CIS 4190/5190: Applied Machine Learning Fall 2024

**Midterm Exam II**

*Exam Data: December 9, 2024*

**Name:** _____

**Penn ID number:** _____

**Section number (4190 or 5190):** _____

# Instructions

- Write your name, Penn ID number, and section number on this page.
- Write your answer directly on the exam paper. Consider using a pencil and eraser, which will allow you to easily adjust your answers if needed.
- No digital devices are allowed during the exam (e.g., laptops, tablets, phones, watches). Please ensure that all your devices are turned off or set to silent mode.
- This is 75-minute exam, containing 15 questions with 50 points total.
- Each point should take approximately 1.5 minutes; if you find yourself spending too much time on one problem, move on and come back to it.
- At the end of 75 minutes, you will put down your pens and submit your exam.

**Good luck!**

1. (2 pt) Suppose we have a neural network that has high training accuracy but low validation accuracy. Which of the following can you use to improve the validation accuracy? Select all that apply. _____.

A. Decrease learning rate.

B. Add dropout.

C. Increase the number of hidden dimensions of layers.

D. Pretrain the neural network on a related task.

B, D. Half point each for **not** marking A and C. Half point each for marking B and D. Follow this for all the "select all" questions.

2. (2 pt) Which of the following advantages do transformers have over recurrent neural networks (RNNs) for language processing? Select all that apply. _____.

A. Transformers can better handle long-range dependencies.

B. Transformers simplify model training by requiring fewer parameters than RNNs.

C. Transformers requires smaller datasets to train than RNNs.

D. Transformers enable sequential processing of fixed-length inputs.

A: Yes. B: No. C: No. D: No.

3. (2 pt) A machine learning algorithm is said to be *scale invariant* if its performance or results do not change when the scale of any of the input features is altered (the scale is, of course, applied identically to both the training and test data). For instance, doubling the values of a feature should not affect the algorithm's results. Which of the following algorithms is scale invariant? Select all that apply. _____.

A. $\ell_2$-regularized linear regression.

B. Decision Tree.

C. Random Forest.

D. KNN with Euclidean distance.

E. K-means clustering.

A: No. B: Yes. C: Yes. D: No. E: No. 0.4 points for each option, to add up to 2.0.

4. (2 pt) Bob is training a decision tree on a dataset, but both the training and testing loss are high. Bob comes to you for advice on how to improve his model. Which of the following suggestions might help him? Select all that apply. _____.

A. Train an ensemble of decision trees using bagging.

B. Use boosting to iteratively reduce errors of previous models.

C. Prune the decision tree on a hold-out validation set.

D. Collect more training data to improve the model's performance.

E. Increase the maximum depth of the decision tree.

B, E

5. (2pt) Consider two MDPs M=(S,A,P,R, $\gamma$) and $M' = (S, A, P, R', \gamma)$. $M'$ is identical to $M$ except the reward function, where $R'(s, a, s') = wR(s, a, s') + b$ for all $s, a, s'$. How is the value function $V'^{\pi}(s)$ for some arbitrary policy in $M'$ related to the value function $V^{\pi}(s)$ in $M$? Write a concise mathematical expression if possible.

Given that value is a discounted sum of rewards, the scaling factor $wR$ carries straight through to the value, so it becomes $wV$. [1 pt] Now, accounting for the constant bias $b$, which is a reward received at every time instant, its sum is $b(1 + \gamma + \gamma^2 + ...)$ [0.5 pt] which is $b/(1 - \gamma)$ [0.5 pt], following the formula for a geometric progression. So final expression is $V \rightarrow wV + b/(1 - \gamma)$.

6. (2 pt) After factorizing the user-item utility matrix at Neflix, we get user features $p_{Jack} = [2, 3]$ and $p_{Eve} = [1, 4]$, and movie features $q_{starwars} = [1, 4]$ and $q_{godfather} = [2, 3]$. Who among Jack and Eve is likely to have higher utility for *The Godfather*? Show your work in the space provided.

The predicted utility is dot product of user feature and movie feature vectors. So, $< Jack, Godfather >= 4 + 9 = 13$ (0.5 pt), and $< Eve, Godfather >= 2 + 12 = 14$ (0.5 pt), so ans: Eve. (1 pt)

7. (3 pt) You are designing a CNN with the following block, repeated over and over: 3x5 conv (stride 1, zero pad 1) $\rightarrow$ Batchnorm $\rightarrow$ 3x3 maxpool (stride 1) $\rightarrow$ ReLU

A. (1 pt) After processing through one such block, what is the largest size of the image region that can affect the output feature map activations at any particular pixel location?

Convolutions and pooling allow one pixel in their output to be influenced by other pixels in the input. As illustrated in the 1-D conv example below (only for explanation, not required to be in the solution), the kernels add up as $k_1 + k_2 - 1$, so along first axis: 3+3-1=5. and along second axis: 5+3-1=7. So **5x7**. 0.75 pts awarded for missing the -1 and saying 6x8 instead.

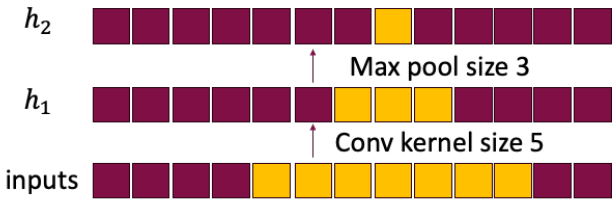



Figure 1: The yellow pixel in the output is influenced by the yellow regions indicated in each layer.

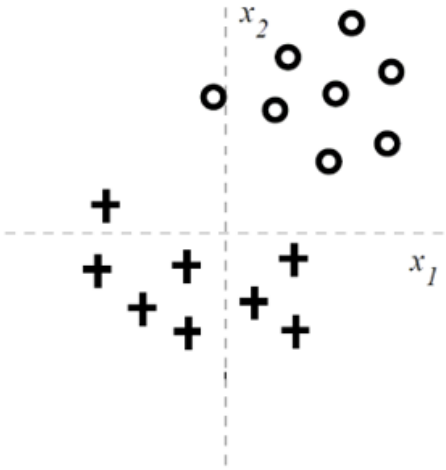
B. (2 pt) A particular image pattern requires observing a spatial extent of 8x6 pixels in the input image in order to recognize it. After how many blocks can one of the feature maps within the CNN correctly identify and locate this pattern? Show your work in 1-2 lines.

Exactly as above, each conv *block* can now be treated as having a region of influence of size 5x7. This is not sufficient for an output pixel to see an 8x6 pattern (since $5 < 8$) in the image. But after 2 blocks, the region of influence becomes (5+5-1=9)x(7+7-1=13), using the $k_1 + k_2 - 1$ formula along each axis. This region is >= the 6x8 pattern on each axis, so **2 blocks suffices**.

8. (4 pt) Consider a logistic regression classifier $P(y = 1|x, \beta) = \sigma(\beta_0 + \beta_1 x_1 + \beta_2 x_2)$, trained to maximize the following objective:

$$\sum_{i=1}^{n} \log P(y_i|x;\beta) - \lambda_1\beta_1^2 - \lambda_2\beta_2^2$$

For the training dataset in the figure, what can you say about the decision boundary (you can sketch it approximately if you like) and training loss if:




A. (2 pt) $\lambda_2$ becomes very large (approaching $\infty$) and $\lambda_1$ is small and positive? As $\lambda_2 \to \infty$, the classifier cannot rely on feature 2 ($\beta_2 \to 0$), so the decision boundary becomes purely vertical. The training loss becomes non-zero i.e. increases (even though the dataset is linearly separable).

B. (2 pt) $\lambda_1$ becomes very large (approaching $\infty$) and $\lambda_2$ is small and positive? Decision boundary becomes purely horizontal. In this case, the training loss remains 0 i.e. does not change, since the dataset is linearly separable even purely along the $x_2$ axis.

9. (5 pt) You are considering two alternative designs of a neural network to map from 100x100x500 (height x width x channels) inputs $x$ to 100x100x500 outputs $y$.

A. (2 pt) Your first option is a single 3x3 convolution layer that maps directly from $x$ to $y$: $x \rightarrow$ 3x3 conv $\rightarrow y$. How many learnable parameters are in this layer? (you may skip biases to approximate) (3x3x500+1)x500=2,250,500 parameters. approx. 2.25M parameters. Full points for getting the approximate answer too.

B. (3 pt) Your second option is the following block: $x \rightarrow$ 1x1 conv $\rightarrow h_1 \rightarrow$ 3x3 conv $\rightarrow h_2 \rightarrow$ 1x1 conv $\rightarrow y$. $h_1$ and $h_2$ each have exactly 10 channels. How many learnable parameters are in this block? (once again, you may skip biases to approximate) (1x1x500+1)x10+(3x3x10+1)x10+(1x1x10+1)x500 = 5010+910+5500=11,420 parameters. Skipping biases, works out to 10,900 parameters. Full points for approx 11k.

(Just for explanation) Note the parameter savings by doing this kind of dimensionality reduction with the 1x1 convolution!

10. (5 pt) In this problem, we consider a self-attention layer using dot-product attention. Suppose the input to this layer is three word embeddings. After applying the query ($q$), key ($k$), and value ($v$) projections, we obtain the following vectors for the input embeddings:

$$q_1 = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad k_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad v_1 = \begin{bmatrix} 2 & 1 \end{bmatrix}$$

$$q_2 = \begin{bmatrix} 0 & 1 \end{bmatrix}, \quad k_2 = \begin{bmatrix} 0 & 1 \end{bmatrix}, \quad v_2 = \begin{bmatrix} 0 & 3 \end{bmatrix}$$

$$q_3 = \begin{bmatrix} 1 & 1 \end{bmatrix}, \quad k_3 = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad v_3 = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

A. (1 pts) Compute the attention score matrix $S$, where each element $s_{i,j}$ is the dot-product attention score between $q_i$ and $k_j$.

The score matrix is:

$$S = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 2 & 1 & 1 \end{bmatrix}$$

(Full points if the matrix is transposed)

B. (2 pts) Compute the attention weights based on the attention score matrix $S$. To simplify computation, assume this self-attention layer uses a modified softmax function with base 2, i.e.,

$$\text{softmax}(\mathbf{z})_i = \frac{2^{z_i}}{\sum_j 2^{z_j}}.$$

The attention weight matrix is:

$$\begin{bmatrix} 2/5 & 1/5 & 2/5 \\ 2/5 & 2/5 & 1/5 \\ 1/2 & 1/4 & 1/4 \end{bmatrix}$$

(Full points if the matrix is transposed)

C. (2 pts) Using the attention weights and the value vectors, compute the output representation $y_i$ for each word embedding.

The output representations are:

$$y_1 = \begin{bmatrix} 1.2 \\ 1.4 \end{bmatrix} = \begin{bmatrix} 6/5 \\ 7/5 \end{bmatrix}, \quad y_2 = \begin{bmatrix} 1.0 \\ 1.8 \end{bmatrix} = \begin{bmatrix} 1 \\ 9/5 \end{bmatrix}, \quad y_3 = \begin{bmatrix} 1.25 \\ 1.5 \end{bmatrix} = \begin{bmatrix} 5/4 \\ 3/2 \end{bmatrix}$$

11. (5 pt) Gradient boosting builds an ensemble of base models iteratively. At each step, the pseudo-residuals (negative gradients of the loss with respect to current predictions) are computed to train the next base model. Suppose the input to a gradient boosting model is a dataset $\{(x_i, y_i)\}_{i=1}^n$, where $x_i$ is the input feature, $y_i$ is the target label, and $F_t(x_i)$ is the current model's prediction. The pseudo-residuals $z_i$ are used to construct the dataset $\{(x_i, z_i)\}_{i=1}^n$ for training the next base model. Write the expressions for $z_i$ for the following loss functions.

A. (2 pt) Gradient boosting with the Mean Squared Error (MSE) loss:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{2} \sum_{i=1}^{n} \left(F_t(x_i) - y_i\right)^2 .$$

The pseudo-residuals $z_i$ are the negative gradients of the loss with respect to the current predictions $F_t(x_i)$:

$$z_i = -\frac{\partial \mathcal{L}_{\text{MSE}}}{\partial F_t(x_i)} = -\left(F_t(x_i) - y_i\right) = y_i - F_t(x_i).$$

B. (3 pt) Gradient boosting with the Negative Log-Likelihood (NLL) loss:

$$\mathcal{L}_{\text{NLL}} = -\sum_{i=1}^{n} \left[y_i \log \sigma(F_t(x_i)) + (1 - y_i) \log(1 - \sigma(F_t(x_i)))\right],$$

where $\sigma(z)$ is the sigmoid function, with $\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z))$.

The pseudo-residuals $z_i$ are the negative gradients of the loss with respect to the current predictions $F_t(x_i)$:

$$z_i = -\frac{\partial \mathcal{L}_{\text{NLL}}}{\partial F_t(x_i)}.$$

First, compute the gradient:

$$\frac{\partial \mathcal{L}_{\text{NLL}}}{\partial F_t(x_i)} = -\left[y_i \frac{\partial \log \sigma(F_t(x_i))}{\partial F_t(x_i)} + (1 - y_i) \frac{\partial \log(1 - \sigma(F_t(x_i)))}{\partial F_t(x_i)}\right].$$

Using the derivatives of the sigmoid function:

$$\frac{\partial \log \sigma(F_t(x_i))}{\partial F_t(x_i)} = 1 - \sigma(F_t(x_i)), \quad \frac{\partial \log(1 - \sigma(F_t(x_i)))}{\partial F_t(x_i)} = -\sigma(F_t(x_i)),$$

we get:

$$\frac{\partial \mathcal{L}_{\text{NLL}}}{\partial F_t(x_i)} = -\left[y_i(1 - \sigma(F_t(x_i))) - (1 - y_i)\sigma(F_t(x_i))\right].$$

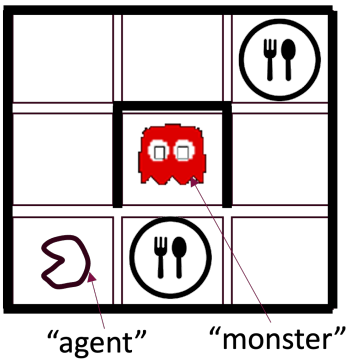
Simplify:

$$\frac{\partial \mathcal{L}_{\text{NLL}}}{\partial F_t(x_i)} = \sigma(F_t(x_i)) - y_i.$$

Therefore, the pseudo-residuals are:

$$z_i = -\frac{\partial \mathcal{L}_{\text{NLL}}}{\partial F_t(x_i)} = y_i - \sigma(F_t(x_i)).$$

12. (5 pt) Imagine a simplified version of Pacman, played over a 3x3 grid, with the agent, one ghost, and food particles scattered over states. The agent's objective is to consume as much food as possible while avoiding the monster – once the agent reaches a cell with food, the food is consumed and vanishes. The agent can only move horizontally or vertically one cell at a time. The monster too has the same movement capabilities as the agent, and moves uniformly at random at each step. Food spawns randomly in any state at any step with a small probability. The thick lines in the grid denote fixed walls that cannot be crossed by the agent or the monster. The game ends if the monster and the agent are in the same cell.




A. (2 pt) Describe the state space. Can you count the total number of states? Show your work, okay to approximate.
The state consists of the agent cell, ghost cell, whether each grid cell has food or not. A quick count produces approx $9x9x2^9$ states= 41472. Full points for approximation to 30-50k with full points, as long as they multiply the right things.

B. (1 pt) Describe the agent's action space. Can you count the total number of actions?
There are just 4 actions for the agent at any given time, up, down, left, right.

C. (1 pt) Describe the transition function. Is it deterministic or stochastic?
Agent state updates deterministically according to action, as permitted by walls. Monster state updates stochastically. Food updates stochastically (except when consumed by agent, in which case it vanishes deterministically). Full points for saying most of these things **(grade leniently)**, plus the overall transition function is stochastic.

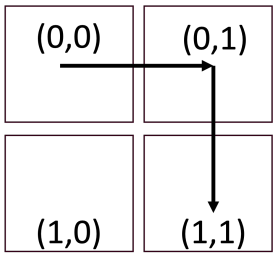
D. (1 pt) For the reward function, is it sufficient to reward the agent positively each time it gets food? Do we need any other reward terms for the reward-optimal agent to behave correctly? Yes, it is sufficient, no other terms are needed to specifically avoid the monster because optimal food-gathering entails avoiding the monster.

13. (5 pt) An agent is dropped into an unknown 2x2 grid world. The agent can select four actions at each step: move up, down, left, or right. The agent makes the following two moves, depicted in Figure A:

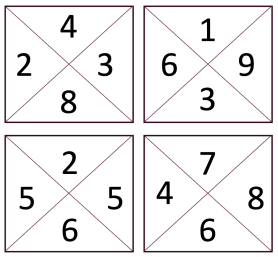
- it starts at (0,0), chooses "right" at the first step, moves to (0,1) and gets a reward -1.
- it then chooses "down" at the next step, moves to (1,1), and gets a reward +10.

As it follows this trajectory, the agent performs a Q-Learning update once after step 1, and then a second time after step 2. Its initial Q-values before any learning (initialized at random) are shown in Figure B. Compute the updated Q values after each step. Show your work. Assume that the discount factor is 0.9, and the learning rate is 0.1.

A




B




After first step, only $Q((0,0), right)$ updates per the Q-learning rule from 3 to: $3 + 0.1(-1 + 0.9 \times 9 - 3) = 3 + 0.1 \times 4.1 = 3.41$. No other Q values change. (2.5 pt, assign partial credit for correct work towards the answer)

Similarly, after second step, only $Q(0,1), down)$ updates from 3 to $3 + 0.1(10 + 0.9 \times 8 - 3) = 3 + 0.1 \times 14.2 = 4.42$ (2.5 pt, assign partial credit for correct work towards the answer)

14. (6 pt) A diagonal co-variance d-dimensional gaussian is defined by the probability density function:

$$P(x) = \frac{1}{(\sigma\sqrt{2\pi})^d} \exp\left(-\sum_{j=1,\ldots,d} \frac{(x_j - \mu_j)^2}{\sigma_j^2}\right)$$

A. (2pt) What is the log-likelihood of a sample $x$ under $P(x)$? It is okay to leave hard-to-simplify logarithms in your answer.

$$\log P_\theta(x) = -d\log(\sigma\sqrt{2\pi}) - \sum_{j=1,\ldots,d} \frac{(x_j - \mu_j)^2}{\sigma^2}$$

NOTE: it was announced in exam that you could assume $\sigma_j = \sigma$ for all j, but full points if the above solution had $\sigma_j$ instead of $\sigma$.

B. (1 pt) How does this log-likelihood expression change if $\sigma_j = 1$ for all $j$, corresponding to a unit-spherical gaussian?

$$\log P_\theta(x) = -d\log(\sqrt{2\pi}) - \sum_{j=1,\ldots,d} (x_j - \mu_j)^2$$

C. (1 pt) Now consider a mixture of $K$ unit-spherical gaussians. What is the log-likelihood of $x$ under this mixture?

Likelihood is easy to write down, looks like:

$$P(x) = \sum_k P(z=k) \frac{1}{(\sqrt{2\pi})^d} \exp\left(-\sum_{j=1,\ldots,d} (x_j - \mu_j^k)^2\right)$$

Its logarithm is not easy to simplify, beyond just writing:

$$\log\left(\sum_k P(z=k) \frac{1}{(\sqrt{2\pi})^d} \exp\left(-\sum_{j=1,\ldots,d} (x_j - \mu_j^k)^2\right)\right)$$

D. (2 pt) Recall that k-means clustering optimizes the sum of squared distances loss function: $\min_S \sum_{k=1}^K \sum_{x\in S_k} \|x - \mu_k\|_2^2$, where each $S_k$ corresponds to a cluster. Can this be interpreted as maximizing the log-likelihood from part C? Why or why not?

"(0.5 pt for saying something like:) The k-means clustering objective does not quite maximize the GMM likelihood (0.5 pt), but it makes some assumptions to approximate it.
(1.5 pt for saying something like the following) It replaces the $\sum_k$ with a $\max_k$, and assumes $P(z = k)$ is same for all $k$ i.e. it assumes a uniform mixture of gaussians. This means that each point's likelihood is approximated by computing it from only the gaussian with the closest mean."

(More elaborate explanation of the solution above, for your interest (not expected in student solutions).) It clearly does not maximize the log-likelihood exactly. But if each sample's likelihood can be approximated as coming from only a single gaussian in the mixture (to which it is assigned), then the summation in the log in part C goes away, and it looks like

$$\log\left(\max_k P(z=k)\frac{1}{(\sqrt{2\pi})^d}\exp\left(-\sum_{j=1,\ldots,d}(x_j-\mu_j^k)^2\right)\right)$$

.
Now, if you further assume that $P(z = k)$ is the same for all $k$, then the expression inside the logarithm i.e. the likelihood of each point, looks like (ignoring constants):

$$\max_k \exp\left(-\sum_{j=1,\ldots,d}(x_j-\mu_j^k)^2\right)$$

. At this point, each data point is "assigned" to one gaussian, and summing over the data points closely recovers the k-means clustering objective.

Extra space

Extra space

Extra space