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

Homework 4

This homework is due on Friday, October 3, 2025, at 10:59PM.

1. Newton-Schulz Runtime

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

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

(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 $d_{\mathsf{out}}, d_{\mathsf{in}}$) 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 $d_{\mathsf{out}} \gg d_{\mathsf{in}}$. **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^TW instead of WW^T. What is the runtime of computing W^TW? Is there a way to rewrite $p(W)$ so that it uses W^TW instead of WW^T?)

2. MuP at the Unit Scale

By now, we have seen how the maximal-update parameterization allows us to properly scale updates to weights, based on the shapes of the dense layers. In last week's homework, we hinted at how this procedure can applied either as a layer-wise learning rate, or as a direct adjustment of the forward pass graph.

This time, we will consider how these same principles can be used in the context of low-precision training. GPUs that support low-precision training tend to exhibit a linear speedup in computation time as precision is lowered (i.e. doing a matrix multiplication in fp16 is twice as fast as in fp32). For this reason, we would like to design training algorithms that can remain numerically stable even at precisions as low as 8 bits.

(a) You are designing a neural network training algorithm that will trained with fp8 parameters (assume activations will be calculated in full precision and we don't need to worry about them). **Why would it make sense for parameters to be initialized from $N(0, 1)$ as oppossed to e.g. Xavier initialization?** As a starting point, note that fp8 can only represent 255 possible values, and you can view these values at: https://asawicki.info/articles/fp8_tables.php.

(b) We have initialized our parameters from $N(0, 1)$. However, we have now lost the desirable properties of Xavier initialization, and our activations are exploding as they propagate deeper into the network. To solve this, we can assign a constant (float) scalar to be multiplied with the activations:

$$\mathbf{y} = cW\mathbf{x}.$$

What should the constant scalar c be to recover the benefits of standard Xavier initialization?

(c) Now, let us consider an update to the weights ΔW. We would like to properly scale this update, such that the resulting $\Delta\mathbf{y} = c\Delta W x$ is controlled. Assume that $\mathbf{x}$ has an RMS norm of 1. **What should the maximum spectral norm of ΔW be such that $\Delta\mathbf{y}$ has an RMS norm no larger than 1?**

(d) Let us consider the case of SignSGD. Assume that our minibatch is of size 1. You saw in discussion that $\text{sign}(\nabla_W L)$ is a rank-1 matrix. **What learning rate α is required to ensure that the overall update of $\alpha \cdot \textbf{sign}(\nabla_W L)$ satisfies the spectral norm constraint from part (c)?** Ensure your answer works on rectangular weight matrices.

(e) Let us consider the usage of Muon-style methods to orthogonalize our gradients. Consider the following orthogonalized update rule:

$$U, \Sigma, V^T = \text{SVD}(\nabla_W L) \tag{1}$$

$$\Delta W = \alpha \cdot UV^T. \tag{2}$$

where we use the compact form of the SVD. **What learning rate α is required to ensure that the overall update ΔW satisfies the spectral norm constraint from part (c)?**

(f) SignGD, Adam, and Muon share a similar property that the global scale of raw gradients does not affect the final update direction. Now consider the backwards pass of a series of dense layers, where each layer follows the scaled definition from part (b): $\mathbf{x}_{n+1} = c_n W_n \mathbf{x}_n$. You may assume there is no activation function for simplicity. Recall that $\nabla_{x_n} L$ can be recursively calculated from $\nabla_{x_{n+1}} L$. **Is there a setting where the scale of these *intermediate backpropagated gradients* can also be ignored?**

(g) In the setting above, without any adjustments, **will intermediate backpropagated gradients suffer from an explosion or vanishing effect as they are backpropagated?** You may assume that W is rank-1, such that the spectral norm is equal to the Frobenius norm, and each parameter is unit scaled. What constants should the intermediate backpropagated gradients be multiplied by to ensure that they remain stable?

3. Understanding Convolution as Finite Impulse Response Filter

For the discrete time signal, the output of linear time invariant system is defined as:

$$y[n] = x[n] * h[n] = \sum_{i=-\infty}^{\infty} x[n-i] \cdot h[i] = \sum_{i=-\infty}^{\infty} x[i] \cdot h[n-i] \tag{3}$$

where x is the input signal, h is impulse response (also referred to as the filter). Please note that the convolution operations is to 'flip and drag'. But for neural networks, we simply implement the convolutional layer without flipping and such operation is called correlation. Interestingly, in CNN those two operations are equivalent because filter weights are initialized and updated. Even though you implement 'true' convolution, you just ended up with getting the flipped kernel. **In this question, we will follow the definition in 3.**

Now let's consider rectangular signal with the length of L (sometimes also called the "rect" for short, or, alternatively, the "boxcar" signal). This signal is defined as:

$$x(n) = \begin{cases} 1 & n = 0, 1, 2, ..., L-1 \\ 0 & \text{otherwise} \end{cases}$$

Here's an example plot for $L = 7$, with time indices shown from -2 to 8 (so some implicit zeros are shown):

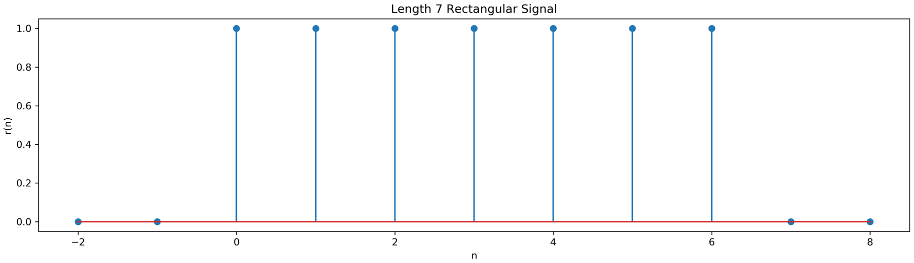

Figure 1: The rectangular signal with the length of 7

(a) The impulse response is define as:

$$h(n) = (\frac{1}{2})^n u(n) = \begin{cases} (\frac{1}{2})^n & n = 0, 1, 2, ... \\ 0 & \text{otherwise} \end{cases}$$

Compute and plot the convolution of $x(n)$ and $h(n)$. For illustrative purposes, your plot should start at -6 and end at +12.

(b) Now let's shift $x(n)$ by N, i.e. $x_2(n) = x(n - N)$. Let's put $N = 5$ **Then, compute $y_2(n) = h(n) * x_2(n)$. Which property of the convolution can you find?**

Now, let's extend 1D to 2D. The example of 2D signal is the image. The operation of 2D convolution is defined as follows:

$$y[m,n] = x[m,n] * h[m,n] = \sum_{i,j=-\infty}^{\infty} x[m-i, n-j] \cdot h[i,j] = \sum_{i,j=-\infty}^{\infty} x[i,j] \cdot h[m-i, n-j] \tag{4}$$

, where x is input signal, h is FIR filter and y is the output signal.

(c) 2D matrices, x and h are given like below:

$$x = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix} \tag{5}$$

$$h = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{6}$$

Then, evaluate y. Assume that there is no pad and stride is 1.

(d) Now let's consider striding and padding. Evaluate y for following cases:

i. stride, pad = 1, 1

ii. stride, pad = 2, 1

4. Feature Dimensions of Convolutional Neural Network

In this problem, we compute output feature shape of convolutional layers and pooling layers, which are building blocks of CNN. Let's assume that input feature shape is $W \times H \times C$, where W is the width, H is the height and C is the number of channels of input feature.

(a) A convolutional layer has 4 architectural hyperparameters: the filter size(K), the padding size (P), the stride step size (S) and the number of filters (F). **How many weights and biases are in this convolutional layer? And what is the shape of output feature that this convolutional layer produces?**

(b) A max pooling layer has 2 architectural hyperparameters: the stride step size(S) and the "filter size" (K). **What is the output feature shape that this pooling layer produces?**

(c) Let's assume that we have the CNN model which consists of L successive convolutional layers and the filter size is K and the stride step size is 1 for every convolutional layer. Then **what is the receptive field size of the last output**?

(d) Consider a downsampling layer (e.g. pooling layer and strided convolution layer). In this problem, we investigate pros and cons of downsampling layers. This layer reduces the output feature resolution and this implies that the output features lose a certain amount of spatial information. Therefore when we design CNNs, we usually increase tchannel length to compensate this loss. For example, if we apply a max pooling layer with a kernel size of 2 and a stride of 2, we increase the output feature size by a factor of 2. **If we apply this max pooling layer, how much does the receptive field increases? Explain the advantage of decreasing the output feature resolution with the perspective of reducing the amount of computation.**

(e) Let's take a real example. We are going to describe a convolutional neural net using the following pieces:

- CONV3-F denotes a convolutional layer with F different filters, each of size $3 \times 3 \times C$, where C is the depth (i.e. number of channels) of the activations from the previous layer. Padding is 1, and stride is 1.
- POOL2 denotes a 2×2 max-pooling layer with stride 2 (pad 0)
- FLATTEN just turns whatever shape input tensor into a one-dimensional array with the same values in it.
- FC-K denotes a fully-connected layer with K output neurons.

Note: All CONV3-F and FC-K layers have biases as well as weights. **Do not forget the biases when counting parameters.**

Now, we are going to use this network to do inference on a single input. **Fill in the missing entries in this table of the size of the activations at each layer, and the number of parameters at each layer. You can/should write your answer as a computation (e.g.** $128 \times 128 \times 3$**) in the style of the already filled-in entries of the table.**

Layer	Number of Parameters	Dimension of Activations
Input	0	$28 \times 28 \times 1$
CONV3-10		$28 \times 28 \times 10$
POOL2	0	$14 \times 14 \times 10$
CONV3-10	$3 \times 3 \times 10 \times 10 + 10$	
POOL2		
FLATTEN	0	490
FC-3		3

(f) Consider a new architecture:

CONV2-3 → ReLU → CONV2-3 → ReLU → GAP (Global Average Pool) → FC-3

Each CONV2-3 layer has stride of 1 and padding of 1. Note that we use **circular padding** (i.e. wrap-around) for this task. Instead of using zeros, circular padding makes it as though the virtual column before the first column is the last column and the virtual row before the first row is the last row — treating the image as though it was on a torus.

Here, the GAP layer is an average pooling layer that computes the per-channel means over the entire input image.

You are told the behavior for an input image with a horizontal edge, $\mathbf{x}_1$ and an image with a vertical edge, $\mathbf{x}_2$:

$$\mathbf{x}_1 = \begin{bmatrix} 0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \end{bmatrix} \qquad \mathbf{x}_2 = \begin{bmatrix} 0&1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \end{bmatrix}$$

Suppose we knew that the GAP output features when fed $\mathbf{x}_1$ and $\mathbf{x}_2$ are

$$\mathbf{g}_1 = f(\mathbf{x}_1) = \begin{bmatrix} 0.8 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{g}_2 = f(\mathbf{x}_2) = \begin{bmatrix} 0 \\ 0.8 \\ 0 \end{bmatrix}$$

Use what you know about the invariances/equivariances of convolutional nets to compute the $\mathbf{g}_i$ corresponding to the following $\mathbf{x}_i$ images.

- $\mathbf{x}_3 = \begin{bmatrix} 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1&0 \end{bmatrix}$

- $\mathbf{x}_4 = \begin{bmatrix} 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0 \end{bmatrix}$

5. Desigining 2D Filter (Coding Question)

Convolutional layer, which is the most important building block of CNN, actively utilizes the concept of filters used in traditional image processing. Therefore, it is quite important to know and understand the types and operation of image filters.

Look at HandDesignFilters.ipynb. In this notebook, we will design two convolution filters by hand to understand the operation of convolution:

(a) **Blurring filter.**

(b) **Edge detection filter.**

For each type of filter, please **include the image generated by the Jupyter Notebook in your submission to the written assignment**. The image should be added to the PDF document that you will be submitting.

6. Inductive Bias of CNNs (Coding Question)

In this problem, you will follow the edge_detection.ipynb notebook to understand the inductive bias of CNNs.

1 Overfitting Models to Small Dataset: **Fill out notebook section (Q1).**

   (i) Can you find any interesting patterns in the learned filters?

2 Sweeping the Number of Training Images: **Fill out notebook section (Q2).**

   (i) Compare the learned kernels, untrained kernels, and edge-detector kernels. What do you observe?

   (ii) We freeze the convolutional layer and train only final layer (classifier). In a high data regime, the performance of CNN initialized with edge detectors is worse than CNN initialized with random weights. Why do you think this happens?

3 Checking the Training Procedures: **Fill out notebook section (Q3).**

   (i) List every epochs that you trained the model. Final accuracy of CNN should be at least 90% for 10 images per class.

   (ii) Check the learned kernels. What do you observe?

   (iii) (optional) You might find that with the high number of epochs, validation loss of MLP is increasing whild validation accuracy increasing. How can we interpret this?

   (iv) (optional) Do hyperparameter tuning. And list the best hyperparameter setting that you found and report the final accuracy of CNN and MLP.

   (v) How much more data is needed for MLP to get a competitive performance with CNN?

4 Domain Shift between Training and Validation Set: **Fill out notebook section (Q4).**

   (i) Why do you think the confusion matrix looks like this?

   (ii) Why do you think MLP fails to learn the task while CNN can learn the task? (Hint: Think about the model architecture.)

5 When CNN is Worse than MLP: **Fill out notebook section (Q5).**

   (i) What do you observe? What is the reason that CNN is worse than MLP? (Hint: Think about the model architecture.)

   (ii) Assuming we are increasing kernel size of CNN. Does the validation accuracy increase or decrease? Why?

   (iii) How do the learned kernels look like? Explain why.

6 Increasing the Number of Classes: **Fill out notebook section (Q6).**

(i) Compare the performance of CNN with max pooling and average pooling. What are the advantages of each pooling method?

7. Weights and Gradients in a CNN

In this homework assignment, we aim to accomplish two objectives. Firstly, we seek to comprehend that the weights of a CNN are a weighted average of the images in the dataset. This understanding is crucial in answering a commonly asked question: does a CNN memorize images during the training process? Additionally, we will analyze the impact of spatial weight sharing in convolution layers. Secondly, we aim to gain an understanding of the behavior of max-pooling and avg-pooling in backpropagation. By accomplishing these objectives, we will enhance our knowledge of CNNs and their functioning.

Let's consider a convolution layer with input matrix $\mathbf{X} \in \mathbb{R}^{n \times n}$,

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \cdots & x_{n,n} \end{bmatrix}, \tag{7}$$

weight matrix $\mathbf{w} \in \mathbb{R}^{k \times k}$,

$$\mathbf{w} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,k} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1} & w_{k,2} & \cdots & w_{k,k} \end{bmatrix}, \tag{8}$$

and output matrix $\mathbf{Y} \in \mathbb{R}^{m \times m}$,

$$\mathbf{Y} = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,m} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m,1} & y_{m,2} & \cdots & y_{m,m} \end{bmatrix}. \tag{9}$$

For simplicity, we assume the number of the input channel (of $\mathbf{X}$ is) and the number of the output channel (of output $\mathbf{Y}$) are both 1, and the convolutional layer has no padding and a stride of 1.

Then for all i, j,

$$y_{i,j} = \sum_{h=1}^{k} \sum_{l=1}^{k} x_{i+h-1,j+l-1} w_{h,l}, \tag{10}$$

or

$$\mathbf{Y} = \mathbf{X} * \mathbf{w}, \tag{11}$$

, where $*$ refers to the convolution operation. For simplicity, we omitted the bias term in this question. Suppose the final loss is $\mathcal{L}$, and the upstream gradient is $d\mathbf{Y} \in \mathbb{R}^{m,m}$,

$$d\mathbf{Y} = \begin{bmatrix} dy_{1,1} & dy_{1,2} & \cdots & dy_{1,m} \\ dy_{2,1} & dy_{2,2} & \cdots & dy_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ dy_{m,1} & dy_{m,2} & \cdots & dy_{m,m} \end{bmatrix}, \tag{12}$$

where $dy_{i,j}$ denotes $\dfrac{\partial \mathcal{L}}{\partial y_{i,j}}$.

(a) **Derive the gradient to the weight matrix** $d\mathrm{w} \in \mathbb{R}^{k,k}$,

$$d\mathbf{w} = \begin{bmatrix} dw_{1,1} & dw_{1,2} & \cdots & dw_{1,k} \\ dw_{2,1} & dw_{2,2} & \cdots & dw_{2,k} \\ \vdots & \vdots & \ddots & \vdots \\ dw_{k,1} & dw_{k,2} & \cdots & dw_{k,k} \end{bmatrix}, \tag{13}$$

where $dw_{h,l}$ denotes $\dfrac{\partial \mathcal{L}}{\partial w_{h,l}}$. Also, **derive the weight after one SGD step with a batch of a single image**.

(b) The objective of this part is to investigate the effect of spatial weight sharing in convolution layers on the behavior of gradient norms with respect to changes in image size.

For simplicity of analysis, we assume $x_{i,j}, dy_{i,j}$ are independent random variables, where for all i, j:

$$\mathbb{E}[x_{i,j}] = 0, \tag{14}$$

$$\mathrm{Var}(x_{i,j}) = \sigma_x^2, \tag{15}$$

$$\mathbb{E}[dy_{i,j}] = 0, \tag{16}$$

$$\mathrm{Var}(dy_{i,j}) = \sigma_g^2. \tag{17}$$

Derive the mean and variance of $dw_{h,l} = \dfrac{\partial \mathcal{L}}{\partial W_{h,l}}$ **for each** i, j **a function of** n, k, σ_x, σ_g**. What is the asymptotic growth rate of the <u>standard deviation</u> of the gradient on** $dw_{h,l}$ **with respect to the length and width of the image** n**?**

Hint: there should be no m in your solution because m can be derived from n and k.

Hint: you cannot assume that $x_{i,j}$ and $dy_{i,j}$ follow normal distributions in your derivation or proof.

(c) **For a network with only 2x2 max-pooling layers (no convolution layers, no activations), what will be** $d\mathrm{X} = [dx_{i,j}] = [\dfrac{\partial \mathcal{L}}{\partial x_{i,j}}]$**? For a network with only 2x2 average-pooling layers (no convolution layers, no activations), what will be** $d\mathrm{X}$**?**

HINT: Start with the simplest case first, where $\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{bmatrix}$. *Further assume that top left value is selected by the max operation. i.e.*

$$y_{1,1} = x_{1,1} = \max(x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2}) \tag{18}$$

Then generalize to higher dimension and arbitrary max positions.

(d) Following the previous part, **discuss the advantages of max pooling and average pooling** in your own words.

Hint: you may find it helpful to finish the question "Inductive Bias of CNNs (Coding Question)" before working on this question

8. 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:

- Joey Hong.
- Anant Sahai.
- Kevin Frans.
- Suhong Moon.
- Dominic Carrano.
- Babak Ayazifar.
- Sukrit Arora.
- Romil Bhardwaj.
- Fei-Fei Li.
- Sheng Shen.
- Jake Austin.
- Kevin Li.
- Kumar Krishna Agrawal.
- Peter Wang.

- Qiyang Li.
- Linyuan Gong.
- Long He.