

Deep Neural Networks: Written Questions (HW5)

Special Participation A
Anders Vestrum (3041972833)

October 2025

Overview of Grok Performance

I evaluated Grok’s responses across all questions, treating each subpart (e.g., 1a, 1b) as a separate task. Each task is assigned to exactly one of the following categories:

1. **One-Shot Correct:** Answered correctly on the first attempt without any nudging.
2. **Minor Misconception / Hallucination:** Small errors or missteps that require minor nudging or clarification.
3. **Larger Error:** Significant errors in reasoning or calculations that require multiple clarifications.
4. **Did Not Solve:** Unable to produce the correct solution even after multiple prompts.

Performance Summary by Task

Task	Category
1a	Minor Misconception / Hallucination
1b	
1c	
2a	One-Shot Correct
2b	One-Shot Correct
3a	One-Shot Correct
3b	One-Shot Correct
3c	One-Shot Correct
4a	Minor Misconception / Hallucination
4b	
4c	
4d	

Overall Summary of Strengths and Weaknesses

Very impressed with the answers. A major strength is that Grok consistently includes conceptual intuition sections, providing a clear overview of the key ideas in each problem. For example, in 3b, it highlighted that depthwise separable convolutional networks implement convolutional nets with fewer parameters for faster training. In 4b, it made clear how dropout mimics Tikhonov regularization. These insights go beyond simply performing algebraic transformations or derivations, with it emphasizing the underlying principles and the “why” behind the mathematics. Another strength is Grok’s adaptability: when prompted about conventions (e.g., the factor of $1/2$ in the loss or the scaling of norms), it quickly corrected its approach and provided accurate, detailed derivations. It also integrates comparisons, such as relating generalized transpose convolutions to standard convolutions without explicit instructions, helping to reinforce understanding. On the weaker side, some outputs occasionally included more computation or explanation than necessary, covering basic concepts that may be obvious to an expert audience. A few steps could have been highlighted more explicitly (e.g., the derivation of diagonal vs. off-diagonal elements in the dropout regularization), which would make the reasoning easier to follow. Additionally, while conceptual intuition was strong, some outputs could have included formulas

or general expressions more explicitly for convenience, though this did not affect correctness. Overall, Grok's strengths lie in its consistent focus on intuition, clear identification of core ideas, and accurate derivations, while minor weaknesses are related to occasional over-explanation and not always highlighting certain steps explicitly. Its responses provide both correct solutions and conceptual clarity, making them highly valuable for learning and understanding the material.

Recommendation and Tips for Using Grok

Based on my experience, Grok is a powerful tool for learning and understanding complex topics, particularly in deep learning and mathematical derivations. Its strength lies in consistently providing correct solutions while emphasizing conceptual intuition, helping to clarify the underlying principles behind each problem. To get the most out of Grok, it is important to prompt it clearly and specifically, guiding it toward the desired level of detail or focus. Iteratively checking conventions, asking for clarifications, or probing certain steps, such as scaling factors or weight transformations, ensures that their responses align with expectations. Additionally, giving context or pointing out when a high-level abstraction is preferred helps produce concise, insightful explanations. Using it interactively in this way maximizes both correctness and conceptual clarity, making it an effective learning and problem solving assistant.

1 Introduction

ME: You are an assistant helping a student understand theoretical questions in Deep Neural Networks. The questions explore core topics in deep learning such as convolutional networks, batch normalization, dropout, depthwise separable convolutions, and regularization techniques, focusing on how these methods affect learning dynamics and model behavior. At this stage, do not solve or preview any question. You will later receive only non-coding, theory-based questions; when solving, provide clear, step-by-step reasoning and concise conceptual explanations at a graduate level, emphasizing intuition, derivation, and interpretation without any implementation or code. Begin solving only after the user sends a specific question.

GROK: Got it! I'm here to help you with theoretical questions on Deep Neural Networks. Please send me the specific question you'd like me to explain.

2 Problem 1: Convolutional Networks

Part (a)

ME: Question 1: Convolutional Networks. Throughout this problem, we use the convention of not flipping the filter before applying it over the signal (this matches the standard neural network convention where the filter is already flipped). (a) List two reasons why convolutional layers are typically preferred over fully connected layers when working with image data.

GROK: To address why convolutional layers are typically preferred over fully connected layers for image data, let's dive into the conceptual advantages of convolutional layers, focusing on their design and how they align with the properties of images. Below, I'll provide two key reasons with clear, step-by-step reasoning and intuitive explanations tailored for a graduate-level understanding.

Reason 1: Parameter Efficiency and Local Connectivity

Convolutional layers exploit the spatial structure of images by using local connectivity, which significantly reduces the number of parameters compared to fully connected layers. In a fully connected layer, every input pixel is connected to every neuron in the next layer, leading to a dense weight matrix. For an image of size $H \times W \times C$ (height, width, and channels), a fully connected layer with K output units requires $H \times W \times C \times K$ weights, which grows quadratically with the image size and becomes computationally infeasible for large images (e.g., a 224×224 RGB image has 150,528 pixels, leading to millions of parameters even for a modest K).

In contrast, a convolutional layer uses a small, shared filter (e.g., $3 \times 3 \times C$) that slides across the image, connecting each output unit to only a local region of the input (the receptive field). This filter is shared across all spatial locations, meaning the number of parameters depends only on the filter size and the number of filters, not the image size. For example, a $3 \times 3 \times C$ filter with F filters requires $3 \times 3 \times C \times F + F$ parameters (including biases), which is orders of magnitude smaller. This parameter efficiency reduces memory requirements, speeds up computation, and mitigates overfitting by limiting model complexity.

Intuition: Images have local spatial patterns (e.g., edges, textures) that are consistent across different regions. Convolutional layers capture these patterns efficiently by learning shared filters, whereas fully connected layers treat each pixel independently, ignoring spatial relationships and requiring excessive parameters to model them.

Reason 2: Translation Invariance

Convolutional layers inherently incorporate translation invariance, making them well-suited for image data where patterns can appear anywhere in the image. Because the same filter is applied across all spatial locations, a convolutional layer detects features (e.g., edges, corners) regardless of their position in the image. This property allows the network to recognize objects in different parts of the image without needing to learn separate weights for each location.

In contrast, fully connected layers lack this property, as each input pixel is tied to specific weights, meaning the same feature in a different location would require relearning entirely new weights. This translation invariance is further enhanced in convolutional neural networks (CNNs) through pooling layers (e.g., max pooling), which aggregate features over local regions, making the network robust to small translations or distortions. Fully connected layers, by treating the input as a flattened vector, lose spatial information and cannot naturally generalize across translations without significantly increasing the number of parameters.

Intuition: In image recognition, an object like a cat looks the same whether it's in the top-left or bottom-right corner. Convolutional layers naturally handle this by sharing weights across locations, while fully connected layers would need to learn the cat's appearance separately for every possible position, which is inefficient and impractical.

Summary

- 1. Parameter Efficiency and Local Connectivity:** Convolutional layers use small, shared filters to exploit local spatial correlations in images, drastically reducing the number of parameters compared to fully connected layers, which connect every input to every output.
- 2. Translation Invariance:** Convolutional layers apply the same filters across the image, enabling them to detect features regardless of their spatial location, unlike fully connected layers, which lack this property and require relearning features for different positions.

These properties make convolutional layers computationally efficient and better aligned with the spatial and hierarchical nature of image data, enabling effective feature extraction and robust generalization.

ME: Good explanation. Could you express the same reasoning more concisely, emphasizing the conceptual intuition rather than numerical examples?

GROK: Reason 1: Parameter Efficiency via Local Connectivity

Convolutional layers are preferred for images because they use small, shared filters that focus on local pixel patterns, drastically reducing parameters compared to fully connected layers, which connect every input to every output.

Intuition: Images have local structures like edges that repeat across regions. Convolutions efficiently capture these by sharing weights, while fully connected layers redundantly model each pixel independently, inflating complexity.

Reason 2: Translation Invariance

Convolutional layers apply the same filter across an image, detecting features like edges or textures regardless of their position, unlike fully connected layers, which treat pixels as unrelated and lose spatial context.

Intuition: Objects in images (e.g., a dog) look the same anywhere in the frame. Convolutions naturally generalize across locations, while fully connected layers require relearning features for each position, making them less effective.

Reflection

The response could have been prompted more efficiently from the start, avoiding unnecessary explanations of fairly fundamental concepts. However, the output was accurate and would provide valuable insights for someone less familiar with CNN architectures. Overall, the summary effectively captured the key ideas at a high level of abstraction, which was appropriate for this question.

Part (b)

ME: Thanks, that was helpful! Now for the part 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?

(Hint: Just to help you check your work, the first entry in the filter that you should find is 2. However, if you try to use this hint directly to solve for the answer, you will not get credit since this hint only exists to help you check your work.)

GROK: To find the length-3 filter used in the 1D convolution of the given signal $[1, 4, 0, -2, 3]$ that produces the output $[-2, 2, 11]$ with no padding and stride=1, we need to set up and solve a system of equations based on the convolution operation. Below, I'll provide a clear, step-by-step derivation with concise conceptual explanations, emphasizing the intuition behind the process, as requested for a graduate-level understanding.

Step 1: Understand the Convolution Operation

In 1D convolution with no padding and stride=1, a filter of length k slides over the input signal, computing the dot product of the filter with each valid window of k consecutive input elements. The input signal is $[x_1, x_2, x_3, x_4, x_5] = [1, 4, 0, -2, 3]$, and the filter is of length 3, denoted as $[f_1, f_2, f_3]$. The output sequence is $[y_1, y_2, y_3] = [-2, 2, 11]$. Since there's no padding, only "valid" convolutions are computed, where the filter fully overlaps with the input.

For an input of length $n = 5$ and filter of length $k = 3$, the output length is given by:

$$n - k + 1 = 5 - 3 + 1 = 3$$

This matches the output length of 3, confirming our setup.

The convolution operation at position i of the output is:

$$y_i = \sum_{j=1}^k x_{i+j-1} \cdot f_j$$

Since the problem specifies that the filter is not flipped (following the neural network convention), we directly use the filter coefficients $[f_1, f_2, f_3]$ in the dot product.

Step 2: Set Up the Convolution Equations

For each output element, the filter slides over a window of 3 input elements:

- First output ($y_1 = -2$): The filter $[f_1, f_2, f_3]$ is applied to the input window $[x_1, x_2, x_3] = [1, 4, 0]$:

$$y_1 = x_1 f_1 + x_2 f_2 + x_3 f_3 = 1 \cdot f_1 + 4 \cdot f_2 + 0 \cdot f_3 = f_1 + 4f_2 = -2$$

- Second output ($y_2 = 2$): The filter slides to $[x_2, x_3, x_4] = [4, 0, -2]$:

$$y_2 = x_2 f_1 + x_3 f_2 + x_4 f_3 = 4 \cdot f_1 + 0 \cdot f_2 + (-2) \cdot f_3 = 4f_1 - 2f_3 = 2$$

- Third output ($y_3 = 11$): The filter slides to $[x_3, x_4, x_5] = [0, -2, 3]$:

$$y_3 = x_3 f_1 + x_4 f_2 + x_5 f_3 = 0 \cdot f_1 + (-2) \cdot f_2 + 3 \cdot f_3 = -2f_2 + 3f_3 = 11$$

This gives us the system of linear equations:

1. $f_1 + 4f_2 = -2$
2. $4f_1 - 2f_3 = 2$
3. $-2f_2 + 3f_3 = 11$

Step 3: Solve the System of Equations

- From Equation 3:

$$3f_3 = 11 + 2f_2 \implies f_3 = \frac{11 + 2f_2}{3}$$

- From Equation 2:

$$4f_1 - 2f_3 = 2 \implies 4f_1 = 2 + 2f_3 \implies f_1 = \frac{1 + f_3}{2}$$

Substitute $f_3 = \frac{11 + 2f_2}{3}$:

$$f_1 = \frac{7 + f_2}{3}$$

- From Equation 1:

$$f_1 + 4f_2 = -2 \implies \frac{7 + f_2}{3} + 4f_2 = -2$$

Multiply through by 3:

$$7 + f_2 + 12f_2 = -6 \implies 13f_2 = -13 \implies f_2 = -1$$

- Solve for f_1 :

$$f_1 = \frac{7 + (-1)}{3} = 2$$

- Solve for f_3 :

$$f_3 = \frac{11 + 2(-1)}{3} = 3$$

Thus, the filter is $[f_1, f_2, f_3] = [2, -1, 3]$.

Step 4: Verify the Solution

- First output: $[1, 4, 0] \cdot [2, -1, 3] = -2$
- Second output: $[4, 0, -2] \cdot [2, -1, 3] = 2$
- Third output: $[0, -2, 3] \cdot [2, -1, 3] = 11$

The output matches $[-2, 2, 11]$, confirming the filter is correct. Additionally, the first entry $f_1 = 2$ aligns with the hint provided.

Intuition

The convolution process slides a filter over the input, computing weighted sums to produce the output. By setting up equations based on these sums, we reverse-engineer the filter that transforms the input into the given output. The no-flip convention simplifies the dot product, and the system of equations ensures a unique solution since we have three equations for three unknowns.

Final Answer

The filter is $[2, -1, 3]$.

Reflection

A very good response overall, though some of the variable computations were more detailed than necessary.

Part (c)

ME: Thanks for the answer. Your solution is correct, but computing each variable step by step was excessive; it could be more concise by directly solving the linear system and briefly verifying the result. Now we move on to part c: 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 $\text{pad}=0$ and $\text{stride}=2$, 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. (For example, if we did standard convolution on a length-7 sequence with filter size of 3 and $\text{stride}=2$, we would output a length-3 sequence.)

If our 2D input is

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

and the 2D filter is

$$\begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix},$$

what is the output of transpose convolution with $\text{pad}=0$ and $\text{stride}=1$?

GROK: The transpose convolution with $\text{stride}=1$ and $\text{pad}=0$ upsamples the 2×2 input to a 3×3 output, as per the formula

$$O = S(I - 1) + K = 1(2 - 1) + 2 = 3.$$

Each input element scales the entire filter, and these scaled filters are placed starting at output positions $(\text{stride} \times \text{input_row}, \text{stride} \times \text{input_col})$, with overlaps summed. Following the no-flip convention, the computation yields:

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

ME: Thank you for the clear and concise explanation. Could you provide an example illustrating how the output dimensions are computed using non-square (rectangular) matrices as input?

GROK: To illustrate how output dimensions are computed for a transpose convolution (also called deconvolution) with a non-square input, consider the following:

Setup: Input size $I_h \times I_w = 3 \times 5$, filter size $K_h \times K_w = 3 \times 3$, stride $S = 2$, padding $P = 0$.

Output dimension formula:

$$O_h = S(I_h - 1) + K_h - 2P, \quad O_w = S(I_w - 1) + K_w - 2P$$

Compute output dimensions:

$$O_h = 2(3 - 1) + 3 - 0 = 7, \quad O_w = 2(5 - 1) + 3 - 0 = 11$$

Thus, the output is 7×11 .

Intuition: Each input element “plants” a 3×3 filter in the output. Stride S determines spacing between these contributions. The first row’s 5 elements produce filters at output columns 0,2,4,6,8, each adding 3 columns, reaching column 10. Similarly, the 3 input rows produce filters at output rows 0,2,4, each adding 3 rows, reaching row 6. Hence, 7×11 .

Comparison to standard convolution: A standard convolution with the same filter, stride, and no padding on a 7×11 input gives

$$O_h = \lfloor \frac{7-3}{2} \rfloor + 1 = 3, \quad O_w = \lfloor \frac{11-3}{2} \rfloor + 1 = 5$$

recovering the original input size. This shows how transpose convolution reverses dimensionality reduction.

Summary: Transpose convolution upsamples by spacing out input contributions and adding the filter span, effectively expanding the input dimensions, unlike standard convolution which reduces them. This property is useful for tasks like image upsampling in generative models.

Reflection

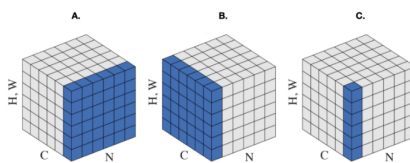
Effectively shows the solution and demonstrates clear adaptation to previous feedback. It compared the transpose convolution to standard convolution without explicit prompting, which strengthened the explanation by highlighting the upsampling property. After I asked an additional question to encourage generalization, it successfully expanded the solution while maintaining focus and clarity.

3 Problem 2: Batch Normalization for CNN

Part (a)

ME: Consider the following diagram where the shaded blocks represent entries participating in one normalization step for a CNN-type architecture. N represents the mini-batch, H, W represent the spatial dimensions (pixels), and C represents channels.

- Which one denotes the process of batch normalization? Please use ☐ for your selection.
A B C
- Which one denotes layer normalization? Please use ☐ for your selection.
A B C



GROK: To determine which diagram represents batch normalization and which represents layer normalization, we need to understand the key differences in how these normalization techniques operate within a CNN architecture. The diagrams show 3D grids with axes labeled N (mini-batch), H (height), W (width), and C (channels), where the shaded blue regions indicate the entries participating in a single normalization step.

Conceptual Intuition

- **Batch Normalization (BN)** normalizes across the mini-batch dimension (N) for each feature map, treating all pixels ($H \times W$) and channels (C) within a single mini-batch sample together. The statistics (mean and variance) are computed over the N samples for a given (H, W, C) position.
- **Layer Normalization (LN)** normalizes across the feature dimension (C) for each spatial location (H, W), treating all channels at a single pixel position, regardless of the mini-batch.

Analysis of Diagrams

- **Diagram A:** Shaded region spans the entire $H \times W$ plane for one channel (C) and all N samples — aligns with batch normalization.
- **Diagram B:** Shaded region spans all C channels for one spatial location (H, W) and all N samples — aligns with layer normalization.
- **Diagram C:** Shaded region spans N for one channel (C) and a thin slice along H or W — does not match standard BN or LN.

Final Answer

- Which one denotes the process of batch normalization? ☒ A ☐ B ☐ C
- Which one denotes layer normalization? ☐ A ☒ B ☐ C

Reflection

A strong answer with a clear and structured explanation. The inclusion of conceptual intuition before diving into the detailed analysis was great. It arrived at the correct solution, and after providing the figures, the model interpreted them accurately and integrated them well into the reasoning.

Part (b)

ME: That was a well-structured answer. Now move on to part B: Consider a simplified batch normalization 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: $[x_1, x_2, \dots, x_n]$. Specifically, we let $\hat{x}_i = x_i - \mu$, where $\mu = \frac{1}{n} \sum_{j=1}^n x_j$, and output $[y_1, y_2, \dots, y_n]$ where $y_i = \gamma \hat{x}_i + \beta$ to the next layer. Assume we have a final loss L somewhere downstream.

Calculate $\frac{\partial L}{\partial x_i}$ in terms of $\frac{\partial L}{\partial y_j}$ for $j = 1, \dots, n$ as well as γ and β as needed. Numerically, what is $\frac{\partial L}{\partial x_1}$ 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 $n \rightarrow \infty$? (Feel free to assume here that all relevant quantities are bounded.)

GROK: We need to compute the partial derivative $\frac{\partial L}{\partial x_i}$ for a simplified batch normalization (BN) process where the data is de-meant but not divided by the standard deviation. The output is then scaled and shifted with γ and β . Let's proceed step-by-step with clear reasoning and intuition.

Step-by-Step Reasoning

Problem Setup

- Input batch: $[x_1, x_2, \dots, x_n]$, batch size n .
- Mean: $\mu = \frac{1}{n} \sum_{j=1}^n x_j$.
- De-meant value: $\hat{x}_i = x_i - \mu$.
- Output: $y_i = \gamma \hat{x}_i + \beta$, with learnable γ and β .
- Final loss: L , function of $[y_1, y_2, \dots, y_n]$.
- Goal: compute $\frac{\partial L}{\partial x_i}$ in terms of $\frac{\partial L}{\partial y_j}$ for $j = 1, \dots, n$, γ , and β .

Derivative Computation By the chain rule:

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

Relating y_j to x_i :

$$y_j = \gamma \hat{x}_j + \beta, \quad \hat{x}_j = x_j - \mu.$$

Since μ depends on all x_k , $\frac{\partial \mu}{\partial x_i} = \frac{1}{n}$. Thus,

$$\frac{\partial \hat{x}_j}{\partial x_i} = \frac{\partial x_j}{\partial x_i} - \frac{\partial \mu}{\partial x_i} = \delta_{ji} - \frac{1}{n}.$$

Hence,

$$\frac{\partial y_j}{\partial x_i} = \gamma \frac{\partial \hat{x}_j}{\partial x_i} = \gamma \left(\delta_{ji} - \frac{1}{n} \right).$$

Applying the chain rule:

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

Conceptual Intuition The derivative accounts for the direct effect of x_i on y_i and the indirect effect via the batch mean μ , scaled by γ .

Numerical Case: $n = 1$

- Input batch: $[x_1]$, so $\mu = x_1$.
- De-meaned: $\hat{x}_1 = 0$.
- Output: $y_1 = \beta$.
- Using the formula:

$$\frac{\partial L}{\partial x_1} = \gamma \left(\frac{\partial L}{\partial y_1} - \frac{1}{1} \frac{\partial L}{\partial y_1} \right) = 0.$$

3.0.1 Limit as $n \rightarrow \infty$

$$\frac{\partial L}{\partial x_i} \approx \gamma \left(\frac{\partial L}{\partial y_i} - \mathbb{E} \left[\frac{\partial L}{\partial y_j} \right] \right),$$

where \mathbb{E} denotes the expectation over the batch. The derivative stabilizes and depends on the deviation of the local gradient from the batch-averaged gradient.

Final Answer

$$\frac{\partial L}{\partial x_i} = \gamma \left(\frac{\partial L}{\partial y_i} - \frac{1}{n} \sum_{j=1}^n \frac{\partial L}{\partial y_j} \right), \quad \frac{\partial L}{\partial x_1} = 0 \text{ for } n = 1, \quad \frac{\partial L}{\partial x_i} \approx \gamma \left(\frac{\partial L}{\partial y_i} - \mathbb{E} \left[\frac{\partial L}{\partial y_j} \right] \right) \text{ as } n \rightarrow \infty.$$

Reflection

The response is correct, thorough, and well-structured

4 Problem 3: Depthwise Separable Convolutions

Part (a)

ME: Okay, now for Question 3 about 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.

(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?

GROK: To determine the number of learnable parameters in a traditional convolution layer, we need to analyze the structure of the operation and the given specifications. Since biases are ignored as per the problem statement, we focus solely on the weights.

Step-by-Step Reasoning

1. Understand Traditional Convolution:

- In a traditional convolution layer, a single filter (or kernel) operates on the input by sliding over both the spatial dimensions (height and width) and the channel dimension simultaneously.
- The filter has a spatial size (kernel size) and a depth equal to the number of input channels. It produces a single output channel per filter.
- The total number of filters determines the number of output channels.

2. Extract Problem Details:

- Input: 3-channel image (e.g., RGB), with resolution 224×224 . The number of channels (3) is the depth of the input.
- Kernel size: 3×3 , meaning each filter has a 3×3 spatial footprint.
- Output channels: 4, meaning there are 4 filters, each producing one output channel.

3. Calculate Parameters per Filter:

- Each filter has a depth of 3 (to match the 3 input channels).
- The spatial size of the filter is 3×3 , so the number of weights per filter is $3 \times 3 \times 3 = 27$.
- This 27-element filter is applied to the input to produce one output channel.

4. Account for Multiple Filters:

- Since there are 4 output channels, we need 4 such filters.
- The total number of parameters is the number of weights per filter multiplied by the number of filters: $27 \times 4 = 108$.

5. Verify Intuition:

- In a traditional convolution, all input channels are convolved together by a single filter to produce an output channel. The parameter count reflects the interaction of the kernel size with the input depth and the number of desired outputs.
- The 224×224 resolution affects the output size but not the number of parameters, as parameters depend only on the filter design (kernel size, input channels, output channels).

Final Answer

The number of learnable parameters in the traditional convolution layer is 108.

Reflection

Correct solution. It could have included the general formula explicitly for convenience, but the step-by-step derivation was accurate and error-free.

Part (b)

ME: That is correct. Now for part (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?

GROK: To calculate the total number of learnable parameters in a depthwise separable convolution layer, we need to determine the parameters for both the depthwise convolution and the pointwise convolution separately, then sum them. Since biases are ignored as per the problem statement, we focus only on the weights.

Step-by-Step Reasoning

1. Understand Depthwise Separable Convolution

- **Depthwise Convolution:** Applies a separate 3×3 filter to each input channel, producing the same number of output channels as input channels. There is no “cross talk” between channels, meaning each channel is convolved independently.
- **Pointwise Convolution:** A 1×1 convolution applied to the output of the depthwise convolution, combining the channels to produce the desired number of output channels (4 in this case). It acts as a linear combination across channels.

2. Extract Problem Details

- Input: 3-channel 224×224 image.
- Depthwise convolution: Uses 3×3 filters, with the number of output channels equal to the number of input channels (3).
- Pointwise convolution: Uses 1×1 filters, with 4 output channels.
- The spatial resolution (224×224) affects the output size but not the parameter count.

3. Calculate Parameters for Depthwise Convolution

- Each of the 3 input channels is convolved with its own 3×3 filter.
- Number of parameters per filter: $3 \times 3 = 9$.
- Total parameters for 3 filters: $9 \times 3 = 27$.

4. Calculate Parameters for Pointwise Convolution

- Pointwise convolution uses 1×1 filters to combine the 3 channels from the depthwise output into 4 output channels.
- Each 1×1 filter has depth equal to the number of input channels (3), so parameters per filter: $1 \times 1 \times 3 = 3$.
- With 4 output channels, total parameters: $3 \times 4 = 12$.

5. Total Parameters

$$\text{Total} = 27 + 12 = 39$$

6. Verify Intuition

Depthwise separable convolution reduces parameters compared to a traditional convolution (108 from part a) by separating spatial and channel-wise operations. The depthwise step handles spatial filtering per channel, while the pointwise step handles channel mixing, resulting in a lower-rank approximation that is computationally efficient.

Final Answer

The total number of learnable parameters of the depthwise separable convolution layer is 39.

Reflection

Correct answer, and it verified the intuition behind depthwise separable convolutional networks by comparing the number of learnable parameters with the standard convolutional network from part (a). This reinforces the understanding by comparing the results and the motivation behind this approach.

5 Problem 4: Regularization and Dropout

5.1 Part (a)

ME: Correct! Now on to question 4 about regularization and dropout, part (a): Recall that linear regression optimizes the following learning objective:

$$L(w) = \|y - Xw\|_2^2$$

One way of using dropout during SGD on the d -dimensional input features x_i involves keeping each feature at random \sim i.i.d Bernoulli(p) (and zeroing it out if not kept) and then performing a traditional SGD step. It turns out that such dropout makes our learning objective effectively become

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

where \odot is the element-wise product and the random binary matrix $R \in \{0, 1\}^{n \times d}$ is such that $R_{i,j} \sim$ i.i.d Bernoulli(p). We use \check{w} to remind you that this is learned by dropout.

Recalling how Tikhonov-regularized (generalized ridge-regression) least-squares problems involve solving:

$$L(w) = \|y - Xw\|_2^2 + \|\Gamma w\|_2^2$$

for some suitable matrix Γ , it turns out we can manipulate (2) to eliminate the expectations and get:

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

with $\check{\Gamma}$ being a diagonal matrix whose j -th diagonal entry is the norm of the j -th column of the training matrix X .

(a) Show that we can manipulate (2) to eliminate the expectations and get:

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

with $\check{\Gamma}$ being a diagonal matrix whose j -th diagonal entry is the norm of the j -th column of the training matrix X .

GROK: To show that the expected loss from dropout regularization (Equation 2) can be manipulated to eliminate the expectation and yield the form in Equation 5, we need to derive the expression step-by-step. The goal is to transform the expected loss

$$L(\hat{\mathbf{w}}) = \mathbb{E}_{R \sim \text{Bernoulli}(p)} \left[\frac{1}{2} \|\mathbf{y} - (R \odot X) \hat{\mathbf{w}}\|_2^2 \right]$$

into

$$\frac{1}{2} \|\mathbf{y} - pX\hat{\mathbf{w}}\|_2^2 + \frac{1}{2} p(1-p) \|\hat{\Gamma}\hat{\mathbf{w}}\|_2^2,$$

where $\hat{\Gamma}$ is a diagonal matrix with its j -th diagonal entry being the norm of the j -th column of the training matrix X . Let's proceed with clear reasoning and intuition.

Step-by-Step Reasoning

1. Define the Problem:

- Input data: X is an $n \times d$ matrix, where n is the number of samples and d is the number of features.
- Target: \mathbf{y} is an $n \times 1$ vector.
- Random matrix: R is an $n \times d$ binary matrix with $R_{i,j} \sim \text{Bernoulli}(p)$, meaning each element is 1 with probability p and 0 with probability $1 - p$, independently.
- Dropout operation: $R \odot X$ is the element-wise product, where each feature $X_{i,j}$ is kept with probability p and zeroed otherwise.
- Loss with dropout:

$$L(\hat{\mathbf{w}}) = \mathbb{E}_{R \sim \text{Bernoulli}(p)} \left[\frac{1}{2} \|\mathbf{y} - (R \odot X) \hat{\mathbf{w}}\|_2^2 \right],$$

where $\hat{\mathbf{w}}$ is the weight vector learned with dropout.

- Target form:

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

with $\hat{\Gamma}$ diagonal and its j -th entry being $\|X_{:,j}\|_2$, the L_2 norm of the j -th column of X .

2. Expand the Expected Loss:

$$\begin{aligned} \frac{1}{2} \|\mathbf{y} - (R \odot X)\hat{\mathbf{w}}\|_2^2 &= \frac{1}{2} (\mathbf{y} - (R \odot X)\hat{\mathbf{w}})^T (\mathbf{y} - (R \odot X)\hat{\mathbf{w}}), \\ &= \frac{1}{2} \mathbf{y}^T \mathbf{y} - \mathbf{y}^T (R \odot X)\hat{\mathbf{w}} + \frac{1}{2} \hat{\mathbf{w}}^T (R \odot X)^T (R \odot X)\hat{\mathbf{w}}. \end{aligned}$$

3. Since $\mathbf{y}^T \mathbf{y}$ is constant with respect to R , the expectation is

$$L(\hat{\mathbf{w}}) = \frac{1}{2} \mathbf{y}^T \mathbf{y} - \mathbb{E}_R[\mathbf{y}^T (R \odot X)\hat{\mathbf{w}}] + \frac{1}{2} \mathbb{E}_R[\hat{\mathbf{w}}^T (R \odot X)^T (R \odot X)\hat{\mathbf{w}}].$$

4. Compute the Linear Term:

$$\mathbb{E}_R[\mathbf{y}^T (R \odot X)\hat{\mathbf{w}}] = \mathbf{y}^T (pX)\hat{\mathbf{w}} = p\mathbf{y}^T X\hat{\mathbf{w}}.$$

Hence,

$$-\mathbb{E}_R[\mathbf{y}^T (R \odot X)\hat{\mathbf{w}}] = -p\mathbf{y}^T X\hat{\mathbf{w}}.$$

5. Compute the Quadratic Term:

$$\frac{1}{2} \mathbb{E}_R[\hat{\mathbf{w}}^T (R \odot X)^T (R \odot X)\hat{\mathbf{w}}] = \frac{1}{2} \hat{\mathbf{w}}^T [p^2 X^T X + p(1-p)\text{diag}(\|X_{:,j}\|_2^2)] \hat{\mathbf{w}}.$$

6. Combine Terms:

$$\begin{aligned} L(\hat{\mathbf{w}}) &= \frac{1}{2} \mathbf{y}^T \mathbf{y} - p\mathbf{y}^T X\hat{\mathbf{w}} + \frac{1}{2} p^2 \hat{\mathbf{w}}^T X^T X\hat{\mathbf{w}} + \frac{1}{2} p(1-p) \hat{\mathbf{w}}^T \text{diag}(\|X_{:,j}\|_2^2) \hat{\mathbf{w}}. \\ \frac{1}{2} \mathbf{y}^T \mathbf{y} - p\mathbf{y}^T X\hat{\mathbf{w}} + \frac{1}{2} p^2 \hat{\mathbf{w}}^T X^T X\hat{\mathbf{w}} &= \frac{1}{2} \|\mathbf{y} - pX\hat{\mathbf{w}}\|_2^2, \\ \frac{1}{2} p(1-p) \hat{\mathbf{w}}^T \text{diag}(\|X_{:,j}\|_2^2) \hat{\mathbf{w}} &= \frac{1}{2} p(1-p) \|\hat{\Gamma}\hat{\mathbf{w}}\|_2^2. \end{aligned}$$

7. Final Form:

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

Conceptual Intuition

Dropout introduces randomness by zeroing features, which, in expectation, scales the input by p and adds a regularization effect. The first term $\frac{1}{2} \|\mathbf{y} - pX\hat{\mathbf{w}}\|_2^2$ accounts for the scaled input effect, while the second term $\frac{1}{2} p(1-p) \|\hat{\Gamma}\hat{\mathbf{w}}\|_2^2$ reflects the variance-induced penalty, proportional to the column norms of X , mimicking Tikhonov regularization.

Final Answer

We have shown that

$$L(\hat{\mathbf{w}}) = \mathbb{E}_{R \sim \text{Bernoulli}(p)} \left[\frac{1}{2} \|\mathbf{y} - (R \odot X)\hat{\mathbf{w}}\|_2^2 \right]$$

can be manipulated to eliminate the expectations and yield

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

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

ME: Could you use the convention used in the problem description?

GROK: To adjust the derivation to match the convention of the problem description by removing the factor of $\frac{1}{2}$ from both the expected loss and the target form, we re-derive the expression step-by-step. The goal is to transform the expected loss

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

into

$$\|\mathbf{y} - pX\hat{\mathbf{w}}\|_2^2 + p(1-p)\|\hat{\Gamma}\hat{\mathbf{w}}\|_2^2,$$

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

Step-by-Step Reasoning

1. Define the Problem:

- Input data: X is an $n \times d$ matrix.
- Target: \mathbf{y} is an $n \times 1$ vector.
- Random matrix: R is an $n \times d$ binary matrix with $R_{i,j} \sim \text{Bernoulli}(p)$.
- Dropout operation: $R \odot X$ is the element-wise product.
- Loss with dropout:

$$L(\hat{\mathbf{w}}) = \mathbb{E}_R [\|\mathbf{y} - (R \odot X)\hat{\mathbf{w}}\|_2^2].$$

- Target form:

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

2. Expand the Expected Loss:

$$\|\mathbf{y} - (R \odot X)\hat{\mathbf{w}}\|_2^2 = \mathbf{y}^T \mathbf{y} - 2\mathbf{y}^T (R \odot X)\hat{\mathbf{w}} + \hat{\mathbf{w}}^T (R \odot X)^T (R \odot X)\hat{\mathbf{w}}.$$

Taking expectation:

$$L(\hat{\mathbf{w}}) = \mathbf{y}^T \mathbf{y} - 2\mathbb{E}_R[\mathbf{y}^T (R \odot X)\hat{\mathbf{w}}] + \mathbb{E}_R[\hat{\mathbf{w}}^T (R \odot X)^T (R \odot X)\hat{\mathbf{w}}].$$

3. Compute the Linear Term:

$$\mathbb{E}_R[\mathbf{y}^T (R \odot X)\hat{\mathbf{w}}] = \mathbf{y}^T (pX)\hat{\mathbf{w}} = p\mathbf{y}^T X\hat{\mathbf{w}},$$

$$-2\mathbb{E}_R[\mathbf{y}^T (R \odot X)\hat{\mathbf{w}}] = -2p\mathbf{y}^T X\hat{\mathbf{w}}.$$

4. Compute the Quadratic Term:

$$\mathbb{E}_R[\hat{\mathbf{w}}^T (R \odot X)^T (R \odot X)\hat{\mathbf{w}}] = p^2 \hat{\mathbf{w}}^T X^T X \hat{\mathbf{w}} + p(1-p) \hat{\mathbf{w}}^T \text{diag}(\|X_{:,j}\|_2^2) \hat{\mathbf{w}}.$$

5. Combine Terms:

$$L(\hat{\mathbf{w}}) = \mathbf{y}^T \mathbf{y} - 2p\mathbf{y}^T X\hat{\mathbf{w}} + p^2 \hat{\mathbf{w}}^T X^T X \hat{\mathbf{w}} + p(1-p) \hat{\mathbf{w}}^T \text{diag}(\|X_{:,j}\|_2^2) \hat{\mathbf{w}}.$$

The first three terms form:

$$\|\mathbf{y} - pX\hat{\mathbf{w}}\|_2^2,$$

and the last term is:

$$p(1-p)\|\hat{\Gamma}\hat{\mathbf{w}}\|_2^2.$$

6. Final Form:

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

Conceptual Intuition

Dropout randomly zeros features, scaling the input by p in expectation and introducing a regularization effect. The term $\|\mathbf{y} - pX\hat{\mathbf{w}}\|_2^2$ reflects the loss with scaled inputs, while $p(1-p)\|\hat{\Gamma}\hat{\mathbf{w}}\|_2^2$ acts as a variance-based regularization penalty proportional to the column norms of X .

Final Answer

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

ME: That is better. Could you also highlight where the $(1-p)$ factor comes from in the expression?

GROK: We manipulate the dropout-regularized objective

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

to obtain

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

where $\hat{\Gamma}_{j,j} = \|X_{:,j}\|_2$.

Step 1: Expand the Expectation

$$\|y - (R \odot X)\hat{w}\|_2^2 = \sum_{i=1}^n \left(y_i - \sum_{j=1}^d R_{i,j} X_{i,j} \hat{w}_j \right)^2.$$

Let $z_i = \sum_{j=1}^d R_{i,j} X_{i,j} \hat{w}_j$:

$$(y_i - z_i)^2 = y_i^2 - 2y_i z_i + z_i^2, \quad \mathbb{E}_R[(y_i - z_i)^2] = y_i^2 - 2\mathbb{E}_R[y_i z_i] + \mathbb{E}_R[z_i^2].$$

Step 2: Compute Expectations

$$\mathbb{E}_R[z_i] = p(X\hat{w})_i, \quad \mathbb{E}_R[y_i z_i] = y_i p(X\hat{w})_i.$$

$$z_i^2 = \sum_{j=1}^d \sum_{k=1}^d R_{i,j} R_{i,k} X_{i,j} X_{i,k} \hat{w}_j \hat{w}_k$$

$$\mathbb{E}_R[z_i^2] = \sum_{j=1}^d p X_{i,j}^2 \hat{w}_j^2 + \sum_{j=1}^d \sum_{k \neq j}^d p^2 X_{i,j} X_{i,k} \hat{w}_j \hat{w}_k = \sum_{j=1}^d (p-p^2) X_{i,j}^2 \hat{w}_j^2 + p^2 (X\hat{w})_i^2 = p(1-p) \sum_{j=1}^d X_{i,j}^2 \hat{w}_j^2 + p^2 (X\hat{w})_i^2.$$

Origin of $(1-p)$: The $(1-p)$ factor arises from $p-p^2 = p(1-p)$, which is the variance of $R_{i,j}$, i.e., $\text{Var}(R_{i,j}) = p(1-p)$. This reflects the stochastic variability introduced by dropout in the squared predictions.

$$\mathbb{E}_R[(y_i - z_i)^2] = y_i^2 - 2py_i(X\hat{w})_i + p^2(X\hat{w})_i^2 + p(1-p) \sum_{j=1}^d X_{i,j}^2 \hat{w}_j^2.$$

Step 3: Sum Over i

$$L(\hat{w}) = \|y - pX\hat{w}\|_2^2 + p(1-p) \sum_{j=1}^d \hat{w}_j^2 \|X_{:,j}\|_2^2.$$

Step 4: Relate to $\hat{\Gamma}$

$$\|\hat{\Gamma}\hat{w}\|_2^2 = \sum_{j=1}^d \|X_{:,j}\|_2^2 \hat{w}_j^2, \quad L(\hat{w}) = \|y - pX\hat{w}\|_2^2 + p(1-p)\|\hat{\Gamma}\hat{w}\|_2^2.$$

Thus, the $(1-p)$ factor, combined with p , forms the coefficient $p(1-p)$ scaling the regularization term due to dropout's stochasticity.

Reflection

The answer initially used an incorrect $\frac{1}{2}$ factor in the loss, but corrected it immediately after feedback. The model likely included the $\frac{1}{2}$ factor by default because it has learned the common MSE convention from training, where this factor simplifies derivatives, even though the problem did not specify it. I asked again specifically to check whether it would stick to the convention I wanted, and it successfully retraced the independent computations of the diagonal and off-diagonal elements of $X^T X$, deriving the $(1 - p)$ part of the scaling on the diagonal.

Part (b)

ME: Okay, thanks a lot! Now for part b: How should we transform the $\hat{\mathbf{w}}$ we learn using Equation (5) (i.e., with dropout) to get something that looks like a solution to the traditionally regularized problem (3)?

(Hint: This is related to how we adjust weights learned using dropout training for using them at inference time. PyTorch by default does this adjustment during training itself, but here, we are doing dropout slightly differently with no adjustments during training.)

GROK: To determine how to transform the weight vector $\hat{\mathbf{w}}$ learned using the dropout-modified loss (Equation 5) to resemble a solution to the traditionally regularized problem (Equation 3), we need to bridge the two formulations. The hint suggests a connection to inference-time adjustments for dropout, where weights are typically scaled to account for the dropout probability p . Let's derive the transformation step-by-step with clear reasoning and intuition.

Step-by-Step Reasoning

1. Recap the Losses:

- **Dropout-Modified Loss (Equation 5):**

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

where $\hat{\mathbf{w}}$ is learned with dropout, p is the keep probability, and $\hat{\Gamma}$ is a diagonal matrix with $\hat{\Gamma}_{j,j} = \|X_{:,j}\|_2$.

- **Tikhonov-Regularized Loss (Equation 3):**

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

where \mathbf{w} is the weight vector, and Γ is a suitable regularization matrix.

2. Interpret the Dropout Effect:

- The term $\|\mathbf{y} - pX\hat{\mathbf{w}}\|_2^2$ suggests that the effective input during training is scaled by p due to dropout.
- The regularization term $p(1 - p)\|\hat{\Gamma}\hat{\mathbf{w}}\|_2^2$ arises from the variance of the dropout mask.
- At inference, dropout is typically disabled, and weights learned with dropout are scaled by $1/p$ to compensate for reduced input magnitude during training.

3. Relate to Traditional Regularization:

- To match Equation 3, transform $\hat{\mathbf{w}}$ so that the data term becomes $\|\mathbf{y} - X\mathbf{w}\|_2^2$ and the regularization term aligns with $\|\Gamma\mathbf{w}\|_2^2$.
- The scaling factor p in $\|\mathbf{y} - pX\hat{\mathbf{w}}\|_2^2$ indicates that $\hat{\mathbf{w}}$ was optimized for a scaled input. To undo this scaling, adjust $\hat{\mathbf{w}}$ by $1/p$.

4. Derive the Transformation:

$$\mathbf{w} = \frac{\hat{\mathbf{w}}}{p} \quad \Rightarrow \quad \|\mathbf{y} - X\mathbf{w}\|_2^2 = \|\mathbf{y} - X \cdot \frac{\hat{\mathbf{w}}}{p}\|_2^2.$$

This restores the data term to the unscaled input.

For the regularization term:

$$\|\Gamma \mathbf{w}\|_2^2 = \left\| \Gamma \cdot \frac{\hat{\mathbf{w}}}{p} \right\|_2^2 = \frac{1}{p^2} \|\Gamma \hat{\mathbf{w}}\|_2^2.$$

In Equation 5, the regularization is $p(1-p)\|\hat{\Gamma}\hat{\mathbf{w}}\|_2^2$. Matching forms implies $\Gamma = \hat{\Gamma}\sqrt{\frac{1-p}{p}}$, but the primary weight transformation is $\mathbf{w} = \hat{\mathbf{w}}/p$.

5. Inference-Time Adjustment Insight:

- During training, the input is effectively pX . The learned weights $\hat{\mathbf{w}}$ fit this scaled input.
- Scaling $\hat{\mathbf{w}}$ by $1/p$ at inference compensates for the full input X , so $\mathbf{w} = \hat{\mathbf{w}}/p$ predicts $\mathbf{y} \approx X\mathbf{w}$.
- The regularization term transforms to $\frac{1-p}{p}\|\hat{\Gamma}\hat{\mathbf{w}}\|_2^2$, resembling a Tikhonov penalty with $\Gamma = \hat{\Gamma}\sqrt{\frac{1-p}{p}}$.

6. Final Transformation:

$$\mathbf{w} = \frac{\hat{\mathbf{w}}}{p}.$$

This adjusts the weights learned with dropout to fit the unscaled input, aligning the data term with Equation 3.

Conceptual Intuition

Dropout trains $\hat{\mathbf{w}}$ on a stochastically reduced input (pX). Scaling $\hat{\mathbf{w}}$ by $1/p$ restores the model's sensitivity to the full input X , mimicking the unregularized part of the Tikhonov problem. The regularization term naturally adjusts to a form comparable to $\|\Gamma\mathbf{w}\|_2^2$ with an appropriate Γ , reflecting dropout's implicit regularization.

Final Answer

To transform the weight $\hat{\mathbf{w}}$ learned using Equation 5 to resemble a solution to the traditionally regularized problem (Equation 3), set

$$\mathbf{w} = \frac{\hat{\mathbf{w}}}{p}.$$

Reflection

correct answer and well-structured. Good part about the conceptual intuition.

Part(c)

ME: That is correct! Now for part (c): With the understanding that the Γ in (3) is an invertible matrix, change variables in (3) to make the problem look like classical ridge regression:

$$L(\tilde{\mathbf{w}}) = \|\mathbf{y} - \tilde{X}\tilde{\mathbf{w}}\|_2^2 + \lambda\|\tilde{\mathbf{w}}\|_2^2 \quad (6)$$

Explicitly, what is the changed data matrix \tilde{X} in terms of the original data matrix X and Γ ?

GROK: To transform the Tikhonov-regularized least-squares problem (Equation 3) into the form of classical ridge regression (Equation 6) by changing variables, we need to redefine the weight vector and data matrix such that the regularization term becomes a simple L_2 norm of the new weights, scaled by a hyperparameter λ . Since Γ is invertible, we can use this property to perform the variable substitution. Let's derive the solution step-by-step with clear reasoning and intuition.

Step-by-Step Reasoning

1. Recap the Original Problem:

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

where X is the $n \times d$ data matrix, \mathbf{y} is the $n \times 1$ target vector, \mathbf{w} is the $d \times 1$ weight vector, and Γ is an invertible $d \times d$ matrix.

The goal is to rewrite this as:

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

where $\tilde{\mathbf{w}}$ is the new weight vector, \tilde{X} is the transformed data matrix, and λ is the ridge regression penalty.

2. Change of Variables: Define

$$\tilde{\mathbf{w}} = \Gamma\mathbf{w} \quad \Rightarrow \quad \mathbf{w} = \Gamma^{-1}\tilde{\mathbf{w}}.$$

Substitute into the original loss:

$$\begin{aligned} X\mathbf{w} &= X\Gamma^{-1}\tilde{\mathbf{w}}, \\ \|\mathbf{y} - X\mathbf{w}\|_2^2 &= \|\mathbf{y} - X\Gamma^{-1}\tilde{\mathbf{w}}\|_2^2, \\ \|\Gamma\mathbf{w}\|_2^2 &= \|\Gamma\Gamma^{-1}\tilde{\mathbf{w}}\|_2^2 = \|\tilde{\mathbf{w}}\|_2^2. \end{aligned}$$

3. Form the Transformed Loss:

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

4. Adjust Regularization Parameter: To generalize, we can introduce λ to scale the regularization term:

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

with $\tilde{X} = X\Gamma^{-1}$.

5. Conceptual Intuition: The transformation $\mathbf{w} = \Gamma^{-1}\tilde{\mathbf{w}}$ rotates and scales the weight space so that the original regularization $\|\Gamma\mathbf{w}\|_2^2$ becomes a standard L_2 norm on $\tilde{\mathbf{w}}$. The data matrix X is adjusted by Γ^{-1} to maintain consistency with the new weight space, aligning the problem with classical ridge regression.

Final Answer

The changed data matrix \tilde{X} in terms of the original data matrix X and Γ is:

$$\tilde{X} = X\Gamma^{-1}.$$

Reflection

Correct answer with a clear explanation, showing how the regularization parameter λ is effectively incorporated through the scaling of the Γ matrix.

Part (d)

ME: Correct! Now for part d: Continuing the previous part, with the further understanding that Γ is a diagonal invertible matrix with the j -th diagonal entry proportional to the norm of the j -th column in X , what can you say about the norms of the columns of the effective training matrix \tilde{X} and speculate briefly on the relationship between dropout and batch-normalization.

GROK: The norms of the columns of the effective training matrix \tilde{X} are all equal, specifically $\sqrt{\frac{p}{1-p}}$ for each column.

Speculatively, this reveals a connection between dropout and batch normalization: dropout's regularization implicitly rescales input features to uniform norms in the transformed space (akin to whitening), which mirrors batch normalization's standardization of activations to unit variance across batches. Both techniques introduce noise—multiplicative for dropout, additive via batch statistics for batch norm—mitigating covariate shift, stabilizing gradients, and enhancing generalization in deep networks.

Reflection

Answers the question very well, keeping the speculative discussion brief yet concise.