CS 182, Fall 2025.

Kian Hekmatnejad

Special Participation A: HW5 Using Mistral AI

0 Introduction

I will be using Mistral to solve the non-coding parts of HW5. We will show our prompts for each question and analyze the results. My annotations will be marked in blue and everything else will be generated by Mistral

1 Problem 1

1.1 Prompt/Explanation

Here is my prompt for the first question. I aim to be as straightforward as possible with my prompting, essentially repeating the questions in their form on the homework sheet and asking the model to solve them:
I need you to solve problems on deep neural networks. The first question is on convolutional neural networks. Throughout this problem, we will use the convention of NOT flipping the filter before dragging it over the signal. Here are the individual parts of the first question. List your solutions and work to get them:
a) List two reasons we typically prefer convolutional layers instead of fully connected layers when working with image data.
b) Consider the following 1D signal: [1, 4, 0, -2, 3]. After convolution with a length-3 filter, no padding, stride=1, we get the following sequence: [-2, 2, 11]. What was the filter?
c) If our 2D input is the 2x2 matrix [-1, 2: 3, 1] and the 2D filter is the 2x2 matrix [1, -1; 0, 1], what is the output of transpose convolution with pad=0 and stride=1? Note that transpose convolution is an operation to help us upsample a signal (increase the resolution). For example, if our original signal were [a, b, c] and we perform transpose convolution with pad=0 and stride=2, with the filter [x, y, z], the output would be [ax, ay, az +bx, by, bz +cx, cy, cz]. Notice that the entries of the input are multiplied by each of the entries of the filter. Overlaps are summed. Also notice how for a fixed filter size and stride, the dimensions of the input and output are swapped compared to standard convolution

1.2 Answer

Let's tackle each part of your question step by step.

a) Two reasons we typically prefer convolutional layers instead of fully connected layers when working with image data:**

1. **Parameter Sharing:** Convolutional layers use the same filter (kernel) across the entire input, drastically reducing the number of parameters compared to fully connected layers. This makes the model more efficient and less prone to overfitting.

2. **Spatial Hierarchy Preservation:** Convolutional layers exploit the spatial locality of pixels in images. They capture local patterns (e.g., edges, textures) and hierarchically combine them into more complex features, which is ideal for image data where nearby pixels are highly correlated.

These responses for part (a) are correct. It properly identifies the reduction in number of parameters and identifies inductive bias in "spatial hierarchy preservation"

b) Finding the length-3 filter for the 1D signal:**

Given:

- Input signal: $[1, 4, 0, -2, 3]$
- Output after convolution: $[-2, 2, 11]$
- Filter length: 3

- Stride: 1
- No padding

Let the filter be $[w_1, w_2, w_3]$.
The convolution operation (without flipping the filter) is:

$$\text{output}[i] = \text{input}[i] \cdot w_1 + \text{input}[i+1] \cdot w_2 + \text{input}[i+2] \cdot w_3$$

For the given output:

$$-2 = 1 \cdot w_1 + 4 \cdot w_2 + 0 \cdot w_3 2 = 4 \cdot w_1 + 0 \cdot w_2 + (-2) \cdot w_3 11 = 0 \cdot w_1 + (-2) \cdot w_2 + 3 \cdot w_3$$

This gives the system of equations:

$$\begin{cases} w_1 + 4w_2 = -2 \\ 4w_1 - 2w_3 = 2 \\ -2w_2 + 3w_3 = 11 \end{cases}$$

Solving: From the first equation: $w_1 = -2 - 4w_2$. From the third equation: $w_3 = \frac{11+2w_2}{3}$.
Substitute into the second equation:

$$4(-2 - 4w_2) - 2\left(\frac{11 + 2w_2}{3}\right) = 2$$

$$-8 - 16w_2 - \frac{22 + 4w_2}{3} = 2$$

Multiply by 3:

$$-24 - 48w_2 - 22 - 4w_2 = 6$$

$$-52w_2 = 52 \implies w_2 = -1$$

Now, $w_1 = -2 - 4(-1) = 2$ and $w_3 = \frac{11+2(-1)}{3} = 3$.
The filter is $[2, -1, 3]$.

The model correctly identifies the filter using the proper system of equations. The methodology in the system of equations is identical to the work done in the answer key

c) Transpose Convolution for the 2D Input:**

Given: - Input: $\begin{bmatrix} -1 & 2 \\ 3 & 1 \end{bmatrix}$ - Filter: $\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$ - Pad: 0 - Stride: 1

Transpose convolution (also called deconvolution) upsamples the input by "spreading" each input value according to the filter, summing overlaps.

For 2D transpose convolution with stride 1 and no padding, the output size is:

$$\text{output\_size} = \text{input\_size} + \text{filter\_size} - 1 = 2 + 2 - 1 = 3$$

So, the output is a 3×3 matrix.

Step-by-step: - Each input value is multiplied by the entire filter and placed in the output grid, summing overlaps.

Let's denote the input as I and the filter as F:

$$I = \begin{bmatrix} -1 & 2 \\ 3 & 1 \end{bmatrix}, \quad F = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

The output O is computed as:

$$O[i,j] = \sum_{m=0}^{1} \sum_{n=0}^{1} I[m,n] \cdot F[i-m, j-n]$$

where i, j range over the output indices.

Calculating each element: - $O[0,0] = I[0,0] \cdot F[0,0] = -1 \cdot 1 = -1$ - $O[0,1] = I[0,0] \cdot F[0,1] + I[0,1] \cdot F[0,0] = -1 \cdot (-1) + 2 \cdot 1 = 1 + 2 = 3$ - $O[0,2] = I[0,1] \cdot F[0,1] = 2 \cdot (-1) = -2$ - $O[1,0] = I[1,0] \cdot F[0,0] = 3 \cdot 1 = 3$ - $O[1,1] = I[1,0] \cdot F[0,1] + I[1,1] \cdot F[0,0] + I[0,0] \cdot F[1,0] + I[0,1] \cdot F[1,1] = 3 \cdot (-1) + 1 \cdot 1 + (-1) \cdot 0 + 2 \cdot 1 = -3 + 1 + 0 + 2 = 0$ - $O[1,2] = I[1,1] \cdot F[0,1] + I[0,1] \cdot F[1,1] = 1 \cdot (-1) + 2 \cdot 1 = -1 + 2 = 1$ - $O[2,0] = I[1,0] \cdot F[1,0] = 3 \cdot 0 = 0$ - $O[2,1] = I[1,0] \cdot F[1,1] + I[1,1] \cdot F[1,0] = 3 \cdot 1 + 1 \cdot 0 = 3$ - $O[2,2] = I[1,1] \cdot F[1,1] = 1 \cdot 1 = 1$

Final output:

$$O = \begin{bmatrix} -1 & 3 & -2 \\ 3 & 0 & 1 \\ 0 & 3 & 1 \end{bmatrix}$$

This solution is incorrect, particularly in the middle entry of the output matrix. From the start, the model uses a different approach from my own solution and the staff solution by summing over individual terms in the input and filter matrices instead of adding scalar multiples of the shifted 3x3 filter matrix based on the current input. This leads to an error in the calculation of O[1, 1]. It should have been

$$= I[0,0] \cdot F[1,1] + I[0,1] \cdot F[1,0] + I[1,0] \cdot F[0,1] + I[1,1] \cdot F[0,0] = 3$$

To correct this issue, I prompted Mistral to identify and alleviate the mistake with the following prompt: you may have miscalculated the output of the transpose convolution. check your work. It answered:
Let's carefully re-calculate the transpose convolution step-by-step for the 2D case.

—

c) Transpose Convolution for the 2D Input (Revisited):

Given: - Input: $\begin{bmatrix} -1 & 2 \\ 3 & 1 \end{bmatrix}$ - Filter: $\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$ - Pad: 0 - Stride: 1

Transpose convolution upsamples the input by "spreading" each input value according to the filter, summing overlaps. The output size is:

$$\text{output\_size} = \text{input\_size} + \text{filter\_size} - 1 = 2 + 2 - 1 = 3$$

So, the output is a 3×3 matrix.

Step-by-step: For each position in the output, we place the filter centered at that position (with zero-padding outside the input) and multiply by the input values, summing the results.

Let's denote the input as I and the filter as F:

$$I = \begin{bmatrix} -1 & 2 \\ 3 & 1 \end{bmatrix}, \quad F = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

The output O is computed as:

$$O[i,j] = \sum_{m=0}^{1} \sum_{n=0}^{1} I[m,n] \cdot F[i-m, j-n]$$

where i, j range over the output indices.

Calculating each element:

- $O[0,0] = I[0,0] \cdot F[0,0] = -1 \cdot 1 = -1$ - $O[0,1] = I[0,0] \cdot F[0,1] + I[0,1] \cdot F[0,0] = -1 \cdot (-1) + 2 \cdot 1 = 1 + 2 = 3$ - $O[0,2] = I[0,1] \cdot F[0,1] = 2 \cdot (-1) = -2$

- $O[1,0] = I[1,0] \cdot F[0,0] + I[0,0] \cdot F[1,0] = 3 \cdot 1 + (-1) \cdot 0 = 3 + 0 = 3$ - $O[1,1] = I[1,0] \cdot F[0,1] + I[1,1] \cdot F[0,0] + I[0,0] \cdot F[1,1] + I[0,1] \cdot F[1,0] = 3 \cdot (-1) + 1 \cdot 1 + (-1) \cdot 1 + 2 \cdot 0 = -3 + 1 - 1 + 0 = -3$ - $O[1,2] = I[1,1] \cdot F[0,1] + I[0,1] \cdot F[1,1] = 1 \cdot (-1) + 2 \cdot 1 = -1 + 2 = 1$

- $O[2,0] = I[1,0] \cdot F[1,0] + I[1,1] \cdot F[0,0] = 3 \cdot 0 + 1 \cdot 1 = 0 + 1 = 1$ - $O[2,1] = I[1,1] \cdot F[1,1] + I[1,0] \cdot F[1,0] = 1 \cdot 1 + 3 \cdot 0 = 1 + 0 = 1$ - $O[2,2] = I[1,1] \cdot F[1,1] = 1 \cdot 1 = 1$

Final output:

$$O = \begin{bmatrix} -1 & 3 & -2 \\ 3 & -3 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

—

Corrected Summary of Answers: - **(a)** Parameter sharing and spatial hierarchy preservation. - **(b)** The filter is $[2, -1, 3]$. - **(c)** The output of the transpose convolution is:

$$\begin{bmatrix} -1 & 3 & -2 \\ 3 & -3 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

This time, Mistral correctly calculated $O[1, 1]$, still using the same element-wise operations but using the proper elements this time. This misconception is difficult to understand - maybe it just miswrote some numbers for the middle element or there is some kind of deeper confusion.

2 Problem 2.

2.1 Prompt

Here is my prompt for this question. Again, I pretty much copied over the questions from the homework, also attaching the image and instructing Mistral to solve the problemes with justification:

Let's do another problem, this time on batch normalization for Convolutional Neural Networks. Here are the individual parts, please answer each one, showing your work:

a) Consider the diagram attached to this prompt, where the shaded blocks are the entries participating in one normalization step for a CNN-type architecture. N represents the mini-batch, H, W represent the different pixels of the "image" at this layer, and C represents different channels. Which of the images (A, B, C), denotes the process for batch normalization and which one denotes the process of layer normalization

b) Consider a simplified BN where we do not divide by the standard deviation of the data batch. Instead, we just de-mean our data batch before applying the scaling factor γ and shifting factor β. For simplicity, consider scalar data in an n-sized batch: $[x1, x2, ..., xn]$. Specifically, we let $x_i = x_i - \mu$ where μ is the average $1/n \sum_{j=1}^{n} x_j$ across the batch and output $[y1, y2, ..., yn]$ where $y_i = \gamma x_i + \beta$ to the next layer. Assume we have a final loss L somewhere downstream. Calculate $\partial L/\partial x_i$ in terms of $\partial L/\partial y_j$ for j = 1, . . . , n as well as γ and β as needed. Numerically, what is $\partial L/\partial x_1$ when n = 1 and our input batch just consists of $[x_1]$ with an output batch of $[y_1]$? (Your answer should be a real number. No need to justify.) What happens when $n \to \infty$? (Feel free to assume here that all relevant quantities are bounded.)

2.2 Answer

Let's address each part of your question in detail.

—

a) Batch Normalization vs. Layer Normalization in the Diagram

- **Batch Normalization (BatchNorm):** BatchNorm normalizes **across the batch dimension (N)** for each channel and spatial location. In the diagram, this corresponds to **A**, where the shaded block spans the batch dimension (N) for a fixed channel (C) and spatial location (H, W).

- **Layer Normalization (LayerNorm):** LayerNorm normalizes **across the channel dimension (C)** for each spatial location and batch element. In the diagram, this corresponds to **B**, where the shaded block spans the channel dimension (C) for a fixed batch element (N) and spatial location (H, W). *(Note: C is sometimes used for channels in LayerNorm, but the key is that normalization is done across features/channels for each data point, not across the batch.)*

Summary: - **A:** Batch Normalization - **B:** Layer Normalization - **C:** Neither (this would be GroupNorm or InstanceNorm, depending on the exact grouping)
The model correctly identifies which diagram represents layer norm and batch norm.

b) Simplified Batch Normalization Gradient
Given: - Input batch: $[x_1, x_2, \ldots, x_n]$ - Mean: $\mu = \frac{1}{n}\sum_{j=1}^{n} x_j$ - Centered data: $\hat{x}_i = x_i - \mu$ - Output: $y_i = \gamma\hat{x}_i + \beta$ - Loss: L
Goal: Find $\frac{\partial L}{\partial x_i}$ in terms of $\frac{\partial L}{\partial y_j}$, γ, and β.

—

Step 1: Express $\frac{\partial L}{\partial x_i}$ using the chain rule:

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial x_i}$$

Since $y_j = \gamma\hat{x}_j + \beta$ and $\hat{x}_j = x_j - \mu$, we have:

$$\frac{\partial y_j}{\partial x_i} = \gamma\frac{\partial \hat{x}_j}{\partial x_i}$$

Step 2: Compute $\frac{\partial \hat{x}_j}{\partial x_i}$:

$$\hat{x}_j = x_j - \mu = x_j - \frac{1}{n}\sum_{k=1}^{n} x_k$$

So,

$$\frac{\partial \hat{x}_j}{\partial x_i} = \delta_{ij} - \frac{1}{n}$$

where δ_{ij} is the Kronecker delta (1 if $i = j$, else 0).
Step 3: Substitute back:

$$\frac{\partial L}{\partial x_i} = \sum_{j=1}^{n} \frac{\partial L}{\partial y_j}\gamma\left(\delta_{ij} - \frac{1}{n}\right)$$

$$\frac{\partial L}{\partial x_i} = \gamma\left(\frac{\partial L}{\partial y_i} - \frac{1}{n}\sum_{j=1}^{n}\frac{\partial L}{\partial y_j}\right)$$

—

Numerical Value for $n = 1$:
For $n = 1$: - $\mu = x_1$ - $\hat{x}_1 = x_1 - x_1 = 0$ - $y_1 = \gamma \cdot 0 + \beta = \beta$
Since y_1 does not depend on x_1 (because $\hat{x}_1 = 0$):

$$\frac{\partial L}{\partial x_1} = 0$$

—

Behavior as $n \to \infty$:
As $n \to \infty$, the term $\frac{1}{n}\sum_{j=1}^{n}\frac{\partial L}{\partial y_j}$ becomes negligible compared to $\frac{\partial L}{\partial y_i}$ (assuming bounded gradients). Thus:

$$\frac{\partial L}{\partial x_i} \approx \gamma\frac{\partial L}{\partial y_i}$$

The gradient becomes **local** to each x_i, and the dependence on the batch mean vanishes.

— The solution to part (b) is also entirely correct. The model correctly calculates the general gradient and then uses the correct inputs to find it in the special 1 dimensional case. It also identifies the correct behavior as n goes to ∞, decribing what happens to the gradients. This is an improvement on the error on question 1.

Summary of Answers: - **(a)** A: Batch Normalization, B: Layer Normalization - **(b)** - $\frac{\partial L}{\partial x_i} = \gamma \left(\frac{\partial L}{\partial y_i} - \frac{1}{n} \sum_{j=1}^{n} \frac{\partial L}{\partial y_j} \right)$ - For $n = 1$: $\frac{\partial L}{\partial x_1} = 0$ - As $n \to \infty$: $\frac{\partial L}{\partial x_i} \approx \gamma \frac{\partial L}{\partial y_i}$

3 Problem 3.

3.1 Prompt

This problem included some background information, which I did include my prompt, but I did not include the figures that may have been helpful because I wanted to see how the model would behave without them. Here is my prompt:

You will solve another question on depthwise separable convolutions. Depthwise separable convolutions are a type of convolutional operation used in deep learning for image processing tasks. Unlike traditional convolutional operations, which perform both spatial and channel-wise convolutions simultaneously, depthwise separable convolutions decompose the convolution operation into two separate operations: Depthwise convolution and Pointwise convolution. This can be viewed as a low-rank approximation to a traditional convolution. For simplicity, throughout this problem, we will ignore biases while counting learnable parameters. Solve each of these parts:

a) Suppose the input is a three-channel 224 × 224-resolution image, the kernel size of the convolutional layer is 3 × 3, and the number of output channels is 4. What is the number of learnable parameters in the traditional convolution layer?

b) Depthwise separable convolution consists of two parts: depthwise convolutions followed by pointwise convolutions. Suppose the input is still a three-channel 224 × 224-resolution image. The input first goes through depthwise convolutions, where the number of output channels is the same as the number of input channels, and there is no "cross talk" between different channels. Then, this intermediate output goes through pointwise convolutions, which is basically a traditional convolution with the filter size being 1 × 1. Assume that we have 4 output channels. What is the total number of learnable parameters of the depthwise separable convolution layer which consists of both depthwise and pointwise convolutions?

3.2 Answer

Let's calculate the number of learnable parameters for both traditional convolution and depthwise separable convolution.

—

å **a) Traditional Convolution Layer**

Given: - Input: 3-channel, 224×224 image - Kernel size: 3×3 - Number of output channels: 4

Number of learnable parameters: For a traditional convolution, the number of parameters is:

$$(\text{kernel height} \times \text{kernel width} \times \text{input channels} + 1) \times \text{output channels}$$

(We ignore the bias term as per the problem statement.)

So,

$$(3 \times 3 \times 3) \times 4 = 27 \times 4 = 108$$

Answer: The traditional convolution layer has **108** learnable parameters.

—

b) Depthwise Separable Convolution Layer

1. Depthwise Convolution: - Input channels: 3 - Output channels: 3 (same as input) - Kernel size: 3×3

For depthwise convolution, each input channel is convolved with a single filter. So, the number of parameters is:

$$(\text{kernel height} \times \text{kernel width} \times \text{input channels}) = 3 \times 3 \times 3 = 27$$

2. Pointwise Convolution: - Input channels: 3 (from depthwise output) - Output channels: 4 - Kernel size: 1×1

For pointwise convolution, the number of parameters is:

$$(1 \times 1 \times \text{input channels}) \times \text{output channels} = (1 \times 1 \times 3) \times 4 = 12$$

Total Parameters:

$$27 \text{ (depthwise)} + 12 \text{ (pointwise)} = 39$$

Answer: The depthwise separable convolution layer has **39** learnable parameters.
The model correctly calculates the number of learnable parameters in each case, using the proper formula for the number of parameters in a traditional convolution and depthwise convolution

4 Problem 4.

4.1 Prompt

For this prompt, instead of writing over the whole question, I copied over a screenshot of the entire problem and asked Mistral to solve it based on that. Here is my prompt: Now you will solve a question on regularization and dropout. Instead of writing out the question, I have attached an image of all the questions. Please solve them as you have done previously

4.2 Response

Let's tackle each part of the problem step-by-step, using the notation and context from the image.

—

(a) Derive the Dropout Objective
Given:

$$\mathcal{L}(\tilde{\mathbf{w}}) = \mathbb{E}_{R\sim\text{Bernoulli}(p)}\left[\|\mathbf{y} - (R \odot X)\tilde{\mathbf{w}}\|_2^2\right]$$

where $R_{i,j} \sim$ i.i.d Bernoulli(p).
Goal: Show that this can be rewritten as:

$$\mathcal{L}(\tilde{\mathbf{w}}) = \|\mathbf{y} - pX\tilde{\mathbf{w}}\|_2^2 + p(1-p)\|\tilde{\Gamma}\tilde{\mathbf{w}}\|_2^2$$

where $\tilde{\Gamma}$ is a diagonal matrix with the norm of the j-th column of X as its j-th diagonal entry.

—

Step 1: Expand the Expectation

$$\mathbb{E}_R\left[\|\mathbf{y} - (R \odot X)\tilde{\mathbf{w}}\|_2^2\right] = \mathbb{E}_R\left[(\mathbf{y} - (R \odot X)\tilde{\mathbf{w}})^T(\mathbf{y} - (R \odot X)\tilde{\mathbf{w}})\right]$$

Expanding the squared norm:

$$= \mathbb{E}_R\left[\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T(R \odot X)\tilde{\mathbf{w}} + \tilde{\mathbf{w}}^T(R \odot X)^T(R \odot X)\tilde{\mathbf{w}}\right]$$

Step 2: Linearity of Expectation

$$= \mathbf{y}^T\mathbf{y} - 2\mathbb{E}_R\left[\mathbf{y}^T(R \odot X)\right]\tilde{\mathbf{w}} + \mathbb{E}_R\left[\tilde{\mathbf{w}}^T(R \odot X)^T(R \odot X)\tilde{\mathbf{w}}\right]$$

Step 3: Compute $\mathbb{E}_R[R \odot X]$ and $\mathbb{E}_R[(R \odot X)^T(R \odot X)]$
- $\mathbb{E}_R[R \odot X] = pX$ (since each entry of R is 1 with probability p and 0 otherwise). - $\mathbb{E}_R[(R \odot X)^T(R \odot X)] = \mathbb{E}_R[X^T(R \odot R)X] = X^T\mathbb{E}_R[R \odot R]X = X^T(pI)X + X^T(\text{diag}(p(1-p)\mathbf{1}) \odot X^TX)$

But more directly, since R is diagonal and independent for each entry:

$$\mathbb{E}_R[(R \odot X)^T (R \odot X)] = p^2 X^T X + p(1-p)\tilde{\Gamma}^2$$

where $\tilde{\Gamma}$ is a diagonal matrix with $\tilde{\Gamma}_{jj} = \|X_j\|_2$ (the norm of the j-th column of X).

Step 4: Substitute Back

$$\mathcal{L}(\tilde{\mathbf{w}}) = \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T pX\tilde{\mathbf{w}} + \tilde{\mathbf{w}}^T \left(p^2 X^T X + p(1-p)\tilde{\Gamma}^2\right) \tilde{\mathbf{w}}$$

Step 5: Rewrite as a Norm

$$\mathcal{L}(\tilde{\mathbf{w}}) = \|\mathbf{y} - pX\tilde{\mathbf{w}}\|_2^2 + p(1-p)\|\tilde{\Gamma}\tilde{\mathbf{w}}\|_2^2$$

While this solution is mathematically correct, it does skip some essential steps. Namely, the computation of $\mathbb{E}_R[R \odot X]$ and $\mathbb{E}_R[(R \odot X)^T (R \odot X)]$ is somewhat unclear. Instead of showing the element-wise simplification, the model seems to skip over a few steps to arrive at the expression for the expectation, citing properties such as the independence of the entries of R. While this makes sense, it would be better for the model to go more in-depth into this simplification. Perhaps I could have prompted it better to show more detail.

(b) Transforming $\tilde{\mathbf{w}}$ for Traditional Regularization

Goal: Transform $\tilde{\mathbf{w}}$ to $\mathbf{w}$ so that the problem looks like:

$$\mathcal{L}(\mathbf{w}) = \|\mathbf{y} - X\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_2^2$$

Solution: Let $\mathbf{w} = p\tilde{\mathbf{w}}$. Then:

$$\mathcal{L}(\mathbf{w}) = \|\mathbf{y} - X\mathbf{w}\|_2^2 + p(1-p)\|\tilde{\Gamma}\frac{\mathbf{w}}{p}\|_2^2 = \|\mathbf{y} - X\mathbf{w}\|_2^2 + \frac{p(1-p)}{p^2}\|\tilde{\Gamma}\mathbf{w}\|_2^2$$

This resembles ridge regression with $\lambda = \frac{p(1-p)}{p^2}\tilde{\Gamma}^2$, but since $\tilde{\Gamma}$ is data-dependent, it's not exactly ridge regression unless $\tilde{\Gamma}$ is proportional to the identity matrix.

The model identifies the correct choice for w and the reasoning behind the choice is sound

(c) Change of Variables for Classical Ridge Regression

Given:

$$\mathcal{L}(\tilde{\mathbf{w}}) = \|\mathbf{y} - \tilde{X}\tilde{\mathbf{w}}\|_2^2 + \lambda\|\tilde{\mathbf{w}}\|_2^2$$

Goal: Express $\tilde{X}$ in terms of X and Γ.

Solution: Let $\tilde{X} = X\Gamma^{-1}$. Then:

$$\|\mathbf{y} - \tilde{X}\tilde{\mathbf{w}}\|_2^2 = \|\mathbf{y} - X\Gamma^{-1}\tilde{\mathbf{w}}\|_2^2$$

Let $\mathbf{w} = \Gamma^{-1}\tilde{\mathbf{w}}$. Then:

$$\mathcal{L}(\mathbf{w}) = \|\mathbf{y} - X\mathbf{w}\|_2^2 + \lambda\|\Gamma\mathbf{w}\|_2^2$$

This solution is also correct

(d) Norms of Columns of $\tilde{X}$ and Relationship to Batch Normalization

Given: Γ is diagonal and invertible, with $\Gamma_{jj} \propto \|X_j\|_2$.

Effective Training Matrix: $\tilde{X} = X\Gamma^{-1}$

Norms of Columns of $\tilde{X}$:

$$\|\tilde{X}_j\|_2 = \|X_j/\Gamma_{jj}\|_2 = \frac{\|X_j\|_2}{\Gamma_{jj}} \propto \frac{\|X_j\|_2}{\|X_j\|_2} = 1$$

Relationship to Batch Normalization: - Dropout (with this transformation) effectively normalizes the columns of the input matrix X to have unit norm, similar to batch normalization, which also normalizes the input features. - Both techniques aim to stabilize and speed up training by reducing internal covariate shift. Batch normalization explicitly normalizes activations, while dropout with this transformation implicitly normalizes the input features during training.

While the model does correctly identify that dropout normalizes the columns of X in an effect similar to batch norm, it incorrectly claims that the columns have unit norms. In reality, they have a constant norm of c. This error could be from the lack of the model's capability to refer back to part 2c to understand why the norm is c. It may only be thinking in the context of the problem at hand, and go straight to the thought that any normalization is unit normalization.

5 Executive Summary

In general, Mistral AI performed averagely on the written portion of HW5. It incorrectly answered 2/11 questions, and required one follow-up question pointing out the mistake in each incorrect answer to fix the issues. Otherwise, Mistral mostly correctly identified key patterns and was able to perform most computations and proofs. The errors could be attributed to weak prompting from me, but I believe a lack of capability to refer to the history of the conversation was the root of the error in the last question. The error in the first question is more puzzling, it seemed to be more of a hallucination where the model added terms that were not necessary in a summation. It also used a slightly different approach for a couple problems than what I would have done, namely the part in question 1 that it got wrong. Another potential issue was the lack of depth in the model's answer for the derivation of the dropout objective. It seemed to skip a few steps while still arriving at the right answer - this is an example of a situation where I should have been more clear about my expectations as the prompter. The speed of the model was excellent - all answers were generated within 10 seconds.