CS 189/289A, Final

Spring 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 **68 points** spread out over **5 questions** and the Honor Code certification. The exam must be completed in **170 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 exactly one correct answer. Please **fully** shade in the circle to mark your answer. For questions with **square bubbles** you may select multiple options. There is always *atleast* one correct answer. Please **fully** shade in the square(s) to mark your answer. For all math questions, **please simplify your answer**. Please also **show your work** if a large box is provided.

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 Machine Learning Trivia [8 Pts]

Select the best answer for each of the following questions.

(a) **[1 Pt]** Which of the following is the *best* characterization of the Universal Approximation Theorem (UAT)?

- ◯ A sufficiently wide single-hidden-layer network with a suitable activation can approximate any continuous function to arbitrary precision.
- ◯ A sufficiently wide single-hidden-layer network with a suitable activation can approximate any continuous function to arbitrary precision, and gradient descent will find such an approximation.
- ◯ A sufficiently deep network with a suitable activation can approximate any continuous function to arbitrary precision, but a single-hidden-layer network cannot.
- ◯ A sufficiently wide single-hidden-layer network can approximate any function to arbitrary precision, with or without an activation function.

(b) **[1 Pt]** Which of the following is **NOT** an advantage of convolutional layers over fully-connected layers for image processing?

- ◯ Parameter efficiency through weight sharing across spatial positions.
- ◯ Translation equivariance, so that a shifted input produces a correspondingly shifted feature map.
- ◯ Preservation of spatial structure by operating on local neighborhoods rather than flattening the input.
- ◯ The ability to learn complex features without requiring non-linear activation functions between layers.

(c) **[1 Pt]** Which of the following best describes the role of pooling (e.g. max-pooling) layers in a CNN?

- ◯ They apply learnable downsampling filters that adapt during training to select the most informative spatial regions.
- ◯ They increase the spatial resolution of feature maps so that fine-grained details are preserved for later layers.
- ◯ They reduce the spatial dimensions of feature maps, which grows the effective receptive field and introduces a degree of translation invariance.
- ◯ They normalize activations across spatial locations to stabilize training and prevent internal covariate shift.

(d) **[1 Pt]** In a Vision Transformer (ViT), how is an input image converted into a sequence that the transformer encoder can process?

- ◯ The image is passed through a stack of convolutional and pooling layers, and the final feature map is used as the token sequence.
- ◯ Each individual pixel is treated as a separate token in the sequence.
- ◯ The image is split into fixed-size non-overlapping patches, and each patch is linearly projected into a token embedding.
- ◯ A recurrent network scans the image row by row to produce a sequence of hidden states.

(e) **[1 Pt]** Suppose we are instruction-tuning a pre-trained language model using Supervised Fine-Tuning (SFT). How is the next-token prediction loss typically applied across a given `(prompt, response)` training pair?

- ◯ The loss is computed and averaged equally over all tokens in both the `prompt` and the `response`.
- ◯ The loss is computed only over the `prompt` tokens, effectively masking the `response`.
- ◯ The loss is computed only over the `response` tokens, effectively masking the `prompt` tokens.
- ◯ The loss applies a penalty term based on the KL divergence between the `prompt` and `response` distributions.

(f) **[1 Pt]** Suppose you perform Supervised Fine-Tuning (SFT) on a pre-trained language model. You consider two approaches: (1) *full-parameter SFT*, which updates all weights in the model, and (2) *output-head-only SFT*, which freezes the base model and updates weights of the output head only. After fine-tuning, the model is deployed for inference. Which of the following best describes the impact on **inference latency** compared to the original base model?

- ◯ Full-parameter SFT increases inference latency relative to the base model, but output-head-only SFT does not.
- ◯ Both approaches increase inference latency, but full-parameter SFT increases it more.
- ◯ Neither approach changes inference latency compared to the base model.
- ◯ Output-head-only SFT decreases inference latency because fewer parameters were updated during training.

(g) **[1 Pt]** What is one of the most important sources of information used by AlphaFold2 when predicting the structure of a protein?

- ◯ The sequences of proteins thought to be evolutionarily related to it
- ◯ The structure of that same protein at its melting point
- ◯ The name of the organism it came from
- ◯ The sequences of proteins of the same length

(h) **[1 Pt]** GEPA (Generalized Evolving Prompt Adaptation) is a method for improving LLM-based agents without modifying model weights. How does it work?

- ◯ It fine-tunes a small adapter network on top of the frozen model using task-specific demonstrations.
- ◯ It uses reinforcement learning to update the model's policy gradient estimates directly.
- ◯ It has the agent attempt a task, then uses a reflection model to suggest improvements to the agent's prompt based on the trajectory and reward.
- ◯ It retrieves relevant few-shot examples from a database and prepends them to the prompt at inference time.

2 Modeling Binary Data [18 Pts]

We observe n examples $\mathbf{x}_1, \ldots, \mathbf{x}_n$, where each example is a binary vector

$$\mathbf{x}_i = (x_{i1}, \ldots, x_{iD}) \in \{0,1\}^D.$$

Recall the Bernoulli probability mass function: for $x \in \{0,1\}$, $\text{Bern}(x \mid \mu) = \mu^x(1-\mu)^{1-x}$.

Mixture of Bernoulli distributions. Consider a Mixture of Bernoulli distributions (MoB) with K components. Each data point $\mathbf{x}_i$ is generated by first drawing a component assignment $z_i \in \{1, \ldots, K\}$ with

$$p(z_i = k) = \pi_k, \qquad 0 \le \pi_k \le 1, \qquad \sum_{k=1}^{K} \pi_k = 1,$$

and then drawing $\mathbf{x}_i$ conditionally on $z_i = k$:

$$p(\mathbf{x}_i \mid z_i = k) = \prod_{d=1}^{D} \mu_{kd}^{x_{id}} (1-\mu_{kd})^{1-x_{id}}, \qquad \mu_{kd} \in [0,1].$$

(a) **[2 Pts] Posterior responsibility**

For fixed MoB parameters (π, μ), define the posterior responsibility

$$r_{ik} := p(z_i = k \mid \mathbf{x}_i, \pi, \mu).$$

Fill in the boxes below:

$$r_{ik} = \frac{\boxed{\text{A}}}{\sum_{j=1}^{K} \boxed{\text{B}}}.$$

A should be in terms of π_k, μ_{kd}, and x_{id} , and **B** should be in terms of π_j, μ_{jd}, and x_{id}.

Bernoulli generative classifier. Now suppose each example $\mathbf{x}_i$ comes with a binary label $y_i \in \{0, 1\}$. The *Bernoulli generative classifier* mirrors the MoB setup, but the observed label y_i takes the role of the latent assignment z_i:

$$p(y_i = 1) = \phi, \qquad p(y_i = 0) = 1 - \phi,$$

$$p(\mathbf{x}_i \mid y_i = c) = \prod_{d=1}^{D} \mu_{cd}^{x_{id}} (1 - \mu_{cd})^{1 - x_{id}}, \qquad c \in \{0, 1\}.$$

(b) **[4 Pts] From generative model to linear log-odds**

Applying Bayes' rule and substituting the Bernoulli factorization, one can show that the posterior log-odds of the generative classifier are

$$\log \frac{p(y = 1 \mid \mathbf{x})}{p(y = 0 \mid \mathbf{x})} = \log \frac{\phi}{1 - \phi} + \sum_{d=1}^{D} \left[x_d \log \frac{\mu_{1d}}{\mu_{0d}} + (1 - x_d) \log \frac{1 - \mu_{1d}}{1 - \mu_{0d}} \right]. \tag{1}$$

Prove that (1) is an affine function of $\mathbf{x}$, i.e., that there exist $\mathbf{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$ such that

$$\log \frac{p(y = 1 \mid \mathbf{x})}{p(y = 0 \mid \mathbf{x})} = \mathbf{w}^\top \mathbf{x} + b,$$

by stating the resulting w_d and b explicitly and clearly showing your algebraic steps.

Logistic regression takes the same linear log-odds form as its starting point, but fits $\mathbf{w}$ and b directly to data rather than deriving them from (ϕ, μ):

$$p(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b).$$

(c) **[2 Pts] Deriving cross-entropy loss**

Given a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with $y_i \in \{0, 1\}$, let $\hat{y}_i := \sigma(\mathbf{w}^\top \mathbf{x}_i + b) = p(y_i = 1 \mid \mathbf{x}_i)$. Note that $p(y_i \mid \mathbf{x}_i)$ is a Bernoulli distribution with parameter $\hat{y}_i$.

Derive the negative log-likelihood, also known as the *binary cross-entropy* loss. Fill in the box below:

$$\mathcal{L}(\mathbf{w}, b) = -\frac{1}{n} \sum_{i=1}^{n} \log p(y_i \mid \mathbf{x}_i) = -\frac{1}{n} \sum_{i=1}^{n} \Big[\boxed{\text{C}} \Big].$$

C should involve only y_i, $\hat{y}_i$, and logarithms.

(d) **Linearly separable training data**

Suppose the training data is *linearly separable*: there exist $\mathbf{w}^*, b^*$ such that ${\mathbf{w}^*}^\top \mathbf{x}_i + b^* > 0$ for all i with $y_i = 1$ and ${\mathbf{w}^*}^\top \mathbf{x}_i + b^* < 0$ for all i with $y_i = 0$.

Consider scaling these parameters by $\alpha > 0$, i.e., using $(\alpha \mathbf{w}^*, \alpha b^*)$.

(i) **[1 Pt]** As $\alpha \to \infty$, what does the loss $\mathcal{L}(\alpha \mathbf{w}^*, \alpha b^*)$ converge to?

- ◯ 0
- ◯ 1
- ◯ $+\infty$
- ◯ $-\infty$

(ii) **[1 Pt]** Which of the following **most directly** follows from this observation when training on *linearly separable data*?

- ◯ The cross-entropy loss is non-convex for linearly separable data, so gradient descent converges to a local minimum.
- ◯ Gradient clipping is needed to prevent the training loss from diverging.
- ◯ Without regularization, gradient descent on linearly separable data will drive $\|\mathbf{w}\| \to \infty$.
- ◯ The model requires careful initialization near the separating hyperplane to avoid divergence.

(e) **Comparing the Bernoulli generative classifier with logistic regression**

(i) **[1 Pt]** How many trainable parameters does the *Bernoulli generative classifier* from part (b) have?

◯ D ◯ $D+1$ ◯ $2D$ ◯ $2D+1$

(ii) **[1 Pt]** How many trainable parameters does *logistic regression* have?

◯ D ◯ $D+1$ ◯ $2D$ ◯ $2D+1$

(iii) **[1 Pt]** Specifying ϕ and all μ_{cd} determines a posterior $p(y=1 \mid \mathbf{x})$ that can be written in the same functional form as logistic regression:

$$p(y=1 \mid \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$$

for suitable $\mathbf{w}, b$.

◯ True ◯ False

(iv) **[1 Pt]** For every choice of $w \in \mathbb{R}^D$ and $b \in \mathbb{R}$, there is a unique choice of ϕ and μ_{cd} that gives exactly the same posterior $p(y=1 \mid x)$ for all $x \in \{0,1\}^D$.

◯ True ◯ False

(v) **[1 Pt]** The Bernoulli generative classifier has closed-form MLEs for ϕ and the μ_{cd}, while logistic regression generally does not have a closed-form MLE for $\mathbf{w}, b$.

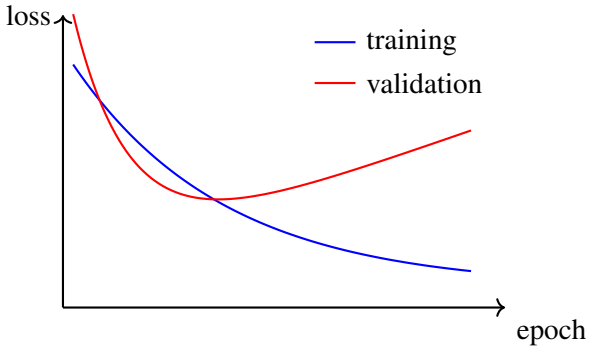
◯ True ◯ False

(vi) **[1 Pt]** Both the Bernoulli generative classifier and logistic regression produce linear decision boundaries in the original feature vector $\mathbf{x}$.

◯ True ◯ False

(f) **Choosing what to do in practice**

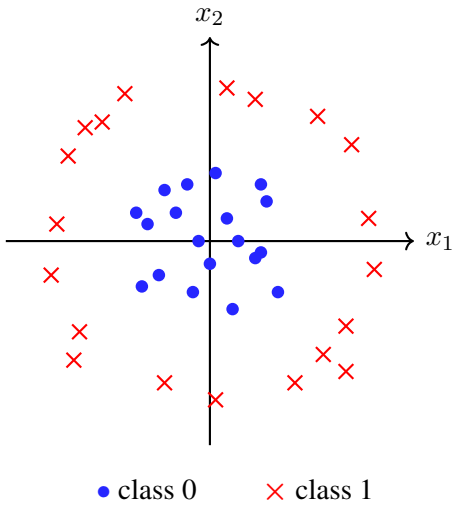
(i) **[1 Pt]** You train logistic regression with gradient descent. As training proceeds, the training loss continues to decrease, but the validation loss begins to increase:



Which change is *most likely* to help?

- ◯ Re-train using a stronger weight-decay penalty
- ◯ Re-train logistic regression after augmenting the input with pairwise product features $x_i x_j$
- ◯ Re-train logistic regression using a larger learning rate
- ◯ Retraining using mean-squared error (MSE) instead of cross-entropy as your loss function.

(ii) **[1 Pt]** The figure below shows a 2D training set with two classes.



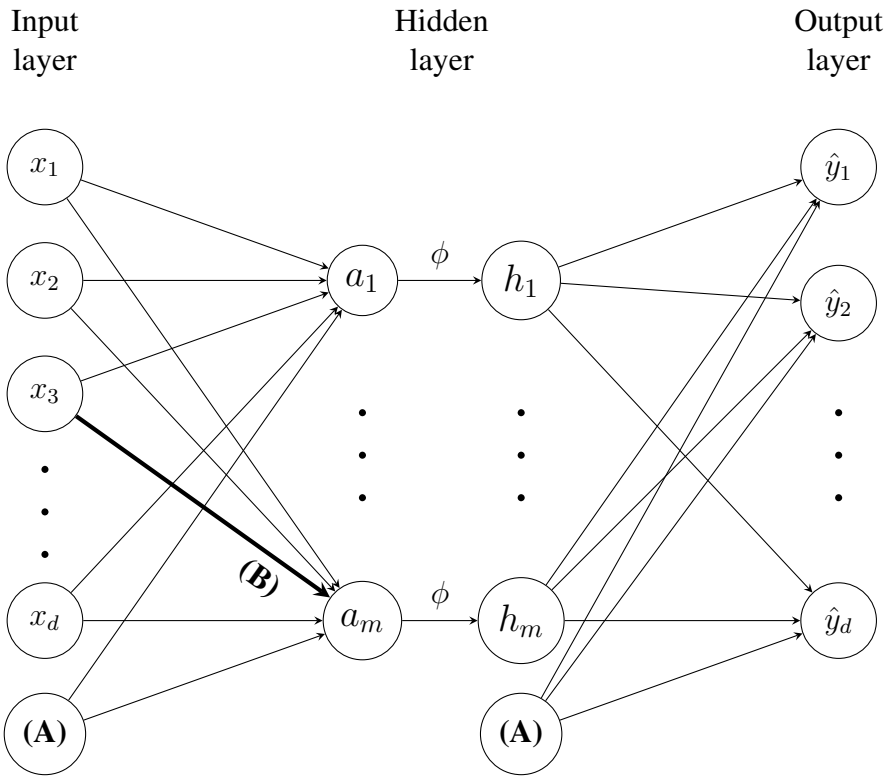
Logistic regression trained on the original two input features gives a straight-line decision boundary and has high training error. Which change is *most likely* to help?

- ◯ Re-train after augmenting the input with the features x_1^2, and x_2^2
- ◯ Replace the sigmoid function in logistic regression with the function $f(x) = \frac{x^2}{1+x^2}$
- ◯ Re-train after augmenting the input with the feature $x_1 x_2$
- ◯ Replace logistic regression with the Bernoulli generative classifier

3 Mirror Neural Network [16 pts]

Consider the following neural network with input $\mathbf{x} \in \mathbb{R}^d$, a single hidden layer of width m, and output $\hat{\mathbf{y}} \in \mathbb{R}^d$. Crucially, the same weight matrix $\mathbf{W} \in \mathbb{R}^{m \times d}$ is used in both layers: $\mathbf{W}$ maps from input to hidden, and $\mathbf{W}^\top$ maps from hidden to output. We use the notation $W_{i,j} \in \mathbb{R}$ to refer to the scalar at the i-th row and j-th column.

Each layer also has its own **bias vector** — $\mathbf{b}^{(1)} \in \mathbb{R}^m$ for the hidden layer, and $\mathbf{b}^{(2)} \in \mathbb{R}^d$ for the output layer.



(a) **Network Fundamentals**

i. **[1 Pt]** What value should appear inside the nodes labeled **(A)**?

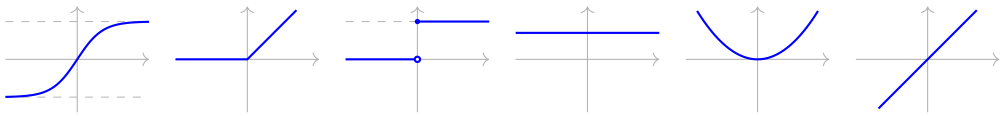
◯ 1 ◯ ϕ

◯ 0 ◯ w_{m+1} (a learnable scalar parameter)

ii. **[1 Pt]** The bold arrow labeled **(B)** connects input x_3 to hidden unit m. What is the scalar weight on this arrow?

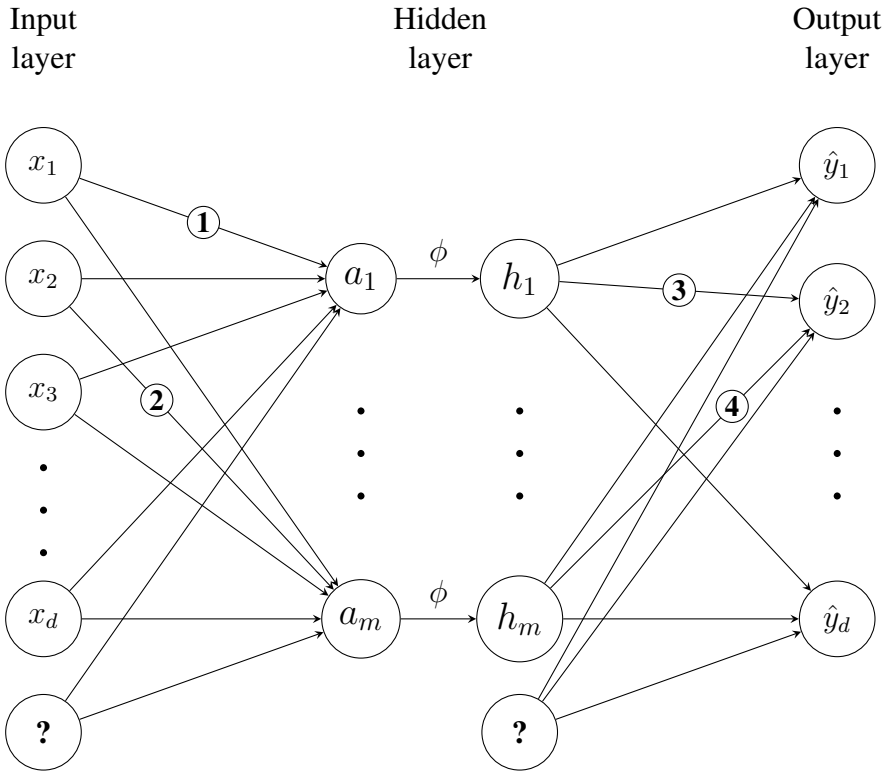
◯ $W_{m,3}$ ◯ $W_{3,m}$ ◯ $W^\top_{3m,m}$ ◯ $b^{(1)}_3$

iii. **[2 Pts]** The activation function ϕ in the hidden layer must satisfy certain properties for the network to be trainable via gradient descent and to be more expressive than a linear model.
Which of the following are valid choices for ϕ? **Select all that apply.**



☐ **(i)** ☐ **(ii)** ☐ **(iii)** ☐ **(iv)** ☐ **(v)** ☐ **(vi)**

(b) **[1 Pt] Mirror weights**
Four arrows in the diagram below are marked with circled numbers ①, ②, ③, and ④. Because the Mirror Network reuses the same weight matrix W in both layers, some pairs of arrows correspond to the **same scalar weight**.



Which of the following pairs of arrows correspond to the same scalar weight?

◯ ① and ② ◯ ② and ③ ◯ ③ and ④
◯ ① and ③ ◯ ② and ④ ◯ ① and ④

For the remaining parts, assume the network uses the sigmoid activation $\phi = \sigma$ where σ is the logistic sigmoid. Given a target $\mathbf{t} \in \mathbb{R}^d$, the loss is

$$L = \tfrac{1}{2} \|\hat{\mathbf{y}} - \mathbf{t}\|_2^2.$$

(c) **[4 Pts] Backpropagation**

Compute $\frac{\partial L}{\partial \mathbf{b}_k^{(1)}}$ for a single bias parameter $\mathbf{b}_k^{(1)}$, for $k \in \{1, \ldots, m\}$. Show your work by clearly identifying each step of the chain rule. You may leave the sigmoid derivative as $\sigma'(\cdot)$ and use $\mathbf{w}_k^\top$ to denote the k-th row of $\mathbf{W}$ (equivalently, the k-th column of $\mathbf{W}^\top$, written as a column vector).

(d) **[2 Pts] Learnable parameters**

How many learnable parameters does this network have?

- ◯ $2md + m + d$
- ◯ $md + m + d$
- ◯ $md + m$
- ◯ md
- ◯ $2md + 2m + 2d$
- ◯ $md + 2m + 2d$

Suppose for that $m < d$. We can view the Mirror Network as an **encoder-decoder**:

- The **encoder** $\mathcal{E}(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}^{(1)})$ encodes an input $\mathbf{x} \in \mathbb{R}_d$ into a latent $\mathbf{h} \in \mathbb{R}^m$.
- The **decoder** $\mathcal{D}(\mathbf{z}) = \mathbf{W}^\top \mathbf{h} + \mathbf{b}^{(2)}$ decodes a latent $\mathbf{h} \in \mathbb{R}^m$ to output $\hat{\mathbf{y}} \in \mathbb{R}^d$.

(e) **[2 Pts] Weight sharing** Which of the following are valid reasons why weight-sharing between the encoder and decoder might be useful? **Select all that apply.**

- ☐ It halves the number of weight parameters compared to using independent encoder and decoder matrices.
- ☐ It ensures that the representation $R(\mathbf{x})$ is a lossless encoding of $\mathbf{x}$, since the decoder can always invert it.
- ☐ It constrains the encoder and decoder to share geometric structure, acting as a form of regularization.
- ☐ It guarantees that the gradient of the loss with respect to $\mathbf{W}$ from the encoder layer equals the gradient with respect to $\mathbf{W}$ from the decoder layer.

(f) **[2 Pts] Batch normalization** Suppose we add a batch normalization layer between the pre-activation $\mathbf{a}$ and the activation σ, with learnable shift $\boldsymbol{\beta} \in \mathbb{R}^m$ and scale $\boldsymbol{\gamma} \in \mathbb{R}^m$. We denote the batch mean as $\boldsymbol{\mu} \in \mathbb{R}^m$ and the batch variance as $\boldsymbol{\nu}^2 \in \mathbb{R}^m$. Which of the following are true? **Select all that apply.**

- ☐ Batch normalization introduces $2m$ new learnable parameters per layer.
- ☐ Batch normalization introduces $4m$ new learnable parameters per layer, since $\boldsymbol{\mu}$ and $\boldsymbol{\nu}^2$ must also be learned.
- ☐ At test time, $\boldsymbol{\mu}$ and $\boldsymbol{\nu}^2$ are recomputed from each test input on the fly.
- ☐ At test time, $\boldsymbol{\mu}$ and $\boldsymbol{\nu}^2$ are replaced by running averages collected during training.
- ☐ At test time, batch normalization is skipped entirely and the pre-activation $\mathbf{a}$ is passed directly into σ.

(g) **[1 Pt] Bias redundancy under batch normalization** A classmate claims that when batch normalization is applied to the pre-activation $\mathbf{a} = \mathbf{W}\mathbf{x} + \mathbf{b}^{(1)}$, the bias $\mathbf{b}^{(1)}$ becomes redundant and can be removed without changing the function the network can express. Do you agree?

- ◯ Disagree: without $\mathbf{b}^{(1)}$, the pre-activation is forced to have zero mean, restricting the network's expressiveness.
- ◯ Disagree: $\mathbf{b}^{(1)}$ is needed to break the symmetry of $\mathbf{W}$ during training.
- ◯ Agree: $\mathbf{b}^{(1)}$ shifts every pre-activation by a constant, which shifts the batch mean $\boldsymbol{\mu}$ by the same amount; the centering step cancels $\mathbf{b}^{(1)}$ exactly, and the learnable shift $\boldsymbol{\beta}$ takes over the role of the bias.
- ◯ Agree, but only when $\boldsymbol{\gamma} = \mathbf{1}$; for other values of $\boldsymbol{\gamma}$, removing $\mathbf{b}^{(1)}$ changes the network's output because the scale parameter interacts with the bias before centering occurs.

4 CNNs and Self-Supervised Learning [11 pts]

(a) **Convolutional layers**

i. **[1 Pt]** An input tensor has shape $3 \times 24 \times 24$ (channels $\times$ height $\times$ width). After applying a convolutional layer with 16 filters, kernel size 7×7, stride 1, and no padding, what is the shape of the output tensor?

- ◯ $16 \times 18 \times 18$
- ◯ $16 \times 30 \times 30$
- ◯ $3 \times 30 \times 30$
- ◯ $16 \times 24 \times 24$

ii. **[1 Pt]** An input tensor has shape $8 \times 20 \times 20$. After applying a 2×2 max-pooling layer with stride 2, what is the shape of the output tensor?

- ◯ $4 \times 10 \times 10$
- ◯ $8 \times 20 \times 20$
- ◯ $8 \times 10 \times 10$
- ◯ $8 \times 9 \times 9$

iii. **[1 Pt]** An input tensor has shape $16 \times 14 \times 14$. After applying a convolutional layer with 32 filters, kernel size 3×3, stride 2, and padding 1, what is the shape of the output tensor?

- ◯ $32 \times 14 \times 14$
- ◯ $32 \times 6 \times 6$
- ◯ $32 \times 7 \times 7$
- ◯ $16 \times 7 \times 7$

iv. **[1 Pt]** The following 3×3 kernel is applied to a single-channel grayscale image (pixel values are nonnegative):

$$\mathbf{K} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Which of the following best describes the behavior of this kernel?

- ◯ It averages nearby pixels, producing a smoothed version of the image where fine details are suppressed.
- ◯ It outputs a large positive value in constant-intensity regions and zero at sharp transitions.
- ◯ It outputs zero in constant-intensity regions and a large value at pixels that differ significantly from their neighbors.
- ◯ It shifts the image by one pixel in the horizontal direction.

(b) **[1 Pt] Pretext tasks**

A student trains a CNN on a large unlabeled dataset of animal images. During training, each image is randomly rotated by one of four angles:

$$0^\circ, \quad 90^\circ, \quad 180^\circ, \quad 270^\circ.$$

The model is trained to predict which rotation was applied. After this training, the student fine-tunes the CNN on a small labeled dataset to distinguish cats from dogs. Which statement identifies pretext task(s) in this setup?

- ◯ The pretext task is cat-vs-dog classification.
- ◯ The pretext task is rotation prediction.
- ◯ Both cat-vs-dog classification and rotation prediction are considered pretext tasks.
- ◯ Neither cat-vs-dog classification nor rotation prediction are considered pretext tasks.

(c) **[1 Pt] Transfer learning with CNNs**

You have a CNN pretrained on ImageNet and only 200 labeled examples for a new bird classification task. What is the most reasonable transfer-learning strategy?

- ◯ Train every layer from scratch on the 200 examples.
- ◯ Freeze the pretrained convolutional layers and fine-tune only the last (classifier) layer.
- ◯ Fine-tune all layers with a high learning rate to adapt quickly to the new task.
- ◯ Freeze all layers and use the pretrained network's outputs directly without any retraining.

(d) **Convolutional autoencoders**

i. **[1 Pt]** A convolutional autoencoder maps an input image $\mathbf{x}$ to a latent code $\mathbf{z} = f(\mathbf{x})$, then reconstructs the image as $\hat{\mathbf{x}} = g(\mathbf{z})$. The model is trained using a reconstruction loss such as $\|g(f(\mathbf{x})) - \mathbf{x}\|_2^2$. Assume that the dimensions of the latent code are smaller than the dimensions of the input image. What is the main purpose of the latent bottleneck $\mathbf{z}$?

- ◯ To force a compressed representation that captures the most important structure in the input.
- ◯ To introduce stochasticity into the encoding, which acts as regularization during training.
- ◯ To ensure the reconstruction loss converges to zero.
- ◯ To make the decoder's job easier by providing a lower-dimensional input to reconstruct from.

ii. **[1 Pt]** Suppose an autoencoder for 28×28 MNIST images uses a latent code $\mathbf{z} \in \mathbb{R}^k$. What is the most likely effect of making k extremely small (e.g. $k = 1$)?

- ◯ Reconstructions become blurry or inaccurate because the bottleneck cannot preserve enough information about the input.
- ◯ Reconstructions remain sharp, but the model takes much longer to converge during training.
- ◯ The encoder learns to ignore the input and the decoder memorizes the training set.
- ◯ The model fails to train entirely because gradients cannot flow through a one-dimensional bottleneck.

(e) **[2 Pts] Training a text-conditioned heatmap model**

A student wants to train a model that takes as input an image $\mathbf{x}$ and a text query q (e.g. $q =$ "red cube") and outputs a heatmap

$$H_q(\mathbf{x}) \in [0,1]^{h \times w},$$

where large values indicate pixels relevant to the text query. The student has a dataset of images paired with text queries and pixel-level binary masks $M_q(\mathbf{x}) \in \{0,1\}^{h \times w}$ marking the queried object. Which training strategy is most appropriate?

- ◯ Train an image encoder that produces a single global feature vector for the image, concatenate it with the text embedding, and predict the heatmap with a fully connected layer supervised by the ground-truth mask.
- ◯ Train an image encoder that produces spatial image features, train a text encoder that embeds the query, compute a similarity score between the query embedding and each spatial feature, and supervise the resulting heatmap against the ground-truth mask.
- ◯ Train an image encoder that produces spatial image features, train a text encoder that embeds the query, compute a similarity score between the query embedding and each spatial feature, and supervise the resulting heatmap using a reconstruction loss that decodes the heatmap back into the original image.
- ◯ Train an image encoder and text encoder jointly using only a contrastive loss between global image and text embeddings, then extract the heatmap from the attention weights at test time without any mask supervision.

(f) **[1 Pt] Positive and negative pairs**

In contrastive learning, an image $\mathbf{x}$ is used as a reference. Which of the following is the best example of a positive pair?

- ◯ $\mathbf{x}$ and the next image in the dataset.
- ◯ $\mathbf{x}$ and a color-distorted version of $\mathbf{x}$.
- ◯ $\mathbf{x}$ and the pixel-nearest neighbor from another class.
- ◯ $\mathbf{x}$ and a random noise image.

(g) **[1 Pt] Zero-shot classification with CLIP**

A CLIP model has a jointly trained image encoder and text encoder. You want to classify an image as one of "cat," "dog," or "plane" without training a new classifier. What should you do?

- ◯ Encode the image, then pass its embedding through a linear classifier trained on the three class labels.
- ◯ Encode the image and each class label as text, then choose the label whose text embedding has the highest cosine similarity to the image embedding.
- ◯ Encode each class label as text and use the text embeddings as cluster centroids, then assign the image to the nearest cluster based on pixel distance.
- ◯ Encode the image three times with different random augmentations, average the embeddings, and pick the class whose text embedding is closest to the average.

(h) **[1 Pt] CLIP for robotics**

Suppose we train a model using the CLIP contrastive objective on a dataset of image-action pairs. A robot must use this model at test time to select an action given the current camera image. What is the *key* limitation? Assume actions are represented as real-valued vectors.

- ◯ The contrastive objective cannot be applied to image-action pairs.
- ◯ The model would not be able to map images and actions to a shared embedding space.
- ◯ The model can score how well a given action matches an image, but cannot generate or sample actions conditioned on an image.
- ◯ The model would require text descriptions of actions as an intermediate step at test time.

5 Transformers [15 Pts]

Transformers are the dominant class of language models. Taking a sequence of token embeddings as input, they repeatedly apply the attention and feed-forward layers to the processed vectors. If set up properly, this model enables generating tokens of new text or images.

(a) **[1 Pt]** Which loss function would be most effective for training a transformer to perform next-token prediction?

- ◯ Mean Squared Error (MSE)
- ◯ InfoNCE (Contrastive Loss)
- ◯ Attention Loss
- ◯ Cross-Entropy Loss

(b) **[1 Pt]** Each transformer block applies an attention layer followed by an MLP (feed-forward) layer. What is the primary role of the MLP layer?

- ◯ It computes pairwise interactions between tokens to propagate information across the sequence.
- ◯ It enables parallel decoding by processing all tokens simultaneously.
- ◯ It applies a learned nonlinear transformation independently to each token's representation, expanding the model's capacity to represent complex features.
- ◯ It reduces the dimensionality of the hidden states to prevent overfitting in deeper layers.

(c) **[1 Pt]** What is the primary benefit of using multi-head attention rather than single-head attention with the same total dimensionality?

- ◯ Each head operates on a distinct section of the input sequence, enabling the model to process longer contexts.
- ◯ The heads share weights with each other, reducing the total parameter count compared to single-head attention.
- ◯ Multiple heads enable parallel token generation during autoregressive decoding.
- ◯ Each head can learn a different attention pattern allowing the model to capture diverse types of dependencies simultaneously.

(d) **[1 Pt]** During next-token prediction training, how many **forward-passes** does a transformer have to do to compute the loss function for a sequence of length N? Assume we can use an arbitrarily large batch size.

- ◯ $\mathcal{O}(1)$, since the entire sequence can be processed in parallel.
- ◯ $\mathcal{O}(N)$, because the model can process one element at a time.
- ◯ $\mathcal{O}(N^2)$, because of the attention layer's quadratic complexity that is computed once for the whole sequence.
- ◯ $\mathcal{O}(N^3)$, because of the attention layer's quadratic complexity that must be recomputed for every processed element of the sequence.

(e) **[1 Pt]** Consider the standard self-attention operation in matrix form

$$\text{Attn}(\mathbf{X}) = \text{softmax}\left(\frac{\mathbf{X}\mathbf{W}_Q\mathbf{W}_K^\top\mathbf{X}^\top}{\sqrt{d_k}}\right)\mathbf{X}\mathbf{W}_V,$$

where $\mathbf{X} \in \mathbb{R}^{N\times d}$ is the matrix of token embeddings (without any positional encoding). Which of the following is true about this operation?

*Note: We use attention **scores** to refer to Attn$(\mathbf{X}) \in \mathbb{R}^{N\times d}$ and attention **weights** to refer to the input to the softmax function in the equation above.*

- ◯ Reordering the rows of X does not change the attention *scores*.
- ◯ Reordering the rows of X reorders the rows of the attention *scores* in the same way.
- ◯ Reordering the rows of X changes the attention *weights* but not the attention *scores*.
- ◯ Reordering the rows of X has an unpredictable effect that depends on the learned weights $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V$.

(f) **[6 pts] Attention Biases**

The full self-attention layer, as originally implemented, would calculate the query of input $\mathbf{x}_i$, and the key of input $\mathbf{x}_j$, for $j = 1, \ldots, N$, as

$$\mathbf{q}_i = \mathbf{W}_Q^\top\mathbf{x}_i + \mathbf{b}_Q \quad \text{and} \quad \mathbf{k}_j = \mathbf{W}_K^\top\mathbf{x}_j + \mathbf{b}_K.$$

Then, the pre-attention scores would be computed as

$$s_{ij} = \frac{\mathbf{q}_i^\top\mathbf{k}_j}{\sqrt{d_k}},$$

where d_k is the dimension of $\mathbf{q}_i$ and $\mathbf{k}_j$, and the softmax would be applied to the vector of pre-attention scores between $\mathbf{q}_i$ and $\mathbf{k}_1, \mathbf{k}_2, \ldots, \mathbf{k}_N$.

i. **[2 pts]** Let $\mathbf{x}$ be a vector and Softmax be the vector-softmax operation. Prove that, for any scalar c

$$\text{Softmax}(\mathbf{x} + c\mathbf{1}) = \text{Softmax}(\mathbf{x})$$

ii. **[4 pts]** Show that the bias term b_K does not affect the final attention score and, thus, can be safely removed.

(g) **[5 pts] Attention Masks**

We are training a transformer-based language model for a specialized application where it is useful to use bidirectional attention for the final M tokens of the generation. For simplicity, assume that every generated sequence has a total length of $N + M$. The first N tokens are generated using standard autoregressive causal attention. The final M tokens are generated in parallel such that they can attend to all prior tokens (1 to N), as well as all other tokens within the final M block bidirectionally. Since the generation of the final M tokens is parallel, not autoregressive, the input token at the final $M - 1$ positions is a fixed special token.

For a standard attention mask matrix, let 1 denote that a query token at row i is allowed to attend to a key token at column j, and 0 denote that the token is masked from attention.

i. **[3 pts]** Write out the full attention mask matrix for $N = 4$ and $M = 2$.

ii. **[2 pts] Inference Complexity** Focus purely on the inference process for generating this sequence from scratch (assume no prefilled prompt; the model must generate the N tokens step-by-step, and then generate the M tokens).

How many **distinct** matrix multiplications are required to compute the key vectors for all tokens in the sequence during a single inference pass?

◯ N ◯ $N + M$ ◯ $N \cdot M$
◯ $N + 1$ ◯ $2N + M$ ◯ $(N + M)^2$

You are done with the final! Congratulations!

Use this page to draw something tuff 🤭