

## Executive Summary:

I used Gemini 2.5 Flash to solve questions 1, 2, 3, and 4 on HW 5. Gemini performed well overall on the homework problems, giving mostly correct mathematical results and generally clear, step-by-step explanations. In the convolution and normalization questions, it showed strong understanding of the core concepts and correctly executed most derivations, including the convolution filter construction, simplified batch-norm derivative, and depthwise-separable convolution parameter counts. It was also effective at organizing its solutions cleanly and justifying its steps. However, its answers occasionally lacked important detail or precision. For example, the convolution weight-sharing explanation missed the deeper point about translation equivariance, the transpose-convolution example contained an incorrect overlap addition, and the pointwise-convolution derivation omitted bias terms even though they were shown in the diagram in the prompt. In a few places, Gemini drifted into excessive general explanation without fully addressing the specific question, and it omitted some of the reasoning behind the expressions it reached in the dropout derivation. Overall, Gemini showed strong mathematical competence and solid explanatory skill, but its responses suffered from occasional computational inaccuracies and incomplete treatment of subtle but important details.

## Prompt:

Hello! Please help me solve this homework for EECS182. Include full details and don't skip any important steps!

## Chat Response:

I can certainly help you work through this homework assignment! Since this is a detailed homework, I'll break down the solutions section by section, providing the necessary steps and explanations.

# 1. Convolutional Networks

## (a) Reasons for preferring Convolutional Layers over Fully Connected Layers for Image Data

We typically prefer convolutional layers (Conv layers) over fully connected layers (FC layers) for image data for two main reasons<sup>1</sup>:

1. **Parameter Sharing (Weight Sharing):** A single **filter** (kernel) is dragged across the entire input image/feature map to compute the output. This means the **same weights** are used for different locations in the image. In contrast, an FC layer would require a unique weight for every connection between an input pixel and an output neuron, leading to a massive number of parameters, especially for high-resolution images. Parameter

sharing drastically **reduces the number of learnable parameters**<sup>2</sup>.

2. **Sparsity of Connections (Local Connectivity):** In a Conv layer, each output feature is only computed from a small, **local region** of the input (defined by the filter size). An FC layer, however, would connect every input pixel to every output neuron. Local connectivity is efficient because, in images, **local pixels are highly correlated** and contain the most relevant information for a given feature<sup>3</sup>. This also makes the model more robust to small shifts in the input.

**Notes: Gemini gave correct answers to the question, but reason #1 (weight sharing) could have been expanded on more to include the fact that weight sharing respects invariance to translations in images.**

---

## (b) Finding the 1D Convolutional Filter

The problem provides:

- Input signal:  $X = [1, 4, 0, -2, 3]$ <sup>4</sup>
- Filter length: 3<sup>5</sup>
- Padding: 0<sup>6</sup>
- Stride: 1<sup>7</sup>
- Output sequence:  $Y = [-2, 2, 11]$ <sup>8</sup>

Let the filter be  $W = [w_1, w_2, w_3]$ . The standard convolution operation (without flipping the filter)<sup>9</sup> is defined as:

$$Y_k = \sum_{i=1}^3 X_{k+i-1} \cdot w_i$$

Applying this to the input and output:

1. First output element ( $Y_1 = -2$ ): The filter is applied to the first three input elements  $[1, 4, 0]$ .

$$Y_1 = X_1 w_1 + X_2 w_2 + X_3 w_3$$

$$-2 = 1 w_1 + 4 w_2 + 0 w_3 \implies w_1 + 4 w_2 = -2$$

2. Second output element ( $Y_2 = 2$ ): The filter is shifted by stride 1 and applied to  $[4, 0, -2]$ .

$$Y_2 = X_2 w_1 + X_3 w_2 + X_4 w_3$$

$$2 = 4 w_1 + 0 w_2 + (-2) w_3 \implies 4 w_1 - 2 w_3 = 2 \implies 2 w_1 - w_3 = 1$$

3. Third output element ( $Y_3 = 11$ ): The filter is shifted again and applied to  $[0, -2, 3]$ <sup>12</sup>.

$$Y_3 = X_3 w_1 + X_4 w_2 + X_5 w_3$$

$$11 = 0 w_1 + (-2) w_2 + 3 w_3 \implies -2 w_2 + 3 w_3 = 11$$

We now have a system of three linear equations:

1.  $w_1 + 4 w_2 = -2$
2.  $2 w_1 - w_3 = 1$
3.  $-2 w_2 + 3 w_3 = 11$

**Solve the system:**

- From (1), we express  $w_1$  in terms of  $w_2$ :

$$w_1 = -2 - 4 w_2$$

- Substitute  $w_1$  into (2):

$$2(-2 - 4 w_2) - w_3 = 1$$

$$-4 - 8 w_2 - w_3 = 1$$

$$-8 w_2 - w_3 = 5 \implies w_3 = -8 w_2 - 5$$

- Substitute  $w_3$  into (3):

$$-2 w_2 + 3(-8 w_2 - 5) = 11$$

$$-2 w_2 - 24 w_2 - 15 = 11$$

$$-26 w_2 = 26 \implies \mathbf{w_2 = -1}$$

- Find  $w_3$ :

$$w_3 = -8(-1) - 5 = 8 - 5 \implies \mathbf{w_3 = 3}$$

- Find  $w_1$ :

$$w_1 = -2 - 4(-1) = -2 + 4 \implies \mathbf{w_1 = 2}$$

The filter was  $\mathbf{[2, -1, 3]}$ <sup>13</sup>.

(Self-check using the hint: The first entry in the filter should be 2. Our answer matches this hint)<sup>14</sup>.

**Notes: Gemini gave correct answers to this question and did a self check using the hint**

as well!

---

## (c) Transpose Convolution Output

The input matrix is<sup>15</sup>:

$$X = \begin{pmatrix} -1 & 2 \\ 3 & 1 \end{pmatrix}$$

The filter (kernel) is<sup>16</sup>:

$$W = \begin{pmatrix} +1 & -1 \\ 0 & +1 \end{pmatrix}$$

We are performing a 2D **transpose convolution** with  $\text{pad}=0$  and  $\text{stride}=1$ <sup>17</sup>.

Transpose convolution can be visualized as performing a standard convolution but with zero padding inserted between the input elements based on the stride, and then performing a standard convolution (or correlation)<sup>18</sup>. Since the stride is 1, no zeros are inserted between elements.

With  $\text{stride}=1$  and  $\text{pad}=0$ , the output size of a transpose convolution (which is an upsampling operation) is the same as the input size plus the filter size minus 1, in each dimension:

$$\text{Output Size} = (\text{Input Size} - 1) \times \text{Stride} - 2 \times \text{Pad} + \text{Filter Size}$$

Since  $\text{Input Size}=2$ ,  $\text{Stride}=1$ ,  $\text{Pad}=0$ ,  $\text{Filter Size}=2$ :

$$\text{Output Size} = (2-1) \times 1 - 2 \times 0 + 2 = 1 + 2 = 3$$

The output matrix  $Y$  will be a  $3 \times 3$  matrix.

The operation can be thought of as placing the filter's top-left corner at each input pixel, multiplying the filter elements by the input pixel value, and summing the overlaps<sup>19</sup>.

$$Y = \begin{pmatrix} y_{1,1} & y_{1,2} & y_{1,3} \\ y_{2,1} & y_{2,2} & y_{2,3} \\ y_{3,1} & y_{3,2} & y_{3,3} \end{pmatrix}$$

The calculation for each output element  $y_{i,j}$ :

- From  $X_{1,1} = -1$ : Multiplies  $W$  and contributes to  $Y$  starting at  $(1, 1)$ .

$$\begin{pmatrix} -1 \cdot 1 & -1 \cdot (-1) \\ -1 \cdot 0 & -1 \cdot 1 \end{pmatrix} = \begin{pmatrix} -1 & 1 \\ 0 & -1 \end{pmatrix}$$

- From  $X_{1,2} = 2$ : Multiplies  $W$  and contributes to  $Y$  starting at  $(1, 2)$ .

$$\begin{pmatrix} 2 \cdot 1 & 2 \cdot (-1) \\ 2 \cdot 0 & 2 \cdot 1 \end{pmatrix} = \begin{pmatrix} 2 & -2 \\ 0 & 2 \end{pmatrix}$$

- From  $X_{2,1} = 3$ : Multiplies  $W$  and contributes to  $Y$  starting at  $(2, 1)$ .

$$\begin{pmatrix} 3 \cdot 1 & 3 \cdot (-1) \\ 3 \cdot 0 & 3 \cdot 1 \end{pmatrix} = \begin{pmatrix} 3 & -3 \\ 0 & 3 \end{pmatrix}$$

- From  $X_{2,2} = 1$ : Multiplies  $W$  and contributes to  $Y$  starting at  $(2, 2)$ .

$$\begin{pmatrix} 1 \cdot 1 & 1 \cdot (-1) \\ 1 \cdot 0 & 1 \cdot 1 \end{pmatrix} = \begin{pmatrix} 1 & -1 \\ 0 & 1 \end{pmatrix}$$

Now, sum the contributions (where overlaps are summed)<sup>20</sup>:

- $Y_{1,1} = -1$
- $Y_{1,2} = 1 + 2 = 3$
- $Y_{1,3} = -2$
- $Y_{2,1} = 0 + 3 = 3$
- $Y_{2,2} = -1 + (-3) + 2 + 1 = -1$
- $Y_{2,3} = -2 - 1 = -3$
- $Y_{3,1} = 0$
- $Y_{3,2} = 3 + 0 = 3$
- $Y_{3,3} = 1$

The output of the transpose convolution is:

$$Y = \begin{pmatrix} -1 & 3 & -2 \\ 3 & -1 & -3 \\ 0 & 3 & 1 \end{pmatrix}$$

**Notes: Gemini gave a very nice explanation of how transpose convolution works and it was actually the approach that I used when solving the homework problem, but it didn't do the addition of the overlaps correctly even though it had found the correct individual matrices.**

## 2. Batch Normalization for CNN

The diagram shows a 4D tensor representing the activations, with dimensions  $N$  (mini-batch size),  $C$  (channels),  $H$  (height), and  $W$  (width)<sup>21</sup>. The shaded blocks represent the elements that participate in one normalization step (i.e., the elements used to calculate the mean and variance)<sup>22</sup>.

## (a) Batch Normalization vs. Layer Normalization

The different normalization schemes are distinguished by which dimensions they normalize over:

- **Batch Normalization (BN):** Normalizes over the **mini-batch** dimension ( $N$ ) and the **spatial** dimensions ( $H, W$ ) for a given channel  $C$ <sup>23</sup>. The normalization is done *across* the batch.
- **Layer Normalization (LN):** Normalizes over the **channel** ( $C$ ) and **spatial** dimensions ( $H, W$ ) for a given sample in the batch ( $N$ ). The normalization is done *within* a single sample.
- **A. Batch Normalization (BN):** In diagram A, the shaded region spans all elements in the  $N, H, W$  dimensions but is restricted to a **single channel**  $C$ . This is the standard definition of **Batch Normalization** for a CNN<sup>24</sup>.
- **B. Layer Normalization (LN):** In diagram B, the shaded region spans all elements in the  $C, H, W$  dimensions but is restricted to a **single sample**  $N$ . This is the standard definition of **Layer Normalization** for a CNN<sup>25</sup>.
- **C. Instance/Channel Normalization:** In diagram C, the shaded region spans all elements in the  $H, W$  dimensions but is restricted to a single channel  $C$  and a single sample  $N$ . This is known as **Instance Normalization** or sometimes **Channel Normalization**<sup>26262626</sup>.
- Which one denotes the process of **batch normalization**?  $\mathbf{A}$ <sup>27</sup>.
- Which one denotes **layer normalization**?  $\mathbf{B}$ <sup>28</sup>.

**Notes: Gemini correctly identified which image matched batch normalization and which matched layer normalization. It even identified what the remaining image's normalization was.**

---

## (b) Simplified Batch Normalization

The simplified BN is  $\hat{x}_i = x_i - \mu$ , where  $\mu = \frac{1}{n} \sum_{j=1}^n x_j$ , and the output is  $y_i = \gamma \hat{x}_i + \beta$ <sup>29</sup>. We need to calculate  $\frac{\partial \mathcal{L}}{\partial x_i}$  in terms of  $\frac{\partial \mathcal{L}}{\partial y_j}$ ,  $\gamma$ ,

and  $\beta$ <sup>30</sup>.

## General Derivative $\frac{\partial \mathcal{L}}{\partial x_i}$

Using the chain rule,  $\frac{\partial \mathcal{L}}{\partial x_i}$  depends on all  $y_j$ 's that  $x_i$  influences:

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

First, let's find the components of  $\frac{\partial y_j}{\partial x_i}$ :

$$y_j = \gamma \hat{x}_j + \beta = \gamma (x_j - \mu) + \beta = \gamma \left( x_j - \frac{1}{n} \sum_{k=1}^n x_k \right) + \beta$$

1. Case 1:  $i = j$

$$\frac{\partial y_i}{\partial x_i} = \frac{\partial}{\partial x_i} \left[ \gamma x_i - \gamma \frac{1}{n} \sum_{k=1}^n x_k + \beta \right] = \gamma - \gamma \frac{1}{n} \frac{\partial}{\partial x_i} \sum_{k=1}^n x_k = \gamma - \frac{\gamma}{n} = \gamma \left( 1 - \frac{1}{n} \right)$$

2. Case 2:  $i \neq j$

$$\frac{\partial y_j}{\partial x_i} = \frac{\partial}{\partial x_i} \left[ \gamma x_j - \gamma \frac{1}{n} \sum_{k=1}^n x_k + \beta \right] = 0 - \gamma \frac{1}{n} \frac{\partial}{\partial x_i} \sum_{k=1}^n x_k = -\frac{\gamma}{n}$$

Now, substitute these back into the expression for  $\frac{\partial \mathcal{L}}{\partial x_i}$ :

$$\frac{\partial \mathcal{L}}{\partial x_i} = \frac{\partial \mathcal{L}}{\partial y_i} \frac{\partial y_i}{\partial x_i} + \sum_{j \neq i} \frac{\partial \mathcal{L}}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

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

$$\frac{\partial \mathcal{L}}{\partial x_i} = \gamma \frac{\partial \mathcal{L}}{\partial y_i} - \frac{\gamma}{n} \frac{\partial \mathcal{L}}{\partial y_i} - \frac{\gamma}{n} \sum_{j \neq i} \frac{\partial \mathcal{L}}{\partial y_j}$$

$$\frac{\partial \mathcal{L}}{\partial x_i} = \gamma \frac{\partial \mathcal{L}}{\partial y_i} - \frac{\gamma}{n} \left( \frac{\partial \mathcal{L}}{\partial y_i} + \sum_{j \neq i} \frac{\partial \mathcal{L}}{\partial y_j} \right)$$

$$\frac{\partial \mathcal{L}}{\partial y_j} \text{ right})$$

Recognizing that  $\sum_{j=1}^n \frac{\partial \mathcal{L}}{\partial y_j}$  is the sum of all  $\frac{\partial \mathcal{L}}{\partial y_j}$ 's:

$$\frac{\partial \mathcal{L}}{\partial x_i} = \gamma \frac{\partial \mathcal{L}}{\partial y_i} - \gamma \sum_{j=1}^n \frac{\partial \mathcal{L}}{\partial y_j}$$

---

### Numerical Value when $n=1$

When the batch size  $n=1$ , the input is  $x_1$  and the output is  $y_1$ <sup>31</sup>.

The mean is  $\mu = \frac{1}{1} x_1 = x_1$ <sup>32</sup>.

The normalized value is  $\hat{x}_1 = x_1 - \mu = x_1 - x_1 = 0$ <sup>33</sup>.

The output is  $y_1 = \gamma \hat{x}_1 + \beta = \gamma(0) + \beta = \beta$ <sup>34</sup>.

Let's use the formula derived above with  $n=1$ :

$$\frac{\partial \mathcal{L}}{\partial x_1} = \gamma \frac{\partial \mathcal{L}}{\partial y_1} - \gamma \sum_{j=1}^1 \frac{\partial \mathcal{L}}{\partial y_j}$$

$$\frac{\partial \mathcal{L}}{\partial x_1} = \gamma \frac{\partial \mathcal{L}}{\partial y_1} - \gamma \frac{\partial \mathcal{L}}{\partial y_1} = 0$$

Alternatively, since  $y_1 = \beta$ , which is a constant and does not depend on  $x_1$ , the derivative  $\frac{\partial y_1}{\partial x_1}$  is 0:

$$\frac{\partial \mathcal{L}}{\partial x_1} = \frac{\partial \mathcal{L}}{\partial y_1} \frac{\partial y_1}{\partial x_1} = \frac{\partial \mathcal{L}}{\partial y_1} \cdot 0 = 0$$

The numerical value of  $\frac{\partial \mathcal{L}}{\partial x_1}$  when  $n=1$  is  $\mathbf{0}$ <sup>35</sup>.

---

### Limit when $n \rightarrow \infty$

We look at the limit of the general derivative expression as  $n \rightarrow \infty$ <sup>36</sup>.

$$\lim_{n \rightarrow \infty} \frac{\partial \mathcal{L}}{\partial x_i} = \lim_{n \rightarrow \infty} \left[ \gamma \frac{\partial \mathcal{L}}{\partial y_i} - \gamma \sum_{j=1}^n \frac{\partial \mathcal{L}}{\partial y_j} \right]$$

Since we assume that all relevant quantities (including the sum of the derivatives) are



bounded<sup>37</sup>, the term  $\frac{\gamma}{n} \sum_{j=1}^n \frac{\partial \mathcal{L}}{\partial y_j}$  goes to zero as  $n$  goes to infinity:

$$\lim_{n \rightarrow \infty} \frac{1}{n} \left( \text{bounded quantity} \right) = 0$$

Therefore, as  $n \rightarrow \infty$ :

$$\lim_{n \rightarrow \infty} \frac{\partial \mathcal{L}}{\partial x_i} = \gamma \frac{\partial \mathcal{L}}{\partial y_i}$$

The large-batch limit recovers a local gradient calculation, as the batch mean  $\mu$  (and thus  $\hat{x}_i$ ) becomes effectively independent of the single sample  $x_i$ , meaning  $\frac{\partial \mu}{\partial x_i} \rightarrow 0$ <sup>38</sup>.

**Notes: Gemini correctly performed the chain rule and separated the two cases accordingly. It also correctly calculated the value when  $n = 1$  and the limit as  $n$  approaches positive infinity.**

### 3. Depthwise Separable Convolutions

Depthwise separable convolution decomposes the traditional convolution into two steps: **Depthwise convolution** (spatial-only) followed by **Pointwise convolution** (channel-only,  $1 \times 1$  kernel)<sup>39</sup>. We ignore biases for parameter counting<sup>40</sup>.

Input image size:  $H_{in} \times W_{in} \times C_{in} = 224 \times 224 \times 3$

Kernel size:  $K \times K = 3 \times 3$

Number of output channels:  $C_{out} = 4$

#### (a) Parameters in Traditional Convolution

In a **traditional convolution** (as shown in Figure 1)<sup>44</sup>, each of the  $C_{out}$  output filters operates on **all  $C_{in}$  input channels**<sup>45</sup>.

- Each filter (kernel) has dimensions  $K \times K \times C_{in} = 3 \times 3 \times 3$ .
- There are  $C_{out} = 4$  such filters<sup>46</sup>.

$$\text{Total Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times \text{Input Channels}) \times \text{Output Channels}$$

$$\text{Total Parameters} = (3 \times 3 \times 3) \times 4$$

$$\text{Total Parameters} = 27 \times 4 = \mathbf{108}$$

The number of learnable parameters in the traditional convolution layer is **108**<sup>47</sup>.

**Notes: Gemini correctly utilized the right equation to find the number of weight parameters and followed the instruction to ignore the bias parameters.**

## (b) Parameters in Depthwise Separable Convolution

Depthwise separable convolution (Figures 2 and 3)<sup>48</sup> has two parts:

1.

Depthwise Convolution (DW) (Figure 2)<sup>49</sup>

- The number of output channels is the same as the input channels:  $C_{in} = 3$ <sup>50</sup>.
- There is **no cross talk between channels**<sup>51</sup>. Each of the  $C_{in}=3$  input channels is convoluted with its **own  $3 \times 3 \times 1$  filter**.
- There are  $C_{in}=3$  filters in total<sup>52</sup>.

$$\text{DW Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times 1) \times \text{Input Channels}$$

$$\text{DW Parameters} = (3 \times 3 \times 1) \times 3$$

$$\text{DW Parameters} = 9 \times 3 = \mathbf{27}$$

The intermediate output has dimensions  $H'_{out} \times W'_{out} \times C_{in}$ , which is  $224 \times 224 \times 3$  for this configuration (stride=1, pad=1)<sup>53</sup>.

2.

Pointwise Convolution (PW) (Figure 3)<sup>54</sup>

- This is a traditional convolution where the filter size is  $1 \times 1$ <sup>55</sup>.

- The input to PW is the output from DW, which has  $C_{in} = 3$  channels.
- The number of output channels is  $C_{out} = 4$ .

$$\text{PW Parameters} = (\text{Kernel Height} \times \text{Kernel Width} \times \text{Input Channels}) \times \text{Output Channels}$$

$$\text{PW Parameters} = (1 \times 1 \times 3) \times 4$$

$$\text{PW Parameters} = 3 \times 4 = 12$$

### Total Parameters

The total number of learnable parameters in the depthwise separable convolution is the sum of the parameters from both parts:

$$\text{Total Separable Parameters} = \text{DW Parameters} + \text{PW Parameters}$$

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

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

(For comparison, the parameter reduction ratio is  $39/108 \approx 0.36$ , showing the efficiency).

**Notes: Gemini found the correct number of weight parameters in the depthwise convolutions and pointwise convolutions, but didn't include the bias parameters on the pointwise convolutions. It was probably continuing to follow the instructions from earlier to ignore biases while counting learnable parameters.**

## 4. Regularization and Dropout

### (a) Derivation of Effective Objective Function

We are asked to show the equivalence between the expected loss with dropout:

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

and the regularized objective:

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

Where  $\tilde{w}$  is the weight vector,  $R \in \{0,1\}^{n \times d}$  has entries  $R_{i,j} \sim \text{Bernoulli}(p)$ ,  $X$  is the  $n \times d$  training data matrix, and  $\tilde{w}$  is a

diagonal matrix whose  $j$ -th diagonal entry is the norm of the  $j$ -th column of  $X$ <sup>606060</sup>.

**Proof:**

The expected loss is:

$$\mathbb{E}[\|y - (R \odot X) \tilde{w}\|_2^2]$$

We expand the squared  $L_2$  norm:  $\|A\|_2^2 = A^{\top} A$ .

$$\mathbb{E}[\|y - (R \odot X) \tilde{w}\|_2^2] = \mathbb{E}[(y - (R \odot X) \tilde{w})^{\top} (y - (R \odot X) \tilde{w})]$$

$$\mathbb{E}[\|y - (R \odot X) \tilde{w}\|_2^2] = \mathbb{E}[y^{\top} y - y^{\top} (R \odot X) \tilde{w} - \tilde{w}^{\top} (R \odot X)^{\top} y + \tilde{w}^{\top} (R \odot X)^{\top} (R \odot X) \tilde{w}]$$

Since  $\mathbb{E}[y^{\top} y] = y^{\top} y$ :

$$\mathbb{E}[\|y - (R \odot X) \tilde{w}\|_2^2] = y^{\top} y - 2 y^{\top} \mathbb{E}[(R \odot X) \tilde{w}] + \tilde{w}^{\top} \mathbb{E}[(R \odot X)^{\top} (R \odot X)] \tilde{w}$$

### 1. Calculate $\mathbb{E}[(R \odot X)]$

$R \odot X$  is the matrix where  $(R \odot X)_{i,j} = R_{i,j} X_{i,j}$ .

Since  $R_{i,j} \sim \text{Bernoulli}(p)$ ,  $\mathbb{E}[R_{i,j}] = p$ .

Thus,  $\mathbb{E}[(R \odot X)_{i,j}] = \mathbb{E}[R_{i,j}] X_{i,j} = p X_{i,j}$ .

Therefore,

$$\mathbb{E}[R \odot X] = pX$$

### 2. Calculate $\mathbb{E}[(R \odot X)^{\top} (R \odot X)]$

Let  $M = R \odot X$ . We want  $\mathbb{E}[M^{\top} M]$ , which is a  $d \times d$  matrix.

The  $(j, k)$ -th entry of  $M^{\top} M$  is  $\sum_{i=1}^n M_{i,j} M_{i,k}$ :

$$(\mathbb{E}[M^{\top} M])_{j,k} = \mathbb{E}[\sum_{i=1}^n M_{i,j} M_{i,k}] = \sum_{i=1}^n \mathbb{E}[M_{i,j} M_{i,k}]$$

- Case 1:  $j=k$  (Diagonal entries)

$$\mathbb{E}[M_{i,j} M_{i,j}] = \mathbb{E}[M_{i,j}^2] = \mathbb{E}[(R_{i,j} X_{i,j})^2] =$$

$$X_{i,j}^2 \mathbb{E}[R_{i,j}^2]$$

Since  $R_{i,j} \in \{0, 1\}$ ,  $R_{i,j}^2 = R_{i,j}$ , so  $\mathbb{E}[R_{i,j}^2] = \mathbb{E}[R_{i,j}] = p$ .

$$\mathbb{E}[M_{i,j}^2] = p X_{i,j}^2$$

Thus, the  $(j, j)$ -th entry of  $\mathbb{E}[M^{\text{top}} M]$  is:

$$\sum_{i=1}^n p X_{i,j}^2 = p \sum_{i=1}^n X_{i,j}^2 = p \|X_{:,j}\|_2^2$$

where  $X_{:,j}$  is the  $j$ -th column of  $X$ .

- Case 2:  $j \neq k$  (Off-diagonal entries)

$$\mathbb{E}[M_{i,j} M_{i,k}] = \mathbb{E}[R_{i,j} X_{i,j} R_{i,k} X_{i,k}] = X_{i,j} X_{i,k} \mathbb{E}[R_{i,j} R_{i,k}]$$

Since  $R_{i,j}$  and  $R_{i,k}$  are i.i.d. Bernoulli( $p$ ) for  $j \neq k$ , they are independent.

$$\mathbb{E}[R_{i,j} R_{i,k}] = \mathbb{E}[R_{i,j}] \mathbb{E}[R_{i,k}] = p \cdot p = p^2$$

Thus,  $\mathbb{E}[M_{i,j} M_{i,k}] = p^2 X_{i,j} X_{i,k}$ .

The  $(j, k)$ -th entry of  $\mathbb{E}[M^{\text{top}} M]$  is:

$$\sum_{i=1}^n p^2 X_{i,j} X_{i,k} = p^2 \sum_{i=1}^n X_{i,j} X_{i,k} = p^2 (X^{\text{top}} X)_{j,k}$$

Combining both cases, we can write:

$$\mathbb{E}[(R \odot X)^{\text{top}} (R \odot X)] = p^2 X^{\text{top}} X + (p - p^2) \text{diag}(\mathbf{v})$$

where  $\mathbf{v}$  is the vector with  $j$ -th component  $\|X_{:,j}\|_2^2$ .

Let  $\tilde{\Gamma}$  be the diagonal matrix whose  $j$ -th diagonal entry is  $\|X_{:,j}\|_2^2$ .

Then  $\tilde{\Gamma}^2$  is the diagonal matrix with diagonal entries  $\|X_{:,j}\|_2^4$ , so  $\tilde{\Gamma}^2 = \text{diag}(\mathbf{v})$ .

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

**3. Substitute back into  $\mathcal{L}(\tilde{\mathbf{w}})$**

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

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

(Note:  $\tilde{\Gamma}$  is diagonal, so  $\tilde{\Gamma}^2 = \tilde{\Gamma}$ )

Now consider the right-hand side of the target equation:

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

$$= (y - pX \tilde{w})^{\top} (y - pX \tilde{w}) + p(1-p) \tilde{w}^{\top} \tilde{\Gamma} \tilde{w}$$

$$= y^{\top} y - y^{\top} pX \tilde{w} - \tilde{w}^{\top} pX^{\top} y + \tilde{w}^{\top} p^2 X^{\top} X \tilde{w} + p(1-p) \tilde{w}^{\top} \tilde{\Gamma} \tilde{w}$$

$$= y^{\top} y - 2 p y^{\top} X \tilde{w} + p^2 \tilde{w}^{\top} X^{\top} X \tilde{w} + p(1-p) \tilde{w}^{\top} \tilde{\Gamma} \tilde{w}$$

Since the expressions are identical, we have successfully shown the desired equivalence<sup>62</sup>.

**Notes: Gemini provided a correct proof, but it was a little confusing because it didn't include the method in which we wanted to subtract and add back in a term which would have been nice for better understanding of the solution.**

---

## (b) Transformation of $\tilde{w}$ for Inference

The objective function solved by dropout training (4) is:

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

This minimizes the loss with respect to a **scaled effective data matrix**  $pX$  and a **scaled effective weight**  $p\tilde{w}$ <sup>63</sup>. The objective assumes the weights are used in a *forward pass* as  $pX\tilde{w}$ <sup>64</sup>.

At **inference (test) time**, the true forward pass is usually  $X w_{\text{inf}}$ , where  $w_{\text{inf}}$  is the weight vector used for prediction. We want the expectation of the output under dropout to match the output at inference, i.e.,  $\mathbb{E}[(R \odot X) \tilde{w}] \approx X w_{\text{inf}}$ .

From part (a), we know  $\mathbb{E}[(R \odot X) \tilde{w}] = (\mathbb{E}[R \odot X]) \tilde{w} = (pX) \tilde{w} = X (p\tilde{w})$ .

To match the inference output  $X w_{\text{inf}}$ , we must have:

$$X w_{\text{inf}} = X (\tilde{w})$$

Therefore, the learned weights  $\tilde{w}$  from dropout training should be transformed by scaling them by the dropout probability  $p$  to get the inference weights  $w_{\text{inf}}$ <sup>65</sup>:

$$w_{\text{inf}} = p \tilde{w}$$

This is the standard transformation applied to weights learned with dropout (often referred to as **Inverted Dropout** when applied during training, but here it's applied after training)<sup>66</sup>.

**Notes: Gemini provided the correct solution.**

## (c) Classical Ridge Regression Form

The Tikhonov-regularized (generalized ridge-regression) problem is:

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

We are asked to change variables to make it look like classical ridge regression<sup>67</sup>:

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

The key to this transformation is to absorb the matrix  $\Gamma$  into a change of variables. Since  $\Gamma$  is invertible, we can define a new weight vector  $\tilde{w}$  such that  $w = \Gamma^{-1} \tilde{w}$ <sup>68</sup>.

### 1. Substitute $w$ into the $\mathcal{L}(w)$

- First term:

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

- Second term:

$$\|\Gamma w\|_2^2 = \|\Gamma(\Gamma^{-1} \tilde{w})\|_2^2 = \|\tilde{w}\|_2^2$$

### 2. Form the new objective

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

3. Match the target form

To match the target form

$\mathcal{L}(\tilde{w}) = \|y - \tilde{X} \tilde{w}\|_2^2 + \lambda \|\tilde{w}\|_2^2$ , we set  $\lambda = 1$  and the new data matrix  $\tilde{X}$  is:

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

(Note: In the more general case, if  $\lambda$  was required to be an arbitrary scalar, we could have used  $w = (\sqrt{\lambda} \Gamma^{-1}) \tilde{w}$ ).

**Notes: Gemini reached the correct solution, utilizing the same approach as the provided solution.**

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

We are given that  $\Gamma$  is a diagonal invertible matrix whose  $j$ -th diagonal entry is proportional to the norm of the  $j$ -th column in  $X$ .

Let  $C_j$  be the  $j$ -th column of  $X$  (the  $j$ -th feature vector across all training examples).

The  $j$ -th diagonal entry of  $\Gamma$  is  $\Gamma_{j,j} = \alpha \|C_j\|_2$ , where  $\alpha$  is the proportionality constant.

The new data matrix is  $\tilde{X} = X \Gamma^{-1}$ .

Since  $\Gamma$  is a diagonal matrix,  $\Gamma^{-1}$  is also diagonal, and its  $j$ -th diagonal entry is  $(\Gamma^{-1})_{j,j} = \frac{1}{\Gamma_{j,j}} = \frac{1}{\alpha \|C_j\|_2}$ .

The columns of  $\tilde{X}$  are formed by scaling the columns of  $X$ :

$$\tilde{C}_j = \frac{1}{\Gamma_{j,j}} C_j = \frac{1}{\alpha \|C_j\|_2} C_j$$

Now, let's find the norm of the  $j$ -th column of  $\tilde{X}$ :

$$\|\tilde{C}_j\|_2 = \left\| \frac{1}{\alpha \|C_j\|_2} C_j \right\|_2$$

$$\|\tilde{C}_j\|_2 = \left| \frac{1}{\alpha \|C_j\|_2} \right| \|C_j\|_2$$

Assuming  $\alpha > 0$ :

$$\|\tilde{C}_j\|_2 = \frac{1}{\alpha}$$



### What we can say about the norms of the columns of $\tilde{X}$ :

The norms of all columns of the effective training matrix  $\tilde{X}$  are **equal** and proportional to  $\frac{1}{\alpha}$ <sup>73</sup>. The transformation effectively **normalizes the norm of each feature (column) vector** to be the same constant value<sup>74</sup>.

### Speculation on the relationship between dropout and batch-normalization:

- **Dropout** (when viewed as regularization) is related to a problem that **normalizes the feature norms** of the data matrix  $X$ <sup>75</sup>.
- **Batch Normalization (BN)** is designed to **normalize the mean and variance** of the activations (features) in the hidden layers<sup>76</sup>.

This connection suggests that both Dropout and Batch Normalization are performing a type of **regularization** by **standardizing the scale** of the features<sup>77</sup>. While dropout achieves this through random multiplication which is equivalent to norm normalization, batch normalization achieves this directly by adjusting the mean and standard deviation of the batch statistics. They both attempt to make the optimization landscape less dependent on the scale of individual features, which can speed up training and improve generalization<sup>78</sup>.

**Notes: Gemini reached the correct solution and provided the right explanation of how dropout and batch normalization are similar.**