

Annotated log of conversation with Mistral AI's Le Chat interface for solving HW3

Prompt 1:

PDF hw3.pdf

This is the homework of deep learning I am currently working on. Help me solve question 1

Le Chat's answer:



Let's break down **Question 1: Maximal Update Parameterization** into its three parts and solve each step-by-step.

1(a) Expected Squared RMS Norm of Output Features

Setup

- Input: $\mathbf{x} \in \mathbb{R}^{d_1}$, entries i.i.d. $\mathcal{N}(0, 1)$
- Weights: $W \in \mathbb{R}^{d_2 \times d_1}$, entries i.i.d. $\mathcal{N}(0, 1)$
- Output: $\mathbf{y} = W\mathbf{x}$

Goal

Find the expected squared RMS norm of \mathbf{y} :

$$\mathbb{E} [\|\mathbf{y}\|_{\text{RMS}}^2] = \mathbb{E} \left[\frac{1}{d_2} \|W\mathbf{x}\|_2^2 \right]$$

Solution

1. **Simplified Case (Single Output Feature):** If $W \in \mathbb{R}^{1 \times d_1}$, then $\mathbf{y} = W\mathbf{x}$ is a scalar. Since W and \mathbf{x} are independent Gaussians:

$$\mathbf{y} = \sum_{i=1}^{d_1} W_i x_i$$

Each $W_i x_i$ is i.i.d. $\mathcal{N}(0, 1)$, so $\mathbf{y} \sim \mathcal{N}(0, d_1)$.

Thus, $\mathbb{E}[\mathbf{y}^2] = d_1$.

2. **General Case:** For $W \in \mathbb{R}^{d_2 \times d_1}$, each output feature is independent and identically distributed as above.

$$\mathbb{E} [\|W\mathbf{x}\|_2^2] = \sum_{j=1}^{d_2} \mathbb{E}[(W_j \mathbf{x})^2] = d_2 \cdot d_1$$

So,

$$\mathbb{E} [\|\mathbf{y}\|_{\text{RMS}}^2] = \frac{1}{d_2} \mathbb{E} [\|W\mathbf{x}\|_2^2] = d_1$$

3. **Scaling for Unit RMS Norm:** To ensure $\mathbb{E} [\|\mathbf{y}\|_{\text{RMS}}^2] = 1$, multiply W by $\frac{1}{\sqrt{d_1}}$.

1(b) Expected RMS Norm of Feature Updates (SignGD)

Setup

- Update rule:

$$W_{t+1} = W_t + \eta \cdot \text{sign}(\mathbf{x}_i \mathbf{g}_i^T)$$

- $\mathbf{x}_i \sim \mathcal{N}(0, I)$, \mathbf{g}_i is a constant vector.
- Feature update: $\mathbf{y}' = \text{sign}(\mathbf{x}_i \mathbf{g}_i^T) \mathbf{x}_i$

Goal

Find the expected squared RMS norm of \mathbf{y}' :

$$\mathbb{E} [\|\mathbf{y}'\|_{\text{RMS}}^2] = \mathbb{E} \left[\frac{1}{d_2} \|\text{sign}(\mathbf{x}_i \mathbf{g}_i^T) \mathbf{x}_i\|_2^2 \right]$$

Solution

1. **Simplify $\text{sign}(\mathbf{x}_i \mathbf{g}_i^T)$:**

Since \mathbf{x}_i is Gaussian, $\text{sign}(\mathbf{x}_i \mathbf{g}_i^T)$ is a random sign matrix, independent of \mathbf{x}_i .

2. **Expected Norm:**

$$\mathbb{E} [\|\text{sign}(\mathbf{x}_i \mathbf{g}_i^T) \mathbf{x}_i\|_2^2] = \mathbb{E} \left[\sum_{j=1}^{d_2} \left(\sum_{k=1}^{d_1} \text{sign}((\mathbf{x}_i \mathbf{g}_i^T)_{jk}) x_{ik} \right)^2 \right]$$

Each term is a sum of d_1 i.i.d. random variables, each with variance 1 (since $\text{sign}(\cdot)$ and x_{ik} are independent).

$$\mathbb{E} \left[\left(\sum_{k=1}^{d_1} \text{sign}(\cdot) x_{ik} \right)^2 \right] = d_1$$

So,

$$\mathbb{E} [\|\text{sign}(\mathbf{x}_i \mathbf{g}_i^T) \mathbf{x}_i\|_2^2] = d_2 \cdot d_1$$

$$\mathbb{E} [\|\mathbf{y}'\|_{\text{RMS}}^2] = \frac{1}{d_2} \cdot d_2 \cdot d_1 = d_1$$

3. **Scaling for Unit RMS Norm:** Multiply the update by $\frac{1}{\sqrt{d_1}}$.

1(c) Why Only d_1 ?

Reasoning

- The update $\text{sign}(\mathbf{x}_i \mathbf{g}_i^T) \mathbf{x}_i$ depends on the outer product of \mathbf{x}_i and \mathbf{g}_i .
- The norm of this update depends on the norm of \mathbf{x}_i , which scales with d_1 (the input dimension).
- The output dimension d_2 does not affect the norm of the update, as the sign operation and the outer product are applied per output feature independently.

Summary Table

Summary of Results		
Part	Expected Squared RMS Norm	Scaling Factor
1(a)	d_1	$\frac{1}{\sqrt{d_1}}$
1(b)	d_1	$\frac{1}{\sqrt{d_1}}$

Follow-up Questions

- For 1(a), the key was recognizing the variance of the output features.
- For 1(b), the sign operation preserves the scaling with d_1 .
- For 1(c), the update only depends on the input dimension, not the output dimension.

Comment 1: For question 1, Le Chat was able to arrive at the correct answer one shot.

Prompt 3.1:

It's time to move onto question 3. Check the link in Q3 and help me answer question a

Le Chat's answer:



Worked for 4s



To answer **Question 3(a)** from your homework, we need to refer to **Figure 1** of the paper "Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer."

Based on the search results and the context provided by the paper, here's what Figure 1 is about:

Figure 1 in "Tensor Programs V"

What the Figure Shows:

- Figure 1 typically consists of two side-by-side plots.
- These plots compare the **spectral norm** (or another relevant norm) of weight matrices (or updates) **before and after applying Maximal Update Parameterization (μP)**.
- The key observation is that **without μP** , the norms of the matrices (or updates) grow with the width of the layer, while **with μP** , the norms remain stable regardless of the layer width.

Observation About Width:

- The figure demonstrates that **μP stabilizes the norms** of weight matrices or updates across different layer widths.
- This stability is crucial for **hyperparameter transfer**, as it allows the same hyperparameters (e.g., learning rate) to work well across different network sizes.

How to Answer the Question

Answer to Question 3(a):

""Figure 1 in the paper shows two side-by-side plots comparing the norms (e.g., spectral norm) of weight matrices or updates for neural network layers of varying widths. The left plot (without μP) shows that the norms increase with the width of the layer, while the right plot (with μP) shows that the norms remain stable regardless of the layer width. This demonstrates that μP ensures the norms of weight matrices or updates are invariant to the width of the network, which is essential for effective hyperparameter transfer.""

References

- The paper discusses how μP stabilizes the norms of weight matrices and updates, enabling zero-shot hyperparameter transfer across different network sizes. This is a key contribution of the μP framework, as it allows hyperparameters tuned on smaller models to be directly applied to larger models without further tuning [arxiv.org +2](https://arxiv.org/+2).

Comment 3.1: Le Chat seems to mistake the y-axis of table 1 in the mP paper by the different norms. This is likely due to its context learning of the norms in Q1.

Prompt 3.2:

but the y-axis of the table says "training loss" instead of norm

Le Chat's answer:

 Thank you for the clarification. Based on your description, **Figure 1** in the paper likely shows **training loss** as a function of network width or another variable, comparing standard parameterization and Maximal Update Parameterization (μP).

Interpretation of Figure 1 (Training Loss vs. Width)

What the Figure Shows:

- **Two side-by-side plots:**
 - **Left plot:** Training loss for networks with standard parameterization as width increases.
 - **Right plot:** Training loss for networks with μP as width increases.

Observation About Width:

- **Without μP (Left Plot):**
 - As the width of the network increases, the training loss may become unstable or increase. This is because the dynamics of the network change with width, making it harder to optimize effectively with a fixed set of hyperparameters.
- **With μP (Right Plot):**
 - With μP , the training loss remains stable or improves consistently as the width increases. μP ensures that the network's training dynamics are invariant to width, allowing for effective optimization regardless of the network size.

How to Answer the Question

Answer to Question 3(a):

""Figure 1 in the paper shows two side-by-side plots of training loss as a function of network width. The left plot, which uses standard parameterization, shows that training loss becomes unstable or increases as the width of the network increases. The right plot, which uses Maximal Update Parameterization (μP), demonstrates that training loss remains stable or improves consistently as the width increases. This indicates that μP stabilizes the training dynamics of the network, making it invariant to changes in width.""

Why This Matters

- **Stable Training Dynamics:**

μP ensures that the training loss does not degrade as the network width increases, which is crucial for scaling up neural networks effectively.

- **Hyperparameter Transfer:**

By stabilizing the training dynamics, μP allows hyperparameters (such as learning rate) tuned on smaller networks to be directly applied to larger networks without further tuning, enabling zero-shot hyperparameter transfer.

Comment 3.2: After insisting the correct y-axis is “training loss” instead of norms, Le Chat was able to recorrect its reasoning to arrive at the correct answer.

Prompt 3.3:

Now help me answer Q3b

Le Chat's answer:



Let's address **Question 3(b)** from your homework, which refers to **Table 3** in the paper "Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer."

Interpretation of Table 3

What Table 3 Shows:

- **Table 3** typically compares the **learning rate scaling rules** for different parameterizations, including **standard parameterization** and **Maximal Update Parameterization (μP)**.
- The table often includes a row for **Adam learning rate scaling** for hidden layers.
- In the context of μP , the rightmost entry in the bottom row (for Adam learning rate scaling for hidden layers) is usually $1/\sqrt{\text{fan}_{\text{in}}}$.

Corresponding Standard Parameterization:

- The **standard parameterization** for the same entry (Adam learning rate scaling for hidden layers) is typically $1/\text{fan}_{\text{in}}$ (without the square root).
- This means that in standard parameterization, the learning rate is scaled by the inverse of the input dimension, while in μP , it is scaled by the inverse square root of the input dimension.

How to Answer the Question

Answer to Question 3(b):

""In Table 3 of the paper, the rightmost entry in the bottom row for Adam learning rate scaling for hidden layers is $1/\sqrt{\text{fan}_{\text{in}}}$ under Maximal Update Parameterization (μP). The corresponding standard parameterization for this entry is $1/\text{fan}_{\text{in}}$. This indicates that μP uses a square root scaling factor, which helps stabilize the training dynamics across different network widths.""

Why This Matters

- **Stabilizing Learning Rates:**

The $1/\sqrt{\text{fan}_{\text{in}}}$ scaling in μP ensures that the learning rate is appropriately adjusted for the input dimension, leading to more stable training dynamics as the network width changes.

- **Hyperparameter Transfer:**

This scaling is a key reason why μP allows for zero-shot hyperparameter transfer, as it makes the network's behavior invariant to width, enabling the same hyperparameters to work well across different network sizes.

Comment 3.3: Le Chat was able to arrive at the correct answer one shot.

Prompt 3.4:

Now let's move onto Q3c)

Le Chat's answer:

 Let's address **Question 3(c)** from your homework, which refers to **Desideratum 1** and **Condition 1** in the paper "A Spectral Condition for Feature Learning."

Desideratum 1

What Desideratum 1 States:

- **Desideratum 1** typically states that for effective feature learning, the **norm of the activations** (or features) at each layer should be **stable** and **not grow or shrink exponentially** with depth or width.
- In other words, the RMS norm of the activations $\|h_\ell(x)\|_2$ should be roughly constant across layers and widths.

Rewriting in Terms of RMS Norm:

- Desideratum 1 can be rewritten as:

"The RMS norm of the activations at each layer, $\|h_\ell(x)\|_{\text{RMS}}$, should be $\Theta(1)$ (i.e., bounded by a constant) regardless of the layer index ℓ or the width of the layer."

Condition 1

What Condition 1 States:

- **Condition 1** usually provides a sufficient condition for Desideratum 1 to hold.
- It states that the **spectral norm** of the weight matrices should be bounded, and the **norm of the updates** should also be controlled.
- Specifically, it often requires that the spectral norm of the weight matrices is $O(1)$ and that the norm of the updates is $O(1/\sqrt{n_\ell})$, where n_ℓ is the width of the layer.

Rewriting in Terms of RMS Norm:

- Condition 1 can be rewritten as:

"The RMS norm of the activations $\|h_\ell(x)\|_{\text{RMS}}$ is $\Theta(\sqrt{n_\ell})$, and the RMS norm of the updates $\|\Delta h_\ell(x)\|_{\text{RMS}}$ is $\Theta(\sqrt{n_\ell})$."

Implications:

- Condition 1 implies that $\|h_\ell(x)\|_2 \leq \Theta(\sqrt{n_\ell})$ and $\|\Delta h_\ell(x)\|_2 \leq \Theta(\sqrt{n_\ell})$.
- This ensures that the norms of the activations and updates grow at most with the square root of the layer width, preventing exponential growth or decay.

Key Assumption for Lower Bounds

- The key assumption that allows us to also get the **lower bounds** on $\|h_\ell(x)\|_2$ and $\|\Delta h_\ell(x)\|_2$ is that the **inputs and weights are sufficiently random and independent**.
- This assumption ensures that the activations and updates do not collapse to zero, providing a meaningful lower bound on their norms.

How to Answer the Question

Answer to Question 3(c):

""Desideratum 1 states that for effective feature learning, the RMS norm of the activations at each layer should be stable and not grow or shrink exponentially with depth or width. In terms of RMS norm, this means $\|h_\ell(x)\|_{\text{RMS}}$ should be $\Theta(1)$."

"Condition 1 provides a sufficient condition for Desideratum 1 to hold, requiring that the RMS norm of the activations $\|h_\ell(x)\|_{\text{RMS}}$ is $\Theta(\sqrt{n_\ell})$ and the RMS norm of the updates $\|\Delta h_\ell(x)\|_{\text{RMS}}$ is $\Theta(\sqrt{n_\ell})$. This implies that $\|h_\ell(x)\|_2 \leq \Theta(\sqrt{n_\ell})$ and $\|\Delta h_\ell(x)\|_2 \leq \Theta(\sqrt{n_\ell})$."

"The key assumption that allows us to also get the lower bounds on $\|h_\ell(x)\|_2$ and $\|\Delta h_\ell(x)\|_2$ is that the inputs and weights are sufficiently random and independent, ensuring that the activations and updates do not collapse to zero.""

Comment 3.4: Le Chat seems to do a good job on the first two parts of the problem. However, for the third part, it seems to draw something parallel to what the problem is asking.

Prompt 4.1:

Now let's move onto question 4. Help me solve each part of it.

Le Chat's answer:



Let's tackle **Question 4: Policy Gradient and the Reparameterization Gradient Estimator** step-by-step. This question involves deriving gradient estimators for an objective function involving a parameterized probability distribution.

Question 4(a): Express $\nabla_\mu \mathcal{F}(\mu)$ as an Expectation

Objective Function

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

where \mathbf{x} is a k -dimensional multivariate Gaussian random variable with mean μ and covariance $\sigma^2 I$.

Probability Density Function

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

Derivation of $\nabla_\mu \mathcal{F}(\mu)$

We want to express $\nabla_\mu \mathcal{F}(\mu)$ as an expectation under $p_\mu(\mathbf{x})$.

1. Express $\mathcal{F}(\mu)$ as an Integral:

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

2. Differentiate Under the Integral Sign:

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

3. Compute $\nabla_\mu p_\mu(\mathbf{x})$:

$$\nabla_\mu p_\mu(\mathbf{x}) = p_\mu(\mathbf{x}) \cdot \frac{\mathbf{x} - \mu}{\sigma^2}$$

4. Substitute Back:

$$\nabla_\mu \mathcal{F}(\mu) = \int p_\mu(\mathbf{x}) \cdot \frac{\mathbf{x} - \mu}{\sigma^2} f(\mathbf{x}) d\mathbf{x}$$

5. Express as an Expectation:

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

Final Answer for 4(a)

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

Comment 4.1: Le Chat was able to arrive at the correct answer one shot.

Question 4(b): Rewrite $\mathcal{F}(\mu)$ with a Standard Normal Distribution

Rewriting the Expectation

We want to rewrite $\mathcal{F}(\mu)$ with an expectation under the standard normal distribution $\mathcal{N}(0, I)$.

1. Change of Variables:

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

2. Rewrite the Expectation:

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

Final Answer for 4(b)

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

Comment 4.2: Le Chat was able to arrive at the correct answer one shot.

Question 4(c): Express $\nabla_\mu \mathcal{F}(\mu)$ with the Standard Normal Distribution**Derivation of $\nabla_\mu \mathcal{F}(\mu)$** **1. Differentiate Under the Expectation:**

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

2. Move the Gradient Inside the Expectation:

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

3. Apply the Chain Rule:

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

Final Answer for 4(c)

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

Comment 4.3: Le Chat was able to arrive at the correct answer one shot.

Question 4(d): General Case for $\nabla \mathcal{F}(\theta)$

General Expression for $\nabla \mathcal{F}(\theta)$

For a general parameterized distribution $p_\theta(\mathbf{x})$, we can write:

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

Final Answer for 4(d)

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

Comment 4.4: Le Chat was able to arrive at the correct answer one shot.

Question 4(e): Reparameterization Gradient Estimator

Derivation of $\nabla_{\theta}\mathcal{F}(\theta)$

Given a function $g(\mathbf{z}, \theta)$ and a distribution $p(\mathbf{z})$ such that $\mathbf{x} = g(\mathbf{z}, \theta)$ has the same distribution as $\mathbf{x} \sim p_{\theta}(\mathbf{x})$, we can express the gradient as:

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

Special Case for Gaussian Distribution

For the Gaussian case in parts (b) and (c):

- $g(\mathbf{z}, \mu) = \mu + \sigma\mathbf{z}$
- $p(\mathbf{z}) = \mathcal{N}(0, I)$

Comment 4.5: Le Chat was able to arrive at the correct answer one shot.

Prompt 5.1:

Now let's move onto question 5. Help me solve each part of it.

Sep 23, 8:34pm

Le chat's answer:



Let's tackle **Question 5: Tensor Rematerialization** step-by-step. This question involves understanding how to train a neural network under memory constraints using tensor rematerialization.

Question 5(a): Number of Forward Passes During Backward Pass

Understanding the Problem

- You have a 10-layer neural network.
- During the backward pass, you need to recompute activations for layers that were not stored in memory.
- The question asks: **How many forward passes (fwd operations) are invoked during a single backward pass through the entire network?**

Given Setup

- Only activations for every 5th layer are stored during the initial forward pass.
- For example, activations for layers 5 and 10 are stored.

Backward Pass Logic

- To compute gradients for a layer, you need its input activations.
- If the input activations are not stored, you must recompute them by performing a forward pass from the nearest stored activation. 

Calculating Forward Passes

- **Layer 10:** Activations are stored. No recomputation needed.
- **Layer 9:** Need activations from layer 5. Recompute layers 6, 7, 8, 9.
- **Layer 8:** Need activations from layer 5. Recompute layers 6, 7, 8.
- **Layer 7:** Need activations from layer 5. Recompute layers 6, 7.
- **Layer 6:** Need activations from layer 5. Recompute layer 6.
- **Layer 5:** Activations are stored. No recomputation needed.
- **Layer 4:** Need activations from the input. Recompute layers 1, 2, 3, 4.
- **Layer 3:** Need activations from the input. Recompute layers 1, 2, 3.
- **Layer 2:** Need activations from the input. Recompute layers 1, 2.
- **Layer 1:** Need activations from the input. Recompute layer 1.

Counting Forward Passes

- **Layers 6-9:** $4 + 3 + 2 + 1 = 10$ forward passes.
- **Layers 1-4:** $4 + 3 + 2 + 1 = 10$ forward passes.
- **Total:** $10 + 10 = 20$ forward passes.

Answer to Question 5(a)

"During a single backward pass through the entire network, **20 forward passes** are invoked to recompute the necessary activations."

Comment 5.1: Le Chat was able to arrive at the correct answer one shot.

Question 5(b): Number of Memory Accesses During Backward Pass

Understanding the Problem

- Each time you load an activation from memory, it counts as a `loadmem` operation.
- You need to count how many times you load activations from memory during the backward pass.

Memory Access Logic

- **Layer 10:** Load activations for layer 5 and layer 10.
- **Layer 9:** Load activations for layer 5.
- **Layer 8:** Load activations for layer 5.
- **Layer 7:** Load activations for layer 5.
- **Layer 6:** Load activations for layer 5.
- **Layer 5:** Load activations for the input and layer 5.
- **Layer 4:** Load activations for the input.
- **Layer 3:** Load activations for the input.
- **Layer 2:** Load activations for the input.
- **Layer 1:** Load activations for the input.

Counting Memory Accesses

- **Layers 6-10:** 5 (for layer 5) + 1 (for layer 10) = 6 memory accesses.
- **Layers 1-5:** 4 (for input) + 1 (for layer 5) = 5 memory accesses.
- **Total:** 6 + 5 = 11 memory accesses.

Answer to Question 5(b)

"During a single backward pass through the entire network, **11 memory accesses** are invoked to load the necessary activations."

Comment 5.2: Le Chat seems to load activations for layer 5 twice.

Question 5(c): Speed of Disk Operations

Understanding the Problem

- You have access to a local disk with practically infinite storage for activations.
- You want to store all activations on disk and load them as needed.
- Each forward pass (`fwd`) takes 20 ns.
- Each memory access (`loadmem`) takes 10 ns.
- You need to determine how fast each disk operation (`loaddisk`) should be to match the total time of tensor rematerialization.

Total Time for Tensor Rematerialization

- **Forward Passes:** $20 \text{ operations} \times 20 \text{ ns} = 400 \text{ ns}$.
- **Memory Accesses:** $11 \text{ operations} \times 10 \text{ ns} = 110 \text{ ns}$.
- **Total Time:** $400 \text{ ns} + 110 \text{ ns} = 510 \text{ ns}$.

Total Time for Disk-Based Approach

- **Disk Operations:** You need to load all activations from disk. There are 10 layers + 1 input = 11 activations to load.
- Let t be the time for each `loaddisk` operation.
- **Total Time:** $11 \times t$.



Equating the Times

- To match the total time of tensor rematerialization, set:

$$11 \times t = 510 \text{ ns}$$

- Solving for t :

$$t = \frac{510 \text{ ns}}{11} = 46.36 \text{ ns}$$

Answer to Question 5(c)

"Each disk operation (`loaddisk`) should take no more than **46.36 ns** to match the total time of tensor rematerialization."

Comment 5.3: Le Chat seems to misunderstand the number of `loadmem` operations.