EECS 182 Deep Neural Networks
Fall 2025 Anant Sahai and Gireeja Ranade

Homework 3

This homework is due on Sep. 26th, 2025, at 10:59PM.

1. Maximal Update Parameterization

During lecture, we learned how SGD and Adam can be seen as constrained optimization problems, defined by some norm over parameter space. In this problem (which builds directly on the last discussion section), we will explore the payoff for defining these norms over the *output features* of a dense layer and then bringing it back to the matrices involved.

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

What is the expected squared RMS norm of the output features $\mathbf{y} = W\mathbf{x}$? 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 $\mathbf{y} = W\mathbf{x}$? What is its variance?

Solution: For simplicity, first consider a single component $\mathbf{y}_i = \sum_{j=1}^{d_{in}} W_{ij}\mathbf{x}_j$ of the output. We have that

$$\mathbb{E}\left[\mathbf{y}_i^2\right] = \text{Var}(\mathbf{y}_i) = \sum_j \text{Var}(W_{ij}\mathbf{x}_j) = \sum_i \text{Var}(W_{ij})\text{Var}(\mathbf{x}_j) = d_1.$$

The first equality holds because $\mathbf{y}$ has zero mean and subsequent equalities hold since variance is both additive and multiplicative for independent zero-mean random variables. Since this holds for each output feature, and we know that RMS norm scales with the magnitude of individual features, the RMS norm of $\mathbf{y}$ scales as $\Theta(d_1)$. Recall that multiplying a random variable by a constant c increases its variance by c^2. To ensure that the expected squared RMS norm is 1, we should multiply W by $\frac{1}{\sqrt{d_1}}$. This recovers the Xavier initialization scaling.

(b) 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 $\mathbf{x}_i \in \mathbb{R}^{d_1}$ 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 $\mathbf{g}_i \in \mathbb{R}^{d_2}$.

$$W_{t+1} \leftarrow W_t + \eta \text{sign}(\mathbf{g}_i \mathbf{x_i}^T).$$

What is the expected RMS norm squared of the (unscaled) change in features $\mathbf{y}' = \text{sign}(\mathbf{g}_i\mathbf{x}_i^T)\mathbf{x}_i$? 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 $\mathbf{y}'$ is 1, regardless of d_1 and d_2?

Solution: Let's consider the update to a single output feature, which in our simplified case is $y = \text{sign}(g \cdot x)^T x$, where g is a scalar constant. We can move g to the outside, resulting in $y = g \cdot \text{sign}(x)^T\mathbf{x} = g\sum_{j=1}^{d_{in}} |\mathbf{x}_j|$. Note that $g^2 = 1$ and each term in the sum has a *positive* expected value

that scales as $\Theta(1)$, and there are d_1 such terms, so the output feature has an expected squared RMS norm that scales as $\Theta(d_1^2)$. To ensure that the expected (squared) RMS norm is 1, we should scale the update by $\frac{1}{d_1}$. This recovers the Maximal Update Parameterization (muP) scaling.

(c) You may notice that the above update rule only depends on d_1. **Why is this the case?**

Solution: When we look at a weight matrix updates, the parameters that affect a single output feature are independent of the other output features. Thus, when we control for the RMS norm, we can consider each output feature independently, and the width m does not affect the scaling.

2. Visualizing Maximal Update Parameterization (Coding Question)

In this question, using simple MLPs, you will explore how the maximal update parameterization preserves feature learning as the width of the network changes. Please follow the instructions in this Google Colab notebook.[1]

3. Maximal Update Parameterization Research

This homework problem talks about the research papers behind "maximal update parameterization," also called μP in the community. This idea is discussed in detail in the paper Tensor Programs V: Tuning Large Neural Networks via Zero-Shot Hyperparameter Transfer. Understanding the full paper is out of scope of the class and requires sophistication that is not a prerequisite, but we want to highlight some points.

(a) 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?**

Solution: Both figures plot the training loss of models of different hidden layer widths vs the log of the learning rate used for training with Adam. Larger width matrices lead to better performance when using the muP scaling.

(b) Look at Table 3 in the paper. In lecture, we gave an elementary argument to support the $1/\text{fan}_{\text{in}}$ scaling for Adam LR for the hidden layers, the rightmost entry in the bottom row of the paper. **What does the table say is the corresponding standard parameterization?**

Solution: The corresponding standard parameterization is 1.

(c) The paper A Spectral Condition for Feature Learning presents the same scaling using elementary linear algebra. While understanding the entirely 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 $\|h_\ell(x)\|_2 \leq \Theta(\sqrt{n_\ell})$ 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 $\|\Delta h_\ell(x)\|_2$?**

Solution: This behavior is desired to so that each element of the hidden vector moves by $O(1)$ and maintains a size of $O(1)$ during training. It is good for features to move significantly since we believe that the initialization is far from the optimal point in the loss landscape.

Condition 1 gives:

[1] This notebook is very newly developed and has not been as extensively tested as previously assigned coding problems. Nonetheless, we believe it is a great aid in learning these timely concepts. If you encounter any issues, please post on Ed or notify the course staff in their office hours.

$$O\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right) = ||W||_2 = \sqrt{\frac{n_\ell}{n_{\ell-1}}}||W||_{\text{RMS}\to\text{RMS}} \Rightarrow ||W||_{\text{RMS}\to\text{RMS}} = O(1). \tag{1}$$

$$\|h_\ell(x)\|_2 = \|W_\ell h_{\ell-1}(x)\|_2 \le ||W_\ell||_2||h_{\ell-1}(x)||_2 \le O\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right) O(\sqrt{n_{\ell-1}}) = O(\sqrt{n_\ell}). \tag{2}$$

$$\begin{aligned}
\|\Delta h_\ell(x)\|_2 &= \|W_\ell \Delta h_{\ell-1}(x) + \Delta W_\ell h_{\ell-1}(x) + \Delta W_\ell \Delta h_{\ell-1}(x)\|_2 && (3)\\
&\le \|W_\ell \Delta h_{\ell-1}(x)\|_2 + \|\Delta W_\ell h_{\ell-1}(x)\|_2 + \|\Delta W_\ell \Delta h_{\ell-1}(x)\|_2 && (4)\\
&\le \|W_\ell\|_2\|\Delta h_{\ell-1}(x)\|_2 + \|\Delta W_\ell\|_2\|h_{\ell-1}(x)\|_2 + \|\Delta W_\ell\|_2\|\Delta h_{\ell-1}(x)\|_2 && (5)\\
&= O\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right) O(\sqrt{n_{\ell-1}}) + O\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right) O(\sqrt{n_{\ell-1}}) + O\left(\sqrt{\frac{n_\ell}{n_{\ell-1}}}\right) O(\sqrt{n_{\ell-1}}) && (6)\\
&= O(\sqrt{n_\ell}). && (7)
\end{aligned}$$

The key assumptions for the lower bounds are Claim 1 and Claim 2 in the paper. Informally, they state that the hidden weight matrices and their updates scale incoming vectors by their spectral norm.

4. Policy Gradient and the Reparameterization Gradient Estimator

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

$$\mathcal{F}(\theta) = \mathbb{E}_{\mathbf{x}\sim p_\theta}\left[f(\mathbf{x})\right], \tag{8}$$

where $\mathbf{x}$ is a random variable that follows the probability distribution of $p_\theta : \mathcal{X} \mapsto \Delta(\mathcal{X})$ that is parameterized by $\theta \in \mathbb{R}^P$, and $f : \mathcal{X} \mapsto \mathbb{R}$ is a function.

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

$$p_\mu(\mathbf{x}) = (2\pi\sigma)^{-n/2}\exp\left(-\|\mathbf{x}-\mu\|_2^2/(2\sigma^2)\right), \tag{9}$$

where $\sigma \in \mathbb{R}$ is a scalar constant. **Express $\nabla_\mu\mathcal{F}(\mu)$ as an expectation under $p_\mu(\mathbf{x})$** (i.e., find $g(\mathbf{x})$ in $\nabla\mathcal{F}(\mu) = \mathbb{E}_{\mathbf{x}\sim p_\mu}\left[g(\mathbf{X})\right]$). *(Hint:* $\mathbb{E}_{\mathbf{x}\sim p_\theta}\left[f(\mathbf{x})\right] = \int p_\theta(\mathbf{x})f(\mathbf{x})d\mathbf{x}$*)*

Solution:

$$\begin{aligned}
\nabla_\mu\mathcal{F}(\mu) &= \nabla_\mu \int_{\mathbf{x}} (2\pi\sigma)^{-n/2}\exp\left(-\|\mathbf{x}-\mu\|_2^2/(2\sigma^2)\right) f(\mathbf{x})d\mathbf{x}\\
&= \int_{\mathbf{x}} \nabla_\mu(2\pi\sigma)^{-n/2}\exp\left(-\|\mathbf{x}-\mu\|_2^2/(2\sigma^2)\right) f(\mathbf{x})d\mathbf{x}\\
&= \int_{\mathbf{x}} (2\pi\sigma)^{-n/2}\exp\left(-\|\mathbf{x}-\mu\|_2^2/(2\sigma^2)\right)\frac{\mathbf{x}-\mu}{\sigma^2} f(\mathbf{x})d\mathbf{x}\\
&= \mathbb{E}_{\mathbf{x}\sim p_\mu(\mathbf{x})}\left[\frac{\mathbf{x}-\mu}{\sigma^2}f(\mathbf{x})\right]
\end{aligned} \tag{10}$$

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

Solution:

$$\mathcal{F}(\mu) = \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,\sigma^2\mathbf{I})}\left[f(\mathbf{z}+\mu)\right] \quad = \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,\mathbf{I})}\left[f(\sigma\mathbf{z}+\mu)\right] \tag{11}$$

(c) Using the expression you obtained from the previous part, **can you express $\nabla_\mu\mathcal{F}(\mu)$ in a similar way** (i.e., with an expectation with the distribution over $p_{\mu=0}(\mathbf{x})$)? *(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.)*

Solution:

$$\begin{aligned}\nabla_\mu\mathcal{F}(\mu) &= \nabla_\mu\mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,\mathbf{I})}\left[f(\sigma\mathbf{z}+\mu)\right] \\ &= \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,\mathbf{I})}\left[\nabla_\mu f(\sigma\mathbf{z}+\mu)\right] \\ &= \mathbb{E}_{\mathbf{z}\sim\mathcal{N}(0,\mathbf{I})}\left[\nabla_\mathbf{x} f|_{\mathbf{x}=\sigma\mathbf{z}+\mu}\right]\end{aligned} \tag{12}$$

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 $\nabla\mathcal{F}(\theta)$ as an expectation under $p_\theta(\mathbf{x})$?** *(Hint: look at your answer in Part (a) and see how the term inside your expectation relates to $\log p_\theta(\mathbf{x})$).*

Solution:

$$\begin{aligned}\nabla_\theta\mathcal{F}(\theta) &= \nabla_\theta\int_\mathbf{x} p_\theta(\mathbf{x})f(\mathbf{x})d\mathbf{x} \\ &= \int_\mathbf{x}\nabla_\theta p_\theta(\mathbf{x})f(\mathbf{x})d\mathbf{x} \\ &= \int_\mathbf{x} p_\theta(\mathbf{x})\frac{\nabla_\theta p_\theta(\mathbf{x})}{p_\theta(\mathbf{x})}f(\mathbf{x})d\mathbf{x} \\ &= \int_\mathbf{x} p_\theta(\mathbf{x})\nabla_\theta\log p_\theta(\mathbf{x})f(\mathbf{x})d\mathbf{x} \\ &= \mathbb{E}_{\mathbf{x}\sim p_\theta}\left[\nabla_\theta\log p_\theta(\mathbf{x})f(\mathbf{x})\right]\end{aligned} \tag{13}$$

(e) Assume there exists a function $g(\mathbf{z}, \theta) : \mathcal{Z}\times\mathbb{R}^P \mapsto \mathcal{X}$ and a distribution over $\mathbf{z}$, $p(\mathbf{z})$, such that $g(\mathbf{z}, \theta)$ has the same distribution as $\mathbf{x} \sim p_\theta(\mathbf{x})$. **Prove that $\nabla_\theta\mathcal{F}(\theta) = \mathbb{E}_{\mathbf{z}\sim\mathcal{Z}}\left[\nabla_\theta g(\mathbf{z})^\top\nabla_\mathbf{x} f|_{\mathbf{x}=g(\mathbf{z},\theta)}\right]$.** In the previous part (b) and (c), we actually prove a special case of this. **Can you determine what g and $p(z)$ are for the special case?**

Solution:

$$\begin{aligned}\nabla_\theta\mathcal{F}(\theta) &= \nabla_\theta\mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[f(g(\mathbf{z},\mu))\right] \\ &= \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\nabla_\theta f(g(\mathbf{z},\theta))\right] \\ &= \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\nabla_\theta g(\mathbf{z})^\top\nabla_\mathbf{x} f|_{\mathbf{x}=g(\mathbf{z},\theta)}\right]\end{aligned} \tag{14}$$

For the special case $p(z)$ is the Gaussian distribution $\mathcal{N}(0, \sigma^2\mathbf{I})$ and $g(\mathbf{z}, \theta) = \mathbf{z} + \theta$.

These kinds of tricks are practically useful because the spirit of stochastic gradient descent plays nicely whenever we have gradients that we can view as expectations.

5. Tensor Rematerialization

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.

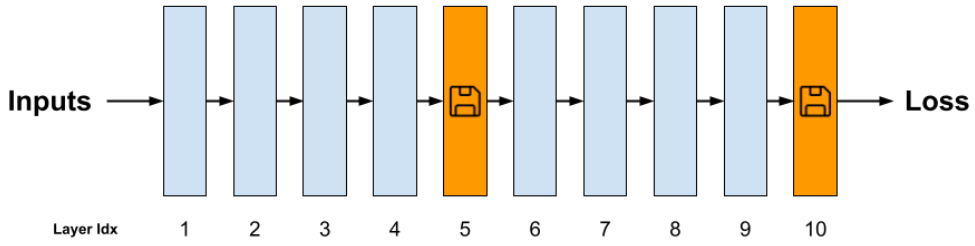

Figure 1: Tensor rematerialization in action - Layer 5 and Layer 10 activations are stored in memory along with the inputs. Activations for other layers are recomputed on-demand from stored activations and inputs.

(a) 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.

Solution: 20.

This question explores trading off memory for compute and computing the break-even point where it's better to use more memory than to re-compute activations.

In this case, computing activations for Layer 10 takes 0 `fwd` operations (loaded from memory), Layer 9 takes 4 (loaded from layer 5 followed by 4 `fwd` passes), Layer 8 takes 3, Layer 7 takes 2, Layer 6 takes 1, Layer 5 takes 0 (loaded from memory), Layer 4 takes 4 (input is loaded and run up to layer 4), Layer 3 takes 3, Layer 2 takes 2 and layer 1 takes 1 forward pass on inputs loaded from memory.

(b) 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?**

Solution: 10. One for each layer.

(c) 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 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.

Solution: 50 ns. This is much lower than the typical times for SSD (2023), which has access time around $[1000, 100000]$ ns. This explains the appeal of tensor rematerialization.

One backward pass with tensor rematerialization takes $20t_{fwd}+10t_{loadmem}$. One backward pass when loading all activations directly from disk takes $10t_{loaddisk}$ (10 activations to be loaded). Solving the below equation gives us $t_{loaddisk}$.

$$20t_{fwd} + 10t_{loadmem} = 10t_{loaddisk}$$

6. Homework Process and Study Group

Citing sources and collaborators are an important part of life, including being a student!
We also want to understand what resources you find helpful and how much time homework is taking, so we can change things in the future if possible.

(a) **What sources (if any) did you use as you worked through the homework?**

(b) **If you worked with someone on this homework, who did you work with?**
List names and student ID's. (In case of homework party, you can also just describe the group.)

(c) **Roughly how many total hours did you work on this homework?**

Contributors:

- Kevin Frans.
- Anant Sahai.
- Gireeja Ranade.
- Qiyang Li.
- Romil Bhardwaj.

