

Thanks for your question, it's hidden from other students until approved by an instructor.

Participation A: HW 3 - Kimi #118



Deena Sun
5 hours ago in Curiosity

STAR WATCHING VIEWS
8



Executive Summary

I tackled homework 3 with Kimi 2. This was my first time trying Kimi's family of LLMs and overall, I was quite impressed with Kimi's derivation capabilities. For the most part, I successfully one-shotted question 1: Maximal Update Parameterization with Kimi 2.

For question 3: Maximal Update Parameterization Research, Kimi often hallucinated with visualizations and technical details. When interpreting figure 1, Kimi 2 hallucinated and guessed what the two loss curves represented. Kimi 1.5 (with vision capabilities) did a better job actually explaining that the right panel was proof of μ Transfer's improvement over standard practice, but incorrectly asserted that even with μ Transfer, wider models couldn't achieve a lower loss. Kimi 1.5 exhibited persistent issues trying to correctly interpret the different colors in table 3 and how they corresponded to standard practice versus μ Transfer. I also noticed that throughout this assignment, Kimi interpreted the induced RMS matrix norm as simply a reformulation of the vector RMS norm that treated the matrix as a flattened vector. While it got the main idea of how the 2 desiderata in the paper ensured that the L_2 -norms of each hidden layer/hidden layer update would scale proportionally to $\Theta(\sqrt{n}l)$, it didn't use the RMS-to-RMS matrix norm we covered in lecture.

When approaching question 4: Policy Gradient and the Reparameterization Gradient Estimator, I used Kimi a lot to clarify additional questions I had about taking derivatives with expectations and score functions. I felt like Kimi had more of a tendency to plug formulae/identities in its derivations here without providing the reasoning or the "why" behind how they fit into the solution. Whereas I would have liked more explicit explanations so that I could extrapolate these concepts to other problems.

Lastly, Kimi's responses for question 5: Tensor Rematerialization were pretty clear to follow. Kimi did have one hiccup where it kept claiming that we would only need 8 loadmems instead of 10, and unlike in question 3, it was much more assertive. It persisted this mistake even in subpart 5(c).

All in all, I think Kimi demonstrated strong mathematical derivation reasoning, but has clear areas of improvement for visual understanding and adapting to feedback. As a side note, Kimi also seems to suffer from random switch-ups in markdown formatting and style that I've noticed in other LLMs.

Annotated logs for each question: to make things more organized, I'll make separate comments for each question!

- Question 1: <https://edstem.org/us/courses/84647/discussion/7077134?comment=16480696>
- Question 3: <https://edstem.org/us/courses/84647/discussion/7077134?comment=16480725>
- Question 4: <https://edstem.org/us/courses/84647/discussion/7077134?comment=16480735>
- Question 5: <https://edstem.org/us/courses/84647/discussion/7077134?comment=16480774>



Deena Sun 5h

Question 1: Maximal Update Parametrization

One shot prompting question

1a

Consider a dense layer with input $x \in \mathbb{R}^{d_1}$, weights $W \in \mathbb{R}^{d_2 \times d_1}$, where W is initialized with i.i.d. standard Gaussian entries, and entries of x are sampled from an i.i.d. unit Gaussian.

What is the expected squared RMS norm of the output features $y = Wx$? How does this scale with d_1 or d_2 ? What constant should we multiply W by to ensure that the expected squared RMS norm of Wx is 1, regardless of d_1 and d_2 ?

Hint: Consider a simplified dense layer with a single output feature, $W \in \mathbb{R}^{1 \times d_1}$.

- What is the distribution of the scalar $y = Wx$? What is its variance?

Let's first tackle the simplified case suggested in the hint.

Step 1: Simplified Case – Single Output Feature

We have:

- $W \in \mathbb{R}^{1 \times d_1}$, where each entry $W_i \sim \mathcal{N}(0, 1)$ (i.i.d. standard Gaussian).
- $x \in \mathbb{R}^{d_1}$, where each entry $x_i \sim \mathcal{N}(0, 1)$ (i.i.d. unit Gaussian).
- The output is a scalar: $y = Wx = \sum_{i=1}^{d_1} W_i x_i$.

Now let's find the distribution and variance of y .

Distribution of y :

The product $W_i x_i$ is the product of two independent standard Gaussian variables. The product of two independent Gaussians is not Gaussian, but the sum of such products might be. However, we can compute the variance directly.

Variance of y :

We have:

$$\text{Var}(y) = \text{Var}\left(\sum_{i=1}^{d_1} W_i x_i\right)$$

Since W_i and x_i are independent, and all entries are i.i.d., we can write:

$$\text{Var}(y) = \sum_{i=1}^{d_1} \text{Var}(W_i x_i)$$

Now, for two independent random variables A and B , the variance of their product is:

$$\text{Var}(AB) = \text{Var}(A)\text{Var}(B) + \text{Var}(A)\mathbb{E}[B]^2 + \text{Var}(B)\mathbb{E}[A]^2$$

For W_i and x_i :

- $\mathbb{E}[W_i] = 0, \text{Var}(W_i) = 1$
- $\mathbb{E}[x_i] = 0, \text{Var}(x_i) = 1$

Thus:

$$\text{Var}(W_i x_i) = (1)(1) + (1)(0)^2 + (1)(0)^2 = 1$$

Therefore:

$$\text{Var}(y) = \sum_{i=1}^{d_1} 1 = d_1$$

So, $y \sim \mathcal{N}(0, d_1)$ (since the sum of independent Gaussians is Gaussian, and we've matched the mean and variance).

Expected Squared Value of y :

For any random variable Z , $\mathbb{E}[Z^2] = \text{Var}(Z) + \mathbb{E}[Z]^2$. Here, $\mathbb{E}[y] = 0$, so:

$$\mathbb{E}[y^2] = \text{Var}(y) = d_1$$

Step 2: General Case – Multiple Output Features

Now, we have:

- $W \in \mathbb{R}^{d_2 \times d_1}$, with entries $W_{ij} \sim \mathcal{N}(0, 1)$.
- $x \in \mathbb{R}^{d_1}$, with entries $x_j \sim \mathcal{N}(0, 1)$.
- The output is $y = Wx \in \mathbb{R}^{d_2}$, where $y_i = \sum_{j=1}^{d_1} W_{ij}x_j$.

From the simplified case, each y_i is $\mathcal{N}(0, d_1)$, and $\mathbb{E}[y_i^2] = d_1$.

RMS Norm of y :

The RMS norm is defined as:

$$\text{RMS}(y) = \sqrt{\frac{1}{d_2} \sum_{i=1}^{d_2} y_i^2}$$

The squared RMS norm is:

$$\text{RMS}(y)^2 = \frac{1}{d_2} \sum_{i=1}^{d_2} y_i^2$$

Expected Squared RMS Norm:

I was impressed that Kimi understood the concept of RMS norms and taking expectations of norms. I think even a year ago, I wouldn't have expected an LLM to have this level of mathematical reasoning.

$$\mathbb{E}[\text{RMS}(y)^2] = \mathbb{E} \left[\frac{1}{d_2} \sum_{i=1}^{d_2} y_i^2 \right] = \frac{1}{d_2} \sum_{i=1}^{d_2} \mathbb{E}[y_i^2] = \frac{1}{d_2} \cdot d_2 \cdot d_1 = d_1$$

So, the expected squared RMS norm is d_1 , independent of d_2 .

Step 3: Scaling W to Normalize the Output

We want $\mathbb{E}[\text{RMS}(y)^2] = 1$, regardless of d_1 and d_2 . Currently, it's d_1 .

Let's multiply W by a constant c , so the new weights are cW . Then, the new output is $y' = cWx$, and:

$$\mathbb{E}[\text{RMS}(y')^2] = \mathbb{E} \left[\frac{1}{d_2} \sum_{i=1}^{d_2} (cy_i)^2 \right] = c^2 \cdot \mathbb{E}[\text{RMS}(y)^2] = c^2 d_1$$

We want this to be 1:

$$c^2 d_1 = 1 \implies c = \frac{1}{\sqrt{d_1}}$$

Thus, we should multiply W by $\frac{1}{\sqrt{d_1}}$ to ensure that the expected squared RMS norm of Wx is 1, regardless of d_1 and d_2 .

Final Answer:

- The expected squared RMS norm of $y = Wx$ is d_1 .
- It scales linearly with d_1 and is independent of d_2 .
- To ensure the expected squared RMS norm is 1, multiply W by $\frac{1}{\sqrt{d_1}}$.

Can you explain how you derived the variance of the product of 2 random variables A, B?

Here is a clean step-by-step derivation of

$$\text{Var}(A B) = \text{Var}(A) \text{Var}(B) + \text{Var}(A) \mathbb{E}[B]^2 + \text{Var}(B) \mathbb{E}[A]^2$$

for two **independent** random variables A and B.

(Independence is essential; without it the formula picks extra covariance terms.)

1. Start with the definition of variance

$$\text{Var}(Z) = \mathbb{E}[Z^2] - (\mathbb{E}[Z])^2.$$

Apply it to $Z = A B$:

$$\text{Var}(A B) = \mathbb{E}[(A B)^2] - (\mathbb{E}[A B])^2.$$

2. Use independence twice.

- Expectation of a product of independent variables factorises:

$$\mathbb{E}[A B] = \mathbb{E}[A] \mathbb{E}[B]. \quad (2)$$

- The same is true for the square:

$$\mathbb{E}[(A B)^2] = \mathbb{E}[A^2 B^2] = \mathbb{E}[A^2] \mathbb{E}[B^2]. \quad (3)$$

3. Express $\mathbb{E}[A^2]$ and $\mathbb{E}[B^2]$ in terms of mean and variance.

For any r.v. X we have $\mathbb{E}[X^2] = \text{Var}(X) + (\mathbb{E}[X])^2$.

Therefore

$$\begin{aligned} \mathbb{E}[A^2] &= \text{Var}(A) + \mathbb{E}[A]^2, \\ \mathbb{E}[B^2] &= \text{Var}(B) + \mathbb{E}[B]^2. \end{aligned} \quad (4)$$

4. Insert (4) into (3):

$$\begin{aligned} \mathbb{E}[(A B)^2] &= (\text{Var}(A) + \mathbb{E}[A]^2)(\text{Var}(B) + \mathbb{E}[B]^2) \\ &= \text{Var}(A) \text{Var}(B) + \text{Var}(A) \mathbb{E}[B]^2 \\ &\quad + \text{Var}(B) \mathbb{E}[A]^2 + \mathbb{E}[A]^2 \mathbb{E}[B]^2. \end{aligned} \quad (5)$$

5. Insert (2) into the second term of (1):

$$(\mathbb{E}[A B])^2 = (\mathbb{E}[A] \mathbb{E}[B])^2 = \mathbb{E}[A]^2 \mathbb{E}[B]^2.$$

6. Subtract (6) from (5):

$$\begin{aligned} \text{Var}(A B) &= [\text{Var}(A) \text{Var}(B) + \text{Var}(A) \mathbb{E}[B]^2 \\ &\quad + \text{Var}(B) \mathbb{E}[A]^2 + \mathbb{E}[A]^2 \mathbb{E}[B]^2] \end{aligned}$$

Here, I was asking Kimi to help explain how to take the variance of the product of two independent RVs

Not sure why Kimi's markdown formatting diverged from the scripted "Var" from earlier, but I think Kimi did a good job explaining the expansion of different terms

$$\begin{aligned}&= \mathbb{E}[A]^2 \mathbb{E}[B]^2 \\&= \text{Var}(A) \text{Var}(B) + \text{Var}(A) \mathbb{E}[B]^2 \\&\quad + \text{Var}(B) \mathbb{E}[A]^2.\end{aligned}$$

Hence, for independent A and B,

$$\text{Var}(A+B) = \text{Var}(A) \text{Var}(B) + \text{Var}(A) \mathbb{E}[B]^2 + \text{Var}(B) \mathbb{E}[A]^2.$$

If either mean is zero, the expression collapses to

$\text{Var}(A+B) = \text{Var}(A) \text{Var}(B)$, the special case used in the neural-network calculation

calculation.

One-shot prompting question

1b

We will now consider how to ensure that RMS norms of features update at a constant rate during training, again regardless of layer width. Assume we are using the SignGD optimizer (which is a simplified version of Adam). Unlike at initialization, where we assume weights and inputs are independent, the updates to weights during training are very much correlated with the inputs. For simplicity, assume that the minibatch training input $x_i \in \mathbb{R}^{d_1}$ are sampled from an i.i.d. unit Gaussian, and the raw

gradient $\nabla_W f(W)$ is an outer product of the input and a constant backpropagated vector $g_i \in \mathbb{R}^{d_2}$

d_2

.

$$W_{t+1} \leftarrow W_t + \eta \text{sign}(x_i g_i)$$

T

i

$).$

What is the expected RMS norm squared of the (unscaled) change in features y

$' = \text{sign}(x_i g_i)$

T

i

$)x_i?$

How does this scale with d_1 or d_2 ? What constant should we multiply the update by to ensure that the expected RMS norm of y

$'$

is 1, regardless of d_1 and d_2 ?

We are given:

- $x_i \in \mathbb{R}^{d_1}$ i.i.d. $\mathcal{N}(0,1)$
- $g_i \in \mathbb{R}^{d_2}$ **fixed** (constant across the mini-batch)
- update matrix $\Delta W = \text{sign}(x_i g_i^T)$ (element-wise sign, so every entry is ± 1)
- new feature vector $y' = \Delta W x_i$ (no weight scaling applied yet)

We need:

1. $\mathbb{E}[\text{RMS}(y')^2] = \mathbb{E}[(1/d_2) \|y'\|^2]$
2. How it scales with d_1, d_2
3. A constant c s.t. after rescaling the update by c , the above expectation becomes 1.

1. Express y'

$$y' = \text{sign}(x_i g_i^T) x_i \quad (\text{size } d_2 \times 1)$$

Entry k of y' is

$$\begin{aligned} y'_k &= \sum_{j=1}^{d_1} \text{sign}(x_i[j] g_i[k]) x_i[j] \\ &= \text{sign}(g_i[k]) \sum_{j=1}^{d_1} \text{sign}(x_i[j]) x_i[j] \\ &= \text{sign}(g_i[k]) \sum_{j=1}^{d_1} |x_i[j]|. \end{aligned}$$

Because $\text{sign}(x_i[j]) x_i[j] = |x_i[j]|$.

The sum $S \triangleq \sum_{j=1}^{d_1} |x_i[j]|$ is **the same for every row k** ; it is independent of g_i .

Hence

$$y' = \text{sign}(g_i) \cdot S \quad (\text{outer product with the constant scalar } S).$$

2. Squared RMS norm

$$\begin{aligned} \text{RMS}(y')^2 &= (1/d_2) \|y'\|^2 \\ &= (1/d_2) \|\text{sign}(g_i) S\|^2 \\ &= (1/d_2) S^2 \cdot \|\text{sign}(g_i)\|^2 \end{aligned}$$

I was confused why Kimi wrote y as $\text{sign}(x * g.T)$ which would cause a dimension mismatch when taking the inner product with x , instead of $\text{sign}(g * x).T$

$$= (1/d_2) S^2 \cdot d_2 \\ = S^2.$$

Therefore

$$\mathbb{E}[\text{RMS}(y')^2] = \mathbb{E}[S^2] = \mathbb{E}[(\sum_{j=1}^{d_1} |x_i[j]|)^2].$$

3. Compute $\mathbb{E}[S^2]$

Let $Z_j \stackrel{\text{def}}{=} |x_i[j]|$. Z_j are i.i.d. half-normal:

$$\mathbb{E}[Z_j] = \sqrt{2/\pi}, \quad \text{Var}(Z_j) = 1 - 2/\pi.$$

For the sum $S = \sum_{j=1}^{d_1} Z_j$,

$$\mathbb{E}[S] = d_1 \sqrt{2/\pi}, \\ \text{Var}(S) = d_1 (1 - 2/\pi).$$

Hence

$$\mathbb{E}[S^2] = \text{Var}(S) + (\mathbb{E}[S])^2 \\ = d_1(1 - 2/\pi) + d_1^2 (2/\pi) \\ = d_1 + (2/\pi)(d_1^2 - d_1).$$

So

$$\mathbb{E}[\text{RMS}(y')^2] = d_1 + (2/\pi)(d_1^2 - d_1).$$

4. Scaling behaviour

- **Linearly** in d_1 for small d_1
 - **Quadratically** in d_1 for large d_1 (dominant term $(2/\pi) d_1^2$)
 - **Independent** of d_2 (the d_2 cancelled out)
-

5. Normalising constant

We want

$$\mathbb{E}[\text{RMS}(c y')^2] = c^2 \mathbb{E}[\text{RMS}(y')^2] = 1.$$

Therefore

$$c = 1 / \sqrt{[d_1 + (2/\pi)(d_1^2 - d_1)]}.$$

This constant makes the expected squared RMS norm of the **rescaled** feature change exactly 1, irrespective of d_2 , and depending only on d_1 .

Here, I was questioning how Kimi calculated the kj -th entry of the outer product between g and x

I noticed you wrote: Entry k of y' is
 $y'_k = \sum_{j=1}^{d_1} g_i[j] \text{sign}(x_i[j]) g_i[k] x_i[j]$

So are you calculating the kj -th entry of $x_i g_i.T$ as $x_i[j] * g_i[k]$? Shouldn't it be that the kj -th entry of $g_i x_i.T = x_i[j] * g_i[k]$ and that therefore we should flip the order of x_i and g_i in the $\text{sign}(x_i g_i.T)$ expression? This way, the dimensions of $\text{sign}(x_i g_i.T) * x_i$ line up

You are absolutely right — thank you for catching the dimension mismatch.

The outer-product gradient is

$$\nabla_W f = g_i x_i.T \quad (\text{shape } d_2 \times d_1),$$

so the SignGD update is

$$\Delta W = \eta \text{sign}(g_i x_i.T) \quad (\text{same shape } d_2 \times d_1).$$

Therefore the raw feature change is

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

Now every entry of the matrix $\text{sign}(g_i x_i.T)$ is

$$[\text{sign}(g_i x_i.T)]_{k,j} = \text{sign}(g_i[k] x_i[j]).$$

With that order the matrix–vector product is well-defined:

$$\begin{aligned} y'k &= \sum_{j=1}^{d_1} \text{sign}(g_i[k]) x_i[j] \\ &= \text{sign}(g_i[k]) \sum_{j=1}^{d_1} |x_i[j]|. \end{aligned}$$

The rest of the derivation proceeds exactly as before, because the scalar factor $\text{sign}(g_i[k])$ still cancels in $\|y'\|^2$ and the sum $\sum |x_i[j]|$ is unchanged.

So the final expected squared RMS norm and the normalising constant remain the same; only the intermediate expression now has consistent dimensions.

When computing $E[\|y'\|^2]$, let's instead think about using the frobenius norm of $\text{sign}(g_i x_i^\top)$ which we know equals its spectral norm when the matrix has rank 1.

Great observation — since $\text{sign}(g_i x_i^\top)$ is a **rank-1 matrix**, its **spectral norm** and **Frobenius norm** are equal, and we can use this to compute $E[\|y'\|^2]$ more cleanly.

Step 1: Express y' using the spectral/Frobenius norm

We have:

$$y' = \text{sign}(g_i x_i^\top) x_i$$

I wanted to try
and push Kimi
to see if it
could reach
the same
derivation
using
Frobenius/
spectral
norms from
lecture

Let's define:

$$M = \text{sign}(g_i x_i^\top) \in \mathbb{R}^{d_2 \times d_1}$$

Then:

$$y' = M x_i$$

So:

$$\|y'\|^2 = \|M x_i\|^2$$

We want:

$$E[\|y'\|^2] = E[\|M x_i\|^2]$$

KIMI was a little spare on its explanations here

Step 2: Use the fact that M is rank-1

Since $M = \text{sign}(g_i) x_i^\top$ is a rank-1 matrix, we can write:

$$M = \text{sign}(g_i) \cdot \text{sign}(x_i)^\top$$

So: Without lecture, I probably wouldn't have understood why we could factor $\text{sign}(g) * x^\top$ into $\text{sign}(g) * \text{sign}(x).^\top$

$$Mx_i = \text{sign}(g_i) \cdot (\text{sign}(x_i)^\top x_i) = \text{sign}(g_i) \cdot \sum_{j=1}^{d_1} |x_i[j]|$$

Thus: Same thing here with how the squared two norm of $\text{sign}(g) = d_2$

$$\|Mx_i\|^2 = \left(\sum_{j=1}^{d_1} |x_i[j]| \right)^2 \cdot \|\text{sign}(g_i)\|^2 = \left(\sum_{j=1}^{d_1} |x_i[j]| \right)^2 \cdot d_2$$

Step 3: Take expectation

$$\mathbb{E}[\|y'\|^2] = d_2 \cdot \mathbb{E} \left[\left(\sum_{j=1}^{d_1} |x_i[j]| \right)^2 \right]$$

As before, we compute:

$$\mathbb{E} \left[\left(\sum_{j=1}^{d_1} |x_i[j]| \right)^2 \right] = d_1 + \frac{2}{\pi} (d_1^2 - d_1)$$

So:

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

Step 4: RMS norm squared

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

✓ Final Result

Using the **Frobenius norm = spectral norm** for rank-1 $\text{sign}(g_i \cdot x_i \cdot \text{top})$ gives the **same result** as before:

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

and the **normalizing constant** is:

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

Here, I was a bit confused on how Kimi derived the expectation of the squared RMS norm. I wasn't very familiar with the half-normal distribution yet

explain how to compute this: E

(
j=1
 Σ
d
1

|x
i
[j])
2

```
=d  
1  
+  
π  
2  
(d  
1  
2  
-d  
1  
)
```

We want

$$\mathbb{E} \left[\left(\sum_{j=1}^{d_1} |x_i[j]| \right)^2 \right]$$

with

$$x_i[j] \sim \mathcal{N}(0, 1) \text{ i.i.d.}$$

Write

$$S = \sum_{j=1}^{d_1} Z_j, \quad Z_j = |x_i[j]|.$$

The Z_j are i.i.d. with the **half-normal** distribution:

- Mean: $\mu = \mathbb{E}[Z_j] = \sqrt{2/\pi}$

- Variance: $\sigma^2 = \text{Var}(Z_j) = 1 - 2/\pi$

Now use the identity

$$\mathbb{E}[S^2] = \text{Var}(S) + (\mathbb{E}[S])^2.$$

Because the Z_j are independent,

$$\text{Var}(S) = d_1\sigma^2 = d_1(1 - 2/\pi)$$

and

$$\mathbb{E}[S] = d_1\mu = d_1\sqrt{2/\pi} \implies (\mathbb{E}[S])^2 = d_1^2 \cdot 2/\pi.$$

Add them:

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

**Intuition on why
the expected
update (ΔW)'s
RMS norm didn't
include d_2**

Why does this update rule only depend on d_1 ?

Because the only randomness left after the rank-1 structure is removed lives in the *input* vector $x_i \in \mathbb{R}^{d_1}$.

1. The update matrix is

$$\Delta W = \text{sign}(g_i x_i^T) = \text{sign}(g_i) \text{ sign}(x_i)^T.$$

Once the sign of every entry is taken, **every row is identical up to a global ± 1** that comes from $\text{sign}(g_i[k])$.

2. When this matrix acts on x_i ,

$$y' = \Delta W x_i = \text{sign}(g_i) \cdot (\text{sign}(x_i)^T x_i) = \text{sign}(g_i) \cdot \sum_{j=1}^{d_1} |x_i[j]|,$$

the entire d_2 -dimensional vector y' is just the *same scalar*

$\sum |x_i[j]|$ multiplied by the fixed vector $\text{sign}(g_i)$.

Consequently

$$\|y'\|^2 = (\sum |x_i[j]|)^2 \cdot \|\text{sign}(g_i)\|^2 = (\sum |x_i[j]|)^2 \cdot d_2,$$

and the RMS norm squared

$$\text{RMS}(y')^2 = (1/d_2) \|y'\|^2 = (\sum |x_i[j]|)^2$$

does not contain d_2 any more.

3. Expectation is taken only over the distribution of x_i , so the final answer inherits the dimension of x_i alone: d_1 .

Generated by Kimi AI

Deena Sun and Kimi chat 

 Reply Edit Delete ...



Deena Sun 5h

Question 3: Maximal Update Parameterization Research

I attempted to one-shot question 3a with a screenshot of figure 1, but I think Kimi 2 hallucinated and simply guessed that both graphs were displaying the losses following the same training paradigms.

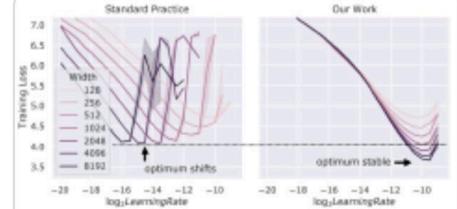


Figure 1: Training loss against learning rate on Transformers of varying d_{model} trained with Adam. Conventionally and in contrast with our technique, different widths do not share the same optimal hyperparameter; wider networks do not always perform better than narrower ones; in fact they underperform the same-width networks in our technique even after tuning learning rate (see dashed line). See Sections 3 and 4 for experimental setup.

This homework problem talks about the research papers behind “maximal update parameterization,” also

called μ P in the community. This idea is discussed in detail in the paper *Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer*.

Look at Figure 1 of the paper. What are the two side-by-side figures plotting? What is the observation you make about the width of a matrix?

Below is a direct reading of what the two panels in Figure 1 of the paper *Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer* show, plus the width-related observation the authors highlight.

1. What the two side-by-side panels plot

Table

Copy

| Panel | x-axis | y-axis |
|-------|--------|--------|
| | | |

| | | |
|-------|--------------------------------|--|
| Left | $\log_2(\text{learning-rate})$ | training loss after a fixed number of steps |
| Right | $\log_2(\text{learning-rate})$ | training loss after the same fixed number of steps |

In both panels every curve is obtained with the **standard PyTorch parameterization** (i.e. no μ P rescaling).

Kimi 2 claimed that both panels showed loss curves with standard practices, despite the right panel displaying μ Transfer's loss curve

2. Observation about matrix width

- Each width has its own, distinct optimal learning-rate — the loss valley moves to smaller learning-rates as the network becomes wider.
- Wider is not automatically better: after individually tuning the learning-rate, the 1024-wide network actually reaches a lower loss than any of the wider ones (4096, 8192).

(The dashed horizontal line in the right panel marks the best loss achieved by the 1024-width net and shows that the wider nets do not beat it even at their own tuned LR.)

Thus, under the conventional parameterization, loss scales monotonically with width, and hyperparameters

Did not explain how the right panel actually shows that training wider nets with μ Transfer can achieve even lower loss

re-done for every width—exactly the inefficiency that μ P is designed to remove.

Not sure why the export translated this into Chinese but this was just a warning that I should switch to Kimi 1.5 for vision capabilities

当前模型仅对图片中的文字进行识别，切换 K1.5 获取更好的视觉理解能力

I tried re-asking question 3a to Kimi 1.5 with vision capabilities to see if Kimi 1.5 could extract more information from the image than what Kimi 2 attempted to summarize from its memory recall alone



This homework problem talks about the research papers behind “maximal update parameterization,” also

called μ P in the community. This idea is discussed in detail in the paper Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer.

Look at Figure 1 of the paper. What are the two side-by-side figures plotting? What is the observation you make about the width of a matrix?

The two side-by-side figures in Figure 1 of the paper "Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer" plot training loss against the logarithm of the learning rate for Transformers of varying widths (d_{model}) trained with Adam. The left figure represents the standard practice, while the right figure shows the results using the authors' technique.

Observations about the width of a matrix:

1. Standard Practice (Left Figure):

- Different widths do not share the same optimal hyperparameter.
- Wider networks do not always perform better than narrower ones.
- There are shifts in the optimal learning rate for different widths, indicating that the optimal hyperparameters vary with the size of the network.

Kimi 1.5 correctly identified that the left panel displayed results with standard practice whereas the right panel displayed results using μ Transfer

2. Our Work (Right Figure):

- The technique presented by the authors results in a more stable and consistent performance across different widths.
- Even after tuning the learning rate, wider networks underperform compared to narrower networks in the authors' technique, as

indicated by the dashed line which represents the best performance achieved by a narrower network.

The key observation is that in the standard hyperparameters are not consistent across and wider networks do not necessarily contrast, the authors' technique shows underperform even after learning rate t method provides a more uniform and potentially more effective approach to hyperparameter tuning across different network sizes.

This second observation still seems to misinterpret the caption. It misses how the authors actually achieved an even lower loss than the best loss achieved by the 4096-wide network with standard practice when they used μ Transfer.

This was a screenshot of table 3, which compared scaling factors for SP vs. μ Transfer. Even with vision capabilities, Kimi 1.5 didn't seem to understand that the gray (1) in the bottom right cell corresponded to the SP of Adam LR for hidden weights.

Q

What does this table say is the corresponding standard parameterization of Adam LR for hidden layers?

The table indicates that for hidden layers, the corresponding standard parameterization (SP) of Adam Learning Rate (LR) is $1/\text{fan_in}$. This means that in the context of standard parameterization, the learning rate for Adam optimizer applied to hidden layers is scaled inversely with the fan-in (the number of input connections to a neuron).

I kept trying to correct Kimi's hallucination

Actually, I believe the purple text represents key differences from standard parameterization whereas the gray text in the table is the standard parameterization practice.

You're correct, and I apologize for the confusion. Let's clarify the table based on your correction:

- **Purple text:** Highlights key differences from standard parameterization (SP).
- **Gray text:** Represents the corresponding standard parameterization practice.

According to your correction, here's what the table indicates for each

category:

Input Weights & All Biases

- Init. Var.: $1/\text{fan_in}$
- SGD LR: fan_out (Purple text indicates a difference from SP)
- Adam LR: 1 (This is the standard practice, as indicated by the gray text)

Output Weights

- Init. Var.: $1/\text{fan_in}^2$
- SGD LR: $(1/\text{fan_in})$ (This is the standard practice, as indicated by the gray text)
- Adam LR: $1/\text{fan_in}$ (This is the standard practice, as indicated by the gray text)

Hidden Weights

- Init. Var.: $1/\text{fan_in}$
- SGD LR: 1 (This is the standard practice, as indicated by the gray text)
Kimi mentions gray text but get the wrong Adam LR value
Adam LR: $1/\text{fan_in}$ (This is the standard practice, as indicated by the gray text)

In summary, the gray text in the table represents the standard parameterization practices for different optimizers and layers, while the purple text highlights the key differences from these practices.

In the Adam LR and Hidden Weights cell, there is a gray "(1)"

Thank you for pointing out the specific detail in the table. Let's clarify the interpretation of the gray text "(1)" in the context of Adam LR for hidden weights:

Table Interpretation:

Hidden Weights

Kimi would acknowledge or apologize for the mistake, but continued to refer to $1/\text{fan_in}$ as standard practice for Adam LR

- Init. Var.: $1/\text{fan_in}$
- SGD LR: 1 (This is the standard parameterization practice)
- Adam LR: $1/\text{fan_in}$ (This is the standard parameterization practice, indicated by the gray text)

The gray "(1)" in the Adam LR for hidden weights cell indicates that this is the standard practice for initializing the Adam learning rate for hidden layers. Specifically, it suggests that the learning rate should be scaled inversely with the fan-in (number of incoming connections to a neuron), which is a common practice in neural network training to ensure stable convergence.

In summary, the table shows that for hidden weights:

- The initialization variance is $1/\text{fan_in}$.
- The standard parameterization for SGD learning rate is 1.
- The standard parameterization for Adam learning rate is $1/\text{fan_in}$, which is highlighted by the gray "(1)" to indicate it as the standard practice.

Kimi addresses the 'gray "(1)" but continues to insist that SD

I believe that the value in gray (i.e. the (1)) is the standard practice

was 1/fan_in

ly right, and I appreciate your patience in clarifying this.

Let's correct the interpretation based on your input:

Corrected Interpretation of the Table:

Hidden Weights

- Init. Var.: 1/fan_in
- SGD LR: 1 (This is the standard parameterization practice, indicated by the gray text)
- Adam LR: 1/fan_in (This is also the standard parameterization practice, indicated by the gray text)

In this context, the gray text indeed represents the standard parameterization practices:

- For SGD, the learning rate for hidden weights is set to 1, which is a common practice.
- For Adam, the learning rate for hidden weights is scaled by 1/fan_in, which is another common practice to adjust the learning rate based on the number of incoming connections.

The purple text in the table highlights the key differences from these standard practices, indicating where the proposed method (μP) deviates from the standard parameterization.

Thank you for your correction, and I hope this clarification aligns with your understanding and the intended meaning of the table.

For question 3c, I included the question prompt as well as the 2 desideratum from the paper

The paper A Spectral Condition for Feature Learning presents the same scaling using elementary linear algebra. While understanding the entirety of the paper is out of scope of the class, we presented some simplified arguments in lecture.

Look at Desideratum 1. Why is this behavior desired for feature learning? Can you rewrite this

Desideratum in terms of the RMS norm? Similarly, rewrite Condition 1 in terms of the RMS norm as well. How does Condition 1 imply that

$$\|h\ell(x)\|_2 \leq \Theta(\sqrt{n\ell})$$

$$\text{and that } \|\Delta h\ell(x)\|_2 \leq$$

$$\Theta(\sqrt{n\ell})$$

? What is the key assumption that allows us to also get the lower bounds on $\|h\ell(x)\|_2$ and $\|\Delta h\ell(x)\|_2$?

<Desideratum>

Desideratum 1 (Feature learning). Let $h\ell(x) \in \mathbb{R}^{n\ell}$

denote the features of input x at layer ℓ of a neural network, and let $\Delta h\ell(x) \in \mathbb{R}^{n\ell}$

denote their change after a gradient step. We desire that:

$$\|h\ell\|_2 = \Theta(\sqrt{n\ell})$$

$$\text{and } \|\Delta h\ell\|_2 = \Theta(\sqrt{n\ell})$$

, at layers $\ell = 1, \dots, L-1$.

</Desideratum>

Desideratum 1 asks for the ℓ_2 -norm of the **pre-activation feature vector** $h\ell(x)$ and of its **update** $\Delta h\ell(x)$ to scale like the square-root of the layer width:

$$\|h\ell(x)\|_2 = \Theta(\sqrt{n\ell}) \text{ and } \|\Delta h\ell(x)\|_2 = \Theta(\sqrt{n\ell}). \quad (1)$$

Why is this “good”?

- $n\ell$ is the number of neurons in layer ℓ .
 - If the **per-coordinate** size of the signal and of its gradient update stay $\Theta(1)$, then the whole vector carries $\Theta(n\ell)$ bits of information and the optimizer can change $\Theta(n\ell)$ coordinates independently — exactly the capacity we want for representation learning.
 - If instead the norm were $\Theta(1)$ (independent of width), the per-coordinate magnitude would be $\Theta(1/\sqrt{n\ell})$ and the network would lose width-proportional information; if it were $\Theta(n\ell)$ the updates would explode.
- Hence (1) is the Goldilocks scaling that keeps both forward activations and backward updates stable **and** informative as the network becomes wider.

Re-writing everything with RMS norms

For any vector $v \in \mathbb{R}^n$ the RMS (root-mean-square) norm is

$$\text{RMS}(v) := \sqrt{(1/n) \sum_i v_i^2} = (1/\sqrt{n}) \|v\|_2. \quad (2)$$

Therefore

$$\|v\|_2 = \sqrt{n} \cdot \text{RMS}(v). \quad (3)$$

This is the correct derivation of the desired RMS norm for $h_{\{l\}}$!

Applying (3) to (1) gives the **RMS version of Desideratum 1**:

$$\text{RMS}(h\ell(x)) = \Theta(1) \text{ and } \text{RMS}(\Delta h\ell(x)) = \Theta(1). \quad (4)$$

Condition 1 in the paper is the **spectral-norm condition** on the weight matrix $W\ell$:

$$\|W\ell\|_2 = \Theta(1) \quad (5)$$

(the largest singular value is $\Theta(1)$).

In RMS language we can restate (5) as

$$\text{RMS}(W\ell) = \Theta(1/\sqrt{(n\ell \ n_{\ell-1}))} \quad (6)$$

I don't think Kimi understood the concept of the induced RMS norm here because it's calculating $\text{RMS}(\|W\ell\|) = 1/\sqrt{n_{\{l\}} * n_{\{l-1\}}} * \|W\ell\|$ instead of $\text{RMS}(\|W\ell\|) = \sqrt{n_{\{l-1\}} / n_{\{l\}}} * \|W\ell\|$. I.e. Kimi seems to be applying the vector RMS norm to a matrix by treating the matrix as a flattened vector with $n_{\{l\}} * n_{\{l-1\}}$ components

because for a matrix of i.i.d. $\Theta(1)$ entries one has $\|W\ell\|_2 \approx \sqrt{(n\ell n\ell_1)} \cdot \text{RMS}(W\ell)$.

Despite the misunderstanding of the RMS -> RMS norm for $W_{\{l\}}$, Kimi still derived the correct upper bounds for $\|\mathbf{h}_{\{l\}}\|$ and $\|\Delta\mathbf{h}_{\{l\}}\|$

How Condition 1 gives the upper bounds

With $\mathbf{h}\ell = W\ell \phi(\mathbf{h}\ell_1)$ (ϕ an element-wise non-linearity with $|\phi(\cdot)| \leq 1$) we have

$$\|\mathbf{h}\ell\|_2 \leq \|W\ell\|_2 \|\phi(\mathbf{h}\ell_1)\|_2 \leq \|W\ell\|_2 \|\mathbf{h}\ell_1\|_2. \quad (7)$$

Assume inductively $\|\mathbf{h}\ell_1\|_2 = \Theta(\sqrt{n\ell_1})$ and use $\|W\ell\|_2 = \Theta(1)$:

$\|\mathbf{h}\ell\|_2 \leq \Theta(1) \cdot \Theta(\sqrt{n\ell_1}) = \Theta(\sqrt{n\ell})$ (since $n\ell = \Theta(n\ell_1)$ in typical wide networks).

The same argument holds for the update $\Delta\mathbf{h}\ell = W\ell \Delta(\phi(\mathbf{h}\ell_1))$, giving

$$\|\Delta\mathbf{h}\ell\|_2 \leq \|W\ell\|_2 \|\Delta\mathbf{h}\ell_1\|_2 = \Theta(1) \cdot \Theta(\sqrt{n\ell_1}) = \Theta(\sqrt{n\ell}).$$

Getting the lower bounds

The inequalities above only give upper bounds. The key extra assumption that

I'm not sure where Kimi got this quote about "sufficiently diverse/non-degenerate activations" because I didn't find it in the paper. The correct assumption for the lower bound should have been that the hidden weight matrices/matrix updates scale vectors by their spectral norm

“sufficiently diverse / non-degenerate activations”, namely

$$\text{RMS}(\mathbf{h}\ell(x)) \geq c > 0 \text{ and } \text{RMS}(\Delta\mathbf{h}\ell(x)) \geq c > 0 \quad (8)$$

for some constant c independent of width.

This is guaranteed (with high probability) by the paper's initialization and the fact that inputs come from a data distribution with non-zero variance and that the network is sufficiently random at initialization. Once (8) holds, (3) immediately gives

$$\|\mathbf{h}\ell\|_2 = \sqrt{n\ell} \cdot \text{RMS}(\mathbf{h}\ell) \geq c \sqrt{n\ell} = \Theta(\sqrt{n\ell}),$$

$$\|\Delta\mathbf{h}\ell\|_2 = \sqrt{n\ell} \cdot \text{RMS}(\Delta\mathbf{h}\ell) \geq c \sqrt{n\ell} = \Theta(\sqrt{n\ell}),$$

completing the $\Theta(\sqrt{n\ell})$ sandwich required by Desideratum 1.

Generated by Kimi AI

Deena Sun and Kimi chat 

 Reply Edit Delete ...

Deena Sun 5h

Question 4: Policy Gradient and the Reparameterization Gradient Estimator

One-shooting question
4a

In this question, you are going to derive two gradient estimators for the following objective function:

$$F(\theta) = \mathbb{E}_{x \sim p\theta}$$

$$f(x)$$

, (1)

where x is a random variable that follows the probability distribution of $p\theta$

: $X \rightarrow \Delta(X)$ that is parameterized by $\theta \in \mathbb{R}^P$, and $f : X \rightarrow \mathbb{R}$ is a function.

(a) Let x be a k -D multivariate Gaussian random variable that is parameterized by the mean $\mu \in \mathbb{R}^K$

under the distribution,

$$p\mu(x) = (2\pi\sigma)^{-n/2}$$

$$\exp \left[-\frac{1}{2\sigma^2} \|x - \mu\|^2 \right]$$

$$\frac{1}{(2\pi\sigma^2)^{n/2}} \exp \left[-\frac{1}{2\sigma^2} \|x - \mu\|^2 \right]$$

where $\sigma \in \mathbb{R}$ is a scalar constant. Express $\nabla_\mu F(\mu)$ as an expectation under $p\mu(x)$ (i.e., find $g(x)$ in

$$\nabla F(\mu) = \mathbb{E}_{x \sim p\mu}$$

$$g(x)$$

). (Hint: $\mathbb{E}x \sim p\theta$

$$\begin{aligned} f(x) \\ = \\ R \\ p\theta(x)f(x)dx \end{aligned}$$

Let's first restate the problem clearly.

We are given the objective function:

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

where the random variable $x \in \mathbb{R}^k$ follows a multivariate Gaussian distribution with mean $\mu \in \mathbb{R}^k$ and covariance matrix $\sigma^2 I$:

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

We want to find the gradient of $F(\mu)$ with respect to μ :

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

and express it as an expectation under the same distribution $p_\mu(x)$:

$$\nabla_\mu F(\mu) = \mathbb{E}_{x \sim p_\mu(x)}[g(x)]$$

for some function $g(x)$.

Step-by-step derivation:

We start from the definition of expectation:

$$F(\mu) = \mathbb{E}_{x \sim p_\mu(x)}[f(x)] = \int_{\mathbb{R}^k} p_\mu(x) f(x) dx$$

We differentiate under the integral sign:

$$\nabla_\mu F(\mu) = \nabla_\mu \int_{\mathbb{R}^k} p_\mu(x) f(x) dx = \int_{\mathbb{R}^k} \nabla_\mu p_\mu(x) f(x) dx$$

Next, we compute the gradient:

$$\nabla_\mu \log p_\mu(x) = \nabla_\mu$$

Thus, we have the identity:

I was a little unclear at first why we could relate $\nabla_\mu p_\mu(x)$ to $p_\mu(x)$. $\nabla_\mu \log(p_\mu(x))$ because Kimi didn't explicitly show that $\nabla_\mu \log(p_\mu(x)) = 1/p_\mu(x) * \nabla_\mu p_\mu(x)$. So I wonder if Kimi memorized this or truly reasoned through it

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

Substitute this back into the integral:

$$\nabla_\mu F(\mu) = \int_{\mathbb{R}^k} p_\mu(x) \frac{x - \mu}{\sigma^2} f(x) dx = \mathbb{E}_{x \sim p_\mu(x)} \left[\frac{x - \mu}{\sigma^2} f(x) \right]$$

Final Answer:

Thus, we have found explicitly the function $g(x)$:

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

with:

$$x - \mu \quad f(x)$$

$$g(x) = \frac{1}{\sigma^2} J(x)$$

One-shottting
question 4b.

Reviewing
the logs now,
I wonder if
Kimi would
have done as
well without
the hint.

Rewrite the expression in $F(\mu)$ with an expectation with the distribution over a standard normal $N(0, I)$? (Hint: if x follows the distribution of $p_\mu(x)$, then $x - \mu$ follows the distribution of $N(0, \sigma^2 I)$)

We are given:

$$F(\mu) = \mathbb{E}_{x \sim p_\mu(x)}[f(x)], \quad p_\mu(x) = \mathcal{N}(x | \mu, \sigma^2 I).$$

Introduce the **standard-normal** variable

$$z = \frac{x - \mu}{\sigma} \implies x = \mu + \sigma z, \quad z \sim \mathcal{N}(0, I).$$

Because this is a one-to-one, measure-preserving transformation, we can rewrite the expectation as an integral over z :

$$F(\mu) = \int_{\mathbb{R}^k} p_\mu(x) f(x) dx = \int_{\mathbb{R}^k} \phi(z) f(\mu + \sigma z) dz = \mathbb{E}_{z \sim \mathcal{N}(0, I)} [f(\mu + \sigma z)]$$

where $\phi(z) = (2\pi)^{-k/2} \exp(-\frac{1}{2}\|z\|^2)$ is the standard normal density.

Hence

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

Question 4c

Using the expression you obtained from the previous part, can you express $\nabla_\mu F(\mu)$ in a similar way (i.e., with an expectation with the distribution over $p_{\mu=0}(x)$)? (Hint: ∇_μ can be safely moved inside the expectation because the expectation no longer follows a distribution that depends on μ and expectations are linear.)

From the previous step we have

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

The distribution of z does **not** depend on μ , so we can interchange derivative and expectation:

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

Hence

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

Can you explain the step where you went from taking the gradient with respect to μ to taking the gradient with respect to x ?

Absolutely — that step is the **chain rule** applied to the composite function

$$f(\mu + \sigma z).$$

1. Variable identification

Inside the expectation we have

$$f(\mu + \sigma z),$$

where

- μ is the parameter we are differentiating with respect to,
- z is fixed (it is the integration variable drawn from $\mathcal{N}(0, I)$),
- $x = \mu + \sigma z$ is an **affine function of μ .**

2. Chain rule

For any differentiable f and any vector a ,

$$\frac{\partial}{\partial \mu} f(\mu + a) = \nabla_x f(x) \Big|_{x=\mu+a}.$$

Here $a = \sigma z$, so

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

3. Result

Because the expectation is over z only, we can push the derivative inside:

$$\nabla_\mu \mathbb{E}_z[f(\mu + \sigma z)] = \mathbb{E}_z[\nabla_\mu f(\mu + \sigma z)] = \mathbb{E}_z[\nabla_x f(\mu + \sigma z)].$$

That is the entire justification for “ $\nabla_\mu \rightarrow \nabla_x$.”

I wanted a

(Hint: ∇_μ can be safely moved

little more
clarification
on
generalizing
how to take
gradients
with
expectation

inside the expectation because the expectation no longer follows a distribution that depends on μ and expectations are linear.)

Please explain this hint more. How does the dependence of the distribution we are taking the expectation with respect to relate to the linearity of expectation?

The key point is **where the parameter μ appears.**

1. Original form

$$F(\mu) = \mathbb{E}_{x \sim p_\mu(x)}[f(x)] = \int p_\mu(x) f(x) dx.$$

Here **both** the integrand **and** the measure $p_\mu(x) dx$ depend on μ .

Therefore

$$\nabla_\mu F(\mu) = \int [\nabla_\mu p_\mu(x)] f(x) dx$$

is **not** the same as

$$\int p_\mu(x) \nabla_\mu f(x) dx,$$

because $\nabla_\mu p_\mu(x) \neq 0$.

You **cannot** move the derivative inside without extra work (score-function estimator, etc.).

2. After reparametrisation

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

I think when Kimi meant the measure $\mathcal{N}(0, I)$ is independent of μ , it meant that $p(\mu + \sigma z) = \phi(z)$ does not contain the parameter μ

Now the measure $\mathcal{N}(0, I)$ is independent of μ ; μ appears **only** inside the function f .

The expectation is a **linear operator** on functions of z , and the derivative with respect to a parameter that appears **only** in the integrand commutes with integration:

$$\nabla_\mu \int \phi(z) f(\mu + \sigma z) dz = \int \phi(z) \nabla_\mu f(\mu + \sigma z) dz.$$

Linearity lets us interchange the order; absence of μ in the measure guarantees that no extra “measure derivative” term arises.

So to summarize, after parametrization the distribution we are taking the expectation with respect to $(N(0, I))$ no longer depends on μ . So $\nabla_{\mu} \phi(z) = 0$, meaning when we apply the product rule $\nabla(g(x) * h(x)) = g'(x)h(x) + g(x)h'(x)$, only the term $\nabla_{\mu} f(\mu + \sigma z)$ is left. Which allows us to "move" the gradient inside the integration in the equation ∇

μ

$$\int_{\mu} \phi(z) f(\mu + \sigma z) dz = \int_{\mu} \phi(z) \nabla f(\mu + \sigma z) dz.$$

Clarifying Kimi's implicit use of the product rule when taking the derivative of the expectation

Exactly — you've captured the essence.

After the reparametrisation the **measure** is $\phi(z)dz$ and **does not contain μ** , so

$$\nabla_{\mu} \phi(z) = 0.$$

When you differentiate under the integral you apply the product rule:

$$\begin{aligned} \nabla_{\mu} [\phi(z) f(\mu + \sigma z)] &= [\nabla_{\mu} \phi(z)] f(\mu + \sigma z) + \phi(z) [\nabla_{\mu} f(\mu + \sigma z)] \\ &= 0 + \phi(z) [\nabla_{\mu} f(\mu + \sigma z)]. \end{aligned}$$

Integrating term-by-term gives

$$\nabla_{\mu} \int \phi(z) f(\mu + \sigma z) dz = \int \phi(z) \nabla_{\mu} f(\mu + \sigma z) dz,$$

which is exactly "moving the gradient inside."

Question 4d

For the following two parts, we are going to generalize our findings from the specific Gaussian case to arbitrary p_{θ} 's (you may assume p_{θ} has well-defined gradient $\nabla_{\theta} p_{\theta}$ that expresses what change in parameters

would make a particular realization more likely).
(d) In the general case, can you still write $\nabla F(\theta)$ as an expectation under $p_\theta(x)$? (Hint: look at your answer in Part (a) and see how the term inside your expectation relates to $\log p_\theta(x)$).

Yes — the same “log-trick” (score-function identity) works for **any** distribution $p_\theta(x)$ that is differentiable w.r.t. θ .

Start from

$$F(\theta) = \mathbb{E}_{x \sim p_\theta}[f(x)] = \int p_\theta(x) f(x) dx.$$

Differentiate under the integral:

$$\nabla_\theta F(\theta) = \int \nabla_\theta p_\theta(x) f(x) dx.$$

Use the identity

$$\nabla_\theta p_\theta(x) = p_\theta(x) \nabla_\theta \log p_\theta(x).$$

Insert it:

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

Thus the general score-function (REINFORCE) estimator is

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

First of all, why were you able to directly move the

gradient inside the integral without using the product rule I was asking about earlier? Doesn't x depend on θ ? So wouldn't $\nabla_\theta f(x)$ be non-zero?

∇

θ

$$F(\theta) = \int \nabla$$

θ

p

θ

$$(x) f(x) dx$$

The integral is w.r.t. the dummy variable x , not w.r.t. θ .

When we write

$$\nabla_\theta \int p_\theta(x) f(x) dx$$

we are differentiating a **parameterised family of integrals**; the variable of integration x is **not a function of θ** .

(Think of x as the integration variable that runs over the whole space; its symbol is unaffected by θ .)

Hence the only θ -dependence inside the integral is **explicitly** in the density $p_\theta(x)$; the function $f(x)$ itself has **no explicit θ -dependence**.

Therefore

$$\nabla_\theta [p_\theta(x) f(x)] = [\nabla_\theta p_\theta(x)] f(x) + p_\theta(x) \underbrace{\nabla_\theta f(x)}_{=0},$$

and the second term vanishes.

That is why we can safely write

$$\nabla_{\theta} F(\theta) = \int \nabla_{\theta} p_{\theta}(x) f(x) dx$$

without invoking any reparametrisation or product-rule complications.

Earlier you said this:

Original form

$$F(\mu) = E$$

$$x \sim p$$

$$\mu$$

$$(x)$$

$$[f(x)] = \int p$$

$$\mu$$

$$(x) f(x) dx.$$

Here both the integrand and the measure p

$$\mu$$

$(x) dx$ depend on μ .

Therefore

$$\nabla$$

$$\mu$$

$$F(\mu) = \int [\nabla$$

$$\mu$$

$$p$$

$$\mu$$

$$(x)] f(x) dx$$

is not the same as

$$\int p$$

$$F(\mu) = E_{\{x \sim p_{\mu}(x)\}}$$

$$[f(x)] = \int p_{\mu}(x) f(x) dx$$

Here, both the integrand $f(x) dx$ and the measure $p_{\mu}(x)$ depend on μ .

Therefore $\nabla_{\{\mu\}} F(\mu) = \int [\nabla_{\{\mu\}} p_{\mu}(x)] f(x) dx$ is not the same as $\int p_{\mu}(x) \nabla_{\{\mu\}} f(x) dx$.

ux

μ

$(x)\nabla$

μ

$f(x)dx,$

because ∇

μ

p

μ

(x)

$=0.$

I wanted more
clarification on how
gradients under the
expectation's integral
generalized to
different
parametrizations of
probability
distributions

Please explain how these 2 cases differ. I'm not quite
sure why $\nabla_\theta f(x)dx$ is zero because $f(x)$ has no
dependence on theta for part d, but not for part a

The difference is where the parameter appears.

TLDR: In part a, $\nabla_{\{\mu\}} f(\mu+\sigma z)$ is non-zero
because $f(\mu+\sigma z)$ is dependent on μ
whereas $\Phi(z)$ is independent of μ . But in
part d, $\nabla_{\{\theta\}} f(x) = 0$ because $f(x)$ doesn't
contain the variable θ we're taking the
gradient w.r.t. and $\nabla_{\{\theta\}} p_\theta(x)$ is non-zero.

Therefore

- Part (a) after reparametrisation:
 $\nabla_\mu f(\mu + \sigma z)$ is non-zero because μ is explicitly inside f .
- Part (d) without reparametrisation:
 $\nabla_\theta f(x)$ is zero because x is just the dummy integration variable;
no θ appears in the symbolic expression $f(x)$.

That is why the product-rule term survives in (a) but vanishes in (d).

I thought that when we had the "Original form

$$F(\mu) = E$$

$$x \sim p$$

$$\mu$$

$$(x)$$

$$[f(x)] = \int p$$

$$\mu$$

$$(x)f(x)dx.$$

Here both the integrand and the measure p

$$\mu$$

$(x)dx$ depend on μ ", we hadn't re-parametrized f yet into $f(\mu + \sigma z)$. Does $f(x)dx$ still depend on μ ?

No — in the **original** (non-reparametrised) integral

$$F(\mu) = \int p_\mu(x) f(x) dx$$

the **symbolic function** $f(x)$ itself does not contain μ ; its only μ -dependence is **implicit** through the **measure** $p_\mu(x)dx$.

That is exactly why we **cannot** write $\int p_\mu(x) \nabla_\mu f(x) dx$ there — $\nabla_\mu f(x)$ is zero because x is the dummy integration variable and no μ appears in the expression $f(x)$.

The phrase “both the integrand and the measure depend on μ ” was meant informally: the **value** of the integral changes with μ , but the **functional form** $f(x)$ is untouched until we **choose** to reparametrise.

1. Where the identity comes from

Start with the **definition** of the logarithmic derivative:

$$\nabla_{\theta} \log p_{\theta}(x) = \frac{1}{p_{\theta}(x)} \nabla_{\theta} p_{\theta}(x).$$

Multiply both sides by $p_{\theta}(x)$:

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

That is all there is to it—just the chain rule written backwards.

2. How the term inside the expectation relates to $\log p_{\theta}(x)$

In part (a) we had a Gaussian with mean μ and obtained

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

Notice that

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

because

$$\log p_{\mu}(x) = \text{const} - \frac{\|x - \mu\|^2}{2\sigma^2} \implies \nabla_{\mu} \log p_{\mu}(x) = \frac{x - \mu}{\sigma^2}.$$

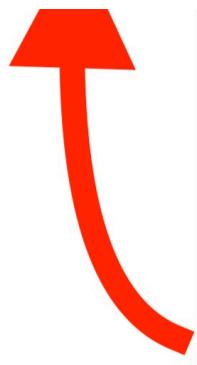
Hence the quantity multiplying $f(x)$ in the expectation is **exactly** the **score function** $\nabla_{\theta} \log p_{\theta}(x)$ for the Gaussian location model.

Part (d) simply **generalises** this observation to **any** differentiable p_{θ} :

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



Kimi's response to my



answer to my
question about the identity
 $\nabla_{\{\theta\}} p_{\{\theta\}}(x) = p_{\{\theta\}}(x)$
 $\nabla_{\{\theta\}} \log(p_{\{\theta\}}(x))$

∇_{θ}
 p
 θ

$(x)=p$

θ

$(x)\nabla$

θ

$\log p$

θ

(x)

Please explain how you got this identity.

Also, please explain how the term inside our
expectation from part relates to $\log p$

θ

(x)

Question 4d

Assume there exists a function $g(z, \theta) : Z \times \mathbb{R}$
 $P \rightarrow X$ and a distribution over z , $p(z)$, such that $g(z, \theta)$
has the same distribution as $x \sim p_\theta(x)$. Prove that

$$\nabla_\theta F(\theta) = E_{z \sim Z} h$$

$$\nabla_\theta g(z)$$

$$+ \nabla_x f|_{x=g(z,\theta)}$$

i

.

we are given the reparameterisation

$$x = g(z, \theta), \quad z \sim p(z) \text{ (independent of } \theta\text{)},$$

so

$$F(\theta) = E_{\{x \sim p(\theta)\}}[f(x)] = E_{\{z \sim p\}}[f(g(z, \theta))].$$

Because the **measure** $p(z)$ does **not** depend on θ , we can differentiate under the expectation:

$$\nabla_\theta F(\theta) = E_{\{z \sim p\}}[\nabla_\theta f(g(z, \theta))].$$

By the **chain rule** for a scalar-valued composite function,

$$\begin{aligned} \nabla_\theta f(g(z, \theta)) &= (\nabla_x f)_{\{x=g(z,\theta)\}} \cdot \nabla_\theta g(z, \theta) \\ &= \nabla_\theta g(z, \theta)^T \nabla_x f_{\{x=g(z,\theta)\}}. \end{aligned}$$

Insert this back:

$$\nabla_\theta F(\theta) = E_{\{z \sim p\}}[\nabla_\theta g(z, \theta)^T \nabla_x f_{\{x=g(z,\theta)\}}],$$

which is the required identity.

I asked some more clarifications on distributions and reparameterizations in this context

the measure $p(z)$ does not depend on θ . Does that mean? I thought we were at the distribution over z , $p(z)$ follows the distribution as $x \sim p_\theta(x)$. So wouldn't $p(z)$ be dependent on θ ?

“ $p(z)$ does **not** depend on θ ” means **the density function $p(z)$ has no θ in its formula**; the **random variable z is fixed** (e.g. standard normal) and θ **only** enters through the **deterministic map $g(z, \theta)$** that turns z into x .

We do **not** assume “ z has the same distribution as x ”.

We assume the **opposite**:

$$x = g(z, \theta) \quad \text{with} \quad z \sim p(z) \quad (\text{independent of } \theta)$$

and we **choose** $g(\cdot, \theta)$ so that the **induced random variable $x = g(z, \theta)$**

has the **same distribution** as the original $x \sim p_{\theta}(x)$.

Thus all θ -dependence is **pushed into the transformation g** , while the **base measure $p(z)$** stays θ -free.

Follow up for question 4d

In the previous parts (b) and (c), we prove a special case of $\nabla_{\theta} F(\theta) = E_{z \sim Z} h$
 $\nabla_{\theta} g(z)$
 $\top \nabla_x f | x=g(z, \theta)$
i

What are g and $p(z)$ for this special case?

For the Gaussian location model

$$x \sim \mathcal{N}(\mu, \sigma^2 I)$$

we reparametrise via

$$z \sim \mathcal{N}(0, I) \quad (\text{independent of } \mu)$$

and set

$$g(z, \mu) = \mu + \sigma z.$$

Kimi derived a similar answer to the solutions, but instead used a identity matrix as the variance of $p(z)$ and applied an additional scaling of σ to z in $g(z, \theta)$

Then

$x = g(z, \mu)$ has the same distribution $\mathcal{N}(\mu, \sigma^2 I)$,

so

$$p(z) = \mathcal{N}(z | 0, I) \quad \text{and} \quad g(z, \mu) = \mu + \sigma z$$

are the base distribution and the transformation used in parts (b)–(c).

Generated by Kimi AI

Deena Sun and Kimi chat



Reply Edit Delete ...



Deena Sun 5h

Question 5: Tensor Rematerialization

First providing all the context for question 5a, and then attempting to one-shot 5a.

Although I didn't include the actual image for figure 1 because I was using Kimi 2 without vision, I still included the figure caption + description

You want to train a neural network on a new chip designed at UC Berkeley. Your model is a 10 layer network, where each layer has the same fixed input and output size of s . The chip your model will be trained on is heavily specialized for model evaluation. It can run forward passes through a layer very fast. However, it is severely memory constrained, and can only fit in memory the following items (slightly more than twice of the data necessary for performing a forward pass):
(a) the inputs;
(b) $2s$ activations in memory;
(c) optimizer states necessary for performing the forward pass through the current layer.
To train despite this memory limitation, your friend suggests using a training method called tensor rematerialization. She proposes using SGD with a batch size of 1, and only storing the activations of every 5th layer during an initial forward pass to evaluate the model. During backpropagation, she suggests recomputing activations on-the-fly for each layer by loading the relevant last stored activation from memory, and rerunning forward through layers up till the current layer.
Figure 1 illustrates this approach. Activations for Layer 5 and Layer 10 are stored in memory from an initial forward pass through all the layers. Consider

when weights in layer 7 are to be updated during backpropagation. To get the activations for layer 7, we would load the activations of layer 5 from memory, and then run them through layer 6 and layer 7 to get the activations for layer 7. These activations can then be used (together with the gradients from upstream) to compute the gradients to update the parameters of Layer 7, as well as get ready to next deal with layer 6.

Assume a forward pass of a single layer is called a fwd operation. How many fwd operations are invoked when running a single backward pass through the entire network? Do not count the initial forward passes required to compute the loss, and don't worry about any extra computation beyond activations to actually backprop gradients.

Understanding the Problem

We have a 10-layer neural network where each layer has the same input and output size s . The chip we're using is memory-constrained, allowing us to store:

1. The inputs.
2. $2s$ activations in memory.
3. Optimizer states necessary for the current layer's forward pass.

For this question, Kimi's response contained a lot more question restating. Maybe because this question was more storyline-based rather than math derivations or paper comprehension?

Given these constraints, we can't store all intermediate activations during a forward pass. Instead, we use **tensor rematerialization**, where we only store activations every 5th layer (i.e., Layer 5 and Layer 10 in this case). During backpropagation, we recompute the necessary activations on-the-fly by loading the last stored activation and running forward passes up to the current layer.

Training Method Details

- **Optimizer:** SGD with batch size 1.
- **Stored Activations:** Only Layer 5 and Layer 10 after the initial forward pass.
- **Backward Pass:** To update weights in a layer, we load the nearest stored activation before it, then run forward passes up to that layer to get its activations.

Goal

Determine how many `fwd` (forward pass of a single layer) operations are invoked during a single backward pass through the entire network, excluding the initial forward passes to compute the loss.

Breaking Down the Backward Pass

During backpropagation, we process layers in reverse order (from Layer 10 down to Layer 1). For each layer, we need its activations to compute gradients. Since we only have Layer 5 and Layer 10 stored, we'll need to recompute activations for other layers.

Let's consider how we handle each layer during the backward pass:

1. Layer 10:

- Activations are already stored (from initial forward pass).
- No need to recompute.
- `fwd` operations: 0.

2. Layer 9:

- Need activations for Layer 9.
- Last stored activation before Layer 9 is Layer 5.
- Load Layer 5, then run forward through Layer 6, Layer 7, Layer 8, Layer 9.
- That's 4 `fwd` operations (Layers 6,7,8,9).

3. Layer 8:

- Need activations for Layer 8.
- Last stored is still Layer 5.
- Run forward through Layer 6, Layer 7, Layer 8.
- 3 fwd operations.

4. Layer 7:

- Need activations for Layer 7.
- Last stored is Layer 5.
- Run forward through Layer 6, Layer 7.
- 2 fwd operations.

5. Layer 6:

- Need activations for Layer 6.
- Last stored is Layer 5.
- Run forward through Layer 6.
- 1 fwd operation.

6. Layer 5:

- Activations are stored.
- No fwd needed.
- 0 fwd operations.

Kimi crafted a long response to this question, but I really liked how step-by-step its response was. It laid out the exact operations needed for every layer, without immediately jumping to the answer by extrapolating. For example, sometimes I feel like LLMs have a tendency to regurgitate theorems/identities/formulas to plug directly into their derivation without always explaining where they came from. But then, students may not learn how to re-apply the same problem-solving strategies or theorems in new, unseen cases

7. Layer 4:

- Need activations for Layer 4.
- Last stored before Layer 4 is none (since we start storing at Layer 5).
- Assuming we have the input stored, we'd run forward from input through Layer 1 to Layer 4.
- But the problem states we store inputs, so we can consider input as "stored".
- Run forward through Layer 1, Layer 2, Layer 3, Layer 4.
- 4 fwd operations.

8. Layer 3:

- Need activations for Layer 3.
- Last stored is input.
- Run forward through Layer 1, Layer 2, Layer 3.
- 3 fwd operations.

9. Layer 2:

- Need activations for Layer 2.
- Last stored is input.
- Run forward through Layer 1, Layer 2.
- 2 fwd operations.

10. Layer 1:

- Need activations for Layer 1.
- Last stored is input.
- Run forward through Layer 1.
- 1 fwd operation.

Summing Up `fwd` Operations

Now, let's add up the `fwd` operations for each layer during the backward pass:

- Layer 10: 0
- Layer 9: 4
- Layer 8: 3
- Layer 7: 2
- Layer 6: 1
- Layer 5: 0
- Layer 4: 4
- Layer 3: 3
- Layer 2: 2
- Layer 1: 1

I think this was an informative way for Kimi to structure its response: provide straightforward step-by-step thinking for this problem, then generalize to a pattern

Total `fwd` operations = $0 + 4 + 3 + 2 + 1 + 0 + 4 + 3 + 2 + 1 = 20$

Verifying the Pattern

Looking at the pattern, it seems like for every group of 5 layers, the `fwd` operations count down from 4 to 0. Specifically:

- Layers 6–10: 4,3,2,1,0 (sum=10)
 - Layers 1–5: 4,3,2,1,0 (sum=10)
- Total: 20

This makes sense because for each layer not stored, we recompute by running forward from the last stored point, and the number of layers to run forward decreases as we move closer to the stored activation.

Final Answer

After carefully working through each layer, the total number of `fwd`

operations invoked during a single backward pass through the entire network is:

20

Trying
question
5b with
Kimi

Assume that each memory access to fetch activations or inputs is called a loadmem operation. How many loadmem operations are invoked when running a single backward pass?

We are now counting **loadmem operations**, defined as:

Each memory access to **fetch activations or inputs** is called a **loadmem** operation.

We are **not** counting:

- Initial forward pass (as before),
- Any memory access for weights, gradients, or optimizer states,
- Any write to memory (only **reads** of activations or inputs).

🔍 Key Insight

During the backward pass, **every time we need to start recomputing activations**, we must **load the last stored activation** (or the input, if no prior stored activation exists).

From the previous analysis, we recompute activations for layers in two blocks:

1. **Layers 6–9**: recompute from **Layer 5** activations.
2. **Layers 1–4**: recompute from **input**.

Each of these recomputation sequences begins with **one loadmem operation** to fetch the starting point (Layer 5 or input).

✓ Counting **loadmem** Operations

Let's go layer-by-layer:

I'm not sure why Kimi omitted the loadmems for layer 10 and layer 5. Kimi states that they

| Table | | <input type="checkbox"/> Copy |
|-------|------------------|-------------------------------|
| Layer | Recompute from | loadmem ? |
| 10 | (already stored) | 0 |
| 9 | Layer 5 | 1 |
| 8 | Layer 5 | 1 |
| 7 | Layer 5 | 1 |
| 6 | Layer 5 | 1 |
| 5 | (stored) | 0 |
| 4 | input | 1 |
| 3 | input | 1 |
| 2 | input | 1 |
| 1 | input | 1 |

Total loadmem Operations

- From Layer 5: 4
- From input: 4

Total = 8

Final Answer

8 loadmem operations are invoked during a single backward pass.

are already stored, but doesn't account for the fact that we still have to load the stored activations for them—which still counts as a loadmem

Even when I questioned why we didn't use loadmem for layer 5 and 10, Kimi restated the problem stem but seemed to misinterpret existing in memory as not need a loadmem, even though that would be incorrect. Unlike when Kimi got the Adam LR parameters in question 3 wrong, this time Kimi was confident in its incorrect answer and refused to change its mind.

Do we not need to also loadmem for layers 5 and 10 because the activations for those are stored in memory?

No — we **do not** count a `loadmem` for layers 5 or 10 in this question, because the definition we were given is

“Each memory access to **fetch activations or inputs** is called a `loadmem` operation.”

but we are **only asked to count the operations that are *invoked*** (i.e. explicitly executed) during the backward pass.

- The activations of layers 5 and 10 are **already sitting in memory** as a result of the **initial forward pass** (which we are told **not** to include).
- During back-prop we **use** those tensors, but we **do not** execute any **new** memory fetch instruction to obtain them; they are simply **already resident**.

Hence no additional `loadmem` is invoked for layers 5 or 10.

Total remains 8.

What if the problem actually does want us to include the fact that we still need to load the activations of layer 5 and 10 from memory when we compute backprop for them? Even though these activations were stored and computed in the forward pass, we still need to load them from memory

Then we simply count **one extra** `loadmem` each time the backward algorithm actually touches those tensors.

- When the back-prop wave reaches **layer 10** we must **read** its activation to compute gradients w.r.t. its inputs/weights → **+1 loadmem**.
- When it later reaches **layer 5** we must **read** its activation → **+1 loadmem**.

All other `loadmem` counts stay the same (the 8 we already had).

New total:

8 (re-computation starts)

+1 (layer 10)

+1 (layer 5)

= **10 `loadmem` operations** in the backward pass.

Question 5c:
for this question, I decided to try a different approach by explicitly breaking the problem down into smaller steps. I started off by asking Kimi to

Say you have access to a local disk which offers practically infinite storage for activations and a `loaddisk` operation for loading activations. You decide to not use tensor rematerialization and instead store all activations on this disk, loading each activation when required. Assuming each fwd operation takes 20ns and each `loadmem` operation (which loads from memory, not local disk) takes 10ns for tensor rematerialization, how fast (in ns) should each `loaddisk` operation be to take the

compute how many loaddisks would be needed, and plan to use that to eventually calculate how fast each loaddisk operation should be.

same time for one backward pass as tensor rematerialization? Assume activations are directly loaded to the processor registers from disk (i.e., they do not have to go to memory first), only one operation can be run at a time, ignore any caching and assume latency of any other related operations is negligible.

First of all, how many loaddisks are needed?

We need the **number of loaddisk operations** that would be executed in the “store–everything–on–disk” scheme.

In that scheme **no activations are recomputed**; instead, **every layer’s activation is written to disk after its forward pass**, and **every layer’s activation is read from disk once during the backward pass**.

There are 10 layers, so:

- **10 activations are stored** (one per layer) — but storing is free for this count.
- **10 activations are loaded** during back–prop (one for each layer, in reverse order).

Hence

10 loaddisk operations are needed.

How fast should each loaddisk be?

We want the **total time** of the disk–based backward pass to equal the **total time** of the tensor–rematerialization backward pass.

1. Rematerialization time (already computed)

- **fwd ops: 20**



- loadmem ops: 8
- Time = $20 \times 20 \text{ ns} + 8 \times 10 \text{ ns} = 400 \text{ ns} + 80 \text{ ns} = 480 \text{ ns}$

2. Disk-based time

- fwd ops: 0 (no recomputation)
- loaddisk ops: 10
- Time = $10 \times t_{\text{disk}}$

Set them equal:

$$10 t_{\text{disk}} = 480 \text{ ns}$$

$$t_{\text{disk}} = 48 \text{ ns}$$

Kimi kept using its incorrect calculation that we would need 8 loadmems instead of 10, even despite my correction attempts. I think it took my earlier prompt about also needing to load the activations of layer 5 and 10 as a hypothetical.

Each loaddisk operation must complete in **48 nanoseconds** to match the rematerialization latency.

We need 10 loadmem operations, not just 8. How fast should each loaddisk be?

Corrected counts with the **two extra reads** (layers 5 and 10):

- fwd ops: 20
- loadmem ops: 10
- Rematerialization time = $20 \times 20 \text{ ns} + 10 \times 10 \text{ ns} = 400 \text{ ns} + 100 \text{ ns} = 500 \text{ ns}$

Disk-based backward:

$$10 \text{ loaddisk ops, time} = 10 \times t_{\text{disk}}$$

Equate:

$$10 t_{\text{disk}} = 500 \text{ ns}$$

$$t_{\text{disk}} = 50 \text{ ns}$$

Finally, if I explicitly ask Kimi to compute the loaddisk time using 10 loadmems, it arrives at the right answer.

Each loaddisk must finish in **50 nanoseconds**.



Reply Edit Delete ...