

CS 189/289A, Midterm

Fall 2026

Name: _____

Email: _____@berkeley.edu

Student ID: _____

Room where you're taking the exam: _____

Name and SID of the person on your right: _____

Name and SID of the person on your left: _____

Instructions:

This exam consists of **48 points** spread out over **7 questions**, plus **1 point** for the Honor Code certification. The exam must be completed in **80 minutes** unless you have accommodations supported by a DSP letter.

Note that you should **select one choice** for questions with **circular bubbles**. There is always at least one correct answer. Please **fully** shade in the circle to mark your answer. For all math questions, **please simplify your answer**. Please also **show your work** if a large box is provided. For all coding questions, you may use commas and/or one or more function calls in each blank.

You MUST write your Student ID number at the top of each page.

Honor Code [1 Pt]:

As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others. I am the person whose name is on the exam, and I completed this exam in accordance with the Honor Code.

Signature: _____

This page has been intentionally left blank.

1 Lasso's Evil Twin [8 points]

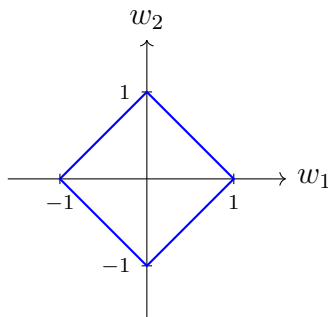
Throughout this question, $X \in \mathbb{R}^{N \times D}$ is a design matrix, $\mathbf{t} \in \mathbb{R}^N$ is a vector of targets, and $\lambda \geq 0$ is a regularization parameter. The *infinity norm* of a vector is its largest absolute entry,

$$\|\mathbf{w}\|_\infty = \max_i |w_i|.$$

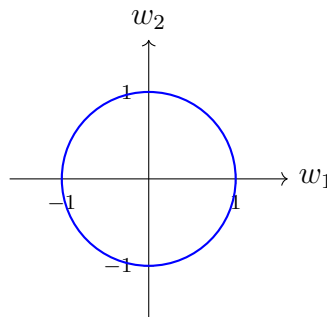
We study least squares with an infinity norm penalty,

$$\mathbf{w}_\lambda = \operatorname{argmin}_{\mathbf{w} \in \mathbb{R}^D} \frac{1}{2} \|X\mathbf{w} - \mathbf{t}\|_2^2 + \lambda \|\mathbf{w}\|_\infty. \quad (1)$$

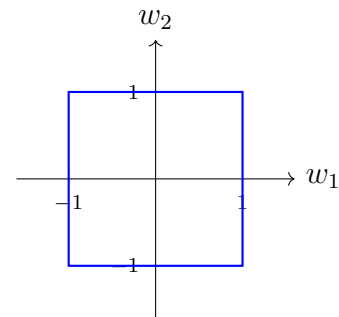
(a) [3 Pts] Sketch the boundary of each inequality in $\mathbb{R}^2$ on the axes provided.



(i) $\|\mathbf{w}\|_1 \leq 1$



(ii) $\|\mathbf{w}\|_2 \leq 1$



(iii) $\|\mathbf{w}\|_\infty \leq 1$

Explanation: (i) A diamond with corners at $(\pm 1, 0)$ and $(0, \pm 1)$. (ii) The unit circle. (iii) The square with corners at $(\pm 1, \pm 1)$.

+1 pt: diamond for ℓ_1 .

+1 pt: circle for ℓ_2 .

+1 pt: square for ℓ_∞ .

- (b) The infinity-norm regularized model (1) was fit at several values of λ , and its mean squared error was measured on the training set and on a validation set.

λ	0	1	3	10	30
Training MSE	10.0	10.4	11.5	15.0	24.0
Validation MSE	14.8	13.9	12.6	13.5	20.1

- (i) [1 Pt] Which λ should be deployed?

☐ 0 ☐ 1 ☒ 3 ☐ 10 ☐ 30

Explanation: Lowest validation error. Training error is not the criterion, since it always favors the least regularized model.

+1 pt: correct answer.

- (ii) [1 Pt] Compared with the model at $\lambda = 3$, the model at $\lambda = 0$ has

☒ lower bias and higher variance. ☐ lower bias and lower variance.
☐ higher bias and lower variance. ☐ higher bias and higher variance.

Explanation: Less regularization fits the training data more closely (lower bias) and is more sensitive to the particular training sample (higher variance). The large train/validation gap at $\lambda = 0$ is the variance showing up.

+1 pt: correct answer.

- (c) [1 Pt] Let $\mathbf{w}^* = (X^\top X)^{-1} X^\top \mathbf{t}$ be the ordinary least squares solution. Suppose the noisy ground-truth labels we observe are generated by a fixed true weight vector $\mathbf{w}_{\text{true}}$ plus zero-mean noise, $\mathbf{t} = X\mathbf{w}_{\text{true}} + \epsilon$. If ϵ 's variance is larger, which property of the least squares solution $\mathbf{w}^*$ changes?

- ☐ its expected value ☐ both
☒ its variance ☐ neither

Explanation: $\mathbf{w}^* = (X^\top X)^{-1} X^\top (X\mathbf{w}_{\text{true}} + \epsilon) = \mathbf{w}_{\text{true}} + (X^\top X)^{-1} X^\top \epsilon$. The noise has mean zero, so $\mathbb{E}[\mathbf{w}^*] = \mathbf{w}_{\text{true}}$ at any noise level, while for noise with covariance $\sigma^2 I$, $\text{Cov}(\mathbf{w}^*) = \sigma^2 (X^\top X)^{-1}$ grows with σ^2 .

+1 pt: correct answer.

- (d) [1 Pt] Which penalty is not guaranteed to increase when the magnitude of an individual weight (an entry of $\mathbf{w}$) increases?

- ☐ ridge (ℓ_2) ☒ infinity norm (ℓ_∞)
☐ lasso (ℓ_1) ☐ none

Explanation: Ridge is $\sqrt{\sum_i w_i^2}$, so raising any $|w_i|$ raises it. Lasso is $\sum_i |w_i|$, so raising any $|w_i|$ raises it too. The infinity norm is $\max_i |w_i|$, so raising a coefficient that is not the largest, and that stays below the largest, leaves it unchanged.

+1 pt: correct answer.

- (e) [1 Pt] Which of the following are linear regression models? **Select all that apply.**

- ☒ $y = w_0 + w_1 x + w_2 x^2$
☒ $y = w_0 + w_1 \exp(-(x - 3)^2)$
☐ $y = \sigma(w_0 + w_1 x)$, with σ the logistic sigmoid
☐ $y = w_0 + w_1 \sin(w_2 x)$

Explanation: Linear in the parameters means each w_j multiplies a fixed function of x , however nonlinear that function is; polynomial and Gaussian basis functions both qualify. Wrapping the parameters in a sigmoid, or putting w_2 inside the sine, makes the prediction nonlinear in $\mathbf{w}$, so those two are not linear regression models.

+1 pt: exactly the first two options marked (all or nothing).

2 Baby Steps [8 points]

For a small regression problem, the training loss written out as a function of the weights $\mathbf{w}$ is

$$L(\mathbf{w}) = w_1^2 - 4w_1 + 4w_2^2 + 8w_2 + 8, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}.$$

We minimize L using gradient descent:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla L(\mathbf{w}^{(\tau)}).$$

Gradient descent starts from the initial weights and uses the learning rate

$$\mathbf{w}^{(0)} = \begin{bmatrix} 7 \\ 4 \end{bmatrix}, \quad \eta = \frac{1}{10}.$$

- (a) [2 Pts] Compute $\frac{\partial L}{\partial w_1}$ and $\frac{\partial L}{\partial w_2}$. Then write $\nabla L(\mathbf{w})$ as a column vector in terms of w_1 and w_2 . Do **not** plug in $\mathbf{w}^{(0)}$. Show your work in the box, then write your answer in the answer box.

Solution:

$$\frac{\partial L}{\partial w_1} = 2w_1 - 4, \quad \frac{\partial L}{\partial w_2} = 8w_2 + 8.$$

Stacking the partial derivatives into a gradient vector:

$$\nabla L(\mathbf{w}) = \begin{bmatrix} 2w_1 - 4 \\ 8w_2 + 8 \end{bmatrix}.$$

+1 pt: $\partial L / \partial w_1 = 2w_1 - 4$.

+1 pt: $\partial L / \partial w_2 = 8w_2 + 8$.

$$\nabla L(\mathbf{w}) = \begin{bmatrix} 2w_1 - 4 \\ 8w_2 + 8 \end{bmatrix}$$

- (b) [2 Pts] Regardless of your answer to (a), suppose that the gradient at the initial weights is

$$\nabla L(\mathbf{w}^{(0)}) = \begin{bmatrix} 10 \\ 40 \end{bmatrix}.$$

Compute $\mathbf{w}^{(1)}$, the weights after one step of gradient descent with $\eta = \frac{1}{10}$. Show your work in the box, then write each coordinate in its answer box.

Solution: One step of the update is

$$\mathbf{w}^{(1)} = \mathbf{w}^{(0)} - \eta \nabla L(\mathbf{w}^{(0)}) = \begin{bmatrix} 7 \\ 4 \end{bmatrix} - \frac{1}{10} \begin{bmatrix} 10 \\ 40 \end{bmatrix} = \begin{bmatrix} 6 \\ 0 \end{bmatrix}.$$

+1 pt: $w_1^{(1)} = 6$.

+1 pt: $w_2^{(1)} = 0$.

$$w_1^{(1)} =$$

6

$$w_2^{(1)} =$$

0

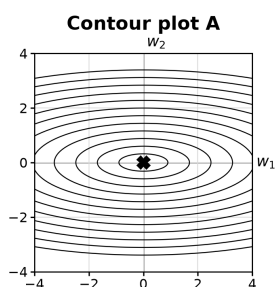
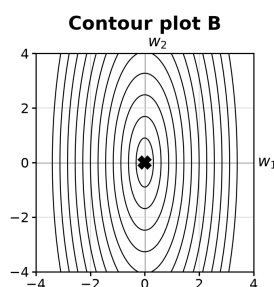
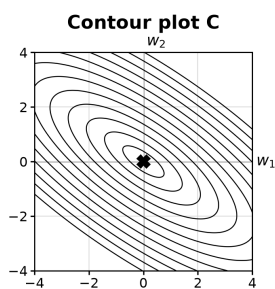
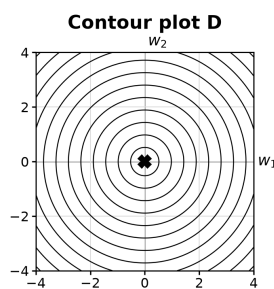
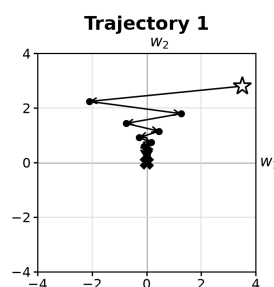
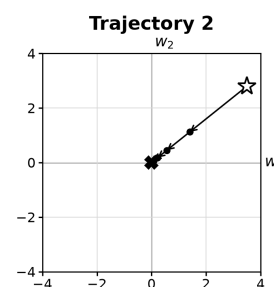
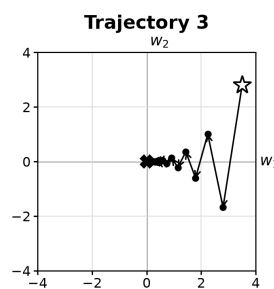
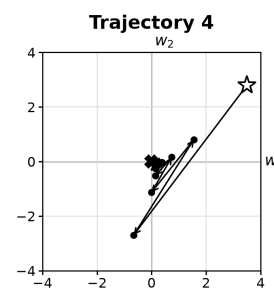
(c) [2 Pts] Contour plots A–D each show the isocontours of a quadratic loss of the form

$$L(\mathbf{w}) = \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^\top H(\mathbf{w} - \mathbf{w}^*)$$

for a different symmetric positive definite matrix H , where the cross marks the minimizer $\mathbf{w}^*$. Trajectories 1–4 show gradient descent run on these four losses, one trajectory per loss. All four runs use the same initial point (the star), minimizer (the cross), and learning rate:

$$\mathbf{w}^{(0)} = \begin{bmatrix} 3.5 \\ 2.8 \end{bmatrix}, \quad \mathbf{w}^* = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \eta = 0.2.$$

Match each contour plot to the trajectory that gradient descent produces on its loss by writing the trajectory number (1, 2, 3, or 4) in each answer box. Each trajectory is used exactly once.



Contour plot A:

3

Contour plot B:

1

Contour plot C:

4

Contour plot D:

2

Solution: The correct matching is

A → 3, B → 1, C → 4, D → 2.

The eigenvectors of H are the axes of the elliptical contours. The short axis of an ellipse is the direction of greater curvature and larger eigenvalue; the long axis is the flatter direction

with the smaller eigenvalue. Along the eigenvector with eigenvalue λ_i , the error α_i evolves as

$$\alpha_i^{(\tau)} = (1 - \eta\lambda_i)^\tau \alpha_i^{(0)},$$

so a large $\eta\lambda_i$ makes the trajectory jump back and forth across the steep direction (and diverge if $\eta\lambda_i > 2$), while a small $\eta\lambda_i$ makes progress along the flat direction slow. In these figures plots A, B, and C have eigenvalues 1 and 8, and plot D has eigenvalue 3 in every direction, so with $\eta = 0.2$ the per-step factors are 0.8 along the long axis and -0.6 along the short axis for A, B, and C, and 0.4 in every direction for D.

- (A) Contour plot A is elongated horizontally, so w_1 is the flat direction and w_2 the steep one. The trajectory should zigzag vertically across the w_1 -axis while drifting slowly toward the minimizer horizontally. This is trajectory 3.
- (B) Contour plot B is elongated vertically, so w_2 is the flat direction and w_1 the steep one. The trajectory should zigzag horizontally across the w_2 -axis while drifting slowly down toward the minimizer. This is trajectory 1.
- (C) Contour plot C has rotated ellipses, so its eigenvectors are not aligned with the w_1 - and w_2 -axes. The trajectory zigzags across the short, high-curvature direction (up-right to down-left) while progressing along the long, low-curvature direction. This is trajectory 4.
- (D) Contour plot D has circular contours, so every direction has the same curvature and the same factor 0.4. The gradient always points straight at the minimizer, so the iterates stay on the line from $w^{(0)}$ to w^* . This is trajectory 2.

+2 pts: all four matches correct.

+1.5 pts: exactly three matches correct (possible only if a trajectory number is repeated).

+1 pt: exactly two matches correct.

+0.5 pts: exactly one match correct.

(d) In practice, plain gradient descent is replaced by one of its variants. For each description, fill in the bubble of the optimizer it describes.

- (i) [1 Pt] This optimizer keeps a single learning rate η for every coordinate, but replaces the gradient in the update by an exponentially weighted sum of the past gradients.

☒ momentum ☐ RMSProp ☐ Adam ☐ AdamW

Explanation: Momentum (lecture 11): $\Delta \mathbf{w}^{(\tau)} = -\eta \nabla E + \mu \Delta \mathbf{w}^{(\tau-1)}$, an exponentially weighted sum of past gradients with one learning rate for all coordinates. SGD changes which data the gradient is computed on, not how the step is taken. RMSProp, Adam, and AdamW scale each coordinate by its own running average of squared gradients, which is a per-coordinate learning rate.

+1 pt: correct answer.

- (ii) [1 Pt] This optimizer applies weight decay by shrinking the parameters directly, $\mathbf{w} \leftarrow \mathbf{w} - \eta \lambda \mathbf{w}$, as a step separate from its gradient-based update, rather than adding $\lambda \mathbf{w}$ to the gradient.

☐ momentum ☐ RMSProp ☐ Adam ☒ AdamW

Explanation: Decoupled weight decay is what defines AdamW (HW3): the decay step is applied to the parameters outside the Adam update. Adam with an ℓ_2 penalty would instead put $\lambda \mathbf{w}$ into the gradient, where it is rescaled by the second-moment estimate; the other optimizers have no built-in weight decay at all.

+1 pt: correct answer.

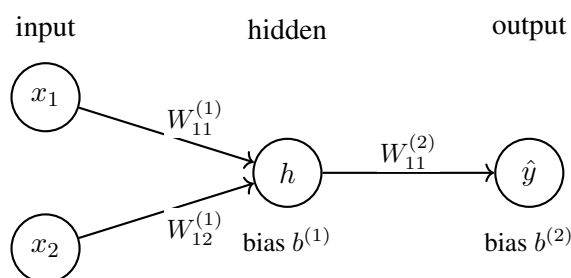
3 GooberNet: A 9-Parameter Model [8 points]

GooberNet is a small fully-connected neural network with one hidden layer. For an input $\mathbf{x} = [x_1 \ x_2]^\top$ it computes

$$\mathbf{z} = W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}, \quad \mathbf{h} = \phi(\mathbf{z}), \quad u = W^{(2)}\mathbf{h} + b^{(2)}, \quad \hat{y} = \sigma(u),$$

where the hidden activation ϕ is applied elementwise and is specified in each part, and $\sigma(t) = \frac{1}{1+e^{-t}}$ is the logistic sigmoid.

- (a) [2 Pts] Suppose the hidden layer has a single unit with the identity activation $\phi(t) = t$, so $W^{(1)} \in \mathbb{R}^{1 \times 2}$, $b^{(1)} \in \mathbb{R}$, $W^{(2)} \in \mathbb{R}^{1 \times 1}$, and $b^{(2)} \in \mathbb{R}$.



Let the inputs be bits, $x_1, x_2 \in \{0, 1\}$. Which parameter setting makes the network compute OR, i.e. $\hat{y} \geq \frac{1}{2}$ when at least one of x_1, x_2 is 1, and $\hat{y} < \frac{1}{2}$ when both are 0?

	$W_{11}^{(1)}$	$W_{12}^{(1)}$	$b^{(1)}$	$W_{11}^{(2)}$	$b^{(2)}$
☐	1	1	0	1	$-\frac{3}{2}$
☒	2	2	-1	1	0
☐	1	1	0	-1	$\frac{1}{2}$
☐	1	-1	0	1	$-\frac{1}{2}$

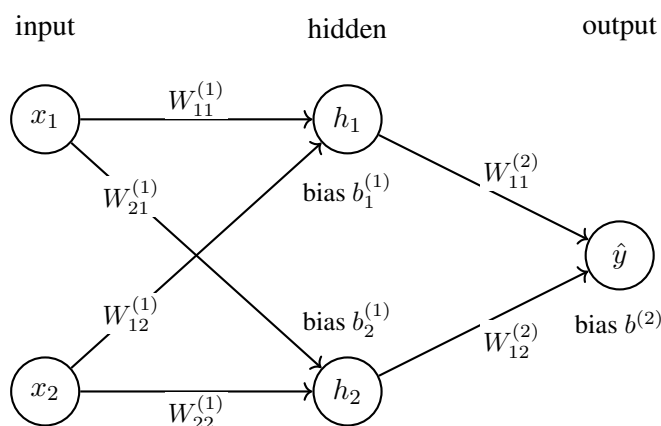
Explanation: Since $\hat{y} \geq \frac{1}{2}$ exactly when $u \geq 0$, check

$u = W_{11}^{(2)}(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + b^{(1)}) + b^{(2)}$ on the four inputs. The second setting gives $u = 2x_1 + 2x_2 - 1$: -1 on (0, 0) and at least 1 on the other three, so it computes OR. The first gives $u = x_1 + x_2 - \frac{3}{2}$, which is negative on (1, 0) and (0, 1) (AND). The third gives $u = -(x_1 + x_2) + \frac{1}{2}$, which is positive only on (0, 0) (NOR). The fourth gives $u = x_1 - x_2 - \frac{1}{2}$, which is negative on (0, 1) and (1, 1) (x_1 AND NOT x_2).

+2 pts: the second setting (all or nothing).

- (b) Now suppose the hidden layer has two units with the ReLU activation, $\phi(t) = \text{ReLU}(t) = \max(0, t)$, so the network computes

$$\mathbf{z} = W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}, \quad \mathbf{h} = \text{ReLU}(\mathbf{z}), \quad u = W^{(2)}\mathbf{h} + b^{(2)}, \quad \hat{y} = \sigma(u).$$



We train with the binary cross-entropy loss

$$L = -[y \log \hat{y} + (1 - y) \log(1 - \hat{y})], \quad \text{for which} \quad \frac{\partial L}{\partial u} = \hat{y} - y.$$

- (i) [1 Pt] Let $W_{11}^{(1)}$ be the weight from x_1 to h_1 . By the chain rule, $\frac{\partial L}{\partial W_{11}^{(1)}}$ equals which of the following?

- ☐ $(\hat{y} - y) \cdot W_{11}^{(2)} \cdot \text{ReLU}'(z_1) \cdot x_2$
☐ $(\hat{y} - y) \cdot W_{11}^{(2)} \cdot x_1$
☒ $(\hat{y} - y) \cdot W_{11}^{(2)} \cdot \text{ReLU}'(z_1) \cdot x_1$
☐ $(\hat{y} - y) \cdot \sigma'(u) \cdot W_{11}^{(2)} \cdot \text{ReLU}'(z_1) \cdot x_1$
☐ $(\hat{y} - y) \cdot h_1$

Explanation: The path from $W_{11}^{(1)}$ to L runs through z_1 , h_1 , and u , and the chain rule multiplies one factor per step:

$$\frac{\partial L}{\partial W_{11}^{(1)}} = \frac{\partial L}{\partial u} \cdot \frac{\partial u}{\partial h_1} \cdot \frac{\partial h_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial W_{11}^{(1)}} = (\hat{y} - y) \cdot W_{11}^{(2)} \cdot \text{ReLU}'(z_1) \cdot x_1, \text{ since}$$

$u = W_{11}^{(2)} h_1 + W_{12}^{(2)} h_2 + b^{(2)}$, $h_1 = \text{ReLU}(z_1)$, and $z_1 = W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + b_1^{(1)}$. The first option uses the wrong input ($W_{11}^{(1)}$ multiplies x_1 , not x_2), the second skips the derivative of the activation, the fourth multiplies by $\sigma'(u)$ a second time (that factor is already inside $\partial L / \partial u = \hat{y} - y$), and the last is the gradient with respect to the output weight $W_{11}^{(2)}$, one layer too late.

+1 pt: correct answer.

- (ii) [2 Pts] This part asks about a different weight from part (i): $W_{11}^{(2)}$, the weight from h_1 to the output $\hat{y}$. On one training example with target $y = 1$, the forward pass gives $\mathbf{h} = [1 \ 0]^T$ and $\hat{y} = \frac{1}{2}$. Compute $\frac{\partial L}{\partial W_{11}^{(2)}}$ on this example. Show your work in the box, then write your answer in the answer box.

Solution: $u = W_{11}^{(2)}h_1 + W_{12}^{(2)}h_2 + b^{(2)}$, so $\partial u / \partial W_{11}^{(2)} = h_1$ and

$$\frac{\partial L}{\partial W_{11}^{(2)}} = \frac{\partial L}{\partial u} \cdot h_1 = (\hat{y} - y) h_1 = \left(\frac{1}{2} - 1\right) \cdot 1 = -\frac{1}{2}.$$

+1 pt: chain rule $\partial L / \partial W_{11}^{(2)} = (\hat{y} - y) h_1$.
+1 pt: value $-\frac{1}{2}$.

$$\frac{\partial L}{\partial W_{11}^{(2)}} = -\frac{1}{2}$$

(c) Fill in exactly one bubble per item.

- (i) [1 Pt] The universal approximation theorem guarantees that if a one-hidden-layer network is sufficiently wide, gradient descent on the training loss will find parameters that approximate any continuous target function to any desired accuracy.

☐ True ☒ False

Explanation: False. The first half of the statement is what the theorem says: for any continuous target and any tolerance, a one-hidden-layer network with enough hidden units and a nonlinear activation can approximate the target within that tolerance. The false part is the conclusion that gradient descent on the training loss will find those parameters. The theorem only asserts that suitable parameters exist; it says nothing about any training procedure or how many hidden units are enough. Gradient descent can stop in a poor local minimum, and even driving the training loss to zero does not by itself make the network close to the target function away from the training points.

+1 pt: correct answer.

- (ii) [1 Pt] For a hidden unit with sigmoid activation $h = \sigma(z)$, if its pre-activation z is a large positive number, the gradient of the loss with respect to that unit's incoming weights is close to zero even when the loss itself is large.

☒ True ☐ False

Explanation: True. The chain rule through the unit multiplies by $\sigma'(z) = \sigma(z)(1 - \sigma(z))$, which is at most $\frac{1}{4}$ and tends to 0 as $z \rightarrow \pm\infty$: the unit saturates, and almost no error signal reaches its incoming weights, however wrong the prediction is. This is a main reason ReLU, whose derivative is 1 on the active side, is preferred for hidden units.

+1 pt: correct answer.

- (iii) [1 Pt] GooberNet is repurposed for regression: the target y is now a real number rather than a bit. The network currently uses the output layer $\hat{y} = \sigma(u)$ with the binary cross-entropy loss. Which pair of (output layer, loss) should replace it?

- ☒ output layer: $\hat{y} = u$; loss: $(\hat{y} - y)^2$ (squared error)
- ☐ output layer: $\hat{y} = \sigma(u)$; loss: $(\hat{y} - y)^2$ (squared error)
- ☐ output layer: $\hat{y} = \text{ReLU}(u)$; loss: binary cross-entropy
- ☐ output layer: $\hat{y} = \sigma(u)$; loss: binary cross-entropy (no change needed)

Explanation: A real-valued target needs an output that can take any real value, so the output activation is the identity and the loss is squared error. The sigmoid squashes the output into $(0, 1)$, so the second option cannot reach targets outside that interval however the weights are set. Binary cross-entropy is a likelihood for a label in $\{0, 1\}$ and takes $\log \hat{y}$ and $\log(1 - \hat{y})$, so the last two options are undefined for a general real y (and the ReLU output is not even in $(0, 1)$).

+1 pt: correct answer.

4 Triple T [4 points]

You're flipping a coin with your good friend Tung. For each tail, Tung gets a point, and for each head, you get a point. However, you observe that the first three flips are TTT — slightly suspicious!

- (a) [1 Pt] You begin by assuming that the outcome of each coin flip is independent, and follows a Bernoulli distribution with parameter $\theta \in [0, 1]$, where θ is the probability of a head.

Recall the distribution of a Bernoulli random variable, where $x = 1$ denotes a head and $x = 0$ a tail:

$$\text{Bern}(x \mid \theta) = \theta^x (1 - \theta)^{1-x}.$$

Which of the following optimization problems correctly formulates the search for the Maximum Likelihood Estimate $\hat{\theta}_{\text{MLE}}$ for the observed sequence TTT ?

- | | |
|---|---|
| ☐ $\arg \max_{\theta \in [0,1]} \left(3 \log(\theta) \right)$ | ☐ $\arg \max_{\theta \in [0,1]} \left(3 \log(1 - \theta) + \log p(\theta) \right)$ |
| ☒ $\arg \max_{\theta \in [0,1]} \left(3 \log(1 - \theta) \right)$ | ☐ $\arg \max_{\theta \in [0,1]} \left(3 \log(\theta(1 - \theta)) \right)$ |

Explanation: $p(TTT \mid \theta) = (1 - \theta)^3$, so the log-likelihood is $3 \log(1 - \theta)$ and the MLE maximizes it. Option 1 swaps heads and tails. Option 3 adds a prior, which is MAP. Option 4 is the log-likelihood of a sequence with three heads and three tails, not TTT ; its solution is $\theta = \frac{1}{2}$.

+1 pt: correct answer.

- (b) [3 Pts] You begin to suspect that Tung is stalling on purpose to throw you off your rhythm. You instead focus on how long Tung takes to flip the coin each round. You model his waiting time T (in seconds) as an Exponential random variable $T \sim \text{Exponential}(\lambda)$ with rate parameter $\lambda > 0$ and PDF:

$$f(t \mid \lambda) = \lambda e^{-\lambda t}, \quad t \geq 0$$

You record M independent waiting times $\mathcal{D} = \{t_1, t_2, \dots, t_M\}$.

Write down the log-likelihood function $\ell(\lambda) = \log p(\mathcal{D} \mid \lambda)$ in terms of M , λ , and the t_i . Your final answer should **not** contain $\prod$, e , or $\exp$. Show your work in the box, then write your answer in the answer box.

Solution:

$$p(\mathcal{D} \mid \lambda) = \prod_{i=1}^M f(t_i \mid \lambda) = \prod_{i=1}^M \lambda e^{-\lambda t_i} = \lambda^M \exp \left(-\lambda \sum_{i=1}^M t_i \right),$$

so

$$\ell(\lambda) = \log p(\mathcal{D} \mid \lambda) = M \log(\lambda) - \lambda \sum_{i=1}^M t_i.$$

+1 pt: likelihood as a product of the densities.

+1 pt: taking the log.

+1 pt: simplified final form (an equivalent form such as $\sum_i (\log \lambda - \lambda t_i)$ is fine).

An answer box containing $\prod$, e , or $\exp$ scores 0 for the part.

$\ell(\lambda) =$

$$M \log(\lambda) - \lambda \sum_{i=1}^M t_i$$

5 Set Your Boundaries [4 points]

Throughout this question, we consider binary logistic regression with classes C_0 and C_1 . For an input $\mathbf{x} \in \mathbb{R}^D$, the model predicts

$$P(C_1 | \mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}^\top \mathbf{x} + w_0), \quad \sigma(a) = \frac{1}{1 + e^{-a}}.$$

We predict C_1 whenever

$$P(C_1 | \mathbf{x}, \mathbf{w}) \geq T,$$

where $T \in (0, 1)$ is a classification threshold.

- (a) [2 Pts] Suppose C_0 denotes a *legitimate credit card transaction* and C_1 a *fraudulent one*, so fraudulent is the positive class. A false positive incurs a loss of 1 and a false negative a loss of 100. Let $p = P(C_1 | \mathbf{x}, \mathbf{w})$ be the model's probability that $\mathbf{x}$ is fraudulent.

Under the model's distribution, write the expected loss at $\mathbf{x}$ if we predict C_0 and if we predict C_1 , in terms of p . Show your work in the box, then write your answers in the answer boxes.

Solution: Under the model's distribution, $\mathbf{x}$ is fraudulent with probability p and legitimate with probability $1 - p$; the expected loss of a decision is the loss of each outcome weighted by its probability.

Predicting C_0 loses 100 only if the transaction is fraudulent (a false negative):

$$0 \cdot (1 - p) + 100p = \boxed{100p}.$$

Predicting C_1 loses 1 only if it is legitimate (a false positive):

$$1 \cdot (1 - p) + 0 \cdot p = \boxed{1 - p}.$$

+1 pt: $100p$ for predicting C_0 .

+1 pt: $1 - p$ for predicting C_1 .

If both boxes are wrong, 1 pt total when the work shows the expected loss of each decision as the loss weighted by the class probability with a single consistent error: the two decisions swapped ($1 - p$ and $100p$), or p and $1 - p$ interchanged ($100(1 - p)$ and p). 0 if the work does not show the weighting.

Predict C_0 :

$100p$

Predict C_1 :

$1 - p$

- (b) [1 Pt] Suppose we increase the probability threshold T used to convert the outputs of a fixed

logistic regression model into 0/1 predictions. We do not change the fitted model or the test data. As the threshold increases, recall

- ☐ can only decrease. ☐ can only increase or stay the same.
☒ can only decrease or stay the same. ☐ can move in either direction.

Explanation: Since

$$\text{recall} = \frac{TP}{TP + FN},$$

and the number of actual positives $TP + FN$ is fixed while TP can only decrease or stay the same (raising the threshold can only shrink the set of predicted positives), recall can only decrease or stay the same.

+1 pt: correct answer.

(c) [1 Pt] Which of the following metrics is not tied to one classification threshold?

- ☐ accuracy ☐ recall
☐ precision ☒ Area under the curve of ROC

Explanation: Accuracy, precision, and recall depend on a particular threshold. AUC-ROC summarizes the ROC curve obtained by sweeping over thresholds.

+1 pt: correct answer.

6 K-Nearest Players [10.5 points]

The 2026–27 NBA season tips off today! A pandas DataFrame `players` holds one row per NBA player with the columns `player`, `position` (one of "G", "F", "C" for guard, forward, center), `height_in` (inches), and `weight_lb` (pounds). Throughout this question, we try to predict a player's position from height and weight alone. Assume `numpy` is imported as `np`, `pandas` as `pd`, and that `train_test_split`, `StandardScaler`, and `KNeighborsClassifier` are imported from `scikit-learn`.

- (a) A KNN classifier has predicted the position of every player in `players`, and the predictions are stored in a new column of `players` called `predicted_position`. A few rows of `players` are shown below.

player	height_in	weight_lb	position	predicted_position
Stephen Curry	74	185	G	G
Brandin Podziemski	76	205	G	G
Moses Moody	77	211	G	G
Jimmy Butler III	78	230	F	F
Draymond Green	78	230	F	F
Al Horford	80	240	C	F
...	...	...	...	...

- (i) [2 Pts] Write a single Python expression that computes the accuracy of the predictions in `predicted_position` over all rows of `players`, using only `players` and `pandas` and/or `NumPy` operations. Do **not** use `len`. Write your answer in the box.

Solution: `(players["predicted_position"] == players["position"]).mean()`. The comparison produces a boolean Series, and the mean of a boolean Series is the fraction of True entries.

+2 pts: an expression that evaluates to the fraction of rows where `predicted_position` equals `position`.

+1 pt: an expression that becomes correct by changing, adding, or deleting exactly one token (one operator, one operand, or one method or attribute call), e.g. a missing `.mean()`, a sum without the division, a tuple `(matches, rows)` in place of the quotient, `.size` or `.shape[1]` as the denominator, a misspelled column name.

0 pts: uses `len`, needs two or more edits, or does not compare the two columns.

- (ii) [1 Pt] The classifier is a `KNeighborsClassifier` with $k = 50$, and `knn.classes_` is `["C", "F", "G"]` (center, forward, guard). For Al Horford's row, `knn.predict_proba` returns `[0.20, 0.74, 0.06]`. Write a single number in each answer box.

The classifier's confidence in this prediction

0.74

The number of Horford's 50 nearest training neighbors that are **guards** ("G")

3

Explanation: Confidence is the largest predicted probability, 0.74, which belongs to the predicted class F. KNN's probabilities are vote fractions among the k neighbors, so $0.06 \times 50 = 3$ of the 50 nearest neighbors are guards (37 are forwards and 10 are centers).

+0.5 pts: confidence 0.74.

+0.5 pts: 3 guards.

- (b) [3 Pts] Andrew wrote the pipeline below to choose k for a KNN position classifier and report its test accuracy. It runs without error and prints best $k = 1$, test accuracy = 0.786, and he is about to report that number. There are **three** major bugs. None of them is a syntax error, and each is fixed by rewriting a single line. For each bug, give the line number and the corrected version of that line.

```

1 # split into train, validation, and test
2 X = players[["height_in", "weight_lb"]].to_numpy()
3 y = players["position"].to_numpy()

4 split = train_test_split(X, y, test_size=0.2)
5 X_train, X_test, y_train, y_test = split

6 split = train_test_split(X, y, test_size=0.25)
7 X_train, X_val, y_train, y_val = split

8 # standardize
9 scaler = StandardScaler().fit(X)
10 X_train_sc = scaler.transform(X_train)
11 X_val_sc = scaler.transform(X_val)
12 X_test_sc = scaler.transform(X_test)

13 # choose k
14 best_k, best_acc = None, 0.0
15 for k in range(1, 101):
16     knn = KNeighborsClassifier(n_neighbors=k).fit(X_train_sc, y_train)
17     acc = knn.score(X_train_sc, y_train)
18     if acc > best_acc:
19         best_k, best_acc = k, acc

20 # final model
21 knn = KNeighborsClassifier(n_neighbors=best_k).fit(X_train_sc, y_train)
22 test_acc = knn.score(X_test_sc, y_test)
23 print(f"best k = {best_k}, test accuracy = {test_acc:.3f}")

```

Line #	Corrected line

Solution: Line 6: the second split draws from all of X instead of the training rows from line 5, so test players can land in the new training set and validation set. Validation rows should be pulled out of the training split only.

Corrected line: `split = train_test_split(X_train, y_train, test_size=0.25)`

Line 9: the scaler is fit on all of X , so the validation and test rows leak into the standardization statistics. Preprocessing should be fit on the training rows only.

Corrected line: `scaler = StandardScaler().fit(X_train)`

Also accepted: line 10 rewritten as `X_train_sc = scaler.fit_transform(X_train)`, which refits the scaler on the training rows and makes line 9's fit irrelevant. `fit_transform` on lines 11 or 12 is not accepted.

Line 17: k is chosen by training accuracy. Every training player is its own nearest neighbor, so $k = 1$ has the highest training accuracy of any k and the loop always picks it. Hyperparameters should be chosen on the validation split.

Corrected line: `acc = knn.score(X_val_sc, y_val)`

With all three fixes the same run prints `best k = 11, test accuracy = 0.821`.

Not accepted: changing the range of k , or adding `stratify` or `random_state`.

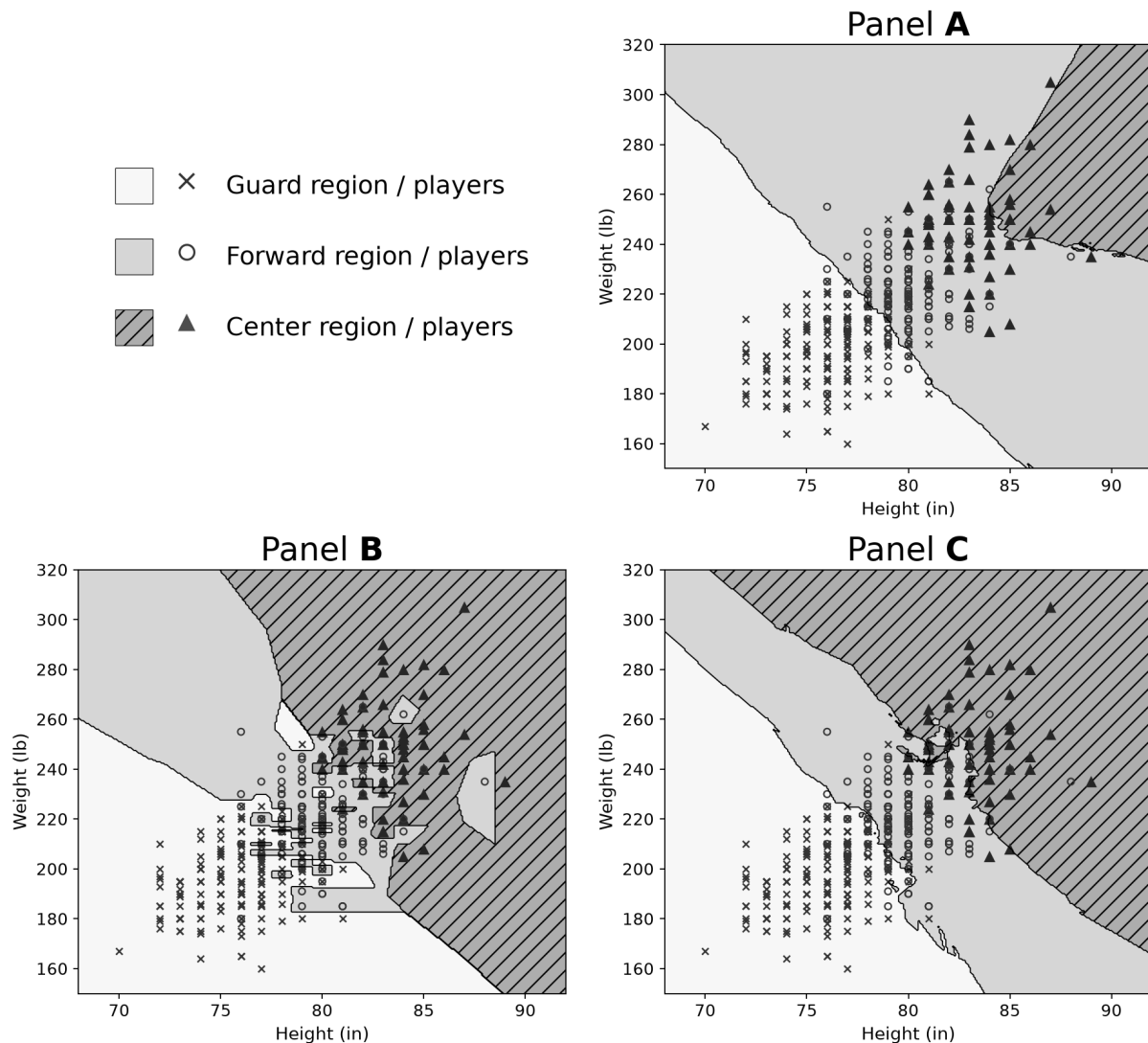
+1 pt: line 6 with its corrected line.

+1 pt: line 9 with its corrected line.

+1 pt: line 17 with its corrected line.

The three bugs may be listed in any order.

- (c) [1.5 Pts] The three panels below show the decision regions of KNN classifiers trained on standardized height and weight for 446 players, with $k \in \{1, 15, 130\}$ in some order. Each marker ($\times$, $\circ$, $\blacktriangle$) is one of these players; the legend in the top-left corner shows each position's marker and region shading.



Match each panel to its k by writing the panel letter (A, B, or C) in each answer box. Each letter is used exactly once.

$k = 1$:

B

$k = 15$:

C

$k = 130$:

A

Explanation: $k = 1$ is B, $k = 15$ is C, $k = 130$ is A. The two tells are the ones from lecture 2's penguin plots: small k emphasizes individual points while large k over-smooths. With

$k = 1$ every training player can create its own cell, so panel B becomes a patchwork of islands around individual players. With $k = 130$ more than a quarter of the 446 players vote on every prediction, so panel A has smooth boundaries and the centers, the least common position, are outvoted almost everywhere: only a sliver of center region survives in the top-right corner. $k = 15$ is in between: three clean bands with only a little raggedness where guards and forwards overlap.

+0.5 pts: $k = 1$ is B.

+0.5 pts: $k = 15$ is C.

+0.5 pts: $k = 130$ is A.

- (d) [1 Pt] A standardized KNN classifier was fit on 446 players and evaluated once on the 112 held-out test players. Rows are true positions, columns are predicted positions.

True position	G	48	4	0
	F	3	38	2
	C	0	5	12
		G	F	C
		Predicted position		

Write the precision and the recall of the classifier for the forward class (F), each as a fraction of two integers. You do **not** need to simplify the fractions.

Precision for F:

38/47

Recall for F:

38/43

Explanation: Precision is the fraction of predicted forwards who are forwards, the F column: $38/(4 + 38 + 5) = 38/47$. Recall is the fraction of true forwards that were predicted as forwards, the F row: $38/(3 + 38 + 2) = 38/43$.

+0.5 pts: precision 38/47.

+0.5 pts: recall 38/43.

- (e) Tall players tend to be heavy, so height and weight carry overlapping information. PCA on the standardized training data (z_{height} , z_{weight}) gives the principal components below.

	eigenvalue	unit eigenvector
first principal component	$\lambda_1 = 1.6$	$\mathbf{v}_1 = \frac{1}{\sqrt{2}}(1, 1)$
second principal component	$\lambda_2 = 0.4$	$\mathbf{v}_2 = \frac{1}{\sqrt{2}}(1, -1)$

- (i) [1 Pt] Two more players are listed below with their raw and standardized features. A player's coordinate along a principal component is the projection of their standardized features onto it. Which statement is correct?

	raw		standardized	
	height_in	weight_lb	z_{height}	z_{weight}
Nikola Jokić	83	284	1.3	2.7
Chet Holmgren	85	208	2.0	-0.3

- ☐ They have about the same coordinate along $\mathbf{v}_1$, since they are about the same height.
☐ Jokić's coordinate along $\mathbf{v}_2$ is larger than Holmgren's, since he is much heavier.
☒ Holmgren's coordinate along $\mathbf{v}_2$ is larger than Jokić's, since he is taller and lighter.
☐ Holmgren's coordinate along $\mathbf{v}_2$ is close to zero, since his standardized weight is close to zero.

Explanation: Along $\mathbf{v}_1 = \frac{1}{\sqrt{2}}(1, 1)$: Jokić $(1.3 + 2.7)/\sqrt{2} \approx 2.83$, Holmgren $(2.0 - 0.3)/\sqrt{2} \approx 1.20$.

Along $\mathbf{v}_2 = \frac{1}{\sqrt{2}}(1, -1)$: Jokić $(1.3 - 2.7)/\sqrt{2} \approx -0.99$, Holmgren $(2.0 + 0.3)/\sqrt{2} \approx 1.63$. $\mathbf{v}_1$ adds the two features, so it measures overall size; $\mathbf{v}_2$ subtracts weight from height, so it measures build, and Holmgren (taller and lighter) comes out higher without any arithmetic. The first option treats $\mathbf{v}_1$ as height, the second has the sign of $\mathbf{v}_2$'s weight entry backwards, and the fourth confuses a feature with a coordinate.

+1 pt: correct answer.

- (ii) [1 Pt] What fraction of the total variance is kept by projecting onto the first principal component $\mathbf{v}_1$? Give your answer as a fraction or decimal. You do **not** need to simplify it.

Explanation: The variance along a principal component is its eigenvalue (lecture 3: the loss along $\mathbf{w}$ works out to $-\lambda$), so the first component keeps 1.6 out of a total of $1.6 + 0.4 = 2$, which is 0.8.

+1 pt: 0.8 (accept 1.6/2, 4/5, or 80%).

Fraction of the total variance kept:

$1.6/2 = 0.8$

7 I Gauss We Are Parking Here? [5.5 points]

A wheeled robot with a fixed arm has to open an oven door. The arm only succeeds from certain base positions, so before reaching, the robot drives along a straight rail that runs parallel to the oven door and stops there (it *parks*). Erica models the good parking spots as a two-component Gaussian mixture over the position $x \in \mathbb{R}$ along the rail, in meters, with $x = 0$ directly in front of the oven (negative on one side, positive on the other).

Let $z \in \{1, 2\}$ be the latent (unobserved) mixture component, i.e. the cluster, that a parking spot comes from, and let $\theta = \{\pi_1, \pi_2, \mu_1, \mu_2\}$ be the mixture's parameters. Both components share a known variance σ^2 :

$$p(z = k) = \pi_k, \quad p(x \mid z = k) = \mathcal{N}(x; \mu_k, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x - \mu_k)^2}{2\sigma^2}\right).$$

- (a) [2 Pts] Find the posterior probability that a parking spot at position x came from component k , $p(z = k \mid x, \theta)$. You may leave Gaussian densities in $\mathcal{N}(\cdot; \cdot, \cdot)$ notation. You do **not** need to write out or simplify the density formula. Show your work in the box.

Solution: By Bayes' rule,

$$p(z = k \mid x, \theta) = \frac{p(x \mid z = k) p(z = k)}{p(x \mid \theta)} = \frac{\pi_k \mathcal{N}(x; \mu_k, \sigma^2)}{\pi_1 \mathcal{N}(x; \mu_1, \sigma^2) + \pi_2 \mathcal{N}(x; \mu_2, \sigma^2)}.$$

+0.5 pts: recognizing Bayes' rule.

+0.5 pts: prior π_k in the numerator.

+0.5 pts: component density $\mathcal{N}(x; \mu_k, \sigma^2)$ in the numerator.

+0.5 pts: denominator summed over both components.

A small slip within a piece loses 0.25 pts. Writing $1 - \pi_1$ for π_2 is fine.

- (b) [2.5 Pts] Erica fits θ from four logged positions at which the arm succeeded,

$$x_1 = -2, \quad x_2 = -1, \quad x_3 = 1, \quad x_4 = 2,$$

and runs EM with $\sigma^2 = 1$ held fixed, initialized at $\pi_1 = \pi_2 = \frac{1}{2}$, $\mu_1 = -0.5$, $\mu_2 = 0.5$.

Rounded to one decimal place, the first E-step gives the responsibilities

$$\gamma_{nk} = p(z_n = k \mid x_n, \theta)$$

$$\gamma_{11} = 0.9, \quad \gamma_{21} = 0.7, \quad \gamma_{31} = 0.3, \quad \gamma_{41} = 0.1$$

for component 1 (and $\gamma_{n2} = 1 - \gamma_{n1}$ for component 2). Use these rounded values. The M-step sets $N_k = \sum_n \gamma_{nk}$, $\pi_k = N_k/4$, and $\mu_k = \frac{1}{N_k} \sum_n \gamma_{nk} x_n$. Compute π_1 and μ_1 after this M-step. Show your work in the box, then write your answers in the answer boxes.

Solution:

$$N_1 = 0.9 + 0.7 + 0.3 + 0.1 = 2, \quad \pi_1 = \frac{2}{4} = 0.5,$$

unchanged, and

$$\begin{aligned} \mu_1 &= \frac{1}{2} [0.9(-2) + 0.7(-1) + 0.3(1) + 0.1(2)] \\ &= \frac{1}{2} (-1.8 - 0.7 + 0.3 + 0.2) = -1, \end{aligned}$$

with $\mu_2 = 1$ by symmetry: one EM step moves the means from ± 0.5 to ± 1 . (The unrounded responsibilities $1/(1 + e^{x_n})$ give $\mu_1 \approx -0.99$.)

+0.5 pts: $N_1 = 2$.

+1 pt: $\pi_1 = 0.5$.

+1 pt: $\mu_1 = -1$.

Follow-through: π_1 and μ_1 computed correctly from a wrong N_1 earn full credit for those lines.

$\pi_1 =$

0.5

$\mu_1 =$

-1

- (c) [1 Pt] Erica considers using K-means (Lloyd's algorithm) instead of EM. Which of the following is true of K-means on a general dataset (not just the four positions above)?

- ☒ Each iteration minimizes the objective $\sum_n \|x_n - \mu_{z_n}\|^2$ over one set of variables with the other held fixed: first the assignments z with the centers fixed, then the centers μ with the assignments fixed. The objective therefore never increases.
- ☐ Like the E-step of EM, the assignment step gives each point a probability of belonging to each cluster, and the center update weights the points by those probabilities.
- ☐ Because the objective never increases, Lloyd's algorithm converges to the global minimum of the objective from any initialization.
- ☐ The number of clusters can be chosen by running K-means for several values of K and keeping the K with the smallest objective.

Explanation: Lloyd's algorithm is alternating minimization: with the centers fixed, assigning each point to its nearest center minimizes the objective over z ; with the assignments fixed, the mean of each cluster minimizes it over μ (lecture 2). Each step can only lower the objective, and there are finitely many assignments, so it terminates. The second option describes EM's soft assignments; K-means assigns each point entirely to one cluster. The third is wrong because it stops at a local optimum, which depends on the initialization (hence scikit-learn's `n_init`). The fourth is wrong because the objective can only decrease as K grows (it is 0 at $K = N$), so K is chosen by other means, such as domain knowledge or the elbow method.

+1 pt: correct answer.

You are done with the midterm! Congratulations!

Use this page to draw your favorite CS 189/289A moment!