Paul Struble
3038299344
Special Participation A

HW3 (Non-Coding Parts) - GPT 5.1 Thinking (Extended)

Executive Summary

In this investigation, I used GPT 5.1 Thinking (Extended) to solve the non-coding parts of Homework 3. Overall, the model was very effective at solving each problem and explaining its reasoning. It was able to one-shot all parts of all problems. I prompted the model in the ChatGPT web frontend by providing a PDF attachment of the original homework assignment, a brief explanation of the task, and some additional prompts/attachments throughout the conversation to provide additional contextual resources (from the assignment). This is all recorded in the conversation log in this document. I found no misconceptions or hallucinations in the model's output although some responses took a different approach than the reference solutions (ultimately still arriving at a valid solution).

Link to Conversation: https://chatgpt.com/share/693a069f-44a8-8007-bed7-a4db5aceaa8f

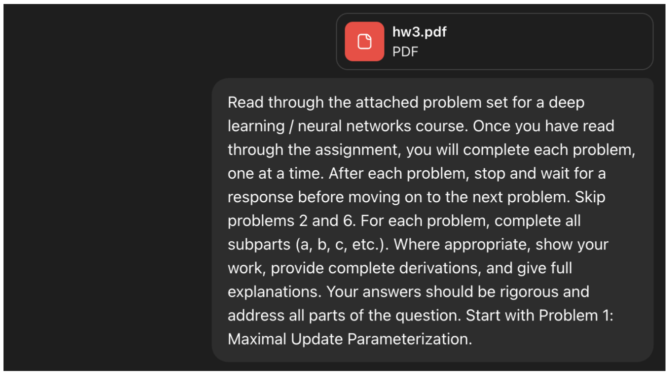
Model Details:

- Model: ChatGPT 5.1 Thinking (Extended)
- Used ChatGPT web frontend (chatgpt.com)
- Disabled all personalization features
- Used "default" personality
- Chain of thought ("Thinking") traces can be found in the original conversation (linked above)

Initial Prompt

User Prompt:

[Attachment: Homework 3 PDF]

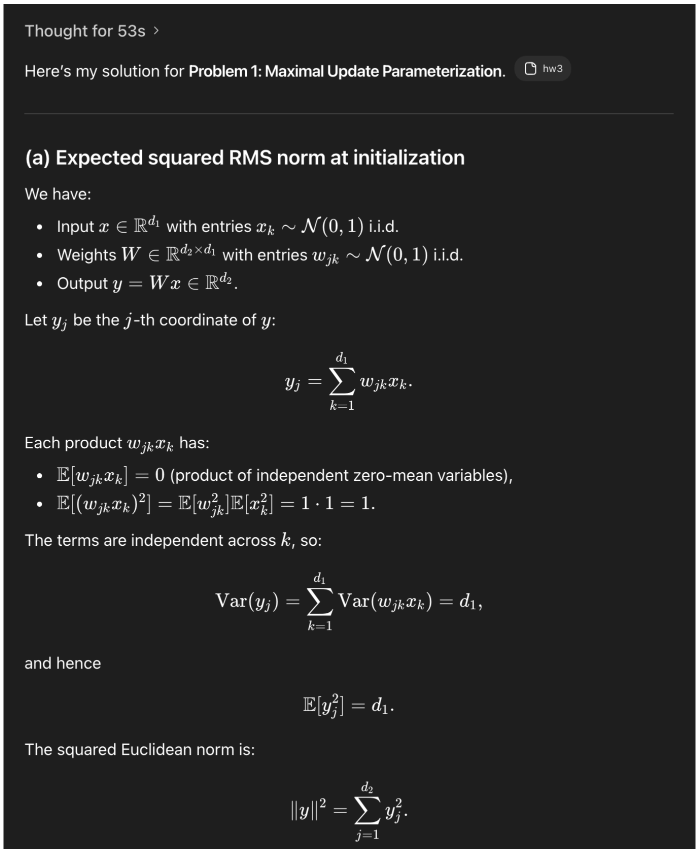


hw3.pdf
PDF

Read through the attached problem set for a deep learning / neural networks course. Once you have read through the assignment, you will complete each problem, one at a time. After each problem, stop and wait for a response before moving on to the next problem. Skip problems 2 and 6. For each problem, complete all subparts (a, b, c, etc.). Where appropriate, show your work, provide complete derivations, and give full explanations. Your answers should be rigorous and address all parts of the question. Start with Problem 1: Maximal Update Parameterization.

Problem 1: Maximal Update Parameterization

AI Response:



Thought for 53s ›

Here's my solution for **Problem 1: Maximal Update Parameterization.** hw3

(a) Expected squared RMS norm at initialization

We have:

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

Let y_j be the j-th coordinate of y:

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

Each product $w_{jk} x_k$ has:

- $\mathbb{E}[w_{jk} x_k] = 0$ (product of independent zero-mean variables),
- $\mathbb{E}[(w_{jk} x_k)^2] = \mathbb{E}[w_{jk}^2]\mathbb{E}[x_k^2] = 1 \cdot 1 = 1.$

The terms are independent across k, so:

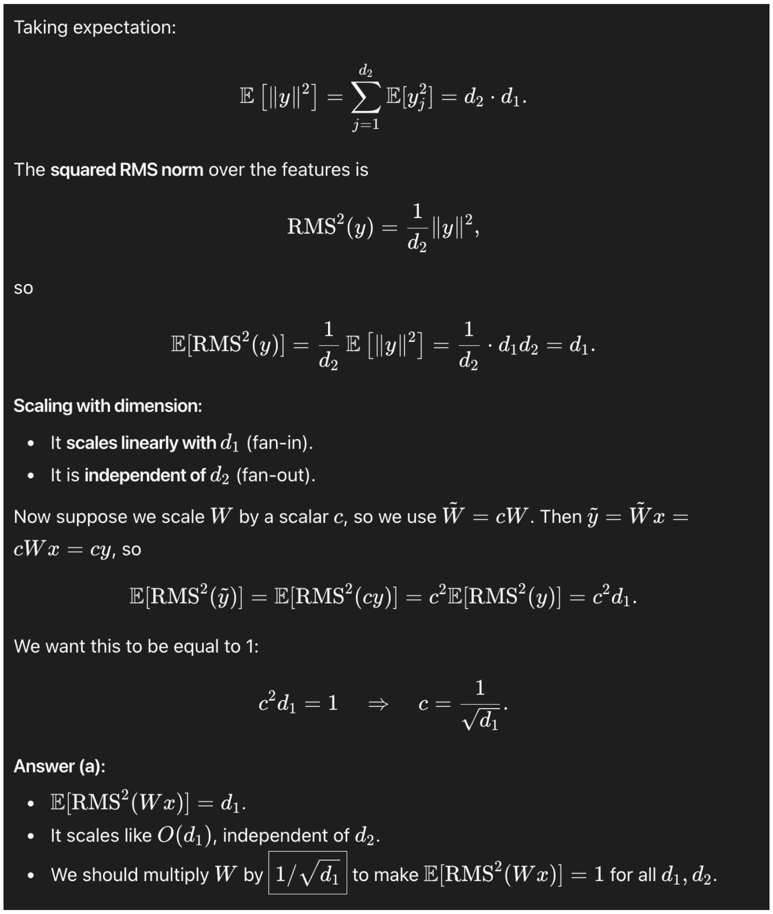
$$\mathrm{Var}(y_j) = \sum_{k=1}^{d_1} \mathrm{Var}(w_{jk} x_k) = d_1,$$

and hence

$$\mathbb{E}[y_j^2] = d_1.$$

The squared Euclidean norm is:

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



Taking expectation:

$$\mathbb{E}\left[\|y\|^2\right] = \sum_{j=1}^{d_2} \mathbb{E}[y_j^2] = d_2 \cdot d_1.$$

The **squared RMS norm** over the features is

$$\mathrm{RMS}^2(y) = \frac{1}{d_2}\|y\|^2,$$

so

$$\mathbb{E}[\mathrm{RMS}^2(y)] = \frac{1}{d_2}\,\mathbb{E}\left[\|y\|^2\right] = \frac{1}{d_2} \cdot d_1 d_2 = d_1.$$

Scaling with dimension:

- It **scales linearly with** d_1 (fan-in).
- It is **independent of** d_2 (fan-out).

Now suppose we scale W by a scalar c, so we use $\tilde{W} = cW$. Then $\tilde{y} = \tilde{W}x = cWx = cy$, so

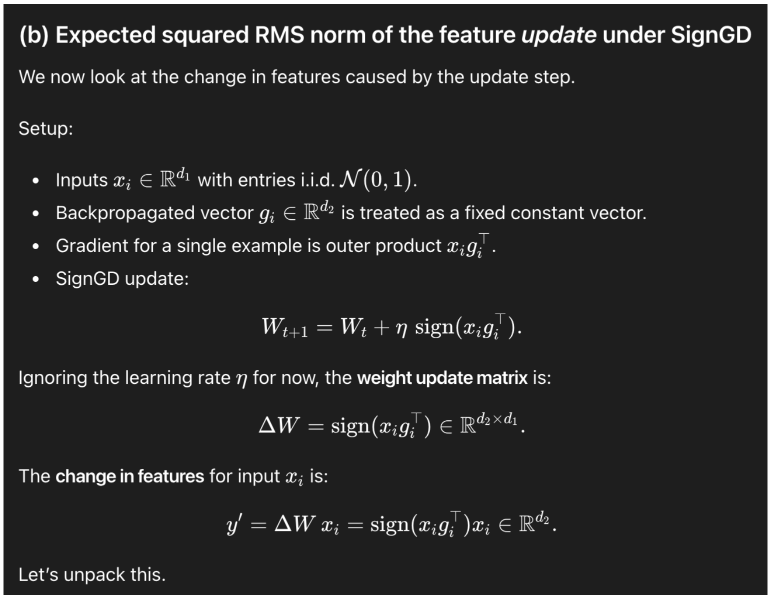
$$\mathbb{E}[\mathrm{RMS}^2(\tilde{y})] = \mathbb{E}[\mathrm{RMS}^2(cy)] = c^2\mathbb{E}[\mathrm{RMS}^2(y)] = c^2 d_1.$$

We want this to be equal to 1:

$$c^2 d_1 = 1 \quad \Rightarrow \quad c = \frac{1}{\sqrt{d_1}}.$$

Answer (a):

- $\mathbb{E}[\mathrm{RMS}^2(Wx)] = d_1$.
- It scales like $O(d_1)$, independent of d_2.
- We should multiply W by $\boxed{1/\sqrt{d_1}}$ to make $\mathbb{E}[\mathrm{RMS}^2(Wx)] = 1$ for all d_1, d_2.



(b) Expected squared RMS norm of the feature *update* under SignGD

We now look at the change in features caused by the update step.

Setup:

- Inputs $x_i \in \mathbb{R}^{d_1}$ with entries i.i.d. $\mathcal{N}(0,1)$.
- Backpropagated vector $g_i \in \mathbb{R}^{d_2}$ is treated as a fixed constant vector.
- Gradient for a single example is outer product $x_i g_i^\top$.
- SignGD update:

$$W_{t+1} = W_t + \eta\ \mathrm{sign}(x_i g_i^\top).$$

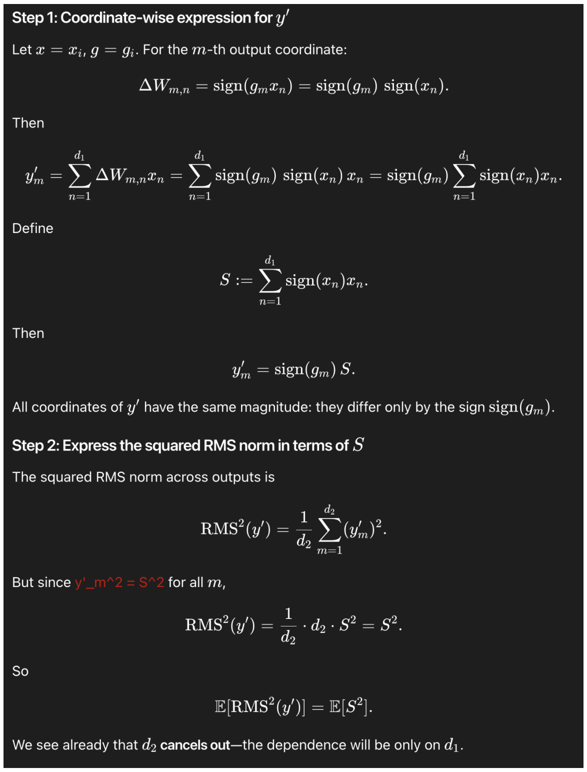
Ignoring the learning rate η for now, the **weight update matrix** is:

$$\Delta W = \mathrm{sign}(x_i g_i^\top) \in \mathbb{R}^{d_2 \times d_1}.$$

The **change in features** for input x_i is:

$$y' = \Delta W\, x_i = \mathrm{sign}(x_i g_i^\top) x_i \in \mathbb{R}^{d_2}.$$

Let's unpack this.



Step 1: Coordinate-wise expression for y'

Let $x = x_i$, $g = g_i$. For the m-th output coordinate:

$$\Delta W_{m,n} = \operatorname{sign}(g_m x_n) = \operatorname{sign}(g_m)\ \operatorname{sign}(x_n).$$

Then

$$y'_m = \sum_{n=1}^{d_1} \Delta W_{m,n} x_n = \sum_{n=1}^{d_1} \operatorname{sign}(g_m)\ \operatorname{sign}(x_n)\, x_n = \operatorname{sign}(g_m) \sum_{n=1}^{d_1} \operatorname{sign}(x_n) x_n.$$

Define

$$S := \sum_{n=1}^{d_1} \operatorname{sign}(x_n) x_n.$$

Then

$$y'_m = \operatorname{sign}(g_m)\, S.$$

All coordinates of y' have the same magnitude: they differ only by the sign $\operatorname{sign}(g_m)$.

Step 2: Express the squared RMS norm in terms of S

The squared RMS norm across outputs is

$$\mathrm{RMS}^2(y') = \frac{1}{d_2} \sum_{m=1}^{d_2} (y'_m)^2.$$

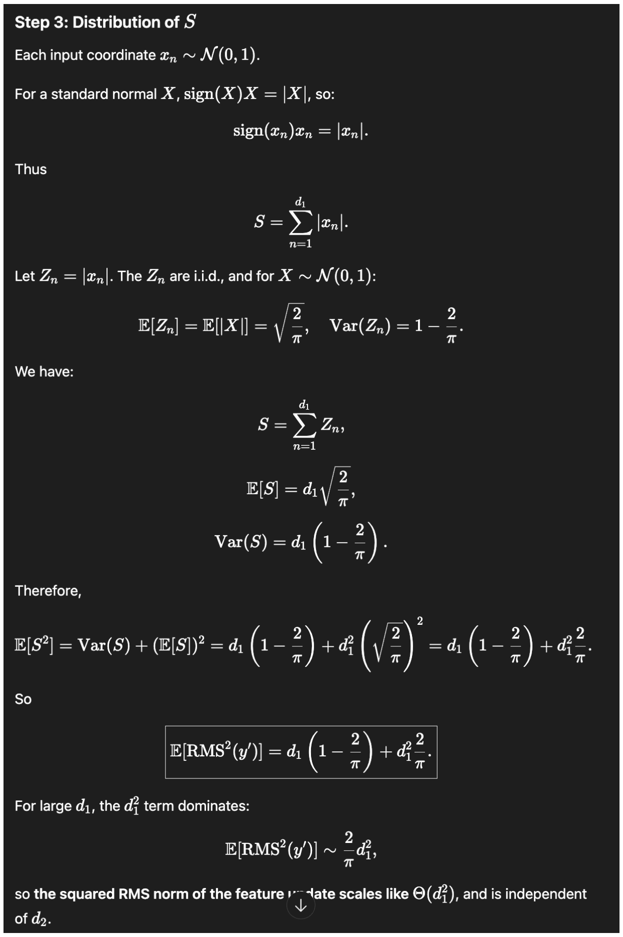
But since y'_m^2 = S^2 for all m,

$$\mathrm{RMS}^2(y') = \frac{1}{d_2} \cdot d_2 \cdot S^2 = S^2.$$

So

$$\mathbb{E}[\mathrm{RMS}^2(y')] = \mathbb{E}[S^2].$$

We see already that d_2 **cancels out**—the dependence will be only on d_1.



Step 3: Distribution of S

Each input coordinate $x_n \sim \mathcal{N}(0, 1)$.

For a standard normal X, $\operatorname{sign}(X)X = |X|$, so:

$$\operatorname{sign}(x_n) x_n = |x_n|.$$

Thus

$$S = \sum_{n=1}^{d_1} |x_n|.$$

Let $Z_n = |x_n|$. The Z_n are i.i.d., and for $X \sim \mathcal{N}(0, 1)$:

$$\mathbb{E}[Z_n] = \mathbb{E}[|X|] = \sqrt{\frac{2}{\pi}}, \quad \mathrm{Var}(Z_n) = 1 - \frac{2}{\pi}.$$

We have:

$$S = \sum_{n=1}^{d_1} Z_n,$$

$$\mathbb{E}[S] = d_1 \sqrt{\frac{2}{\pi}},$$

$$\mathrm{Var}(S) = d_1 \left(1 - \frac{2}{\pi}\right).$$

Therefore,

$$\mathbb{E}[S^2] = \mathrm{Var}(S) + (\mathbb{E}[S])^2 = d_1 \left(1 - \frac{2}{\pi}\right) + d_1^2 \left(\sqrt{\frac{2}{\pi}}\right)^2 = d_1 \left(1 - \frac{2}{\pi}\right) + d_1^2 \frac{2}{\pi}.$$

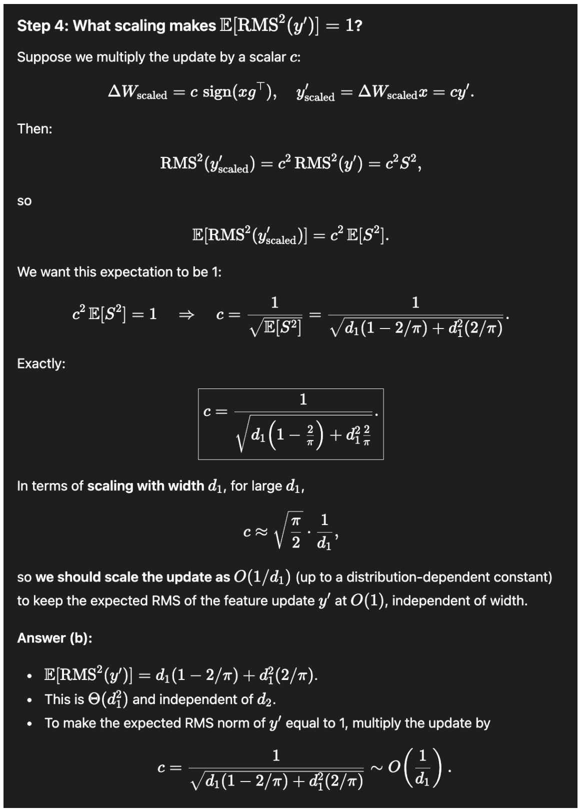
So

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

For large d_1, the d_1^2 term dominates:

$$\mathbb{E}[\mathrm{RMS}^2(y')] \sim \frac{2}{\pi} d_1^2,$$

so **the squared RMS norm of the feature update scales like $\Theta(d_1^2)$**, and is independent of d_2.



Step 4: What scaling makes $\mathbb{E}[\mathrm{RMS}^2(y')] = 1$?

Suppose we multiply the update by a scalar c:

$$\Delta W_{\text{scaled}} = c\ \mathrm{sign}(xg^\top), \quad y'_{\text{scaled}} = \Delta W_{\text{scaled}} x = cy'.$$

Then:

$$\mathrm{RMS}^2(y'_{\text{scaled}}) = c^2\,\mathrm{RMS}^2(y') = c^2 S^2,$$

so

$$\mathbb{E}[\mathrm{RMS}^2(y'_{\text{scaled}})] = c^2\,\mathbb{E}[S^2].$$

We want this expectation to be 1:

$$c^2\,\mathbb{E}[S^2] = 1 \quad \Rightarrow \quad c = \frac{1}{\sqrt{\mathbb{E}[S^2]}} = \frac{1}{\sqrt{d_1(1 - 2/\pi) + d_1^2(2/\pi)}}.$$

Exactly:

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

In terms of **scaling with width** d_1, for large d_1,

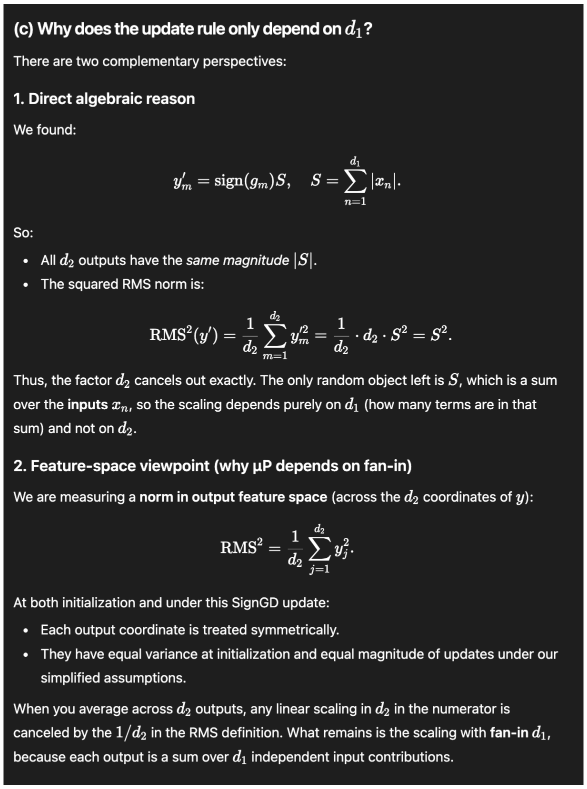
$$c \approx \sqrt{\frac{\pi}{2}} \cdot \frac{1}{d_1},$$

so **we should scale the update as** $O(1/d_1)$ (up to a distribution-dependent constant) to keep the expected RMS of the feature update y' at $O(1)$, independent of width.

Answer (b):

- $\mathbb{E}[\mathrm{RMS}^2(y')] = d_1(1 - 2/\pi) + d_1^2(2/\pi)$.
- This is $\Theta(d_1^2)$ and independent of d_2.
- To make the expected RMS norm of y' equal to 1, multiply the update by

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



(c) Why does the update rule only depend on d_1?

There are two complementary perspectives:

1. Direct algebraic reason

We found:

$$y'_m = \mathrm{sign}(g_m) S, \quad S = \sum_{n=1}^{d_1} |x_n|.$$

So:

- All d_2 outputs have the *same magnitude* $|S|$.
- The squared RMS norm is:

$$\mathrm{RMS}^2(y') = \frac{1}{d_2} \sum_{m=1}^{d_2} y'^2_m = \frac{1}{d_2} \cdot d_2 \cdot S^2 = S^2.$$

Thus, the factor d_2 cancels out exactly. The only random object left is S, which is a sum over the **inputs** x_n, so the scaling depends purely on d_1 (how many terms are in that sum) and not on d_2.

2. Feature-space viewpoint (why µP depends on fan-in)

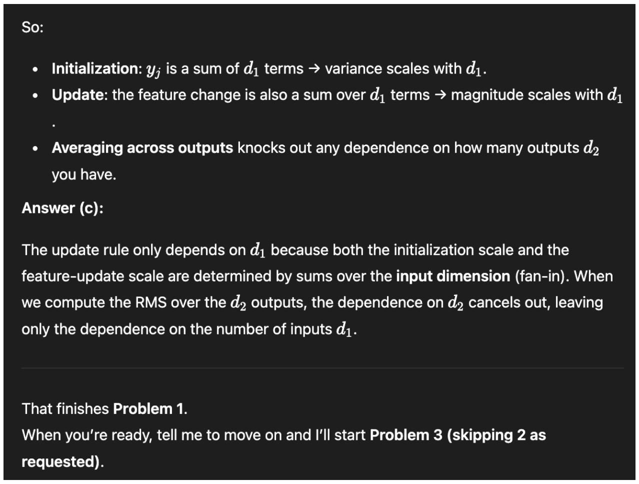
We are measuring a **norm in output feature space** (across the d_2 coordinates of y):

$$\mathrm{RMS}^2 = \frac{1}{d_2} \sum_{j=1}^{d_2} y_j^2.$$

At both initialization and under this SignGD update:

- Each output coordinate is treated symmetrically.
- They have equal variance at initialization and equal magnitude of updates under our simplified assumptions.

When you average across d_2 outputs, any linear scaling in d_2 in the numerator is canceled by the $1/d_2$ in the RMS definition. What remains is the scaling with **fan-in** d_1, because each output is a sum over d_1 independent input contributions.



So:

- **Initialization**: y_j is a sum of d_1 terms → variance scales with d_1.
- **Update**: the feature change is also a sum over d_1 terms → magnitude scales with d_1 .
- **Averaging across outputs** knocks out any dependence on how many outputs d_2 you have.

Answer (c):

The update rule only depends on d_1 because both the initialization scale and the feature-update scale are determined by sums over the **input dimension** (fan-in). When we compute the RMS over the d_2 outputs, the dependence on d_2 cancels out, leaving only the dependence on the number of inputs d_1.

That finishes **Problem 1**.
When you're ready, tell me to move on and I'll start **Problem 3 (skipping 2 as requested)**.

Comments & Observations:

Part A: The model one-shotted this part. The model first considers a single coordinate of the output, which is the same logic used by the reference solution. It then writes its proof, using some basic natural language to connect each step of its proof, and it summarizes the correct answers.

Part B: The model one-shotted this part. It provides a full proof. Interestingly, the model provides the full constant term (not included in the reference solution), but also summarizes its order as $O(1/d_1)$, recognizing that only the $1/d_1$ term is significant here. The model also makes some minor mistake in the formatting of some of its math that causes a slight rendering error in the web frontend, but this is not particularly significant.

Part C: The model one-shotted this part. Its explanation is valid/correct but uses a different line of reasoning than the reference solution. The model's explanation is split into two parts: an algebraic justification and an intuitive "feature-space" viewpoint. Both perspectives are still valid explanations that answer the question.

3: Maximal Update Parameterization Research

User Prompt:

[Attachment: PDF of Arxiv 2203.03466]

[Attachment: Image of Figure 1 from paper]

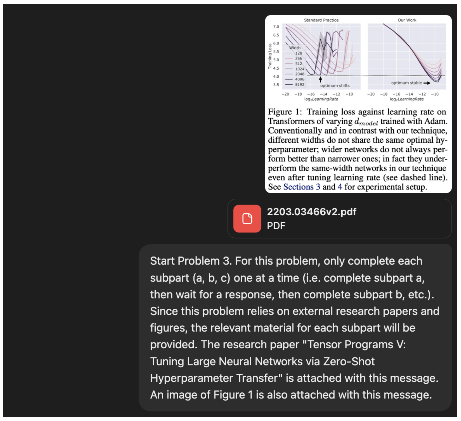
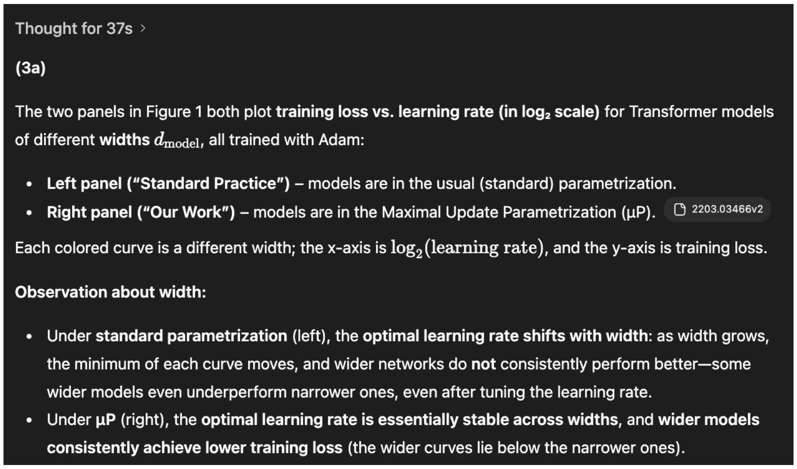


Figure 1: Training loss against learning rate on Transformers of varying d_{model} trained with Adam. Conventionally and in contrast with our technique, different widths do not share the same optimal hyperparameter; wider networks do not always perform better than narrower ones; in fact they underperform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

2203.03466v2.pdf
PDF

Start Problem 3. For this problem, only complete each subpart (a, b, c) one at a time (i.e. complete subpart a, then wait for a response, then complete subpart b, etc.). Since this problem relies on external research papers and figures, the relevant material for each subpart will be provided. The research paper "Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer" is attached with this message. An image of Figure 1 is also attached with this message.

AI Response:



(3a)

The two panels in Figure 1 both plot **training loss vs. learning rate (in $\log_2$ scale)** for Transformer models of different **widths d_{model}**, all trained with Adam:

- **Left panel ("Standard Practice")** – models are in the usual (standard) parametrization.
- **Right panel ("Our Work")** – models are in the Maximal Update Parametrization (µP). 2203.03466v2

Each colored curve is a different width; the x-axis is $\log_2(\text{learning rate})$, and the y-axis is training loss.

Observation about width:

- Under **standard parametrization** (left), the **optimal learning rate shifts with width**: as width grows, the minimum of each curve moves, and wider networks do **not** consistently perform better—some wider models even underperform narrower ones, even after tuning the learning rate.
- Under **µP** (right), the **optimal learning rate is essentially stable across widths**, and **wider models consistently achieve lower training loss** (the wider curves lie below the narrower ones).

User Prompt:

[Attachment: Image of Table 3 from paper]

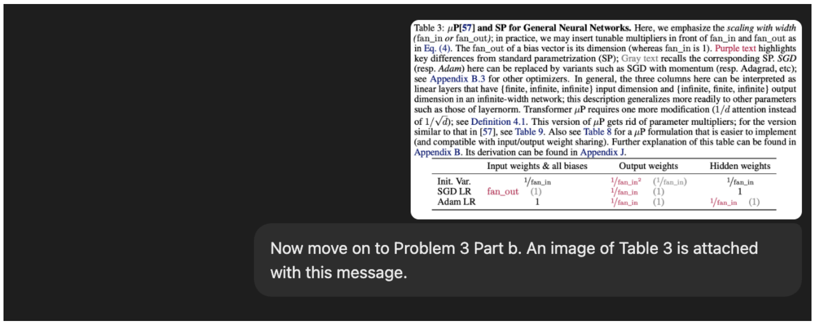
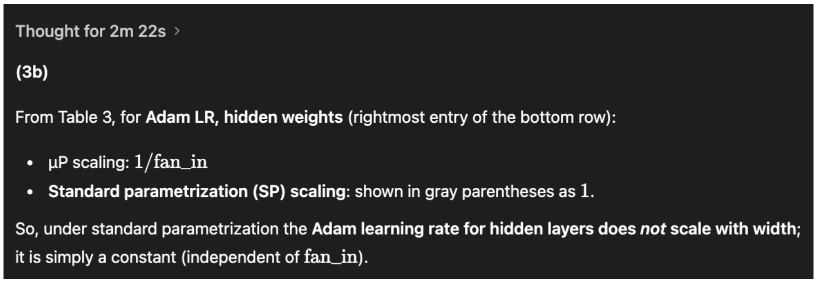


Table 3: μ**P[57] and SP for General Neural Networks.** Here, we emphasize the *scaling with width (*fan_in *or* fan_out*)*; in practice, we may insert tunable multipliers in front of fan_in and fan_out as in Eq. (4). The fan_out of a bias vector is its dimension (whereas fan_in is 1). Purple text highlights key differences from standard parametrization (SP); Gray text recalls the corresponding SP. *SGD* (resp. *Adam*) here can be replaced by variants such as SGD with momentum (resp. Adagrad, etc); see Appendix B.3 for other optimizers. In general, the three columns here can be interpreted as linear layers that have {finite, infinite, infinite} input dimension and {infinite, finite, infinite} output dimension in an infinite-width network; this description generalizes more readily to other parameters such as those of layernorm. Transformer μP requires one more modification ($1/d$ attention instead of $1/\sqrt{d}$); see Definition 4.1. This version of μP gets rid of parameter multipliers; for the version similar to that in [57], see Table 9. Also see Table 8 for a μP formulation that is easier to implement (and compatible with input/output weight sharing). Further explanation of this table can be found in Appendix B. Its derivation can be found in Appendix J.

	Input weights & all biases	Output weights	Hidden weights
Init. Var.	$1/\text{fan\_in}$	$1/\text{fan\_in}^2$ $(1/\text{fan\_in})$	$1/\text{fan\_in}$
SGD LR	fan_out (1)	$1/\text{fan\_in}$ (1)	1
Adam LR	1	$1/\text{fan\_in}$ (1)	$1/\text{fan\_in}$ (1)

Now move on to Problem 3 Part b. An image of Table 3 is attached with this message.

AI Response:



Thought for 2m 22s ›

(3b)

From Table 3, for **Adam LR, hidden weights** (rightmost entry of the bottom row):

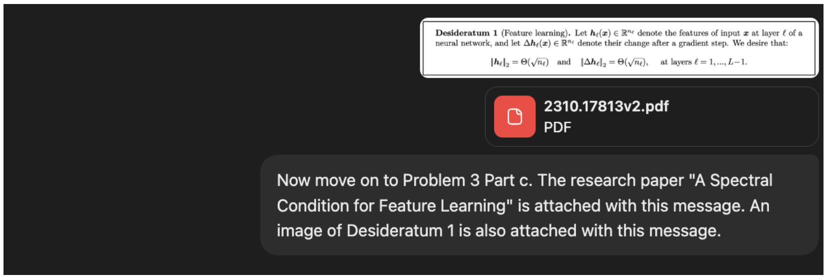
- µP scaling: $1/\text{fan\_in}$
- **Standard parametrization (SP) scaling**: shown in gray parentheses as 1.

So, under standard parametrization the **Adam learning rate for hidden layers does *not* scale with width**; it is simply a constant (independent of fan_in).

User Prompt:

[Attachment: PDF of Arxiv 2310.17813]

[Attachment: Desideratum 1 from paper]



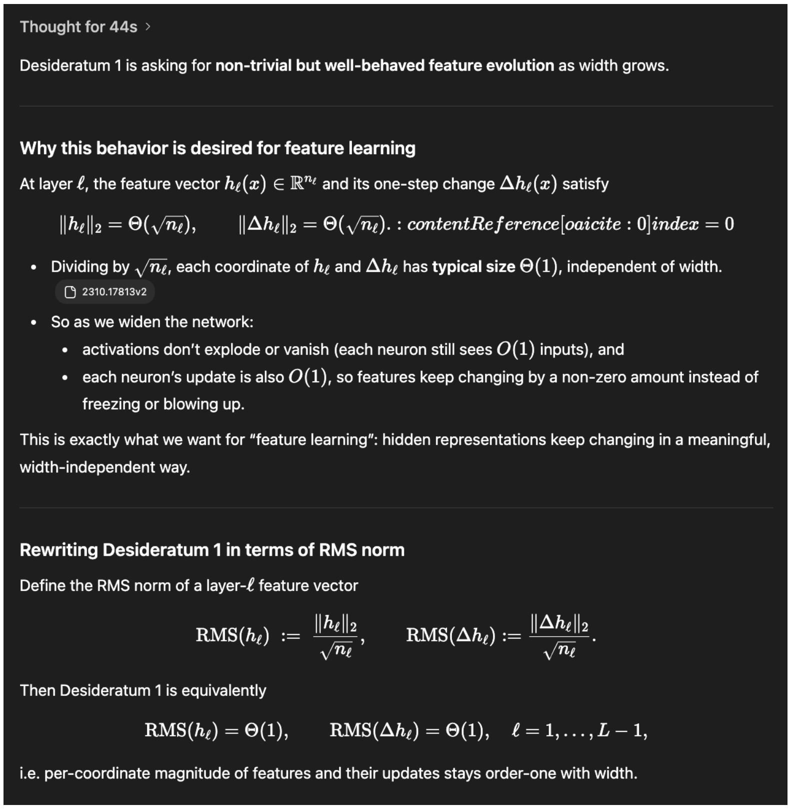
Desideratum 1 (Feature learning). Let $\boldsymbol{h}_\ell(\boldsymbol{x}) \in \mathbb{R}^{n_\ell}$ denote the features of input $\boldsymbol{x}$ at layer ℓ of a neural network, and let $\Delta\boldsymbol{h}_\ell(\boldsymbol{x}) \in \mathbb{R}^{n_\ell}$ denote their change after a gradient step. We desire that:

$$\|\boldsymbol{h}_\ell\|_2 = \Theta(\sqrt{n_\ell}) \quad \text{and} \quad \|\Delta\boldsymbol{h}_\ell\|_2 = \Theta(\sqrt{n_\ell}), \quad \text{at layers } \ell = 1, ..., L-1.$$

2310.17813v2.pdf
PDF

Now move on to Problem 3 Part c. The research paper "A Spectral Condition for Feature Learning" is attached with this message. An image of Desideratum 1 is also attached with this message.

AI Response:



Thought for 44s ›

Desideratum 1 is asking for **non-trivial but well-behaved feature evolution** as width grows.

Why this behavior is desired for feature learning

At layer ℓ, the feature vector $h_\ell(x) \in \mathbb{R}^{n_\ell}$ and its one-step change $\Delta h_\ell(x)$ satisfy

$$\|h_\ell\|_2 = \Theta(\sqrt{n_\ell}), \qquad \|\Delta h_\ell\|_2 = \Theta(\sqrt{n_\ell}). : contentReference[oaicite:0]index=0$$

- Dividing by $\sqrt{n_\ell}$, each coordinate of h_ℓ and Δh_ℓ has **typical size** $\Theta(1)$, independent of width.
  2310.17813v2
- So as we widen the network:
  - activations don't explode or vanish (each neuron still sees $O(1)$ inputs), and
  - each neuron's update is also $O(1)$, so features keep changing by a non-zero amount instead of freezing or blowing up.

This is exactly what we want for "feature learning": hidden representations keep changing in a meaningful, width-independent way.

Rewriting Desideratum 1 in terms of RMS norm

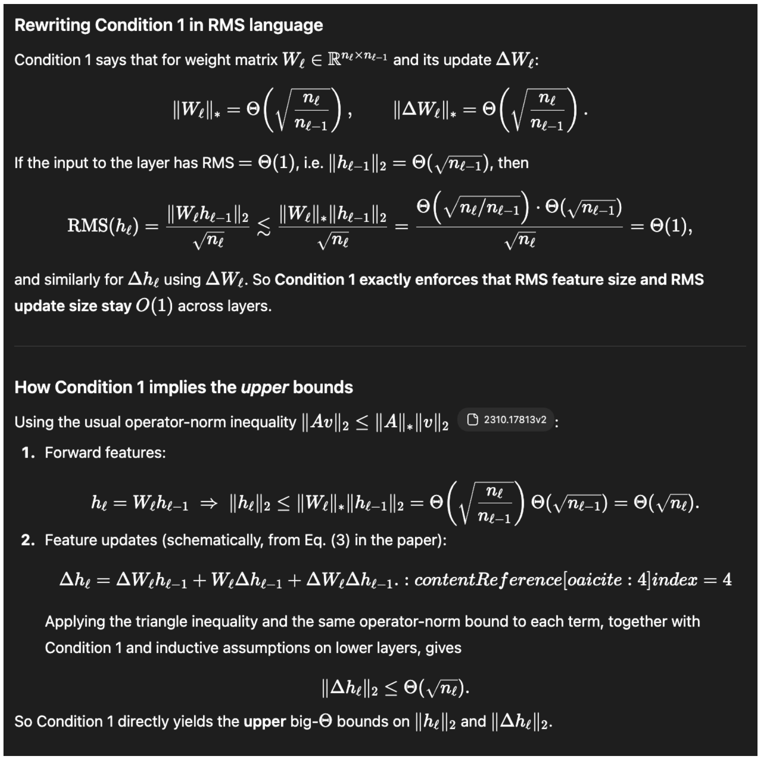
Define the RMS norm of a layer-ℓ feature vector

$$\text{RMS}(h_\ell) := \frac{\|h_\ell\|_2}{\sqrt{n_\ell}}, \qquad \text{RMS}(\Delta h_\ell) := \frac{\|\Delta h_\ell\|_2}{\sqrt{n_\ell}}.$$

Then Desideratum 1 is equivalently

$$\text{RMS}(h_\ell) = \Theta(1), \qquad \text{RMS}(\Delta h_\ell) = \Theta(1), \qquad \ell = 1, \dots, L-1,$$

i.e. per-coordinate magnitude of features and their updates stays order-one with width.



Rewriting Condition 1 in RMS language

Condition 1 says that for weight matrix $W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and its update ΔW_ℓ:

$$\|W_\ell\|_* = \Theta\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right), \qquad \|\Delta W_\ell\|_* = \Theta\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right).$$

If the input to the layer has RMS $= \Theta(1)$, i.e. $\|h_{\ell-1}\|_2 = \Theta(\sqrt{n_{\ell-1}})$, then

$$\mathrm{RMS}(h_\ell) = \frac{\|W_\ell h_{\ell-1}\|_2}{\sqrt{n_\ell}} \lesssim \frac{\|W_\ell\|_* \|h_{\ell-1}\|_2}{\sqrt{n_\ell}} = \frac{\Theta\left(\sqrt{n_\ell/n_{\ell-1}}\right) \cdot \Theta(\sqrt{n_{\ell-1}})}{\sqrt{n_\ell}} = \Theta(1),$$

and similarly for Δh_ℓ using ΔW_ℓ. So **Condition 1 exactly enforces that RMS feature size and RMS update size stay** $O(1)$ across layers.

How Condition 1 implies the *upper* bounds

Using the usual operator-norm inequality $\|Av\|_2 \le \|A\|_* \|v\|_2$ 2310.17813v2 :

1. Forward features:

$$h_\ell = W_\ell h_{\ell-1} \ \Rightarrow\ \|h_\ell\|_2 \le \|W_\ell\|_* \|h_{\ell-1}\|_2 = \Theta\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right)\Theta(\sqrt{n_{\ell-1}}) = \Theta(\sqrt{n_\ell}).$$

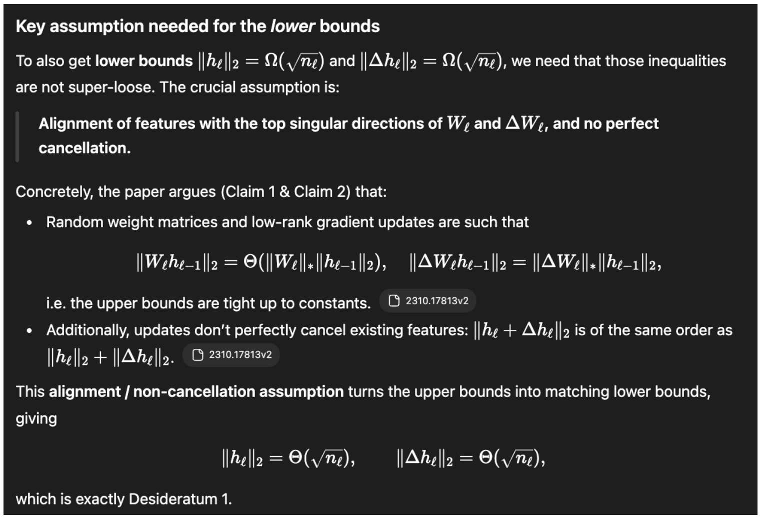
2. Feature updates (schematically, from Eq. (3) in the paper):

$$\Delta h_\ell = \Delta W_\ell h_{\ell-1} + W_\ell \Delta h_{\ell-1} + \Delta W_\ell \Delta h_{\ell-1}. : contentReference[oaicite:4]index = 4$$

   Applying the triangle inequality and the same operator-norm bound to each term, together with Condition 1 and inductive assumptions on lower layers, gives

$$\|\Delta h_\ell\|_2 \le \Theta(\sqrt{n_\ell}).$$

So Condition 1 directly yields the **upper** big-Θ bounds on $\|h_\ell\|_2$ and $\|\Delta h_\ell\|_2$.



Key assumption needed for the *lower* bounds

To also get **lower bounds** $\|h_\ell\|_2 = \Omega(\sqrt{n_\ell})$ and $\|\Delta h_\ell\|_2 = \Omega(\sqrt{n_\ell})$, we need that those inequalities are not super-loose. The crucial assumption is:

> **Alignment of features with the top singular directions of W_ℓ and ΔW_ℓ, and no perfect cancellation.**

Concretely, the paper argues (Claim 1 & Claim 2) that:

- Random weight matrices and low-rank gradient updates are such that

$$\|W_\ell h_{\ell-1}\|_2 = \Theta(\|W_\ell\|_* \|h_{\ell-1}\|_2), \quad \|\Delta W_\ell h_{\ell-1}\|_2 = \|\Delta W_\ell\|_* \|h_{\ell-1}\|_2,$$

  i.e. the upper bounds are tight up to constants. 2310.17813v2
- Additionally, updates don't perfectly cancel existing features: $\|h_\ell + \Delta h_\ell\|_2$ is of the same order as $\|h_\ell\|_2 + \|\Delta h_\ell\|_2$. 2310.17813v2

This **alignment / non-cancellation assumption** turns the upper bounds into matching lower bounds, giving

$$\|h_\ell\|_2 = \Theta(\sqrt{n_\ell}), \qquad \|\Delta h_\ell\|_2 = \Theta(\sqrt{n_\ell}),$$

which is exactly Desideratum 1.

Comments & Observations:

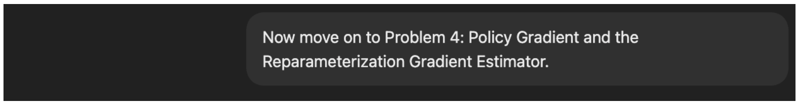
Part A: The model one-shotted this part. It provides a correct/complete analysis of the figure that aligns with the reference solution, and provides some additional detail.

Part B: The model one-shotted this part. Interestingly, the chain of thought trace reveals that the model had some issues using some of its tools for reading the PDF that caused a long chain of thought time for this subpart (around 2.5 minutes). It's possible that the attached PDF may have caused the conversation to exceed the context window, but I am unsure. Overall, the model was able to recover and still provide a valid solution.

Part C: The model one-shotted this part. It gave a valid proof/reasoning for the problem that aligned with the reference solution.

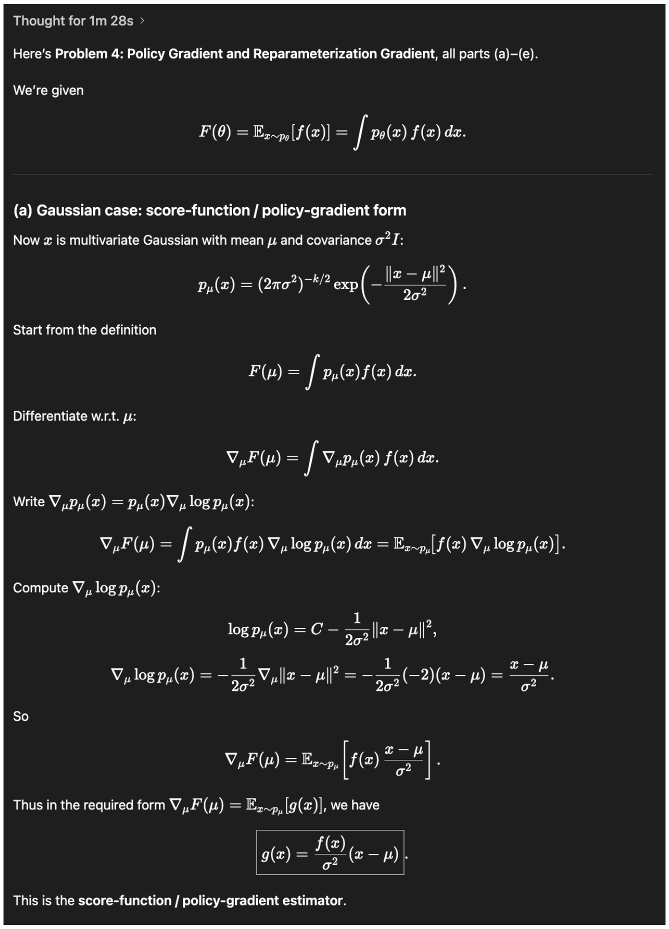
4: Policy Gradient and the Reparameterization Gradient Estimator

User Prompt:



Now move on to Problem 4: Policy Gradient and the Reparameterization Gradient Estimator.

AI Response:



Here's **Problem 4: Policy Gradient and Reparameterization Gradient**, all parts (a)–(e).

We're given

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

(a) Gaussian case: score-function / policy-gradient form

Now x is multivariate Gaussian with mean μ and covariance $\sigma^2 I$:

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

Start from the definition

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

Differentiate w.r.t. μ:

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

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

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

Compute $\nabla_\mu \log p_\mu(x)$:

$$\log p_\mu(x) = C - \frac{1}{2\sigma^2}\|x-\mu\|^2,$$

$$\nabla_\mu \log p_\mu(x) = -\frac{1}{2\sigma^2}\nabla_\mu\|x-\mu\|^2 = -\frac{1}{2\sigma^2}(-2)(x-\mu) = \frac{x-\mu}{\sigma^2}.$$

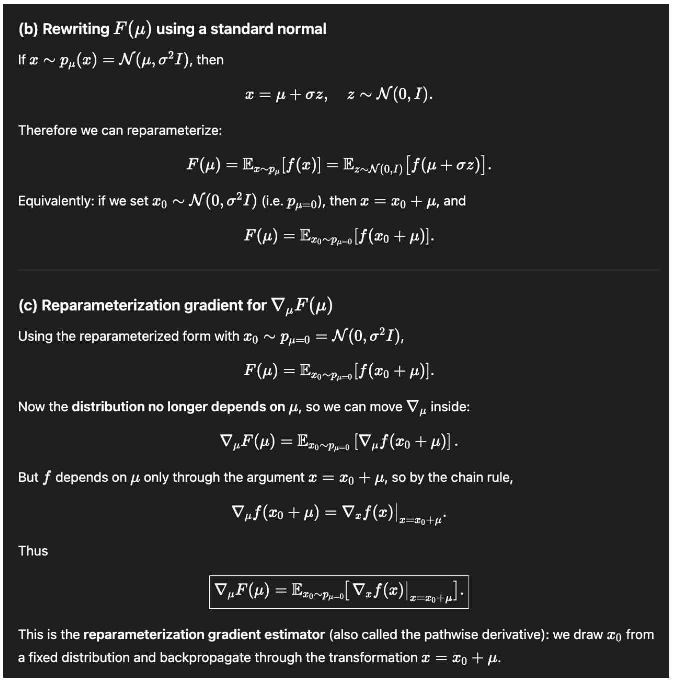
So

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

Thus in the required form $\nabla_\mu F(\mu) = \mathbb{E}_{x\sim p_\mu}[g(x)]$, we have

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

This is the **score-function / policy-gradient estimator**.



(b) Rewriting $F(\mu)$ using a standard normal

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

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

Therefore we can reparameterize:

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

Equivalently: if we set $x_0 \sim \mathcal{N}(0, \sigma^2 I)$ (i.e. $p_{\mu=0}$), then $x = x_0 + \mu$, and

$$F(\mu) = \mathbb{E}_{x_0\sim p_{\mu=0}}[f(x_0 + \mu)].$$

(c) Reparameterization gradient for $\nabla_\mu F(\mu)$

Using the reparameterized form with $x_0 \sim p_{\mu=0} = \mathcal{N}(0, \sigma^2 I)$,

$$F(\mu) = \mathbb{E}_{x_0\sim p_{\mu=0}}[f(x_0 + \mu)].$$

Now the **distribution no longer depends on** μ, so we can move ∇_μ inside:

$$\nabla_\mu F(\mu) = \mathbb{E}_{x_0\sim p_{\mu=0}}\left[\nabla_\mu f(x_0 + \mu)\right].$$

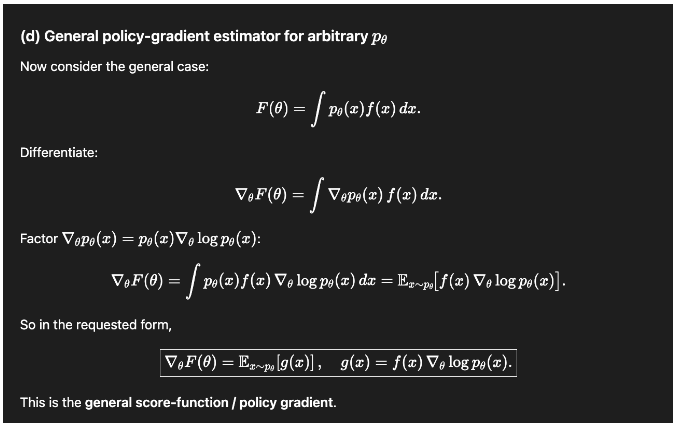
But f depends on μ only through the argument $x = x_0 + \mu$, so by the chain rule,

$$\nabla_\mu f(x_0 + \mu) = \nabla_x f(x)\big|_{x=x_0+\mu}.$$

Thus

$$\boxed{\nabla_\mu F(\mu) = \mathbb{E}_{x_0\sim p_{\mu=0}}\big[\nabla_x f(x)\big|_{x=x_0+\mu}\big].}$$

This is the **reparameterization gradient estimator** (also called the pathwise derivative): we draw x_0 from a fixed distribution and backpropagate through the transformation $x = x_0 + \mu$.



(d) General policy-gradient estimator for arbitrary p_θ

Now consider the general case:

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

Differentiate:

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

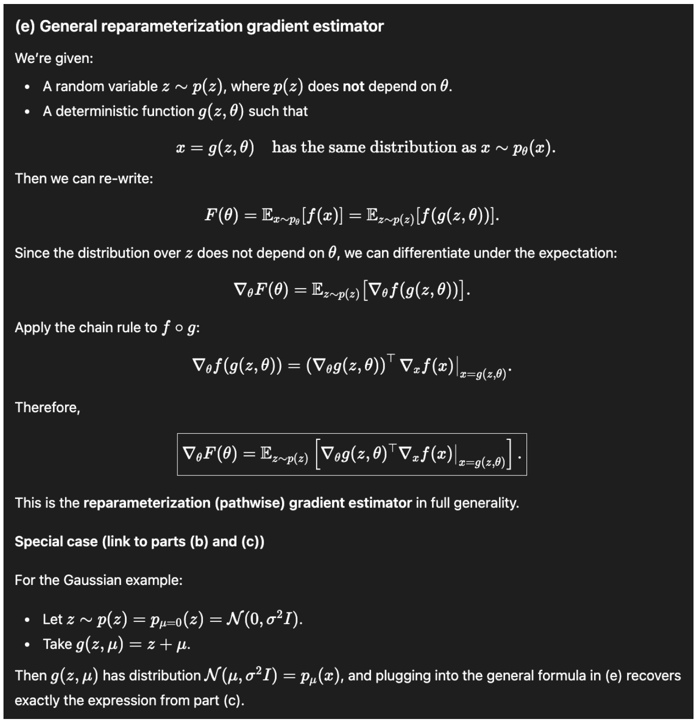
Factor $\nabla_\theta p_\theta(x) = p_\theta(x) \nabla_\theta \log p_\theta(x)$:

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

So in the requested form,

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

This is the **general score-function / policy gradient**.



(e) General reparameterization gradient estimator

We're given:

- A random variable $z \sim p(z)$, where $p(z)$ does **not** depend on θ.
- A deterministic function $g(z, \theta)$ such that

$$x = g(z, \theta) \quad \text{has the same distribution as } x \sim p_\theta(x).$$

Then we can re-write:

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

Since the distribution over z does not depend on θ, we can differentiate under the expectation:

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

Apply the chain rule to $f \circ g$:

$$\nabla_\theta f(g(z, \theta)) = \left(\nabla_\theta g(z, \theta)\right)^\top \nabla_x f(x)\big|_{x = g(z, \theta)}.$$

Therefore,

$$\boxed{\nabla_\theta F(\theta) = \mathbb{E}_{z \sim p(z)}\left[\nabla_\theta g(z, \theta)^\top \nabla_x f(x)\big|_{x = g(z, \theta)}\right].}$$

This is the **reparameterization (pathwise) gradient estimator** in full generality.

Special case (link to parts (b) and (c))

For the Gaussian example:

- Let $z \sim p(z) = p_{\mu=0}(z) = \mathcal{N}(0, \sigma^2 I)$.
- Take $g(z, \mu) = z + \mu$.

Then $g(z, \mu)$ has distribution $\mathcal{N}(\mu, \sigma^2 I) = p_\mu(x)$, and plugging into the general formula in (e) recovers exactly the expression from part (c).

Comments & Observations:

Part A: The model one-shotted this part. It used the score function identity and recognized that solution is the "score-function / policy-gradient estimator." Because it used the score function identity, its work looks a little different from the reference solution, but ultimately its final solution is the same.

Part B: The model one-shotted this part. For its final answer, it added a slightly different (but correct) notation compared to the reference solution, and it used this notation in the follow-up in Part C.

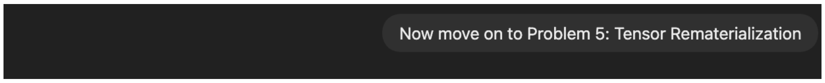
Part C: The model one-shotted this part. It continued with its Part B notation, and noted that "this is the reparameterization gradient estimator (also called the pathwise derivative)."

Part D: The model one-shotted this part. It notes that "this is the general score-function / policy gradient."

Part E: The model one-shotted this part. It was able to identify the correct special cases referenced by the problem (recognizing the link to Part B and Part C).

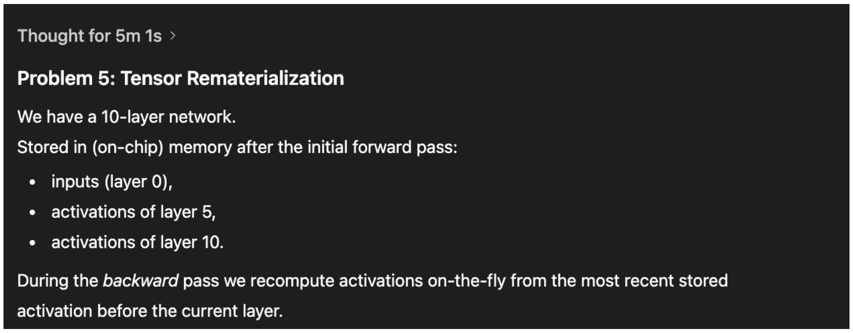
5: Tensor Rematerialization

User Prompt:



Now move on to Problem 5: Tensor Rematerialization

AI Response:



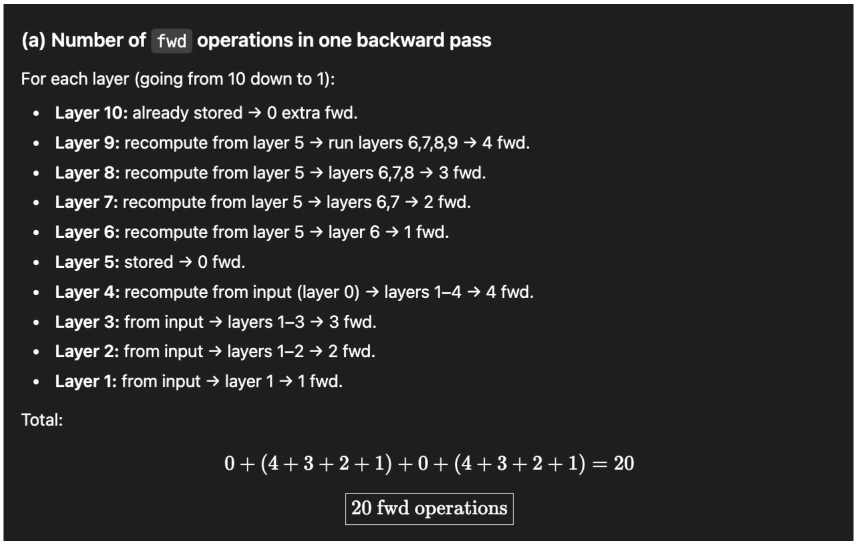
Thought for 5m 1s ›

Problem 5: Tensor Rematerialization

We have a 10-layer network.
Stored in (on-chip) memory after the initial forward pass:

- inputs (layer 0),
- activations of layer 5,
- activations of layer 10.

During the *backward* pass we recompute activations on-the-fly from the most recent stored activation before the current layer.



(a) Number of `fwd` operations in one backward pass

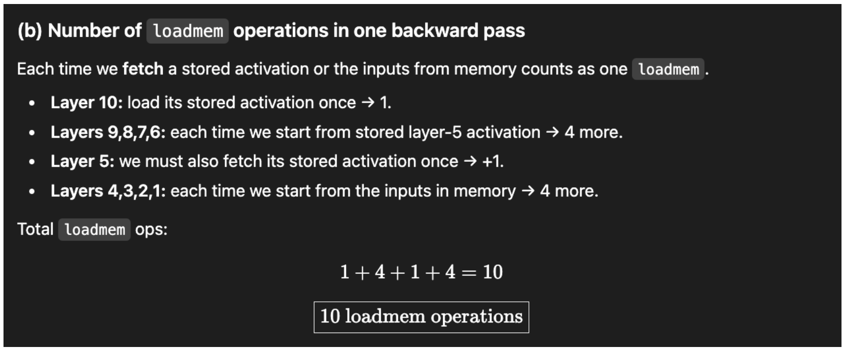
For each layer (going from 10 down to 1):

- **Layer 10:** already stored → 0 extra fwd.
- **Layer 9:** recompute from layer 5 → run layers 6,7,8,9 → 4 fwd.
- **Layer 8:** recompute from layer 5 → layers 6,7,8 → 3 fwd.
- **Layer 7:** recompute from layer 5 → layers 6,7 → 2 fwd.
- **Layer 6:** recompute from layer 5 → layer 6 → 1 fwd.
- **Layer 5:** stored → 0 fwd.
- **Layer 4:** recompute from input (layer 0) → layers 1–4 → 4 fwd.
- **Layer 3:** from input → layers 1–3 → 3 fwd.
- **Layer 2:** from input → layers 1–2 → 2 fwd.
- **Layer 1:** from input → layer 1 → 1 fwd.

Total:

$$0+(4+3+2+1)+0+(4+3+2+1)=20$$

$$\boxed{20\ \text{fwd operations}}$$



(b) Number of `loadmem` operations in one backward pass

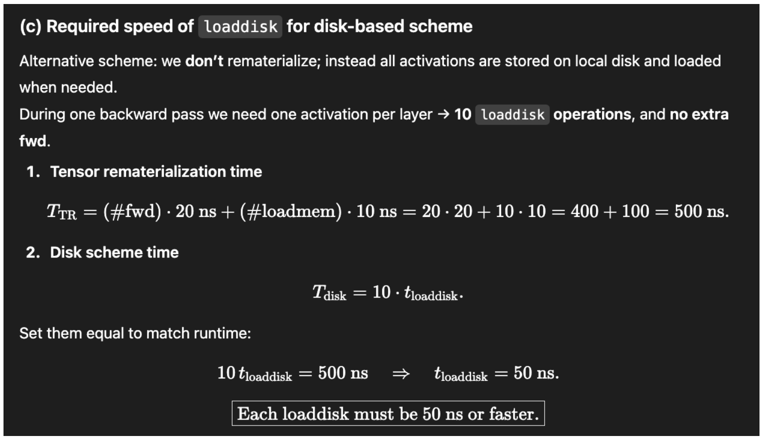
Each time we **fetch** a stored activation or the inputs from memory counts as one `loadmem`.

- **Layer 10:** load its stored activation once → 1.
- **Layers 9,8,7,6:** each time we start from stored layer-5 activation → 4 more.
- **Layer 5:** we must also fetch its stored activation once → +1.
- **Layers 4,3,2,1:** each time we start from the inputs in memory → 4 more.

Total `loadmem` ops:

$$1+4+1+4=10$$

$$\boxed{10\ \text{loadmem operations}}$$



(c) Required speed of `loaddisk` for disk-based scheme

Alternative scheme: we **don't** rematerialize; instead all activations are stored on local disk and loaded when needed.
During one backward pass we need one activation per layer → **10 `loaddisk` operations**, and **no extra fwd**.

1. **Tensor rematerialization time**

$$T_{\text{TR}} = (\#\text{fwd}) \cdot 20 \text{ ns} + (\#\text{loadmem}) \cdot 10 \text{ ns} = 20 \cdot 20 + 10 \cdot 10 = 400 + 100 = 500 \text{ ns}.$$

2. **Disk scheme time**

$$T_{\text{disk}} = 10 \cdot t_{\text{loaddisk}}.$$

Set them equal to match runtime:

$$10\, t_{\text{loaddisk}} = 500 \text{ ns} \quad \Rightarrow \quad t_{\text{loaddisk}} = 50 \text{ ns}.$$

$$\boxed{\text{Each loaddisk must be 50 ns or faster.}}$$

Comments & Observations:
The model's chain of thought reasoning took a particularly long time to run on this problem (around 5 minutes). It seems that most of the chain of thought time was spent reasoning about the arithmetic required by the problem and performing calculations.
Part A: The model one-shotted this part. It first identified the correct number of fwd operations for each individual layer and then calculated the correct total number of fwd operations.
Part B: The model one-shotted this part. Its response/reasoning follows a similar structure to its Part A response, and its final answer for the total number of loadmem operations is correct.
Part C: The model one-shotted this part. The model again writes out its arithmetic and arrives at the correct final answer (50 ns).