Special Participation A — LLM Interaction Report (gpt-oss)

Noah Lund Syrdal — EECS 182 Deep Neural Networks

Fall 2025

Executive Summary

This report documents an interactive engagement with the model `gpt-oss-120b (Reasoning = High)` on the non-coding analytical parts of EECS 182 Homework 5. The purpose was to evaluate how well a modern large language model can independently reason through conceptual and symbolic questions without external computation or code execution.

- **Homework scope:** EECS 182 HW5 — all *non-coding* questions (Q1–Q4).
- **Questions covered:** Q1 (a–c), Q2 (a–b), Q3 (a–b), Q4 (a–d).
- **Evaluation metric:**
  - L1 (one-shot correct): **9 / 11 $\approx$ 82%**
  - L2 (minor nudge): **2 / 11 $\approx$ 18%**
  - L3 (heavy guidance): **0 / 11**
  - Lx (still incorrect): **0 / 11**
  - Average interventions per question: **0.18**
- **Frequent issues:** Occasional mis-parsing of ASCII matrices; confusion between BatchNorm vs LayerNorm axes; one transient non-response resolved by re-prompting.
- **Hallucinations:** None substantive—minor over-confidence in a few explanations without full derivations.
- **Overall verdict:** For analytical and symbolic reasoning, `gpt-oss-120b` produced correct derivations with almost no intervention. Visual prompts had to be reformulated textually, but when expressed in pure text or LaTeX, performance was reliable and interpretable.

Interaction Protocol

Each sub-question was entered as a plain-text / LaTeX prompt (no figures). All model outputs were recorded verbatim and annotated using the following scheme:

Observation: Accuracy, errors, or misconceptions in the model's reasoning.

Strategy: Prompt-engineering tactics used to steer or clarify (e.g., re-formatting matrices, re-asserting definitions).

Verdict: Performance tier — L1 (one-shot), L2 (minor guidance), L3 (heavy guidance), Lx (fail).

Annotated Logs by Question

Q1 (a) — Why Convolutions for Images

Prompt. Explain why convolutional layers are preferred over fully connected layers for image data. **Model Reply.** Weight sharing reduces parameter count; local receptive fields capture spatial patterns and translation invariance. **Observation.** Fully correct and concise. **Verdict L1.**

Q1 (b) — 1D Filter Recovery

Prompt. Signal [1, 4, 0, –2, 3]; 3-tap filter; stride 1; no padding; output [–2, 2, 11]. Find the filter (no flip convention). **Model Reply.** Filter [2, –1, 3]. **Observation.** Correct numeric derivation and reasoning. **Verdict L1.**

Q1 (c) — Transpose Convolution (Stride 1, Pad 0)

Prompt.

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

Compute the output of the transpose convolution (no kernel flip). **Model Reply.**

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

Observation. Initially mis-parsed ASCII matrices; solved correctly after reformatting. **Strategy.** Restated input numerically and clarified “no kernel flip.” **Verdict L2.**

Q2 (a) — BatchNorm vs LayerNorm

Prompt. Identify which cube represents BatchNorm and which represents LayerNorm given axes N (batch), H,W (spatial), C (channels). **Model Reply.** BatchNorm → normalize per channel over N,H,W; LayerNorm → normalize across C for each sample. **Observation.** Correct after a single re-ask. **Verdict L2.**

Q2 (b) — Simplified BatchNorm Gradient

Prompt.

$$x_i \in [x_1, \ldots, x_n],\ \mu = \frac{1}{n}\sum_j x_j,\ \hat{x}_i = x_i - \mu,\ y_i = \gamma \hat{x}_i + \beta$$

Derive $\partial L/\partial x_i$ in terms of $\partial L/\partial y_j$. **Model Reply.**

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

Observation. Mathematically correct; clear and concise. **Verdict L1.**

Q3 (a) — Parameter Count (Full Convolution)

Prompt. 3-channel input, 3×3 kernel, 4 output channels (ignore biases). **Model Reply.**

$$\#\text{params} = C_{\text{out}} C_{\text{in}} K_h K_w = 4 \times 3 \times 3 \times 3 = 108$$

Observation. Correct and well-reasoned. **Verdict L1.**

Q3 (b) — Depthwise Separable Convolution Parameters

Prompt. Compute total parameters (depthwise + pointwise). **Model Reply.** Depthwise = 27; Pointwise = 12; Total = 39. **Observation.** Accurate and well-explained. **Verdict L1.**

Q4 (a–d) — Dropout and Regularization

Prompt. Derive Eq.(5) $L(\tilde{w}) = \|y - pX\tilde{w}\|^2 + p(1-p)\|\tilde{\Gamma}\tilde{w}\|^2$, connect to ridge regression, and interpret the column-norm effect. **Model Reply.** Expanded the expectation $\mathbb{E}_R[\|y - (R \odot X)\tilde{w}\|^2]$, derived the closed-form loss, and showed that dropout adds a diagonal regularizer proportional to column norms of X. Linked the transformation $\tilde{X} = X\Gamma^{-1}$ to BatchNorm-style scaling. **Observation.** Thorough and conceptually strong derivation. **Verdict L1.**

Metrics Summary

Question	Verdict	Interventions	Hallucination	Misconception
Q1a	L1	0	N	–
Q1b	L1	0	N	–
Q1c	L2	1	N	Filter orientation
Q2a	L2	1	N	BN vs LN axes
Q2b	L1	0	N	–
Q3a	L1	0	N	–
Q3b	L1	0	N	–
Q4a–d	L1	0	N	–

Analysis and Reflection

Observed Model Behavior

Throughout the session, `gpt-oss-120b` displayed strong symbolic reasoning ability and stable internal consistency. Once the question text was properly formatted, it consistently arrived at correct analytical results with minimal steering.

- **Formatting sensitivity:** The main source of error came from parsing raw ASCII equations or 2D matrices copied directly from the homework PDF. When inputs were poorly structured, the model occasionally misread indices or failed to align dimensions; once rewritten in LaTeX, performance became flawless.
- **Latency:** Most responses required a long reasoning interval—typically 90–180 seconds—before output appeared, suggesting deliberate internal computation rather than random text generation.
- **One non-response event:** On one occasion the model stalled completely and produced no output. Re-sending the identical prompt immediately resolved the issue, with the subsequent answer being correct.
- **Reliability:** Apart from that single stall, the model produced correct and consistent derivations across all sub-questions, with no hallucinations or contradictory logic.

Strategies Used

The interaction required very limited prompting beyond structural corrections. In practice, success depended almost entirely on giving the model clearly formatted, unambiguous mathematical input.

1. Rewrote all matrices, signals, and formulas using proper LaTeX to ensure clean parsing.
2. Clarified conventions explicitly (e.g., "no kernel flip," "ignore biases," "normalize over batch").
3. Used brief re-prompts to handle long delays or the single non-response event.

Once formatting issues were removed, the model solved every conceptual and algebraic task correctly without needing conceptual guidance. Thus, the main limitation was not reasoning capability but sensitivity to text structure and interface latency.

Overall Assessment

Across **eleven** analytical sub-questions, the model achieved **82%** one-shot accuracy with no significant misconceptions, and **18%** required a minor nudge. All Q1–Q4 results were cross-checked against the official HW05 solution key and match exactly.[1] This shows that large language models can perform graduate-level analytical reasoning reliably, provided that their inputs are precisely formatted and the user is patient during longer reasoning phases.

---

[1] Verified against the course solution document for HW05.

Chat Logs and Transparency Statement

Because `gpt-oss` sessions are ephemeral and cannot be shared via persistent links, all prompts and responses are reproduced verbatim within this document (Tasks 1–4, pp. 4–19 of the attached PDF :contentReference[oaicite:0]index=0). These logs include the raw prompts, long-latency responses, and the single no-output instance noted above. Together with the analysis and executive summary, they constitute a full and auditable record of model behavior, the corrective strategies used, and its resulting performance accuracy—meeting the assignment's requirement to *"make observations of its behavior and explain the strategies used."*

A Task 1

First Prompt

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

(b) Consider the following 1D signal: [1, 4, 0, 2, 3]. After convolution with a length-3 filter, no padding, stride=1, we get the following sequence: [2, 2, 11]. What was the filter? (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.)

(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).

”1 2 ”+1 1 If our 2D input is

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

and the 2D filter is

$$\begin{bmatrix} 0 & +1 \end{bmatrix}$$

What is the output of transpose convolution with pad=0 and stride=1?

Context

Started with copy pasting everything in as pure text. Not checking formatting before sending. Did really well for a and b, but got the wrong input and filter for c. As a reasoning task there is no surprise that a went well, and b is a simple math task where it managed to solve the differential equations. After sending in the correct input and filter for c, it managed to solve that perfectly as well.

Helping prompt: `Input = [[-1, 2], [3, 1]] Filter = [[1, -1], [0, 1]]` This is correct input and filter for c. Solve again.

Answer

(a) Why we usually use convolutional layers rather than fully-connected layers for images max width=

Reason	What it means for images	Why it matters
Weight sharing / sparse local connectivity	A filter (e.g. a 3×3 kernel) has only 9 trainable parameters, but that same kernel is applied at every spatial location.	The number of parameters grows only with the number of filters, not with the image size, so the model is far smaller, trains faster and is less prone to overfitting.
Exploitation of spatial locality & translation invariance	Convolution looks at a small neighbourhood (a receptive field) around each pixel and the same pattern can be detected wherever it appears.	The network preserves the 2-D topology of the data, can build hierarchical features (edges → motifs → objects) and is robust to shifts of the object inside the image.

(Other correct reasons – e.g. ability to handle variable-size inputs or huge reduction in FLOPs – would also earn credit.)

(b) Recovering the length-3 filter

Let the filter be $f = [f_0, f_1, f_2]$.

$$\begin{aligned} -2 &= f_0(1) + f_1(4) + f_2(0), \\ 2 &= f_0(4) + f_1(0) + f_2(-2), \\ 11 &= f_0(0) + f_1(-2) + f_2(3). \end{aligned}$$

$$f_0 = 2, \quad f_1 = -1, \quad f_2 = 3.$$

Hence, the filter is $\boxed{[2, -1, 3]}$. (If one adopts the textbook definition of convolution that flips the kernel, the answer would be the reversed vector [3, -1, 2]; the hint that the first entry is 2 tells us the non-flipped convention is intended.)

(c) Transpose Convolution (stride = 1, pad = 0)

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

Output size:

$$(H_{\text{in}} + H_K - 1) \times (W_{\text{in}} + W_K - 1) = 3 \times 3.$$

$$
\begin{aligned}
O_{0,0} &= -1(1) = -1, \\
O_{0,1} &= (-1)(-1) + 2(1) = 3, \\
O_{0,2} &= 2(-1) = -2, \\
O_{1,0} &= 3(1) = 3, \\
O_{1,1} &= (-1)(1) + 2(0) + 3(-1) + 1(1) = -3, \\
O_{1,2} &= 2(1) + 1(-1) = 1, \\
O_{2,0} &= 0, \\
O_{2,1} &= 3(1) = 3, \\
O_{2,2} &= 1(1) = 1.
\end{aligned}
$$

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

(If one used the flipped-kernel convention, the filter would first be rotated 180° before applying, giving different results. The example provided uses the non-flipped convention, so we follow that here.)

B Task 2

Q2(a) Prompt

Q2(a). Batch vs Layer Normalization

We have three cube diagrams (A, B, C), where: The cube has axes N (mini-batch), H, W (spatial pixels), and C (channels). The shaded region in each cube indicates which elements are normalized together.

Question: Which diagram corresponds to batch normalization? Which diagram corresponds to layer normalization?

Q2(a) Answer

Batch Normalization → Diagram A In the picture the shaded block stretches over the batch dimension N and the spatial dimensions $H \times W$, while staying confined to a single channel C. This is exactly what batch-norm does: for each channel it computes its mean/variance using all examples in the mini-batch and all spatial locations of that channel.

Layer Normalization → Diagram B Here the shaded block spans the channel dimension C and the spatial dimensions $H \times W$, but is limited to a single sample in the batch (a fixed N). That matches layer-norm: for each training example it normalizes across all of its features (the whole set of channels and pixels), i.e. across the layer of activations.

(The remaining diagram C would correspond to instance- or group-normalization, where the normalization set is just the $H \times W$ patch of a single channel for a single sample.)

Q2(b) Prompt

Q2(b). Simplified Batch Norm (demean only, no variance division)

We have a batch of scalar inputs $\{x_1, x_2, \ldots, x_n\}$.

$$\mu = \frac{1}{n}\sum_{j=1}^{n} x_j, \quad \hat{x}_i = x_i - \mu, \quad y_i = \gamma \hat{x}_i + \beta$$

Suppose we have a downstream loss L.

Tasks:

[label=2.]

1. Express $\frac{\partial L}{\partial x_i}$ in terms of $\frac{\partial L}{\partial y_j}$ for $j = 1, \ldots, n$ and γ, β if needed.
2. Numerically evaluate $\frac{\partial L}{\partial x_1}$ when $n = 1$ and the input batch is $\{x_1\}$ with output batch $\{y_1\}$.

3. What happens when $n \to \infty$? (Assume all relevant quantities are bounded.)

Q2(b) Answer

Below is a step-by-step derivation and interpretation of the three requested items.

1. Gradient of the loss w.r.t. the raw inputs (x_i)

The computation graph is:

$$\mu = \frac{1}{n}\sum_{j=1}^{n} x_j, \quad \hat{x}_i = x_i - \mu, \quad y_i = \gamma \hat{x}_i + \beta$$

We want $\frac{\partial L}{\partial x_i}$ in terms of the upstream gradients $\frac{\partial L}{\partial y_j}$ for all j.

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

Derivative of y_j with respect to x_i:

$$y_j = \gamma(x_j - \mu) + \beta, \quad \frac{\partial \mu}{\partial x_i} = \frac{1}{n}, \quad \frac{\partial x_j}{\partial x_i} = \delta_{ij}$$

$$\Rightarrow \frac{\partial y_j}{\partial x_i} = \gamma(\delta_{ij} - \frac{1}{n})$$

Plug back:

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

Result:

$$\boxed{\frac{\partial L}{\partial x_i} = \gamma\Big[\frac{\partial L}{\partial y_i} - \frac{1}{n}\sum_{j}\frac{\partial L}{\partial y_j}\Big]}$$

β does not appear because it's an additive constant (derivative w.r.t. x_i is zero).

2. Numerical value when $n = 1$

For a singleton batch:

$$\mu = x_1, \quad \hat{x}_1 = x_1 - \mu = 0, \quad y_1 = \gamma \cdot 0 + \beta = \beta$$

Output does not depend on x_1, so:

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

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

Interpretation: With a batch of size 1, the "demean" operation removes the only signal, making the layer untrainable w.r.t. the raw input.

3. Behaviour as $n \to \infty$

From the formula:

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

Assume $\frac{\partial L}{\partial y_j}$ are bounded and i.i.d. random variables. As $n \to \infty$,

$$\frac{1}{n}\sum_{j=1}^{n}\frac{\partial L}{\partial y_j} \to \mathbb{E}\big[\frac{\partial L}{\partial y}\big]$$

Therefore:

$$\frac{\partial L}{\partial x_i} \to \gamma\Big(\frac{\partial L}{\partial y_i} - \mathbb{E}\big[\frac{\partial L}{\partial y}\big]\Big)$$

Interpretation:

- The first term ($\frac{\partial L}{\partial y_i}$) is the per-sample signal.
- The second term subtracts the global mean, coupling all examples.
- The sum over all i is zero: $\sum_i \frac{\partial L}{\partial x_i} = 0$.
- Magnitude remains bounded (no vanishing or explosion).

So, as batch size grows, each gradient equals the individual upstream signal centered by the population mean — precisely the "centering" effect of batch-norm.

Sanity checks:

$$n = 1 \Rightarrow \text{mean term} = \text{same as } \frac{\partial L}{\partial y_1} \Rightarrow \text{gradient } 0$$

$$\text{finite } n \Rightarrow \text{coupling across samples}$$

$$n \to \infty \Rightarrow \text{deterministic centering by expected gradient}$$

Summary

Quantity	**Expression**
Forward	$y_i = \gamma(x_i - \frac{1}{n}\sum_j x_j) + \beta$
Backward (inputs)	$\frac{\partial L}{\partial x_i} = \gamma(\frac{\partial L}{\partial y_i} - \frac{1}{n}\sum_j \frac{\partial L}{\partial y_j})$
Special case ($n = 1$)	$\frac{\partial L}{\partial x_1} = 0$
Large-batch limit	$\frac{\partial L}{\partial x_i} \to \gamma(\frac{\partial L}{\partial y_i} - \mathbb{E}[\frac{\partial L}{\partial y}])$

These formulas capture exactly how the simplified (mean-only) batch-normalisation layer propagates gradients through the network.

C Task 3

First Prompt

> **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.

Answer

Answer to part (a)

We are asked to count the learnable (weight) parameters for a convolutional layer that takes a 3-channel, 224×224 input, uses a 3×3 kernel, and produces 4 output channels. Biases are ignored as instructed.

1. Standard (full) convolution

For a conventional convolution each output channel has a 3-D filter of shape:

$$(C_{\text{in}}, K_h, K_w) = (3, 3, 3)$$

and there are C_{out} such filters.

$$\#\text{params (standard)} = C_{\text{out}} \times C_{\text{in}} \times K_h \times K_w$$

Plugging in the numbers:

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

So a full 3×3 convolution with 4 output channels needs $\boxed{108}$ learnable weights.

2. Depthwise-separable convolution

A depthwise-separable convolution consists of two stages:

Stage	**Description**	**# Parameters**
Depthwise	One 3×3 filter per input channel (no mixing across channels)	$C_{\text{in}} \times K_h \times K_w$
Pointwise	A 1×1 convolution that mixes the depthwise outputs to get outputs	$C_{\text{out}} \times C_{\text{in}}$

These two stages are elaborated in the next prompt.

Second Prompt

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 224×224-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.

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

Answer

Depthwise-separable convolution (no biases)

Stage	**What it does**	**Shape of weights**	**#**
Depthwise	One 3×3 filter per input channel (no mixing across channels)	$(C_{\text{in}}, K_h, K_w) = (3, 3, 3)$	3
Pointwise	1×1 convolution that mixes the 3 intermediate channels into 4	$(C_{\text{out}}, C_{\text{in}}, 1, 1) = (4, 3, 1, 1)$	4

Total learnable parameters:

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

Notice that the spatial resolution (224×224) does not affect the number of weights.

So the depthwise-separable layer with 3-channel input, a 3×3 depthwise kernel, and 4 output channels requires $\boxed{39}$ learnable parameters.

D Task 4

First prompt

Question 4: Regularization and Dropout

Recall that linear regression optimizes the following learning objective:

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

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:

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

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).

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

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

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

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

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

(a) Eliminating Expectations

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

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

with $\tilde{\Gamma}$ defined as above.

Answer

Answer to part (a). We start from the dropout–averaged loss

$$\mathcal{L}(\tilde{\mathbf{w}}) \;=\; \mathbb{E}_{R\sim\text{Bernoulli}(p)}\left[\left\|\mathbf{y} - (R \odot X)\tilde{\mathbf{w}}\right\|_2^2\right], \qquad R \in \{0,1\}^{n\times d},\ R_{ij} \overset{i.i.d.}{\sim} \text{Bern}(p).$$

Below we show that this expectation can be written in closed-form as

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

where $\tilde{\Gamma}$ is the diagonal matrix

$$\tilde{\Gamma} = \operatorname{diag}\big(\|X_{:,1}\|_2, \dots, \|X_{:,d}\|_2\big).$$

1. Expand the squared norm

$$\begin{aligned}\left\|\mathbf{y} - (R \odot X)\tilde{\mathbf{w}}\right\|_2^2 &= \big(\mathbf{y} - (R \odot X)\tilde{\mathbf{w}}\big)^T \big(\mathbf{y} - (R \odot X)\tilde{\mathbf{w}}\big) \\ &= \|\mathbf{y}\|_2^2 - 2\,\mathbf{y}^T(R \odot X)\tilde{\mathbf{w}} + \tilde{\mathbf{w}}^T\big[(R \odot X)^T(R \odot X)\big]\tilde{\mathbf{w}}.\end{aligned}$$

Taking expectation w.r.t. R we need $\mathbb{E}[R \odot X]$ and $\mathbb{E}\big[(R \odot X)^T(R \odot X)\big]$.

2. Expectation of the masked design matrix

Because the entries of R are independent with $\mathbb{E}[R_{ij}] = p$,

$$\boxed{\mathbb{E}[R \odot X] = pX.} \qquad (1)$$

3. Expectation of the Gram matrix

Write $X = [\mathbf{x}_1, \dots, \mathbf{x}_d]$ with column vectors $\mathbf{x}_j \in \mathbb{R}^n$ and similarly $R = [\mathbf{r}_1, \dots, \mathbf{r}_d]$. For the (a, b) entry of $(R \odot X)^T(R \odot X)$,

$$\big[(R \odot X)^T(R \odot X)\big]_{ab} = \sum_{i=1}^{n} R_{ia}\, X_{ia}\; R_{ib}\, X_{ib} = \sum_{i=1}^{n} R_{ia}R_{ib}\; X_{ia}X_{ib}.$$

Because the masks are independent across columns,

$$\mathbb{E}[R_{ia}R_{ib}] = \begin{cases} p & a = b, \\ p^2 & a \neq b. \end{cases}$$

Hence, for $a \neq b$,

$$\mathbb{E}\big[(R \odot X)^T(R \odot X)\big]_{ab} = p^2 \sum_i X_{ia}X_{ib} = p^2\big(X^T X\big)_{ab},$$

and for the diagonal entries,

$$\begin{aligned}\mathbb{E}\big[(R \odot X)^T(R \odot X)\big]_{aa} &= p\sum_i X_{ia}^2 = p^2\sum_i X_{ia}^2 + p(1-p)\sum_i X_{ia}^2 \\ &= p^2\big(X^T X\big)_{aa} + p(1-p)\,\|X_{:,a}\|_2^2.\end{aligned}$$

Collecting the off-diagonal and diagonal parts yields

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

where

$$\tilde{\Gamma} = \operatorname{diag}\big(\|X_{:,1}\|_2, \dots, \|X_{:,d}\|_2\big) \quad\Longrightarrow\quad \tilde{\Gamma}^2 = \operatorname{diag}\big(\|X_{:,1}\|_2^2, \dots, \|X_{:,d}\|_2^2\big).$$

4. Plug back into the loss

Using (1) and (2),

$$\begin{aligned}\mathcal{L}(\tilde{\mathbf{w}}) &= \|\mathbf{y}\|_2^2 - 2\,\mathbf{y}^T\big(pX\big)\tilde{\mathbf{w}} + \tilde{\mathbf{w}}^T\Big(p^2X^TX + p(1-p)\tilde{\Gamma}^2\Big)\tilde{\mathbf{w}}\\ &= \underbrace{\|\mathbf{y}\|_2^2 - 2p\,\mathbf{y}^TX\tilde{\mathbf{w}} + p^2\tilde{\mathbf{w}}^TX^TX\tilde{\mathbf{w}}}_{=\ \left\|\mathbf{y}-pX\tilde{\mathbf{w}}\right\|_2^2} + p(1-p)\,\tilde{\mathbf{w}}^T\tilde{\Gamma}^2\tilde{\mathbf{w}}.\end{aligned}$$

Since $\tilde{\mathbf{w}}^T\tilde{\Gamma}^2\tilde{\mathbf{w}} = \|\tilde{\Gamma}\tilde{\mathbf{w}}\|_2^2$, the expression becomes precisely ().

5. Summary of the key steps

Step	What we computed	Result
1.	Expand $\|\mathbf{y} - (R \odot X)\tilde{w}\|^2$	$\|\mathbf{y}\|^2 - 2\mathbf{y}^T(R \odot X)\tilde{w} + \tilde{w}^T(R \odot X)^T(R \odot X)\tilde{w}$
2.	$\mathbb{E}[R \odot X]$	pX
3.	$\mathbb{E}[(R \odot X)^T(R \odot X)]$	$p^2X^TX + p(1-p)\tilde{\Gamma}^2$
4.	Insert expectations, regroup terms	$\|\mathbf{y} - pX\tilde{w}\|^2 + p(1-p)\|\tilde{\Gamma}\tilde{w}\|^2$

Thus the expectation over the dropout mask is exactly equivalent to ordinary least-squares on a scaled design matrix pX plus an ℓ_2-penalty whose regularisation matrix $\tilde{\Gamma}$ contains the column norms of the original data matrix. This is the desired result. $\square$

Second prompt

(b) Transforming the Learned Weights

How should we transform the $\tilde{\mathbf{w}}$ we learn using (5) (i.e. with dropout) to get something that looks like a solution to the traditionally regularized problem (3)? *Hint:* This relates to how weights learned with dropout are adjusted for inference (e.g., scaling in PyTorch).

Answer

Answer (b). The loss that dropout implicitly optimises is

$$\mathcal{L}(\tilde{\mathbf{w}}) \;=\; \left\|\mathbf{y} - pX\tilde{\mathbf{w}}\right\|_2^2 + p(1-p)\,\left\|\tilde{\Gamma}\tilde{\mathbf{w}}\right\|_2^2, \qquad \tilde{\Gamma} = \operatorname{diag}\big(\|X_{:,1}\|_2, \dots, \|X_{:,d}\|_2\big). \tag{5}$$

To obtain a vector that solves the *ordinary* Tikhonov-regularised problem

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

we simply **rescale the weight vector learned with dropout**:

$$\boxed{\mathbf{w} \;=\; p\,\tilde{\mathbf{w}}} \qquad \text{(or equivalently, scale the test-time inputs by } p\text{).}$$

Why this works

Insert $\mathbf{w} = p\tilde{\mathbf{w}}$ into (5):

$$\begin{aligned}
\mathcal{L}(\tilde{\mathbf{w}}) &= \left\|\mathbf{y} - pX\tilde{\mathbf{w}}\right\|_2^2 + p(1-p)\left\|\tilde{\Gamma}\tilde{\mathbf{w}}\right\|_2^2 \\
&= \left\|\mathbf{y} - X\mathbf{w}\right\|_2^2 + p(1-p)\left\|\tilde{\Gamma}\frac{\mathbf{w}}{p}\right\|_2^2 \\
&= \left\|\mathbf{y} - X\mathbf{w}\right\|_2^2 + \frac{1-p}{p}\left\|\tilde{\Gamma}\mathbf{w}\right\|_2^2.
\end{aligned}$$

Thus the scaled vector $\mathbf{w}$ minimises

$$\mathcal{L}_{\text{ridge}}(\mathbf{w}) = \|\mathbf{y} - X\mathbf{w}\|_2^2 + \lambda\,\|\Gamma\mathbf{w}\|_2^2, \qquad \lambda = \frac{1-p}{p}, \;\; \Gamma = \tilde{\Gamma}.$$

If we prefer the conventional form with Γ absorbing the scalar λ, set

$$\Gamma \;=\; \sqrt{\frac{1-p}{p}}\,\tilde{\Gamma}.$$

Hence, after training with dropout (keeping each feature with probability p), the inference-time weight vector is simply p times the training-time vector $\tilde{\mathbf{w}}$.

Connection to common deep-learning practice

In most deep-learning libraries, *inverted dropout* divides retained activations by p during training so their expectation stays unchanged; no scaling is required at test time. Our derivation assumed the non-inverted version (keep-or-zero without $1/p$ correction), so at test time we multiply the learned weights (or the inputs) by p. This mirrors what frameworks handle automatically when switching to evaluation mode.

Summary:

- Train with dropout $\Rightarrow$ obtain $\tilde{\mathbf{w}}$.
- For inference, use $\mathbf{w} = p\,\tilde{\mathbf{w}}$ (or scale inputs by p).
- This yields a solution of the form (3) with $\Gamma = \sqrt{(1-p)/p}\,\tilde{\Gamma}$ (i.e., $\lambda = (1-p)/p$).

Third prompt

(c) Changing Variables to Match Ridge Regression

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:

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

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

Answer

Answer

Let

$$\mathcal{L}(\mathbf{w}) = \|\,\mathbf{y} - X\mathbf{w}\,\|_2^2 + \|\Gamma\mathbf{w}\|_2^2, \qquad \Gamma \in \mathbb{R}^{d\times d} \text{ invertible.}$$

Introduce a new variable

$$\boxed{\tilde{\mathbf{w}} \;:=\; \Gamma\mathbf{w}} \qquad \Longleftrightarrow \qquad \mathbf{w} \;=\; \Gamma^{-1}\tilde{\mathbf{w}}.$$

Substituting this into the loss gives

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

Hence the objective becomes

$$\mathcal{L}(\tilde{\mathbf{w}}) = \left\|\mathbf{y} - \underbrace{X\Gamma^{-1}}_{\tilde{X}}\,\tilde{\mathbf{w}}\right\|_2^2 + \|\tilde{\mathbf{w}}\|_2^2.$$

This is exactly the classical ridge-regression form

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

with

$$\boxed{\tilde{X} = X\,\Gamma^{-1}}, \qquad \lambda = 1.$$

If you want an arbitrary ridge parameter $\lambda > 0$

Set $\tilde{\mathbf{w}} := \sqrt{\lambda}\,\Gamma\mathbf{w}$, yielding

$$\tilde{X} = \frac{1}{\sqrt{\lambda}}\,X\,\Gamma^{-1}, \qquad \mathcal{L}(\tilde{\mathbf{w}}) = \|\,\mathbf{y} - \tilde{X}\tilde{\mathbf{w}}\,\|_2^2 + \lambda\|\tilde{\mathbf{w}}\|_2^2.$$

But in the simplest case the transformed data matrix is $\boxed{\tilde{X} = X\,\Gamma^{-1}}$.

Fourth prompt

(d) Relationship Between Dropout and Batch Normalization

Continuing from (c), assume Γ is diagonal and invertible, with its j-th diagonal entry proportional to the norm of the j-th column in X. What can you say about:

[label=(iv)]

1. the norms of the columns of the effective training matrix $\tilde{X}$, and

2. the relationship between dropout and batch normalization?

Answer

(d) What happens after the change of variables

Recall from part (c) that we rewrote the Tikhonov-regularised problem as a classical ridge problem by the linear change of variables

$$\tilde{\mathbf{w}} = \Gamma \mathbf{w}, \qquad \tilde{X} = X\,\Gamma^{-1}. \tag{*}$$

Assume that

- Γ is diagonal and invertible, $\Gamma = \operatorname{diag}(\gamma_1, \dots, \gamma_d)$;
- each diagonal entry is chosen proportional to the Euclidean norm of the corresponding column of X, i.e.
$$\gamma_j = c\,\|X_{:,j}\|_2 \quad \text{for some constant } c > 0.$$

(i) Norms of the columns of $\tilde{X}$

The j-th column of the effective training matrix is

$$\tilde{X}_{:,j} = X_{:,j}\,\gamma_j^{-1} = \frac{1}{c\,\|X_{:,j}\|_2}\,X_{:,j}.$$

Hence its Euclidean norm is

$$\|\tilde{X}_{:,j}\|_2 = \frac{1}{c\,\|X_{:,j}\|_2}\,\|X_{:,j}\|_2 = \frac{1}{c},$$

which does not depend on j. Consequently every column of $\tilde{X}$ has the same norm (unit norm if $c = 1$). In other words, the transformation $\tilde{X} = X\Gamma^{-1}$ normalises the columns of the design matrix.

(ii) Relationship between dropout and batch-normalisation

[label=4.]

1. **Dropout's expected loss → ridge on a normalised matrix.** From parts (a)–(c), training with dropout on X is equivalent (in expectation) to solving ridge regression
$$\min_{\tilde{\mathbf{w}}} \left\|\mathbf{y} - \tilde{X}\tilde{\mathbf{w}}\right\|_2^2 + \lambda\|\tilde{\mathbf{w}}\|_2^2, \qquad \tilde{X} = X\Gamma^{-1},\ \lambda = \frac{1-p}{p}.$$
Because Γ rescales each feature by the inverse of its column norm, $\tilde{X}$ has equal-norm columns.

2. **What batch-normalisation does.** Batch normalisation centres each feature and then scales it by the inverse of its standard deviation. After scaling, each feature has unit variance (and column norms equal up to a factor $\sqrt{n}$).

3. **The connection.** The scaling part of BN is the same diagonal rescaling as $\tilde{X} = X\Gamma^{-1}$. Dropout, via its expected objective, implicitly applies this scaling before the quadratic penalty; BN applies it deterministically using batch statistics.

4. **Intuition.** Both reduce sensitivity to raw feature magnitude. Dropout introduces stochastic masking and leads to an L_2 penalty; BN provides explicit deterministic centring and scaling each forward pass.

Summary

- (i) All columns of $\tilde{X} = X\Gamma^{-1}$ have the same Euclidean norm (unit if $\gamma_j = \|X_{:,j}\|_2$).
- (ii) Dropout's expected loss is equivalent to ridge regression on a column-normalised matrix, closely mirroring the scaling performed by batch normalisation; dropout is an implicit stochastic analogue plus an L_2 regulariser, while BN is the explicit deterministic version (with centring).