CS 688: Machine Learning
George Mason University, Fall 22 **Homework 3** **Due at 11:59pm, Nov 15**

Problems marked with **E** are graded on effort, which means that they are graded subjectively on the perceived effort you put into them, rather than on correctness. We strongly encourage you to typeset your solutions in LaTeX.

1. (40 points (bonus)) Suppose that we want to answer a counting query (e.g., how may student

$$f(\vec{x}) = \sum_{i=1}^{n} x_i$$

where $\vec{x} = (x_1, \ldots, x_n)$ and $x_i \in \{0, 1\}$ for all $i = 1, \ldots, n$. In week 7, we learned the Laplace mechanism adding Laplace noise with scale parameter $1/\epsilon$ is ϵ-differentially private. Formally, $M_{Lap}(\vec{x}) = f(\vec{x}) + Z$ where the Laplace noise Z equals $z \in \mathbb{R}$ with probability $\Pr[Z = z] = \frac{\epsilon}{2}\exp(-\epsilon|z|)$. However, the output of the Laplace mechanism is generally a real number which is unnatural for counting query (e.g., there are 3.14 students in the class) In this problem, we will study discrete variants of the Laplace mechanism that can always output an integer.

   (a) Given $p \in [0, 1]$ and $\epsilon > 0$, consider the following double geometric mechanism

$$M_{DGeo}(\vec{x}) = f(\vec{x}) + W \text{ where the noise } \Pr[W = w] = \frac{1-p}{1+p}p^{|w|}, \text{ for all } w \in \mathbb{Z}.$$

   What is the minimum possible $p \in [0, 1]$ if we want M_{DGeo} to be ϵ-differentially private?

   (b) We can also use an exponential mechanism to ensure integer output. Formally, we set a scoring function q so that $q(\vec{x}, r) = -|f(\vec{x}) - r|$ for all $r \in \mathbb{Z}$ and datasets $\vec{x}$, and

$$M_{Exp}(\vec{x}) = y, \text{ with probability proportional to } \exp(\frac{\epsilon q(\vec{x}, y)}{2\Delta_q}).$$

   In class, we learn the exponential mechanism can be reduce to a Laplace mechanism when the output is real numbers. Here we would like to simplify M_{Exp}. Specifically, find the distribution of $\hat{W}$ that is independent of $\vec{x}$ so that $M_{Exp}(\vec{x}) = f(\vec{x}) + \hat{W}$ for all input $\vec{x}$.

   (c) The above mechanism ensures that the output is always an integer, but can output negative values (e.g., -2 students in the class). To address this, one may choose some $\rho \in [0, 1]$, sample a noise $\tilde{W}$ so that $\Pr[\tilde{W} = k] = \rho(1-\rho)^k$ for all $k = 1, \ldots,$ and output $\tilde{M}(\vec{x}) = f(\vec{x}) + \tilde{W}$. Can $\tilde{M}$ be ϵ-differentially private for some ρ? *Hint: what happen when $f(\vec{x}) = 0$ and $f(\vec{x}') = 1$*

2. (40 points (bonus)) Consider a classification problem for admission with feature vector (a, x) where $a \in \{0, 1\}$ is sensitive feature (e.g., race) and $x \in \mathbb{R}$ is the other feature (e.g., SAT score), and outcome $t \in \{0, 1\}$ where a, x, t is sampled from a known distribution P. The goal is to decide a "fair" classifier f that output the admission decision $f(x, a) \in \{0, 1\}$ for a student with feature (x, a).

   (a) Given the distribution P on a, x, t, find a classifier f_D so that the positive rate from two groups are equal,

$$\Pr_{a,x,t\sim P}[f_D(a, x) = 1 | a = 0] = \Pr_{a,x,t\sim P}[f_D(a, x) = 1 | a = 1].$$

   (b) Similarly given P, find a classifier f_E so that has equalized odds,

$$\Pr_{a,x,t\sim P}[f_E(a, x) = 1 | a = 0, t = 1] = \Pr_{a,x,t\sim P}[f_E(a, x) = 1 | a = 1, t = 1], \text{ and}$$
$$\Pr_{a,x,t\sim P}[f_E(a, x) = 1 | a = 0, t = 0] = \Pr_{a,x,t\sim P}[f_E(a, x) = 1 | a = 1, t = 0].$$

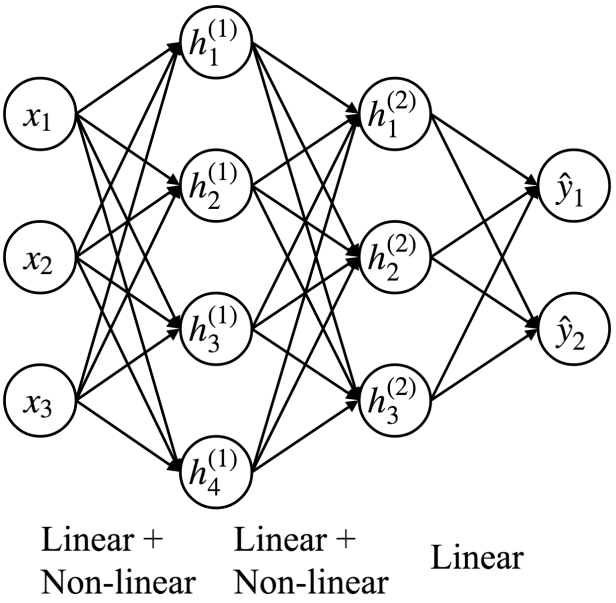

Figure 1: An instance of neural network with two hidden layer

(c) If a classifier f^* has equal positive rate and equalized odd on P and $\Pr[t = 1|a = 0] \neq \Pr[t = 1|a = 1]$, show that the output of f^* is independent of the outcome t and sensitive feature a. Formally, there exists ρ so that

$$\Pr_{a,x,t\sim P}[f^*(a, x) = 1|a = \alpha, t = \tau] = \rho$$

for all $\alpha, \tau \in \{0, 1\}$.

(d) Finally, given above P compute the optimal zero one error of a classifier f^* that has equal positive rate and equalized odd.

3. (30 points) In this question, we will learn how to compute forward propagation using matrix operation, and study the computational complexity.

Suppose that multiplying a matrix of size $n \times k$ by a matrix of $k \times d$ take times $O(nkd)$, and the time to evaluate an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is C_σ. Additionally, though $\sigma(\cdot)$ is defined as a mapping on a scalar, we overload notation and allow us to apply σ to a vector (or matrix) component-wise. For instance, if v is a vector of length d_v then $\sigma(v)$ will be a vector of length d_v and the computational time is $d_v C_\sigma$.

Assume that we have two hidden layered neural network as in Fig 1 where the first hidden level has parameter $W^{(1)} \in \mathbb{R}^{d^{(0)} \times d^{(1)}}$, the second has $W^{(2)} \in \mathbb{R}^{d^{(1)} \times d^{(2)}}$, and the final one has $W^{d^{(2)} \times d^{(3)}}$. Note that $d^{(0)}$ equals the input dimension d and $d^{(3)} = k$ is the output dimension.

(a) Given the input vector $x \in \mathbb{R}^d$, write out pseudocode for the forward propagation in terms of matrix multiplication where you apply the activation function to the vector of activations. Specify the dimensions of all the activations and outputs in terms of $d^{(0)} = d, d^{(1)}, d^{(2)}$ and $d^{(3)} = k$.

(b) Now suppose we want to run the forward propogation on a mini-batch of inputs of size m. Let X be a $m \times d$ matrix so that each row of X corresponds to an input vector. Write out the forward pass in matrix form (you should apply σ componentwise and the output

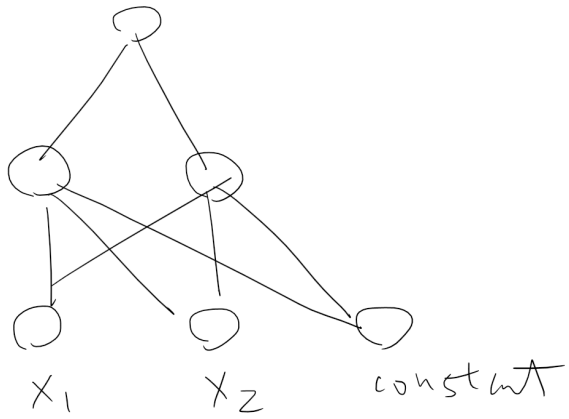

Figure 2: A neural neural network for XOR function

is a matrix of size $m \times k$. Specify the size of all the activation and outputs in terms of m $d^{(0)} = d, d^{(1)}, d^{(2)}$ and $d^{(3)} = k$.

(c) Using the above algorithm, what is the computational time to compute the square loss on the mini-batch X is terms of d, the batch size m and the number of nodes per level $d^{(1)}, d^{(2)}, d^{(3)}$, and the transfer function evaluation cost C_σ?

4. (30 points) Let us consider the parity problem (XOR problem) with two variables. In class, we saw this problem is not linear-separable. In this problem, we ask how can we design an neural network to achieve zero error. Formally, given two dimensional inputs $x = (x_1, x_2) \in \{-1, 1\}^2$, $XOR(x) = \mathbf{1}[x_1 \neq x_2]$.

   (a) Give a feature mapping $\phi(x)$ using only linear and quadratic terms so that the linear classifier $\text{sign}(w^\top \phi(x))$ for some vector w can represent XOR function with 0 zero-one loss for all $x \in \{-1, 1\}^2$.

   (b) Provide a one hidden layer neural network with $\tanh(\cdot)$ activation function that has no more than two hidden nodes which can represent this function of 0 zero-one loss shown in Fig. 2. Specify the parameter in each level, $W^{(1)} \in \mathbb{R}^{3\times 2}$ and $W^{(2)} \in \mathbb{R}^{2\times 1}$. You may assume your prediction is $\text{sign}(\hat{y})$ (you are not allowed break ties in your favor if at 0). Specifically, you must provide your weights, and you are free to use a bias term in your activations.

5. (E 60 points) You will use the dataset of mnist 2 vs 9 (the same as we used in homework 2). Use a one-hidden layer network with $d^{(1)} = 100$ hidden nodes with sigmoid activation functions. The output of the neural net will then be a linear transformation of the outputs of layer 1, i.e., $\hat{y}(x) = w^{(2)} z^{(1)}$ where $w^{(2)}$ is a $2 \times d^{(1)}$ sized matrix. Finally, we will use a sigmoid function on the output to get a prediction.

   You are free to use PyTorch or directly write your own backprop code. Please note which method you used. You will have 2 output nodes, one for class 2 and class 9. You must use (mini-batch) SGD for this problem. We do expect you to find a way to get reasonable performance (compared to what is achievable for the given architecture). Training neural nets can be a bit of an art, until you get used to it. Use regularization if you find it helpful.

   (a) Run stochastic (mini-batch) gradient descent. Specify all your parameter choices: mini-batch size, regularization parameter (if any), and learning rates (if you alter the learning

rates, make sure you precisely state when it is decreased). Also, specify how you initialize your parameters.

(b) Which learning rate do you start observing a non-trivial decrease in your cross entropy error?

(c) Make a plot showing your average cross entropy error for both your training and your validation sets on the y-axis and have the iteration on the x-axis. Both curves should be on the same plot.

(d) Make this plot again (with both curves), except use the zero-one error, as a percentage. Here, make sure to start your x-axis at a slightly later iteration, so that your error starts below 20%, which makes the behavior more easy to view.

(e) Again, it is expected that you obtain a good test error (meaning you train long enough and you regularize appropriately, if needed). Report your lowest test error.