum 1 behavior is desired for feature learning

Desideratum 1 states:

$$\|\mathbf{h}_\ell\|_2 = \Theta(\sqrt{n_\ell}) \quad \text{and} \quad \|\Delta \mathbf{h}_\ell\|_2 = \Theta(\sqrt{n_\ell})$$

This behavior is desired because:

- **Non-vanishing feature evolution:** If $\|\Delta \mathbf{h}_\ell\|_2$ were smaller than $\Theta(\sqrt{n_\ell})$, feature updates would vanish as width increases, preventing meaningful learning
- **Non-exploding features:** If $\|\Delta \mathbf{h}_\ell\|_2$ were larger than $\Theta(\sqrt{n_\ell})$, features would blow up during training, making optimization unstable

- **Proper activation scaling:** Since activation functions are designed to take order-one inputs and produce order-one outputs, having $\|\mathbf{h}_\ell\|_2 = \Theta(\sqrt{n_\ell})$ ensures that individual elements of \mathbf{h}_ℓ are $\Theta(1)$, which is the expected input range for most activation functions

Desideratum 1 in terms of RMS norm

The RMS (root-mean-square) norm is defined as:

$$\|\mathbf{v}\|_{\text{RMS}} = \frac{1}{\sqrt{n}} \|\mathbf{v}\|_2$$

Rewriting Desideratum 1 in RMS norm:

$$\|\mathbf{h}_\ell\|_{\text{RMS}} = \Theta(1) \quad \text{and} \quad \|\Delta \mathbf{h}_\ell\|_{\text{RMS}} = \Theta(1)$$

This formulation is more natural because it directly expresses that the "typical element size" of both features and their updates is $\Theta(1)$, independent of layer width.

Condition 1 in terms of RMS norm

Condition 1 states:

$$\|\mathbf{W}_\ell\|_* = \Theta\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right) \quad \text{and} \quad \|\Delta \mathbf{W}_\ell\|_* = \Theta\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right)$$

In terms of RMS norm (using the natural spectral norm from Appendix E):

$$\|\mathbf{W}_\ell\|_z = \Theta(1) \quad \text{and} \quad \|\Delta \mathbf{W}_\ell\|_z = \Theta(1)$$

where $\|\mathbf{A}\|_z = \frac{\sqrt{n_\ell}}{\sqrt{n_{\ell-1}}} \|\mathbf{A}\|_*$ is the natural spectral norm.

How Condition 1 implies the upper bounds

From the submultiplicativity property of the spectral norm:

$$\|\mathbf{h}_\ell(\mathbf{x})\|_2 \leq \|\mathbf{W}_\ell\|_* \cdot \|\mathbf{h}_{\ell-1}(\mathbf{x})\|_2$$

Assuming $\|\mathbf{h}_{\ell-1}(\mathbf{x})\|_2 = \Theta(\sqrt{n_{\ell-1}})$ (by recursive application) and $\|\mathbf{W}_\ell\|_* = \Theta(\sqrt{n_\ell/n_{\ell-1}})$ (from Condition 1), we get:

$$\|\mathbf{h}_\ell(\mathbf{x})\|_2 \leq \Theta(\sqrt{n_\ell/n_{\ell-1}}) \cdot \Theta(\sqrt{n_{\ell-1}}) = \Theta(\sqrt{n_\ell})$$

Similarly, for the feature updates:

$$\|\Delta \mathbf{h}_\ell(\mathbf{x})\|_2 \leq \|\Delta \mathbf{W}_\ell\|_* \cdot \|\mathbf{h}_{\ell-1}(\mathbf{x})\|_2 + \|\mathbf{W}_\ell\|_* \cdot \|\Delta \mathbf{h}_{\ell-1}(\mathbf{x})\|_2 + \|\Delta \mathbf{W}_\ell\|_* \cdot \|\Delta \mathbf{h}_{\ell-1}(\mathbf{x})\|_2$$

With all terms satisfying the appropriate scaling conditions, this gives:

$$\|\Delta \mathbf{h}_\ell(\mathbf{x})\|_2 \leq \Theta(\sqrt{n_\ell})$$

Key assumption for the lower bounds

The key assumption that allows us to also get the **lower bounds** on $\|\mathbf{h}_\ell(\mathbf{x})\|_2$ and $\|\Delta \mathbf{h}_\ell(\mathbf{x})\|_2$ is:

Matrix-vector alignment: During neural network training, the layer inputs $\mathbf{h}_{\ell-1}(\mathbf{x})$ align with the top singular subspaces of both the weight matrices \mathbf{W}_ℓ and their updates $\Delta \mathbf{W}_\ell$.

Specifically:

- **Claim 1:** For random initial weight matrices, $\|\mathbf{W}_\ell \mathbf{h}_{\ell-1}(\mathbf{x})\|_2 = \Theta(\|\mathbf{W}_\ell\|_* \cdot \|\mathbf{h}_{\ell-1}(\mathbf{x})\|_2)$
- **Claim 2:** For gradient updates with batch size 1, $\|\Delta \mathbf{W}_\ell \mathbf{h}_{\ell-1}(\mathbf{x})\|_2 = \|\Delta \mathbf{W}_\ell\|_* \cdot \|\mathbf{h}_{\ell-1}(\mathbf{x})\|_2$ (exact equality)

This alignment ensures that the upper bounds provided by submultiplicativity are tight, giving us the matching lower bounds needed to conclude that $\|\mathbf{h}_\ell(\mathbf{x})\|_2 = \Theta(\sqrt{n_\ell})$ and $\|\Delta \mathbf{h}_\ell(\mathbf{x})\|_2 = \Theta(\sqrt{n_\ell})$.

solve problem 4.

link <https://chat.deepseek.com/share/wv18jjyyfd012xbtp3>

4(a)

prompt:

solve problem 4(a) from hw3.

output:

Looking at problem 4(a), we need to find the gradient $\nabla_\mu \mathcal{F}(\mu)$ for:

$$\mathcal{F}(\mu) = \mathbb{E}_{\mathbf{x} \sim p_\mu} [f(\mathbf{x})]$$

where

$$p_\mu(\mathbf{x}) = (2\pi\sigma^2)^{-n/2} \exp\left(-\frac{\|\mathbf{x} - \mu\|_2^2}{2\sigma^2}\right)$$

and σ is a constant.

Step 1: Write the expectation explicitly

$$\mathcal{F}(\mu) = \int p_\mu(\mathbf{x}) f(\mathbf{x}) d\mathbf{x}$$

Step 2: Differentiate under the integral sign

$$\nabla_\mu \mathcal{F}(\mu) = \int \nabla_\mu p_\mu(\mathbf{x}) f(\mathbf{x}) d\mathbf{x}$$

Step 3: Use the log-derivative trick

$$\nabla_\mu p_\mu(\mathbf{x}) = p_\mu(\mathbf{x}) \nabla_\mu \log p_\mu(\mathbf{x})$$

So:

$$\nabla_\mu \mathcal{F}(\mu) = \int p_\mu(\mathbf{x}) [\nabla_\mu \log p_\mu(\mathbf{x})] f(\mathbf{x}) d\mathbf{x}$$

$$= \mathbb{E}_{\mathbf{x} \sim p_\mu} [f(\mathbf{x}) \nabla_\mu \log p_\mu(\mathbf{x})]$$

Step 4: Compute $\nabla_{\mu} \log p_{\mu}(\mathbf{x})$

$$\log p_{\mu}(\mathbf{x}) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{\|\mathbf{x} - \mu\|_2^2}{2\sigma^2}$$

$$\nabla_{\mu} \log p_{\mu}(\mathbf{x}) = \nabla_{\mu} \left(-\frac{\|\mathbf{x} - \mu\|_2^2}{2\sigma^2} \right)$$

$$= \frac{\mathbf{x} - \mu}{\sigma^2}$$

Step 5: Final expression

$$\nabla_{\mu} \mathcal{F}(\mu) = \mathbb{E}_{\mathbf{x} \sim p_{\mu}} \left[f(\mathbf{x}) \cdot \frac{\mathbf{x} - \mu}{\sigma^2} \right]$$

So $g(\mathbf{x}) = f(\mathbf{x}) \cdot \frac{\mathbf{x} - \mu}{\sigma^2}$.

Final answer:

$$\boxed{\nabla_{\mu} \mathcal{F}(\mu) = \mathbb{E}_{\mathbf{x} \sim p_{\mu}} \left[f(\mathbf{x}) \cdot \frac{\mathbf{x} - \mu}{\sigma^2} \right]}$$

This is the **policy gradient** (REINFORCE) estimator for the Gaussian mean parameter.

comment:

The solution is correct and clearly explained step-by-step.

4(b)

prompt:

solve problem 4(b) from hw3.

output:

Looking at problem 4(b), we need to rewrite $\mathcal{F}(\mu)$ using a standard normal distribution.

Step 1: Recall the given hint

If $\mathbf{x} \sim p_{\mu}(\mathbf{x})$, then $\mathbf{x} - \mu \sim \mathcal{N}(0, \sigma^2 \mathbf{I})$.

Let $\mathbf{z} = \frac{\mathbf{x} - \mu}{\sigma}$, then $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$.

Step 2: Change of variables

We have $\mathbf{x} = \mu + \sigma \mathbf{z}$ where $\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})$.

The expectation becomes:

$$\mathcal{F}(\mu) = \mathbb{E}_{\mathbf{x} \sim p_\mu} [f(\mathbf{x})] = \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})} [f(\mu + \sigma \mathbf{z})]$$

Step 3: Verify the transformation

Since \mathbf{z} follows a standard normal distribution $\mathcal{N}(0, \mathbf{I})$, and $\mathbf{x} = \mu + \sigma \mathbf{z}$ is an affine transformation of a Gaussian, this preserves the distribution $p_\mu(\mathbf{x})$.

The expectation no longer depends on μ in the distribution, only inside the function f .

Final answer:

$$\boxed{\mathcal{F}(\mu) = \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})} [f(\mu + \sigma \mathbf{z})]}$$

comment:

The solution is correct and clearly explained step-by-step.

4(c)

prompt:

solve problem 4(c) from hw3.

output:

Looking at problem 4(c), we need to find $\nabla_\mu \mathcal{F}(\mu)$ using the expression from part (b).

Step 1: Start with the result from part (b)

$$\mathcal{F}(\mu) = \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})} [f(\mu + \sigma \mathbf{z})]$$

Step 2: Move gradient inside expectation

As hinted: " ∇_μ can be safely moved inside the expectation because the expectation no longer follows a distribution that depends on μ and expectations are linear."

The distribution of \mathbf{z} is $\mathcal{N}(0, \mathbf{I})$, which does **not** depend on μ , so:

$$\nabla_\mu \mathcal{F}(\mu) = \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})} [\nabla_\mu f(\mu + \sigma \mathbf{z})]$$

Step 3: Apply chain rule

Let $\mathbf{x} = \mu + \sigma \mathbf{z}$, then:

$$\nabla_\mu f(\mu + \sigma \mathbf{z}) = \nabla_{\mathbf{x}} f(\mathbf{x}) \cdot \frac{\partial \mathbf{x}}{\partial \mu}$$

Since $\frac{\partial \mathbf{x}}{\partial \mu} = \mathbf{I}$ (the identity matrix), we get:

$$\nabla_{\mu} f(\mu + \sigma \mathbf{z}) = \nabla_{\mathbf{x}} f(\mathbf{x})$$

where $\mathbf{x} = \mu + \sigma \mathbf{z}$.

Step 4: Final expression

$$\nabla_{\mu} \mathcal{F}(\mu) = \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})} \left[\nabla_{\mathbf{x}} f(\mathbf{x}) \Big|_{\mathbf{x}=\mu+\sigma \mathbf{z}} \right]$$

Final answer:

$$\nabla_{\mu} \mathcal{F}(\mu) = \mathbb{E}_{\mathbf{z} \sim \mathcal{N}(0, \mathbf{I})} \left[\nabla_{\mathbf{x}} f(\mathbf{x}) \Big|_{\mathbf{x}=\mu+\sigma \mathbf{z}} \right]$$

This is the **reparameterization gradient** estimator, which typically has lower variance than the policy gradient estimator from part (a).

comment:

The solution is totally correct and clearly explained step-by-step.

4(d)

prompt:

solve problem 4(d) from hw3.

output:

Looking at problem 4(d), we need to find $\nabla_{\theta} \mathcal{F}(\theta)$ for a general distribution $p_{\theta}(\mathbf{x})$.

Step 1: Write the expectation explicitly

$$\mathcal{F}(\theta) = \mathbb{E}_{\mathbf{x} \sim p_{\theta}}[f(\mathbf{x})] = \int p_{\theta}(\mathbf{x}) f(\mathbf{x}) d\mathbf{x}$$

Step 2: Differentiate under the integral sign

$$\nabla_{\theta} \mathcal{F}(\theta) = \int \nabla_{\theta} p_{\theta}(\mathbf{x}) f(\mathbf{x}) d\mathbf{x}$$

Step 3: Use the log-derivative trick

As hinted: "look at your answer in Part (a) and see how the term inside your expectation relates to $\log p_{\theta}(\mathbf{x})$ "

We have:

$$\nabla_{\theta} p_{\theta}(\mathbf{x}) = p_{\theta}(\mathbf{x}) \nabla_{\theta} \log p_{\theta}(\mathbf{x})$$

Substituting:

$$\nabla_{\theta} \mathcal{F}(\theta) = \int p_{\theta}(\mathbf{x}) [\nabla_{\theta} \log p_{\theta}(\mathbf{x})] f(\mathbf{x}) d\mathbf{x}$$

Step 4: Write as expectation

$$\nabla_{\theta} \mathcal{F}(\theta) = \mathbb{E}_{\mathbf{x} \sim p_{\theta}} [f(\mathbf{x}) \nabla_{\theta} \log p_{\theta}(\mathbf{x})]$$

Step 5: Compare with part (a)

In part (a) for the Gaussian case, we found:

$$\nabla_{\mu} \mathcal{F}(\mu) = \mathbb{E}_{\mathbf{x} \sim p_{\mu}} \left[f(\mathbf{x}) \cdot \frac{\mathbf{x} - \mu}{\sigma^2} \right]$$

and indeed $\nabla_{\mu} \log p_{\mu}(\mathbf{x}) = \frac{\mathbf{x} - \mu}{\sigma^2}$, so this matches the general form.

Final answer:

$$\nabla_{\theta} \mathcal{F}(\theta) = \mathbb{E}_{\mathbf{x} \sim p_{\theta}} [f(\mathbf{x}) \nabla_{\theta} \log p_{\theta}(\mathbf{x})]$$

This is the general **score function estimator** or **REINFORCE estimator**, valid for any differentiable parameterized distribution p_{θ} .

comment:

The solution is totally correct and clearly explained step-by-step.

4(e)

prompt:

solve problem 4(e) from hw3.

output:

Looking at problem 4(e), we need to prove the reparameterization gradient formula for a general transformation.

Step 1: Setup

We have:

- $g(\mathbf{z}, \theta) : \mathcal{Z} \times \mathbb{R}^P \rightarrow \mathcal{X}$
- $\mathbf{z} \sim p(\mathbf{z})$ (fixed distribution not depending on θ)
- $g(\mathbf{z}, \theta)$ has the same distribution as $\mathbf{x} \sim p_{\theta}(\mathbf{x})$

Therefore:

$$\mathcal{F}(\theta) = \mathbb{E}_{\mathbf{x} \sim p_{\theta}} [f(\mathbf{x})] = \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} [f(g(\mathbf{z}, \theta))]$$

Step 2: Differentiate

Since the distribution $p(\mathbf{z})$ doesn't depend on θ , we can move the gradient inside the expectation:

$$\nabla_{\theta} \mathcal{F}(\theta) = \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} [\nabla_{\theta} f(g(\mathbf{z}, \theta))]$$

Step 3: Apply chain rule

Using the chain rule:

$$\nabla_{\theta} f(g(\mathbf{z}, \theta)) = \nabla_{\theta} g(\mathbf{z}, \theta)^{\top} \nabla_{\mathbf{x}} f(\mathbf{x}) \Big|_{\mathbf{x}=g(\mathbf{z}, \theta)}$$

Where:

- $\nabla_{\theta} g(\mathbf{z}, \theta)$ is the Jacobian of g with respect to θ (size $P \times \dim(\mathcal{X})$)
- $\nabla_{\mathbf{x}} f(\mathbf{x})$ is the gradient of f with respect to \mathbf{x} (column vector)

The transpose and multiplication give us the proper chain rule.

Step 4: Final expression

$$\nabla_{\theta} \mathcal{F}(\theta) = \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \left[\nabla_{\theta} g(\mathbf{z}, \theta)^{\top} \nabla_{\mathbf{x}} f(\mathbf{x}) \Big|_{\mathbf{x}=g(\mathbf{z}, \theta)} \right]$$

Step 5: Special case from parts (b) and (c)

In parts (b) and (c), for the Gaussian case:

- $g(\mathbf{z}, \mu) = \mu + \sigma \mathbf{z}$
- $p(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$
- $\nabla_{\mu} g(\mathbf{z}, \mu) = \mathbf{I}$
- So $\nabla_{\mu} g(\mathbf{z}, \mu)^{\top} \nabla_{\mathbf{x}} f(\mathbf{x}) = \nabla_{\mathbf{x}} f(\mathbf{x})$

Which matches our result from part (c).

Final answer:

$$\nabla_{\theta} \mathcal{F}(\theta) = \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \left[\nabla_{\theta} g(\mathbf{z}, \theta)^{\top} \nabla_{\mathbf{x}} f(\mathbf{x}) \Big|_{\mathbf{x}=g(\mathbf{z}, \theta)} \right]$$

For the Gaussian special case: $g(\mathbf{z}, \mu) = \mu + \sigma \mathbf{z}$ and $p(\mathbf{z}) = \mathcal{N}(0, \mathbf{I})$.

solve problem 5.

5(a)

link <https://chat.deepseek.com/share/3cr1cpwjaisjhai4t>

prompt:

solve problem 5(a) from hw3.

output:

Let's break down **Problem 5(a)** step by step.

Step 1: Understanding the setup

We have:

- A **10-layer** network.
- Each layer has the same input/output size s .
- Memory can store: inputs, **2s** activations, optimizer states for the current layer.
- **Tensor rematerialization** strategy:
 - Store activations only every **5th layer** (i.e., Layer 5 and Layer 10) during initial forward pass.
 - During backpropagation:
 - To update layer L , load the last stored activation before L (e.g., Layer 5 for layers 6–9, Layer 0 input for layers 1–4), and recompute forward from there up to layer L .

Step 2: Counting forward (fwd) operations during backward pass

We ignore the initial forward pass for computing the loss (already done before backprop).

We only count **recomputation forward passes** during backpropagation.

Let's go layer by layer in **backward order** (layer 10 down to layer 1):

- **Layer 10:**
No recomputation needed — stored activation already available from initial forward pass.
→ **0 fwd.**
- **Layer 9:**
Need activation at layer 9 to compute gradient for layer 9.
Last stored activation before layer 9 is at **layer 5**.
So: recompute layers 6, 7, 8, 9.
→ **4 fwd.**
- **Layer 8:**
Last stored activation before layer 8 is at **layer 5**.
Recompute layers 6, 7, 8.
→ **3 fwd.**
- **Layer 7:**
Last stored activation before layer 7 is at **layer 5**.
Recompute layers 6, 7.
→ **2 fwd.**
- **Layer 6:**
Last stored activation before layer 6 is at **layer 5**.
Recompute layer 6.
→ **1 fwd.**
- **Layer 5:**
Stored activation available.
→ **0 fwd.**
- **Layer 4:**
Last stored activation before layer 4 is at **layer 0 (input)**.
Recompute layers 1, 2, 3, 4.
→ **4 fwd.**

- **Layer 3:**

Last stored activation before layer 3 is at **layer 0**.

Recompute layers 1, 2, 3.

→ **3 fwd.**

- **Layer 2:**

Last stored activation before layer 2 is at **layer 0**.

Recompute layers 1, 2.

→ **2 fwd.**

- **Layer 1:**

Last stored activation before layer 1 is at **layer 0**.

Recompute layer 1.

→ **1 fwd.**

Step 3: Summing fwd operations

$$4 + 3 + 2 + 1 + 0 + 4 + 3 + 2 + 1 + 0$$

Group:

Layers 9–6: $4 + 3 + 2 + 1 = 10$

Layers 4–1: $4 + 3 + 2 + 1 = 10$

Total = $10 + 10 = 20$ fwd operations.

Step 4: Final answer

20

comment:

The solution is totally correct and clearly explained step-by-step.

5(b)

prompt:

solve problem 5(b) from hw3.

output:

Let's break down **Problem 5(b)** step by step.

Step 1: Understanding the setup

We have the same **10-layer network** and **tensor rematerialization** strategy as in part (a):

- Stored activations: **Layer 5** and **Layer 10** (and the input at Layer 0) are stored in memory during the initial forward pass.
- During backpropagation, to compute gradients for layer L , we **load** the last stored activation before L and recompute forward up to layer L .

A **loadmem** operation is counted for each time we fetch stored activations or inputs from memory.

Step 2: Counting loadmem operations during backward pass

We go backward from layer 10 to layer 1:

- **Layer 10:**

Activation already in memory (Layer 10 stored).

But to compute gradient for layer 10, we need **activation of layer 10** (already there) and **upstream gradient**.

Actually, the stored activation of layer 10 is used directly — no loadmem for recomputation here.

But we also need **activation of layer 9**? No — for layer 10's gradient, we need input to layer 10 = activation of layer 9.

Wait — careful: The stored activation at layer 10 is the output of layer 10, but for computing ∇W_{10} we need **input to layer 10** (which is output of layer 9).

So even for layer 10, we must recompute layer 9's output? Let's check the diagram logic.

Actually, in standard backprop, to compute $\frac{\partial L}{\partial W_\ell}$ we need $a_{\ell-1}$ (input to layer ℓ) and δ_ℓ (error signal from layer ℓ output).

The stored activation at layer 5 and 10 are the **outputs** of those layers, i.e., a_5, a_{10} .

So for **layer 10** gradient:

We need a_9 and δ_{10} .

We don't have a_9 stored, so we must recompute from last stored before layer 9 = layer 5.

So actually **all layers** except the stored ones require loading some stored activation and recomputing. Even layer 10's weight gradient requires a_9 , so we must recompute layers 6–9.

But the problem's Figure 1 example says: for layer 7's update, load layer 5 activation, run layer 6 and 7. So indeed **every layer** during backprop requires loading a stored activation **once** per layer's gradient computation.

Let's list systematically:

Backward pass layer by layer:

1. **Layer 10:**

Need a_9 . Load last stored before 9 = layer 5.

→ **1 loadmem** (load layer 5 activation).

2. **Layer 9:**

Need a_8 . Load last stored before 8 = layer 5.

→ **1 loadmem**.

3. **Layer 8:**

Need a_7 . Load last stored before 7 = layer 5.

→ **1 loadmem**.

4. **Layer 7:**

Need a_6 . Load last stored before 6 = layer 5.

→ **1 loadmem**.

5. **Layer 6:**

Need a_5 . Load last stored before 6 = layer 5.

→ **1 loadmem**.

6. **Layer 5:**

Need a_4 . Load last stored before 4 = layer 0 (input).

→ **1 loadmem**.

7. **Layer 4:**

Need a_3 . Load last stored before 3 = layer 0.

→ **1 loadmem**.

8. **Layer 3:**

Need a_2 . Load last stored before 2 = layer 0.

→ **1 loadmem.**

9. **Layer 2:**

Need a_1 . Load last stored before 1 = layer 0.

→ **1 loadmem.**

10. **Layer 1:**

Need a_0 (input). Load input from memory.

→ **1 loadmem.**

Step 3: Total loadmem operations

Each of the 10 layers requires exactly **1 loadmem** to fetch the nearest stored activation (either layer 5, layer 0 input, or layer 10? Wait, did we use layer 10 stored activation at all? Actually no — because for layer 10’s gradient we need a_9 , so we load layer 5, not layer 10. The stored layer 10 activation is only for computing loss initially, not for gradient w.r.t. W_{10} ? That seems odd — maybe they store layer 10 activation for the loss gradient computation, but for ∇W_{10} we still need a_9 . So indeed stored layer 10 activation is useless for backprop in this scheme? Possibly — but the problem says “Layer 5 and Layer 10 activations are stored in memory along with the inputs.” So maybe for layer 10’s gradient, they use stored a_{10} for δ_{10} and recompute a_9 for a_{in} . But a_{10} is the output, not input. So indeed stored a_{10} not needed for ∇W_{10} .

Thus: **Every layer** needs 1 loadmem.

Total loadmem = 10.

Step 4: Final answer

10

comment:

The solution is totally correct and clearly explained step-by-step.

5(c)

prompt:

solve problem 5(c) from hw3.

output:

Let’s go step by step for **Problem 5(c)**.

Step 1: Understanding the setup

We have two scenarios:

1. **Tensor rematerialization** (baseline):
 - Forward pass: store activations for layers 5 and 10 (and input).
 - Backward pass: recompute activations as needed, loading stored ones from memory.
 - From part (a): **20 fwd** operations during backward pass.

- From part (b): **10 loadmem** operations during backward pass.
- Times:
 - **fwd**: 20 ns each
 - **loadmem**: 10 ns each

2. **Store all activations on disk** (alternative):

- Forward pass: store all activations for layers 1–10 on disk.
- Backward pass: load each needed activation directly from disk to registers (bypassing memory).
- No recomputation forward passes.
- **fwd** operations during backward pass = 0.
- **loaddisk** operations:
 - For each layer ℓ during backprop, we need activation $a_{\ell-1}$.
 - So for layers 1 to 10, we load a_0, a_1, \dots, a_9 from disk.
 - That's **10 loaddisk** operations.

Step 2: Compute time for tensor rematerialization

From parts (a) and (b):

$$T_{\text{remat}} = 20 \times (\text{fwd time}) + 10 \times (\text{loadmem time})$$

$$T_{\text{remat}} = 20 \times 20 \text{ ns} + 10 \times 10 \text{ ns}$$

$$T_{\text{remat}} = 400 \text{ ns} + 100 \text{ ns} = 500 \text{ ns}$$

Step 3: Compute time for disk storage method

Let t_d = time per loaddisk operation (ns).

We have 10 loaddisk operations and 0 fwd during backward pass:

$$T_{\text{disk}} = 10 \times t_d$$

We want $T_{\text{disk}} = T_{\text{remat}}$:

$$10 \times t_d = 500$$

$$t_d = 50 \text{ ns}$$

Step 4: Final answer