

EECS 182 Deep Neural Networks
 Fall 2026 Gireeja Ranade

Homework 4

This homework is due on Wednesday, September 30, 2026, at 10:59PM.

1. RMS Norm

We have learned how SGD and Adam can be seen as constrained optimization problems, defined by some norm over parameter space. In this problem, we will explore how it is helpful to have suitable norms over the *output features* of a neural net layer.

- (a) The Euclidean norm is often utilized to define the "scale" of a feature vector:

$$||\mathbf{x}||_2 = \sqrt{\sum_i x_i^2}.$$

However, the *root-mean-squared* (RMS) norm:

$$||\mathbf{x}||_{RMS} = \sqrt{\frac{1}{d} \sum_i x_i^2},$$

is often more appropriate when we care about the average scales of *individual feature elements*.

How do the two norms scale with dimension d (i.e. the number of elements in x), assuming each element is sampled from a standard Gaussian distribution?

- (b) **For a vector space with dimension $d = 2$, draw out the set of points with distance 1 under each norm below.**

- The Euclidean norm (or 2-norm):

$$||\mathbf{x}||_2 = \sqrt{\sum_i x_i^2}.$$

- The RMS Norm:

$$||\mathbf{x}||_{RMS} = \sqrt{\frac{1}{d} \sum_i x_i^2}.$$

- The 1-Norm:

$$||\mathbf{x}||_1 = \sum_i |x_i|.$$

- The Infinity Norm:

$$||\mathbf{x}||_\infty = \max_i |x_i|.$$

2. Optimizers as Penalized Linear Improvement with different norm penalties

In lecture, you saw the locally linear perspective of a neural network and the loss by Taylor expanding the loss around the current value of the parameters. This approximation is only very good in a near neighborhood

of those values. One way to proceed with optimization is to consider the size of the neighborhood as a hyperparameter and to bound our update to stay within that neighborhood while minimizing our linear approximation to the loss. You saw in lecture that the choice of norm in defining that neighborhood also matters.

In this problem, you will work out for yourself a slightly different perspective. Instead of treating the norm as a constraint (with the size of the acceptable norm as a hyperparameter), we can do an unconstrained optimization with a weighted penalty that corresponds to the squared norm — where that weight is a hyperparameter.

At each iteration, we wish to maximize linear improvement of the objective (as defined by the dot-product between the gradient and the update) locally regularized by a penalty on the size of the update. This can be expressed (in traditional minimization form) as:

$$u = \underset{\Delta\theta}{\operatorname{argmin}} \underbrace{g^T \Delta\theta}_{\text{Linear Improvement}} + \frac{1}{\alpha} \underbrace{d(\Delta\theta)}_{\text{Distance Penalty}}, \quad (1)$$

where $g = \nabla f(\theta)$ is the gradient of the loss, α is a scalar, and d is a scalar-output distance function $\mathbb{R}^{\dim(\theta)} \rightarrow \mathbb{R}^+$.

- (a) Let's assume *Euclidean distance* is the norm that captures our sense of relevant neighborhoods in parameter space. Then our objective can be:

$$u = \underset{\Delta\theta}{\operatorname{argmin}} g^T \Delta\theta + \frac{1}{\alpha} \|\Delta\theta\|_2^2. \quad (2)$$

What is the analytical solution for u in the above problem? What standard optimizer does this recover?

- (b) Now, consider an alternative way of capturing local neighborhood size – the squared infinity norm over parameters. Recall that this is defined as $\|x\|_\infty = \max_i |x_i|$. Our objective is now:

$$u = \underset{\Delta\theta}{\operatorname{argmin}} g^T \Delta\theta + \frac{1}{\alpha} \|\Delta\theta\|_\infty^2. \quad (3)$$

What is the analytical solution for u in this case? Which optimizer does this correspond to?

3. Visualizing features from local linearization of neural nets

In the first discussion, you trained a 1-hidden-layer neural network with SGD and visualized how the network fitted the function leveraging the “elbows” of the non-linear activation function ReLU. In this question, we are going to visualize the effective “features” that correspond to the local linearization of this network in the neighborhood of the parameters.

We provide you with some starter code in the [course repo](#), or you can use [Google Colab](#). For this question, please submit the .pdf export of the jupyter notebook when it is completed. In addition, answer the questions below, including plots from the notebook where relevant.

- (a) **Visualize the features corresponding to $\frac{\partial}{\partial w_i^{(1)}} y(x)$ and $\frac{\partial}{\partial b_i^{(1)}} y(x)$ where $w_i^{(1)}$ are the first hidden layer's weights and the $b_i^{(1)}$ are the first hidden layer's biases.** These derivatives should be evaluated at at least both the random initialization and the final trained network. When visualizing these features,

plot them as a function of the scalar input x , the same way that the notebook plots the constituent “elbow” features that are the outputs of the penultimate layer.

- (b) During training, we can imagine that we have a generalized linear model with a feature matrix corresponding to the linearized features corresponding to each learnable parameter. We know from our analysis of gradient descent, that the singular values and singular vectors corresponding to this feature matrix are important.

Use the SVD of this feature matrix to plot both the singular values and visualize the “principle features” that correspond to the d -dimensional singular vectors multiplied by all the features corresponding to the parameters.

(HINT: Remember that the feature matrix whose SVD you are taking has n rows where each row corresponds to one training point and d columns where each column corresponds to each of the learnable features. Meanwhile, you are going to be plotting/visualizing the “principle features” as functions of x even at places where you don’t have training points.)

- (c) Augment the jupyter notebook to add a second hidden layer of the same size as the first hidden layer, fully connected to the first hidden layer. **Allow the visualization of the features corresponding to the parameters in both hidden layers, as well as the “principle features” and the singular values.**

4. Newton-Schulz Runtime

Let us consider the Newton-Schulz update for a parameter matrix $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, using degree-3 odd polynomial p :

$$p(W) = \frac{1}{2} \left(3I_{d_{\text{out}}} - WW^T \right) W.$$

- (a) First, we will analyze the runtime of a single iteration. Assume that the runtime is dominated by matrix multiplication (which is often true for large $d_{\text{out}}, d_{\text{in}}$) and the runtime of multiplying a $n \times m$ matrix by a $m \times p$ matrix takes $cmnp$ runtime. **What is the runtime of each iteration?**
- (b) Now, consider the case where $d_{\text{out}} \gg d_{\text{in}}$. **Is there a way to compute $p(W)$ faster? Explain how this can be done and report the updated runtime of each iteration?**

(Hint: Consider the Gram matrix $W^T W$ instead of WW^T . What is the runtime of computing $W^T W$? Is there a way to rewrite $p(W)$ so that it uses $W^T W$ instead of WW^T ?)

5. Implementing Muon (Coding Question)

In this assignment, you will implement key components of a Muon optimizer in PyTorch. For this assignment, we recommend using Google Colab as some PCs or laptops may not support GPU-based PyTorch. You will be implementing the code at [q_coding_muon.ipynb](#), then answering the questions in the notebook in your submission. The notebook itself does not need to be submitted.

- (a) Notice that in this Newton-Schulz implementation, we initially scale the Frobenius norm to be at most $\sqrt{3}$. **Can you explain why we choose this particular scaling?** Comment on why this is better than using 1.

(Hint: Newton-Schulz acts on each singular value σ of the matrix separately, sending it to $p(\sigma) = (3\sigma - \sigma^3)/2$. Find the fixed points of this map and check which of them are stable, i.e. $|p'(\sigma^)| < 1$. What happens to a singular value that starts above the largest positive root of p ?)*

- (b) Note that Muon requires that parameters are 2D matrices of shape $d_{out} \times d_{in}$. However, we know that parameters that are convolutional kernels have shape $c_{out} \times c_{in} \times k \times k$, where c denotes number of channels and k is kernel size.

Modern implementations of convolutional layers will (implicitly) transform an input image $\mathbf{x}$ of shape $c_{in} \times h \times w$ to $\mathbf{x}'$ such that each column of $\mathbf{x}'$ has size $c_{in} \cdot k \cdot k$ and corresponds to one flattened "receptive field" of the image (or one patch of the image that a filter passes over to compute one output value). This allows the GPU to use heavily optimized matrix-multiplication capabilities directly. You may look up `im2col` for more details and take a look at <https://arxiv.org/abs/2110.03901> if you are interested.

Given this fact, **how do we modify the convolutional kernel into a $d_{out} \times d_{in}$ matrix C such that the output of the convolutional layer can be expressed as $C\mathbf{x}'$?**

- (c) **Which optimizer performed best between Muon, SGD, and AdamW? Also copy the resulting plots into the submission as well.**
- (d) Compare the loss curves between Muon and MuonSVD. **Are the results expected? Explain why.**
- (e) (Optional) Our implementation of Newton-Schulz is suboptimal in the polynomial used for convergence. The community has developed quintic polynomials that converge faster while still being efficient. **Implement an improved Newton-Schulz and compare. Comment on the speed advantage of the improved Muon relative to the MuonSVD.**
(Hint: You can modify the number of iterations by setting the `ns_iters` parameter in the Muon optimizer)
- (f) (Optional) **What were the best choices for hyperparameters for Muon? What about for AdamW?**

6. 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, 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 $W\mathbf{x}$ 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 $y = W\mathbf{x}$? What is its variance?
- (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 ?

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

7. 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.
- Matteo Guarrera.
- Mert Cemri.
- Sizhe Chen.
- Sheng Shen.
- Joey Hong.
- Gireeja Ranade.