NYU TANDON SCHOOL OF ENGINEERING

Deep Learning Fall 2023

Prof. Gustavo Sandoval

Due Date: **5pm Friday September 29, 2023**

- You are encouraged to discuss ideas with each other. But you **must acknowldge** any collaborators, including students, ChatGPT, books, online sources, etc. and you **must** write up your own solutions individually.
- We **require** answers to theory questions to be writen in LaTeX. Figures can be drawn by hand, but **must** be scanned and inserted into your document. We will not accept hand-written answers.
- We **require** code for programming questions to be submitted as a Jupyter notebook. It is important to include sufficient documentation so that the grader can understand your code. Use the text cells in Jupyter notebook to provide explanations.
- Upload your .pdf, .ipynb and any other files to Gradescope in a single PDF.

Problem 1: Warmup: Fun with vector calculus. (8 pts)

(a) If x is a d-dimensional vector variable, write down the gradient of the function $f(x) = \|x\|_2^2$.

(b) Suppose we have n data points that are real d-dimensional vectors. Analytically derive a constant vector μ for which the following function is minimized:

$$\sum_{i=1}^{n} \|x_i - \mu\|_2^2$$

Problem 2: Linear Regression with non-standard losses (12 pts)

In class we derived an analytical expression for the optimal linear regression model using the least squares loss. In this problem we will consider a few other loss functions. If X is the matrix of n training data points (stacked as rows) and y is the vector of n target values, then:

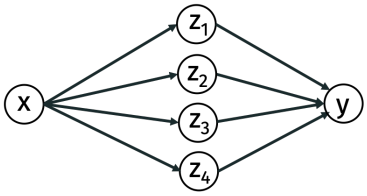
(a) Using matrix/vector notation, write down a loss function that measures the training error in terms of the l_1-norm. Write down that sizes of all matrices/vectors.

(b) Can you simply write down the optimal linear model in closed form, as we did for the standard linear regression model? If so, do it. If not, explain why not.

(c) In about 2-3 sentences, reflect on how you solved this question, and relate any aspects of this to the 3-step recipe for machine learning we discussed in class.

Problem 3: Neural Networks for Curve Fitting (20 pts)

Consider the following 2-layer, feed forward neural network for single variate regression:

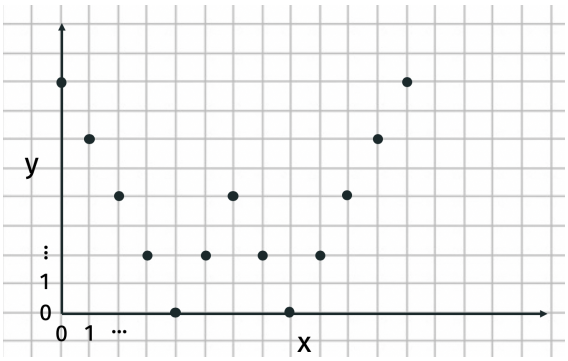


Let $W_{H,1}, W_{H,2}, W_{H,3}, W_{H,4}$ and $b_{H,1}, b_{H,2}, b_{H,3}, b_{H,4}$ be weights and biases for the hidden layer. Let $W_{O,1}, W_{O,2}, W_{O,3}, W_{O,4}$ and b_O be weights and bias for the output layer. The hidden layer uses rectified linear unit (ReLU) non-linearities and the output layer uses no non-linearity.

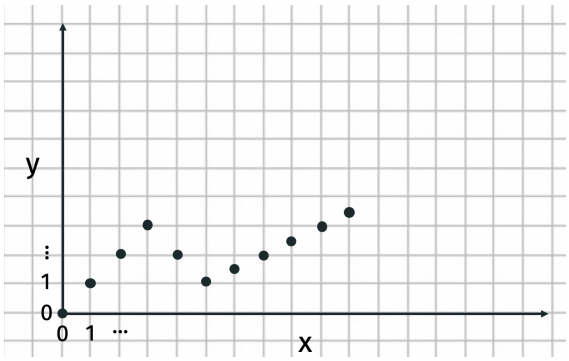
Specifically, for $i = 1, \ldots, 4$, $z_i = \max(0, \bar{z}_i)$ where $\bar{z}_i = W_{H,i}x + b_{H,i}$. And

$$y = b_O + \sum_{i=1}^{4} W_{O,i} z_i.$$

(a) For each of the two datasets below, determine values for weights and biases which would allow this network to perfectly fit the data.



Dataset 1



Dataset 2

(b) For input parameters $\vec{\theta}$ let $f(x, \vec{\theta})$ denote the output of the neural network for a given input x. We want to train the network under the squared loss. Specifically, given a training dataset $(x_1, y_1), \ldots, (x_n, y_n)$, we want to choose $\vec{\theta}$ to minimize the loss:

$$\mathcal{L}(\vec{\theta}) = \sum_{i=1}^{n} (y_i - f(x_i, \vec{\theta}))^2.$$

Write down an expression for the gradient $\nabla\mathcal{L}(\vec{\theta})$ in terms of $\nabla f(x, \vec{\theta})$. **Hint:** Use chain rule.

(c) Suppose we randomly initialize the network with ± 1 random numbers:

$$\begin{aligned} W_{H,1} = -1, W_{H,2} = 1, W_{H,3} = 1, W_{H,4} = -1 \\ b_{H,1} = 1, b_{H,2} = 1, b_{H,3} = -1, b_{H,4} = 1 \\ W_{O,1} = -1, W_{O,2} = -1, W_{O,3} = -1, W_{O,4} = 1 \\ b_O = 1 \end{aligned}$$

Call this initial set of parameter $\vec{\theta}_0$. Use forward-propagation to compute $f(x, \vec{\theta}_0)$ for $x = 2$.

(d) Use back-propagation to compute $\nabla f(x, \vec{\theta}_0)$ for $x = 2$. To do the computation you will need to use the derivative of the ReLU function, $\max(0, z)$. You can simply use:

$$\frac{\partial}{\partial z} \max(0, z) = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{if } z > 0 \end{cases}$$

This derivative is discontinuous, but it turns out that is fine for use in gradient descent.

Problem 4: Improving the FashionMNIST classifier (30 pts)

In the first demo, we classified the MNIST dataset with a logistic regression architecture. Now, we'll use a neural network to improve the performance.

Repeat the same experiment in the demo but with the FashionMNIST dataset. You can load the dataset using the following pytorch code:

```
trainingdata = torchvision.datasets.FashionMNIST('./FashionMNIST/',
        train=True,download=True,transform=torchvision.transforms.ToTensor())

testdata = torchvision.datasets.FashionMNIST('./FashionMNIST/',
        train=False,download=True,transform=torchvision.transforms.ToTensor())
```

Now use a (dense) neural network with three hidden layers. The first layer should have 256 neurons, the second should have 128, and the third should have 64, all with ReLU activations. Remember to then use one last layer so the output is 10-dimensional (for the 10 classes). Display train- and test-loss curves. You may have to tweak the total number of training epochs to get reasonable performance.

Finally, sample any three images from the validation set and visualize the predicted class probabilities for each sample. Comment on what you observe from the plots.

Problem 5: Implementing Backpropagation (30 pts)

Open the (incomplete) Jupyter notebook provided as an attachment to this homework in Google Colab (or other Python IDE of your choice) and complete the missing items.

In the second demo, we will work with autodiff. Autodiff enables us to implicitly store how to calculate the gradient when we call backward. We implemented some basic operations (addition, multiplication, power, and ReLU).

In this homework problem, you will implement backprop for more complicated operations directly. Instead of using autodiff, you will manually compute the gradient of the loss function for each parameter.