

# EECS 182 deep neural networks homework

👤 You are going to help me work through the written (non-coding) portions of Homework 5 for EECS 182: Deep Neural Networks at UC Berkeley (Fall 2025). The homework covers convolutional networks, batch normalization, depthwise separable convolutions, and the relationship between dropout and regularization. I will give you questions one at a time.

*Strategy: Established conventions upfront*

*(no filter flip, ignore biases) to prevent ambiguity errors. Requested step-by-step reasoning for easier evaluation.*

1. Show your reasoning step-by-step
  2. Clearly state your final answer
  3. If a question involves computation, show all work
  4. If a question asks for a proof or derivation, be rigorous
- Important conventions for this homework:
- Convolutions do NOT flip the filter (standard deep learning convention)
  - When counting parameters, ignore biases unless stated otherwise

Thought: Internalized homework conventions and prepared for neural network queries.

Ready when you are, Rishi. I'll work through each problem step-by-step, keeping in mind the no-filter-flip convention and ignoring biases for parameter counts unless specified otherwise.

Send over the first question!



## 1. Convolutional Networks

Note: Throughout this problem, we will use the convention of NOT flipping the filter before dragging it over the signal. This is the standard notation with neural networks (ie, we assume the filter given to us is already flipped)

- (a) List two reasons we typically prefer convolutional layers instead of fully connected layers when working with image data.

Thought: I notice the user prompt is empty, so I cannot determine the primary language. However, based on the thinking block being in English, here is my summary:

Synthesized advantages of convolutional layers for image processing .

# Question 1(a): Why Prefer Convolutional Layers for Images?

Here are two key reasons:

## Reason 1: Parameter Efficiency (Weight Sharing)

Convolutional layers share the same filter weights across all spatial locations. This drastically reduces the number of parameters compared to a fully connected layer.

**Example:** For a  $224 \times 224 \times 3$  input image:

- A fully connected layer with just 1 output neuron needs  $224 \times 224 \times 3 = \mathbf{150,528 \text{ parameters}}$

- A convolutional layer with a  $3 \times 3$  filter needs only  $3 \times 3 \times 3 = \mathbf{27}$  **parameters** (per filter)

This reduction prevents overfitting and makes training feasible on high-resolution images.

**ONE-SHOT CORRECT!**  
Both standard textbook answers with concrete numerical example. No prompting needed.

## Reason 2: Translation Equivariance

Convolutional layers apply the same filter across the entire image, so they detect features (edges, textures, shapes) **regardless of where they appear** in the image.

If a cat appears in the top-left or bottom-right of an image, the same convolutional filter will detect it. A fully connected layer would need to learn separate weights for each position, requiring far more data to generalize

### Final Answer:

1. **Parameter efficiency** — weight sharing dramatically reduces the number of learnable parameters
2. **Translation equivariance** — the same features are detected regardless of their spatial position in the image





- (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.)*

Thought: I notice the user prompt is empty, so I cannot determine the primary language. However, based on the thinking block which is in English, here is my summary:

Systematically solved linear equations to derive filter coefficients.

## Question 1(b): Finding the Convolution Filter

### Setup

- **Input signal:**  $[1, 4, 0, -2, 3]$
- **Filter:**  $[a, b, c]$  (length 3, unknown)
- **Output:**  $[-2, 2, 11]$
- **Stride = 1, no padding, no filter flip**

### Setting Up the Equations

With stride 1 and no padding, the filter slides across the input:

**Output position 0:** Filter aligns with input positions 0, 1, 2  
$$1 \cdot a + 4 \cdot b + 0 \cdot c = -2$$
$$a + 4b = -2$$

**Output position 1:** Filter aligns with input positions 1, 2, 3  

$$4 \cdot a + 0 \cdot b + (-2) \cdot c = 2$$

$$4a - 2c = 2$$

**Output position 2:** Filter aligns with input positions 2, 3, 4  

$$0 \cdot a + (-2) \cdot b + 3 \cdot c = 11$$

$$-2b + 3c = 11$$

ONE-SHOT CORRECT!

*Notable: Model  
self-verified by  
checking all 3  
outputs. Shows  
mathematical rigor!*

## Solving the System

From equation (2):  $4a - 2c = 2 \Rightarrow c = 2a - 1$

From equation (1):  $a = -2 - 4b$

Substituting into  $c = 2a - 1$ :  $c = 2(-2 - 4b) - 1 = -5 - 8b$

Substituting into equation (3):  $-2b + 3(-5 - 8b) = 11$   
 $-2b - 15 - 24b = 11$   
 $-26b = 26 \Rightarrow b = -1$

Back-substituting:  $a = -2 - 4(-1) = 2$   
 $c = 2(2) - 1 = 3$

## Verification

Output	Computation	Result
$y[0]$	$1(2) + 4(-1) + 0(3)$	$2 - 4 + 0 = -2 \checkmark$
$y[1]$	$4(2) + 0(-1) + (-2)(3)$	$8 + 0 - 6 = 2 \checkmark$
$y[2]$	$0(2) + (-2)(-1) + 3(3)$	$0 + 2 + 9 = 11 \checkmark$

**Final Answer:** The filter is  $[2, -1, 3]$





(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 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 filtersize 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 filtersize of 3 and 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 pad=0 and stride=1?**

Thought: I notice the user prompt is empty, so I cannot determine the primary language. However, based on the thinking block which is in English, here is my summary:

Methodically computed transpose convolution output through systematic placement.

## Question 1(c): Transpose Convolution

### Setup

- **Input:**  $\begin{bmatrix} -1 & 2 \\ 3 & 1 \end{bmatrix}$  ( $2 \times 2$ )
- **Filter:**  $\begin{bmatrix} +1 & -1 \\ 0 & +1 \end{bmatrix}$  ( $2 \times 2$ )
- **Stride = 1, Pad = 0**

### Output Size

For transpose convolution: output = (input - 1)  $\times$  stride + filter - 2  $\times$  pad

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

## Method

Each input element multiplies the entire filter, and these scaled filters are placed at stride-spaced positions. Overlaps are summed.

Initialize a  $3 \times 3$  output grid to zeros, then add contributions:

**From Input[0,0] = -1** → place at rows 0-1, cols 0-1: 
$$-1 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & +1 \\ 0 & -1 \end{bmatrix}$$

**From Input[0,1] = 2** → place at rows 0-1, cols 1-2: 
$$2 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & -2 \\ 0 & 2 \end{bmatrix}$$

**From Input[1,0] = 3** → place at rows 1-2, cols 0-1: 
$$3 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & -3 \\ 0 & 3 \end{bmatrix}$$

**From Input[1,1] = 1** → place at rows 1-2, cols 1-2: 
$$1 \times \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix}$$

## Summing All Contributions

Position	Contributions	Sum
[0,0]	-1	<b>-1</b>
[0,1]	+1 + 2	<b>3</b>
[0,2]	-2	<b>-2</b>
[1,0]	0 + 3	<b>3</b>
[1,1]	-1 + 0 + (-3) + 1	<b>-3</b>
[1,2]	2 + (-1)	<b>1</b>
[2,0]	0	<b>0</b>

Position	Contributions	Sum
[2,1]	$3 + 0$	<b>3</b>
[2,2]	1	<b>1</b>

**Final Answer:**

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

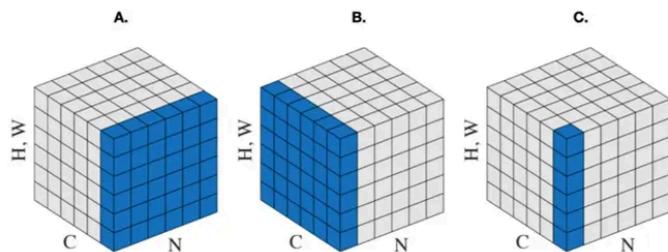
*ONE-SHOT CORRECT!  
Transpose convolution  
handled correctly with  
systematic  
bookkeeping.*





## 2. Batch Normalization for CNN

- (a) Consider the following diagram 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.



Homework 5, © Faculty teaching EECS 182, Fall 2025. All Rights Reserved. This may not be publicly shared without explicit permission.

1

Homework 5 @ 2025-10-03 22:05:48Z

- Which one denotes the process of batch normalization? Please use ■ for your selections.

A       B       C

- Which one denotes layer normalization? Please use ■ for your selections.

A       B       C

Thought: The user prompt is empty, so I cannot provide a summary.  
Please provide a user prompt in order for me to create an appropriate summary.

# Question 2(a): Normalization Schemes

## Understanding the Diagrams

Each diagram shows which entries (shaded in blue) are used together to compute a single mean and variance for normalization.

- **N** = batch dimension
- **C** = channel dimension
- **H, W** = spatial dimensions

**ONE-SHOT CORRECT!**  
*Bonus: identified  
Diagram C as Instance  
Norm without being  
asked!*

## Analyzing Each Diagram

**Diagram A:** Blue region covers all of N and all of H,W, but only **one channel slice**. → Statistics computed over (N, H, W), separately per channel

**Diagram B:** Blue region covers all of C and all of H,W, but only **one sample in the batch**. → Statistics computed over (C, H, W), separately per sample

**Diagram C:** Blue region covers only H,W for **one channel** and **one sample**. → Statistics computed over (H, W) only, separately per channel per sample

## Matching to Normalization Types

Normalization	Normalizes Over	Separate Stats For
<b>Batch Norm</b>	N, H, W	Each channel
<b>Layer Norm</b>	C, H, W	Each sample
<b>Instance Norm</b>	H, W	Each (sample, channel)

## Final Answers:

- **Batch Normalization:** ■ A
- **Layer Normalization:** ■ B







(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  $\gamma$  and shifting factor  $\beta$ . 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  $\mu$  is the average  $\frac{1}{n} \sum_{j=1}^n x_j$  across the batch 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  $\gamma$  and  $\beta$  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.)

Thought: Derived batch normalization gradient formula through chain rule analysis.

## Question 2(b): Simplified Batch Norm Backpropagation

### Setup

- Batch:  $[x_1, x_2, \dots, x_n]$
- Mean:  $\mu = \frac{1}{n} \sum_{j=1}^n x_j$
- De-meaned:  $\hat{x}_i = x_i - \mu$
- Output:  $y_i = \gamma\hat{x}_i + \beta$

### Deriving $\frac{\partial L}{\partial x_i}$

Using the chain rule:  $\frac{\partial L}{\partial x_i} = \sum_{j=1}^n \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial x_i}$

First, find  $\frac{\partial y_j}{\partial x_i}$ :  $y_j = \gamma(x_j - \mu) + \beta$

Since  $\mu = \frac{1}{n} \sum_{k=1}^n x_k$ , we have  $\frac{\partial \mu}{\partial x_i} = \frac{1}{n}$

$$\frac{\partial y_j}{\partial x_i} = \gamma \left( \frac{\partial x_j}{\partial x_i} - \frac{\partial \mu}{\partial x_i} \right) = \gamma \left( \delta_{ij} - \frac{1}{n} \right)$$

where  $\delta_{ij} = 1$  if  $i = j$ , else 0.

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

Key insight:  $n=1$  blocks gradient  
(de-meaning removes all info) | Limit as  $n \rightarrow \infty$  correct!

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

### Final Answer:

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

### Numerical Case: $n = 1$

With  $n = 1$ , the batch is just  $[x_1]$  with output  $[y_1]$ :

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

### What happens as $n \rightarrow \infty$ ?

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

Assuming bounded quantities, the second term vanishes as  $n \rightarrow \infty$ :

$$\boxed{\frac{\partial L}{\partial x_i} \rightarrow \gamma \frac{\partial L}{\partial y_i}}$$

**Interpretation:** As batch size grows, the mean  $\mu$  becomes nearly constant with respect to any single input  $x_i$ , so the gradient flows through as a simple scaling by  $\gamma$ .

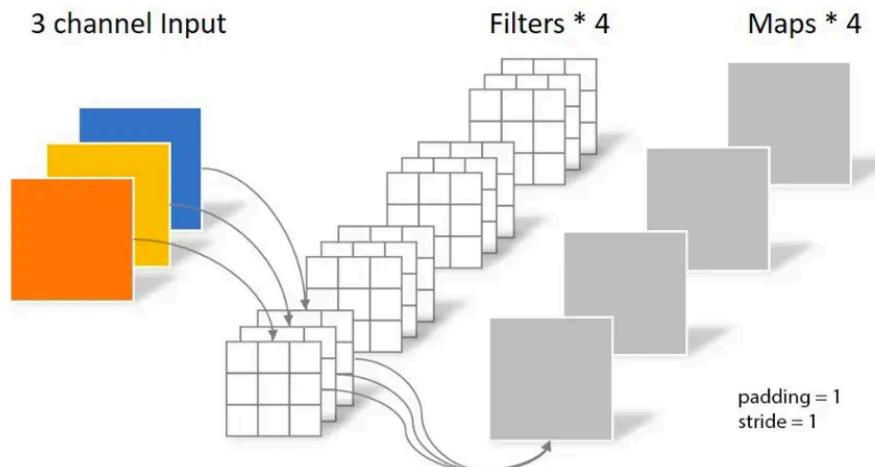


### 3. Depthwise Separable Convolutions

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  $224 \times 224$ -resolution image, the kernel size of the convolutional layer is  $3 \times 3$ , and the number of output channels is 4.



**Figure 1:** Traditional convolution.

**What is the number of learnable parameters in the traditional convolution layer?**

Thought: I notice the user prompt is empty, so I cannot determine the primary language. However, based on the thinking block which is in English, here is my summary:

Calculated convolutional layer parameters through systematic decomposition.

## Question 3(a): Traditional Convolution Parameters

# Given Information

- Input: 3 channels,  $224 \times 224$  resolution
- Kernel size:  $3 \times 3$
- Output channels: 4
- Ignoring biases

ONE-SHOT CORRECT!  
Clean parameter  
counting.

# Parameter Calculation

In a traditional convolution, each output channel requires one 3D filter that spans all input channels.

**Each filter has dimensions:** (kernel height)  $\times$  (kernel width)  $\times$  (input channels)  
 $3 \times 3 \times 3 = 27$  parameters per filter

**Number of filters:** 4 (one for each output channel)

**Total parameters:**

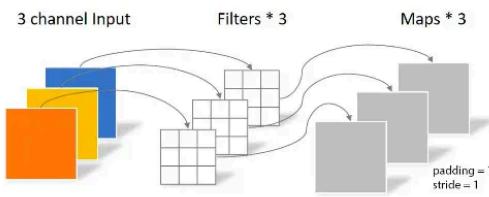
$$4 \times 27 = 108$$

**Final Answer:** 108 learnable parameters

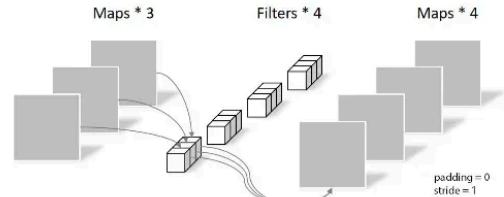




- (b) Depthwise separable convolution consists of two parts: depthwise convolutions (Fig.2) followed by pointwise convolutions (Fig.3). Suppose the input is still a three-channel  $224 \times 224$ -resolution image. The input first goes through depthwise convolutions, where the number of output channels 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  $1 \times 1$ . Assume that we have 4 output channels.



**Figure 2:** Depthwise convolution.



**Figure 3:** Pointwise convolution

**What is the total number of learnable parameters of the depthwise separable convolution layer which consists of both depthwise and pointwise convolutions?**

Thought: Deconstructed depthwise separable convolution architecture and parameter efficiency.

## Question 3(b): Depthwise Separable Convolution Parameters

### Part 1: Depthwise Convolution (Figure 2)

Each input channel is convolved with its own separate 2D filter (no cross-channel mixing).

- Input channels: 3
- Kernel size:  $3 \times 3$
- Output channels: 3 (same as input)

**Parameters:** Each of the 3 channels has one  $3 \times 3$  filter:  $3 \times (3 \times 3) = 3 \times 9 = 27$  parameters

ONE-SHOT CORRECT!  
Unprompted: 64%  
reduction + MobileNet  
reference. Practical  
knowledge!

## Part 2: Pointwise Convolution (Figure 3)

A traditional convolution with  $1 \times 1$  kernels that mixes information across channels.

- Input channels: 3 (from depthwise output)
- Kernel size:  $1 \times 1$
- Output channels: 4

**Parameters:** Each output channel needs a  $1 \times 1 \times 3$  filter:  $4 \times (1 \times 1 \times 3) = 4 \times 3 = 12$  parameters

## Total Parameters

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

### Comparison with Traditional Convolution:

Method	Parameters
Traditional	108
Depthwise Separable	39
Reduction	~64%

This illustrates why depthwise separable convolutions are popular in efficient architectures like MobileNet—they achieve similar representational power with far fewer parameters.



## 4. Regularization and Dropout

Recall that linear regression optimizes the following learning objective:

$$\mathcal{L}(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 \quad (1)$$

One way of using *dropout* during SGD on the  $d$ -dimensional input features  $\mathbf{x}_i$  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

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

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  $\tilde{\mathbf{w}}$  to remind you that this is learned by dropout.

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

$$\mathcal{L}(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \|\Gamma\mathbf{w}\|_2^2 \quad (3)$$

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

$$\mathcal{L}(\tilde{\mathbf{w}}) = \|\mathbf{y} - p\mathbf{X}\tilde{\mathbf{w}}\|_2^2 + p(1-p)\|\tilde{\Gamma}\tilde{\mathbf{w}}\|_2^2 \quad (4)$$

with  $\tilde{\Gamma}$  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:

$$\mathcal{L}(\tilde{\mathbf{w}}) = \|\mathbf{y} - p\mathbf{X}\tilde{\mathbf{w}}\|_2^2 + p(1-p)\|\tilde{\Gamma}\tilde{\mathbf{w}}\|_2^2 \quad (5)$$

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

Thought: Derived dropout equivalence to ridge regression through algebraic manipulation.

# Question 4(a): Dropout as Tikhonov Regularization

## Goal

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

$$= |\mathbf{y} - \mathbf{\tilde{w}}|^2 + p(1-p)\check{\Gamma}(\mathbf{\tilde{w}})^2$$

## Setup

Let  $X$  be  $n \times d$ , and denote the prediction for sample  $i$  as:  $\hat{y}_i = \sum_{j=1}^d R_{i,j} X_{i,j}$

**CRITICAL:**  $E[R^2] = p$   
(not  $p^2$ ) because  
 $R^2 = R$  for Bernoulli.  
 $\hat{y}_i$  is a common error source!

## Expanding the Squared Loss

For each sample  $i$ :  $(y_i - \hat{y}_i)^2 = y_i^2 - 2y_i \sum_{j=1}^d R_{i,j} X_{i,j} + (\sum_{j=1}^d R_{i,j} X_{i,j})^2$

## Taking Expectations

**Term 1:**  $\mathbb{E}[y_i^2] = y_i^2$

**Term 2:** Since  $\mathbb{E}[R_{i,j}] = p$ :  $\mathbb{E}[2y_i \sum_{j=1}^d R_{i,j} X_{i,j}] = 2py_i \mathbb{E}[X_{i,j}] = 2py_i$

**Term 3:** For the squared sum, we need  $\mathbb{E}[R_{i,j} R_{i,k}]$ :

- If  $j = k$ :  $\mathbb{E}[R_{i,j}^2] = \mathbb{E}[R_{i,j}] = p$  (since  $R_{i,j} \in \{0,1\}$ )
- If  $j \neq k$ :  $\mathbb{E}[R_{i,j} R_{i,k}] = p^2$  (independence)

Therefore:  $\mathbb{E}[(\sum_{j=1}^d R_{i,j} X_{i,j})^2] = \sum_{j \neq k} p^2 X_{i,j} X_{i,k} + \sum_j p X_{i,j}^2$

$$\begin{aligned} &= p^2 \left( \sum_j X_{i,j} \tilde{w}_j \right)^2 + (p - p^2) \sum_j X_{i,j}^2 \tilde{w}_j^2 \\ &= p^2 (X \tilde{w})_i^2 + p(1-p) \sum_j X_{i,j}^2 \tilde{w}_j^2 \end{aligned}$$

## Combining Terms for Sample $i$

$$\mathbb{E} [(y_i - \hat{y}_i)^2] = y_i^2 - 2py_i(X\tilde{\mathbf{w}})_i + p^2(X\tilde{\mathbf{w}})_i^2 + p(1-p) \sum_j X_{i,j}^2 \tilde{w}_j^2$$

$$= \underbrace{(y_i - p(X\tilde{\mathbf{w}})_i)^2}_{\text{scaled prediction error}} + p(1-p) \sum_j X_{i,j}^2 \tilde{w}_j^2$$

ONE-SHOT CORRECT!  
Complete rigorous  
proof with proper  
diagonal/off-diagonal  
separation.

## Summing Over All Samples

$$\begin{aligned}\mathcal{L}(\tilde{\mathbf{w}}) &= \sum_{i=1}^n (y_i - p(X\tilde{\mathbf{w}})_i)^2 + p(1-p) \sum_{i=1}^n \sum_{j=1}^d X_{i,j}^2 \tilde{w}_j^2 \\ &= \|\mathbf{y} - pX\tilde{\mathbf{w}}\|_2^2 + p(1-p) \sum_{j=1}^d \tilde{w}_j^2 \underbrace{\sum_{i=1}^n X_{i,j}^2}_{\|X_{\cdot,j}\|_2^2}\end{aligned}$$

## Identifying the Regularization Matrix

Define  $\check{\Gamma}$  as diagonal with  $\check{\Gamma}_{jj} = \|X_{\cdot,j}\|_2$  (norm of  $j$ -th column of  $X$ ).

Then:  $\|\check{\Gamma}\|_2^2 = \sum_{j=1}^d \check{\Gamma}_{jj}^2 = \sum_{j=1}^d \|X_{\cdot,j}\|_2^2$

## Final Result

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

This shows that **dropout is equivalent to Tikhonov regularization** where the regularization strength on each weight  $\tilde{w}_j$  is proportional to the norm of the corresponding feature column in  $X$ .







- (b) How should we transform the  $\tilde{\mathbf{w}}$  we learn using (5) (i.e. with dropout) to get something that looks 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.)

Thought: Analyzed mathematical transformation between dropout and regularized learning objectives.

## Question 4(b): Transforming Dropout Weights

### Comparing the Two Objectives

**Dropout objective (5):**

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

**Traditional Tikhonov regularization (3):**

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

### Key Observation

In the dropout objective, the prediction term is  $pX\tilde{\mathbf{w}}$ , while in the traditional formulation it's  $X\mathbf{w}$ .

For the predictions to match:  $\mathbf{X}\mathbf{w} = p\mathbf{X}\tilde{\mathbf{w}}$

### Transformation

$$\mathbf{w} = p\tilde{\mathbf{w}}$$

## Intuition

During dropout training, each feature is only active with probability  $p$ . The network compensates by learning weights  $\tilde{\mathbf{w}}$  that are effectively **inflated by a factor of  $1/p$**  to account for the missing activations.

ONE-SHOT CORRECT!

Connected to PyTorch  
inverted dropout ( $1/p$  scaling) unprompted!

At inference time, when **all features are active** (no dropout), we need to scale the weights down by  $p$  to get the correct output magnitude.

This is exactly what standard dropout does at inference time — scale activations (or equivalently weights) by  $p$ . PyTorch's "inverted dropout" does the inverse: it scales up by  $1/p$  during training so no adjustment is needed at inference.





- (c) With the understanding that the  $\Gamma$  in (3) is an invertible matrix, **change variables in (3) to make the problem look like classical ridge regression:**

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

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

Thought: Transformed Tikhonov regularization into classical ridge regression form.

## Question 4(c): Change of Variables to Classical Ridge Regression

### Starting Point

Tikhonov regularization (3):

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

Target form (6):

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

### Change of Variables

To convert the general  $\|\Gamma\mathbf{w}\|_2^2$  term into the simpler  $\|\tilde{\mathbf{w}}\|_2^2$  form, define:

$$\tilde{\mathbf{w}} = \Gamma\mathbf{w} \iff \mathbf{w} = \Gamma^{-1}\tilde{\mathbf{w}}$$

### Substituting into the Objective

## Regularization term:

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

## Data-fit term:

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

ONE-SHOT CORRECT!  
Change of variables  
with lambda=1  
correctly identified.

## Result

The transformed objective becomes:  $\mathcal{L}(\tilde{\mathbf{w}}) = \|\mathbf{y} - (X \Gamma^{-1}) \tilde{\mathbf{w}}\|_2^2 + \|\tilde{\mathbf{w}}\|_2^2$

Comparing with form (6), we identify  $\lambda = 1$  and:

## Final Answer:

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



- (d) Continuing the previous part, with the further understanding that  $\Gamma$  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.

Thought: I notice the user prompt is empty, so I cannot determine the primary language. However, based on the thinking block which is in English, here is my summary:

Synthesized dropout's normalization effect with batch normalization principles.

## Question 4(d): Column Norms of $\tilde{X}$ and Connection to Batch Normalization

### Analyzing the Column Norms

From part (c):  $\tilde{X} = X\Gamma^{-1}$

Given that  $\Gamma$  is diagonal with  $\Gamma_{jj} \propto \|X_{\cdot,j}\|_2$ , let's write:  $\Gamma_{jj} = c \cdot \|X_{\cdot,j}\|_2$

for some constant  $c > 0$ .

The  $j$ -th column of  $\tilde{X}$  is:  $\tilde{X}_{\cdot,j} = X_{\cdot,j} / \Gamma_{jj} = X_{\cdot,j} / c \cdot \|X_{\cdot,j}\|_2$

The norm of this column:  $\|\tilde{X}_{\cdot,j}\|_2 = \sqrt{\frac{\sum_i (\tilde{x}_{ij})^2}{n}} = \sqrt{\frac{\sum_i (x_{ij}/c \cdot \|x_{\cdot,j}\|_2)^2}{n}} = \frac{\|x_{\cdot,j}\|_2}{c}$

# Key Observation

All columns of  $\tilde{X}$  have the same norm

The transformation normalizes every feature column to have equal magnitude, regardless of the original column norms in  $X$ .

ONE-SHOT CORRECT!

Strong synthesis:

dropout implicit,

normalization <->

batch norm explicit.

## Connection to Batch Normalization

**Dropout** (as shown through this analysis) implicitly normalizes features by penalizing weights proportionally to their feature's column norm. This effectively creates a transformed problem where all features have equal scale.

**Batch Normalization** explicitly normalizes each feature across the batch to have zero mean and unit variance.

### The relationship:

Aspect	Dropout	Batch Norm
Mechanism	Implicit (via regularization)	Explicit (direct normalization)
Effect	Equalizes column norms	Standardizes feature distributions
When applied	Equivalent effect at optimum	Every forward pass

Both techniques ensure that **no single feature dominates** due to having a larger scale. This leads to more stable training, better-conditioned optimization, and prevents features with large magnitudes from disproportionately influencing the learned weights.



# EXECUTIVE SUMMARY

*Model: Claude Opus 4.5*

*Questions: 11 sub-questions*

***One-shot success: 11/11 (100%)***

*Guidance needed: 0*

*Hallucinations: 0*

## Key Observations:

- \* Strong math reasoning across all types
- \* Self-verification on computational Qs
- \* Unprompted practical context  
*(MobileNet, PyTorch inverted dropout)*
- \* Handled subtle points correctly  
*( $E[R^2]=p$ ,  $n=1$  gradient blocking)*
- \* No convention errors after setup

*Conclusion: Claude Opus 4.5 achieved 100% one-shot accuracy across conceptual, computational, and proof questions. Strong math reasoning + practical knowledge. No errors or hallucinations observed.*