CS689: Machine Learning - Fall 2022

Final Exam

Dec 20, 2022 Name: ______________________________

Instructions: Write your name on the exam sheet. No electronic devices may be used during the exam. You may consult your paper notes and/or a print copy of texts during the exam. Sharing of notes/texts during the exam is strictly prohibited. Show your work for all derivation questions. A suitable explanation must be provided for all questions to earn full credit. Attempt all problems. Partial credit may be given for partially incorrect or incomplete answers. If your answer to a question spans two pages, please make sure to indicate that at the end of the first page of the answer. If you have questions at any time, please raise your hand.

Problem	Topic	Page	Points	Score
1	Activation Functions	1-2	10	
2	Multilayer Perceptrons	3-4	10	
3	Recurrent Neural Networks	5-6	10	
4	Generalized Probabilistic Regression	7-8	10	
5	Heteroskedastic Regression	9-10	10	
6	Product of Marginals	11-12	10	
7	Scale Mixtures	13-14	10	
8	Discrete Non-Linear Factor Analysis	15-16	10	
9	Implementation	17-18	10	
10	Modeling	19-20	10	
Total:			100	

1. (*10 points*) **Activation Functions:** Explain what the vanishing gradient problem is and how the use of rectified linear unit activation functions helps to mitigate this problem. Provide supporting equations.

Example Solution: The vanishing gradient problem is cause by the use of activation functions like the the sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$ whose gradient is always positive, but less than 1 (it is specifically in [0,0.25]). When we have many layers of units with sigmoid activations, each layer contributes a multiplicative factor less than one to the gradient of parameters in layers below it. This usually results in the partial derivative of the loss with respect to parameters lower in a deep network approaching zero. The ReLU activation $\text{relu}(x) = \max(0, x)$ avoids this problem by having the branch of the activation function that is not clipped to 0 have a gradient equal to 1. This helps to stops the partial derivative of the loss with respect to parameters lower in a deep network from decaying towards 0.

2. (*10 points*) **Multilayer Perceptrons:** Suppose we use a one hidden layer sigmoid MLP as the discriminant function $g_\theta(\mathbf{x})$ for a binary classifier with $y \in \{-1, 1\}$. Assume that $\mathbf{x}$ is a $1 \times D$ vector. What is the computational complexity of computing the predicted class as a function of the data dimension D and the hidden layer width K? Assume a simplified computational model where addition, subtraction, multiplication, addition, exponentiation and numerical comparisons all take $O(1)$ time. Explain your answer.

Example Solution: The computation performed in this model is $g_\theta(\mathbf{x}) = \mathbf{hv} + c$ where $h = \sigma(\mathbf{xw} + b)$. We have that $\mathbf{w}$ is a matrix of shape (D, K), b is of shape $(1, K)$, $\mathbf{v}$ is of shape $(K, 1)$ and c is a scalar. The function $\sigma(z) = \frac{1}{1+\exp(-z)}$.

The matrix multiplication $\mathbf{xw}$ requires $O(DK)$ time and produces a vector of shape $(1, K)$. Adding the bias vector requires $O(K)$ time and produces a vector of shape $(1, K)$. Applying the sigmoid function to a single input is an $O(3)$ operation, so applying it to all K inputs takes $O(K)$ time and produces the vector of hidden unit values $\mathbf{h}$ of shape $(1, K)$. The multiplication of $\mathbf{h}$ by $\mathbf{v}$ requires $O(K)$ time and produces a real scalar as output. Finally, the addition of the scalar c takes $O(1)$ time. Predicting the class simply requires checking if the value of $g_\theta(\mathbf{x}) > 0$, which costs $O(1)$. The total complexity is thus $O(DK) + O(K) + O(K) + O(K) + O(1) + O(1) = O(DK + K)$.

3. (*10 points*) **Recurrent Neural Networks:** Describe the primary capability that an RNN has that an MLP does not have. Explain how the RNN architecture enables this capability.

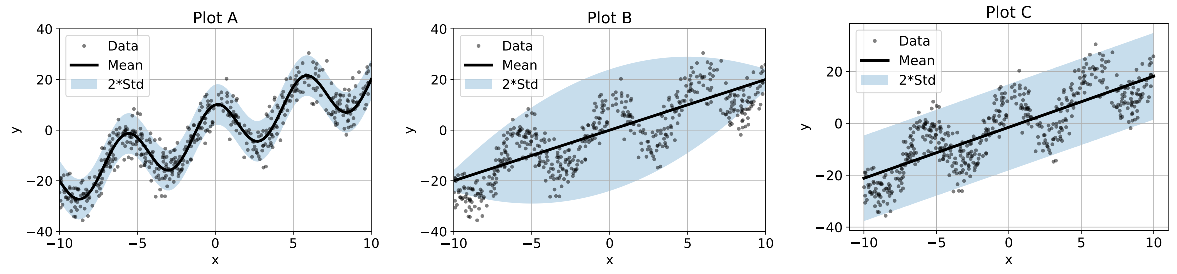
Example Solution: The primary capability than an RNN has that an MLP lacks is the ability to make predictions for inputs of different lengths. An MLP by contrast requires an input $\mathbf{x}$ of fixed dimension D. The RNN architecture accomplishes this by iteratively processing a sequence of input vectors $\mathbf{x}_1, ..., \mathbf{x}_L$. Each vector in the sequence must have the same size D, but the sequence can be any length L. For each position i in the input sequence, the RNN uses the same set of parameters θ to produce a new set of hidden unit values $\mathbf{h}_i$ using the data $\mathbf{x}_i$ at position i and the previous hidden unit values $\mathbf{h}_{i-1}$. Similarly, the RNN uses the same set of parameters ϕ to optionally produce an output y_i for each sequence position i. The combination of processing each position of the input sequence one at a time, producing outputs one at a time, and using the same parameters for every position in the sequence allows the RNN to produce predictions for sequences of any length.

4. (*10 points*) **Generalized Probabilistic Regression:** Consider the Laplace distribution $p(Z = z|\phi) = \frac{1}{2\lambda}\exp\left(-\frac{1}{\lambda}|z-\mu|\right)$ where $z \in \mathbb{R}$, $\mu \in \mathbb{R}$ and $\lambda \in \mathbb{R}^{>0}$. The collection of model parameters is thus $\phi = [\mu, \lambda]$. he Laplace distribution satisfies $\mathbb{E}_{p(Z|\phi)}[z] = \mu$ and has variance $2\lambda^2$. Use the Laplace distribution to define a generalized linear regression model $p(Y = y|\mathbf{X} = \mathbf{x}, \theta)$. Assume $y \in \mathbb{R}$, $\mathbf{x}$ is a $1 \times D$ real vector and θ is the collection of model parameters (you will need to define specific parameters). In the model that you define, the conditional mean of y given $\mathbf{x}$ should be a linear function of $\mathbf{x}$, but the variance of y should not depend on $\mathbf{x}$. Explain your answer.

Example Solution: The mean parameter of the Laplace distribution is μ. The problem requires that the conditional mean of y given $\mathbf{x}$ should be a linear function of $\mathbf{x}$. This means that we need to replace μ in the Laplace distribution with a linear function of $\mathbf{x}$. We use the function $\mathbf{xw} + b$ where $\mathbf{w}$ is a $D \times 1$ real vector and b is a real scalar. The problem requires that the variance of y not depend on $\mathbf{x}$, so we represent it using the parameter λ as in the base Laplace distribution. The final model is as shown below where $\theta = [\mathbf{w}, b, \lambda]$.

$$p(Y = y|\mathbf{X} = \mathbf{x}, \theta) = \frac{1}{2\lambda}\exp\left(-\frac{1}{\lambda}|z - (\mathbf{xw} + b)|\right)$$

5. (*10 points*) **Heteroskedastic Regression:** Consider the heteroskedastic regression model $p(Y = y|X = x, \theta) = \mathcal{N}(y; xw + b, \log(1 + \exp(xv + c))^2)$. In this model $x \in \mathbb{R}$, $y \in \mathbb{R}$, $w \in \mathbb{R}$, $b \in \mathbb{R}$, $v \in \mathbb{R}$, $c \in \mathbb{R}$. The model parameters are $\theta = [w, v, b, c]$. Recall that the notation $\mathcal{N}(z; \mu, \sigma^2)$ indicates the normal probability density function with mean μ and standard deviation σ evaluated at $z \in \mathbb{R}$. Which of the plots below is most likely to correspond to a visualization of the maximum likelihood fit of the model $p(Y = y|X = x, \theta)$ to the data shown. Explain your answer.



Example Solution: The best choice is (C). First, the conditional mean of y given x is specified as $xw + b$, which is a linear function of x. This means that the mean of $p(Y = y|X = x, \theta)$ must be a straight line as a function of x. This eliminates (A) as an option.

Next consider (B). Here the conditional mean relationship is linear. However, we can see that that marginal standard deviation first increases and then decreases as a function of x. Looking at the model, we see that the standard deviation is given by $\log(1 + \exp(xv + c))$. The softplus function is a monotonic function of its input and the input is a linear and thus monotonic function of x. This means that the standard deviation has to either be an increasing function of x, a decreasing function of x or a constant function of x, it can not increase and then decrease as a function of x. This eliminates (B) as an option.

Considering (C), we can see that the conditional mean is linear in x and the standard deviation is constant. This is representable in the model using $v = 0$. Considering the fit, we can see that the structure in the data is interpreted by the model as random variation about the conditional mean which must be the case since the mean is constrained to be a linear function of x. Further, the shaded two standard deviation uncertainty region covers most of the data, as we would expect for a properly fit conditional normal model. As a result, (C) is the best choice.

6. (*10 points*) **Product of Marginals:** Consider the product of normal marginals model defined as $P(\mathbf{X}_n = \mathbf{x}_n|\theta) = \prod_{d=1}^{D} \mathcal{N}(\mathbf{x}_{nd}; \mu_d, \sigma_d^2)$ where $\mathbf{x}_n = [\mathbf{x}_{n1}, ..., \mathbf{x}_{nD}]$, $\theta = [\mu_1, .., \mu_D, \sigma_1, ..., \sigma_D]$. Given a data set $\mathcal{D} = \{\mathbf{x}_n | 1 \leq n \leq N\}$, explain how we can learn the parameters of this model. Provide supporting equations.

Example Solution: We can learn the model by minimizing the negative log likelihood (NLL). We have that $\log P(\mathbf{X}_n = \mathbf{x}_n|\theta) = \sum_{d=1}^{D} \log \mathcal{N}(\mathbf{x}_{nd}; \mu_d, \sigma_d^2)$. We form the log likelihood by summing the log of the joint distribution over the data set:

$$L(\mathcal{D}, \theta) = \sum_{n=1}^{N} \sum_{d=1}^{D} \log \mathcal{N}(\mathbf{x}_{nd}; \mu_d, \sigma_d^2)$$

To learn the model, we could simply apply automatic differentiation to the NLL to obtain a gradient and then minimize the NLL. To do that, we need to make sure the standard deviation parameters are constrained to be positive. We can accomplish this using a parameter transformation such as $\sigma_d = \exp(\eta_d)$, where η_d is an unconstrained real-valued parameter.

However, we can also note that the problem decouples into D different optimization problems, one for each dimension D when we take the partial derivatives with respect to the parameters. That allows us to easily analytically minimize the NLL. We first observe that $\log \mathcal{N}(\mathbf{x}_{nd}; \mu_d, \sigma_d^2) = -\frac{1}{2}\log(2\pi\sigma_d^2) - \frac{1}{2\sigma_d^2}(\mathbf{x}_{nd} - \mu_d)^2$. We use this to solve for the optimal model parameters:

$$\begin{aligned}
\frac{\partial}{\partial \mu_d} L(\mathcal{D}, \theta) &= -\sum_{n=1}^{N} \frac{1}{2\sigma_d^2}(\mathbf{x}_{nd} - \mu_d)(2)(-1) = 0 \\
&\rightarrow 2\sum_{n=1}^{N} \frac{1}{2\sigma_d^2}\mu_d = 2\sum_{n=1}^{N} \frac{1}{2\sigma_d^2}\mathbf{x}_{nd} \\
&\rightarrow N\mu_d = \sum_{n=1}^{N} \mathbf{x}_{nd} \\
&\rightarrow \hat{\mu}_d = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_{nd}
\end{aligned}$$

To estimate σ_d we fix μ_d to it's estimate $\hat{\mu}_d$ and solve the gradient equation:

$$
\begin{aligned}
\frac{\partial}{\partial \sigma_d^2} L(\mathcal{D}, \theta) &= -\frac{N}{2}\frac{1}{\sigma_d^2} + \sum_{n=1}^{N} \frac{1}{2\sigma_d^4}(\mathbf{x}_{nd} - \hat{\mu}_d)^2 = 0 \\
&\rightarrow \frac{N}{2}\frac{1}{\sigma_d^2} = \frac{1}{2\sigma_d^4}\sum_{n=1}^{N}(\mathbf{x}_{nd} - \hat{\mu}_d)^2 \\
&\rightarrow N\sigma_d^2 = \sum_{n=1}^{N}(\mathbf{x}_{nd} - \hat{\mu}_d)^2 \\
&\rightarrow \hat{\sigma}_d^2 = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_{nd} - \hat{\mu}_d)^2
\end{aligned}
$$

7. (*10 points*) **Scale Mixtures:** A scale mixture is a generalized form of mixture model where the latent variable z is continuous. A scale mixture of normals model has the model structure $p(X = x, Z = z|\theta) = p(X = x|Z = z, \mu)p(Z = z|\lambda)$ where $p(X = x|Z = z, \mu) = \mathcal{N}(x; \mu, z^2)$ and $p(Z = z|\lambda) = \lambda \exp(-\lambda z)$, The constraints are $x \in \mathbb{R}$, $z \in \mathbb{R}^{>0}$, $\mu \in \mathbb{R}$, $\lambda \in \mathbb{R}^{>0}$. The parameters are $\theta = [\mu, \lambda]$. Provide an expression for $p(Z = z|X = x, \theta)$ under this model and explain your answer.

Example Solution: Using the definition of conditioning, we have $p(Z = z|X = x, \theta) = p(X = x, Z = z|\theta)/p(X = x|\theta)$. From the model specification, we have that $p(X = x, Z = z|\theta) = p(X = x|Z = z, \mu)p(Z = z|\lambda)$. To obtain $p(X = x|\theta)$, we need to apply marginalization to the joint distribution $p(Z = z, X = x|\theta)$. Since in this model z is a real valued random variable, we marginalize over z using integration. We have that $p(X = x|\theta) = \int p(Z = z', X = x|\theta)dz'$. Putting these pieces together, we have:

$$p(Z = z|X = x, \theta) = \frac{p(X = x|Z = z, \mu)p(Z = z|\lambda)}{\int p(X = x|Z = z', \mu)p(Z = z'|\lambda)dz'}$$

$$= \frac{\mathcal{N}(x; \mu, z^2)\lambda \exp(-\lambda z)}{\int \mathcal{N}(x; \mu, z'^2)\lambda \exp(-\lambda z')dz'}$$

8. (*10 points*) **Discrete Non-Linear Factor Analysis:** In the standard non-linear factor analysis model $p(\mathbf{X}=\mathbf{x},\mathbf{Z}=\mathbf{z}) = p(\mathbf{X}=\mathbf{x}|\mathbf{Z}=\mathbf{z})p(\mathbf{Z}=\mathbf{z})$ with $p(\mathbf{X}=\mathbf{x}|\mathbf{Z}=\mathbf{z}) = \mathcal{N}(\mathbf{x}; f_\theta(\mathbf{z}), \Psi)$ and $p(\mathbf{Z}=\mathbf{z}) = N(\mathbf{z}; 0, I)$. In this model, $\mathbf{x} \in \mathbb{R}^D$, $\mathbf{z} \in \mathbb{R}^K$, Ψ is a $D \times D$ positive diagonal matrix and $f_\theta : \mathbf{R}^K \to \mathbf{R}^D$ is a non-linear neural network generator model with parameters θ. Suppose we are instead interested in a model where the latent code $\mathbf{z} \in \{0,1\}^K$ is a length K binary vector. To accomplish this, we redefine the distribution on $\mathbf{z}$ to $P(\mathbf{Z}=\mathbf{z}) = \prod_{k=1}^K \pi^{z_k}(1-\pi)^{(1-z_k)}$ for a parameter $0 \leq \pi \leq 1$ and leave the distribution $p(\mathbf{X}=\mathbf{x}|\mathbf{Z}=\mathbf{z})$ unchanged. Assuming all parameters are known, show how under this new model we can exactly compute $P(\mathbf{Z}=\mathbf{z}|\mathbf{X}=\mathbf{x})$. Explain your answer. (Hint: the solution does not require an inference network, variational inference or stochastic approximation).

Example Solution: Using the definition of conditioning we have $P(\mathbf{Z}=\mathbf{z}|\mathbf{X}=\mathbf{x}) = P(\mathbf{Z}=\mathbf{z},\mathbf{X}=\mathbf{x})/p(\mathbf{X}=\mathbf{x})$. There is no problem computing the full joint distribution on $\mathbf{z}$ and $\mathbf{x}$ since $\mathbf{z}$ is fully specified. Now consider the computation of the marginal $p(\mathbf{X}=\mathbf{x})$. In the case where $\mathbf{z}$ is real-valued, the marginalization operation requires computing an integral, which is not tractable. In this model, the $\mathbf{z}$ is binary and thus discrete. This means we must sum out over all joint configurations of the binary vector $\mathbf{z}$ to marginalize over it. The final answer is given below:

$$\begin{aligned}
P(\mathbf{Z}=\mathbf{z}|\mathbf{X}=\mathbf{x}) &= \frac{P(\mathbf{Z}=\mathbf{z},\mathbf{X}=\mathbf{x})}{p(\mathbf{X}=\mathbf{x})} \\
&= \frac{\mathcal{N}(\mathbf{x}; f_\theta(\mathbf{z}), \Psi) \cdot \prod_{k=1}^K \pi^{z_k}(1-\pi)^{(1-z_k)}}{\sum_{\mathbf{z}'\in\{0,1\}^K} \mathcal{N}(\mathbf{x}; f_\theta(\mathbf{z}'), \Psi) \cdot \prod_{k=1}^K \pi^{z'_k}(1-\pi)^{(1-z'_k)}} \\
&= \frac{\mathcal{N}(\mathbf{x}; f_\theta(\mathbf{z}), \Psi) \cdot \prod_{k=1}^K \pi^{z_k}(1-\pi)^{(1-z_k)}}{\sum_{z'_1=0}^{1} \cdots \sum_{z'_K=0}^{1} \mathcal{N}(\mathbf{x}; f_\theta(\mathbf{z}'), \Psi) \cdot \prod_{k=1}^K \pi^{z'_k}(1-\pi)^{(1-z'_k)}}
\end{aligned}$$

9. (*10 points*) **Implementation: Implementation:** Consider the probabilistic binary logistic regression model objective function shown below. In this model, each data case $\mathbf{x}_n$ is a $1 \times D$ vector, $y_n \in \{0, 1\}$, $\mathbf{w}$ is a $D \times 1$ real vector and b is a real scalar. Provide a computationally efficient implementation of this function using Python and PyTorch-like tensor operations including tensor arithmetic (multiplication, addition, etc.) and basic element-wise functions (log, exp, powers, etc.). The function must take as input the feature matrix X, a label vector Y, the weight matrix w and the bias b and return as output the objective function value for the binary logistic regression model objective function. You may provide your answer as one function or several functions. Explain your answer and be sure to indicate your assumptions about the shapes and data types of the inputs to the function.

$$-\frac{1}{N}\sum_{n=1}^{N}\log\left(\left(\frac{1}{1+\exp(-(\mathbf{x}_n\mathbf{w}+b))}\right)^{y_n}\cdot\left(1-\frac{1}{1+\exp(-(\mathbf{x}_n\mathbf{w}+b))}\right)^{1-y_n}\right)$$

Example Solution: To start we can break this computation down into the computation of the sigmoid function, the computation of a linear layer, and the computation of the rest of the objective function. The sigmoid function applies elementwise to any tensor. The linear layer takes an X tensor of shape (N, D), a weight vector of shape $(D, 1)$ and a scalar bias. The linear layer uses the torch matrix multiplication operator @ and broadcasting to add the bias. The objective function first computes the probability corresponding to the output of the sigmoid function applied to the linear layer. It then computes a vector of cross entropy terms using the labels and the log probabilities and takes the mean of this vector.

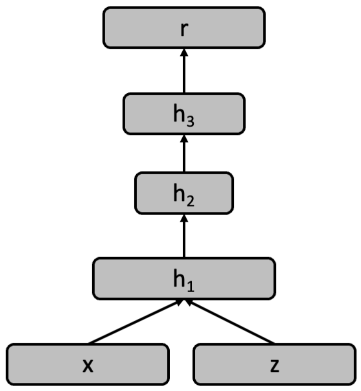
```
def sigmoid(Z):
    return 1/(1+torch.exp(-Z))

def liniear_layer(X,w,b):
    return X@w + b

def obj(X,Y,w,b):
    P = sigmoid(liniear_layer(X,w,b))
    CE = -1*Y*(torch.log(P)) - (1-Y)*torch.log(1-P)
    return CE.mean()
```

10. (*10 points*) **Modeling:** A depth camera is an imaging technology that simultaneously captures a regular RGB image $\mathbf{x}$ and a depth map $\mathbf{z}$ of the same scene. The RGB image is represented as an $H \times W \times 3$ tensor giving the color of the object represented at each pixel location (i, j), while the depth map $\mathbf{z}$ indicates the distance from the camera to the object at each pixel location (i, j). Assume that we have a data set from such a camera consisting of $(\mathbf{x}_n, \mathbf{z}_n)$ pairs. One issue with such data is that the depth maps are often much nosier than the corresponding RGB images. How can we build a machine learning model that denoises the depth maps? Specify (1) the model structure you would use, (2) why you propose to use this structure, (3) what the key parameters and key hyper-parameters of the model are, (4) how the model will be used to denoise depth maps once trained, (5) what objective function you will use to learn the model, and (6) how you will select any hyper-parameters.

Example Solution: (1) Since both the RGB images and the depth images are of the same scene and the RGB images are less noisy, we will construct a model that takes both the RGB and depth images as input, and produces a depth image as output. We will use an autoencoder-like model with the architecture shown below. Since the depth values are non-negative, we will use ReLU activations at all layers including the output. All layers will be fully connected. The final output $\mathbf{r}$ denotes the reconstructed depth image only.



(2) We propose this structure as it allows for a shared code layer between the RGB images and the depth images. The hope is that by conditioning on both image types, the model could learn to use the cleaner RGB images to help identify object edges and smooth surfaces to help clean up the signal from the depth images. The model has has a narrow middle code layer to further help eliminate noise from the depth image.

(3) Each connection shown in the diagram corresponds to a linear layer and has a weight matrix and bias. The key hyper-parameters are the hidden layer widths, which we will call K_1, K_2, K_3. Since the first hidden layer needs to encode information from $\mathbf{x}$ and $\mathbf{z}$, let let it be the largest, followed by h_3 and h_2. We will thus require that $K_2 < K_3 < K_1$.

(4) Once the model is trained, we will provide it with an RGB image and corresponding

depth image and use a standard forward pass to produce the reconstructed depth image $\mathbf{r}$.

(5) To learn the model we will use the MSE between the reconstructed depth image and the input depth image.

(6) To select the layer width hyper-parameters we will use a validation set approach. We will define a grid of layer widths that satisfy the ordering constraint $K_2 < K_3 < K_1$, train a model for each hyper-parameter setting in the grid, evaluate the model on the validation set using MSE, and select the set of layer-width values with the lowest validation set MSE.