HW1 - Neural Networks and Deep Learning

due Feb 9, 2023

The programming part must be done on colab. The submission for programming part is a python notebook file that you should upload to canvas. The math question (and any written part) submission must be *one* pdf file.

1 Problem 1

Look at the attached python notebook. It has (x,y) co-ordinate data in tensor X_data and Y_data and the labels (0 or 1) for these 2d points in the tensor Label. (the data is generated with a fixed seed, so you will get the same data as I have). Do the following

1. Write down the formula of the ground truth classifier boundary. The formula is in the code, you just need to write it on paper (and put it in the submission pdf).
2. Using seaborn or matplotlib to plot a picture that shows the two regions separated by the classifier boundary and also plots the datapoints with different colors for the two classes. This is to be done in colab and I will run your python notebook code to check. This code should be in a separate code block. Your picture should look like:

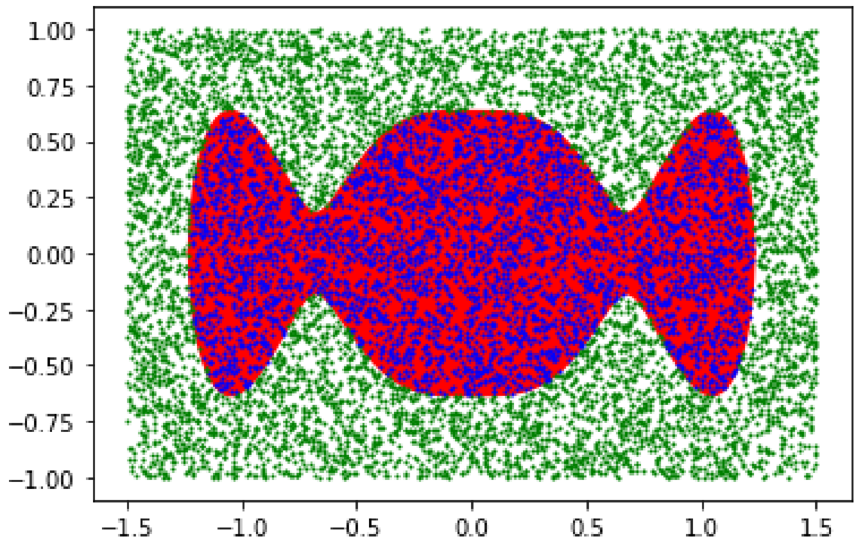

Figure 1: The plot needed

3. Perform a 70:30 split of data into train and test. Your must use the code below with the seed 52 so that the split is same for all of you.

```
generator=torch.Generator().manual_seed(52)
dataset = torch.utils.data.TensorDataset(Feature, Label_tensor)
train_ds, test_ds = torch.utils.data.random_split(dataset, [0.7 ,0.3], generator)
```

Train a neural network classifier using the training data and report the accuracy on test data. For this part,

- Your training of classifier should complete within 5 minutes.
- You must use cpu to train - no gpu.
- Your test accuracy must be over 90%.
- You are free to choose your network, learning rate, etc. I will run your submitted code on colab. You are free to use the code I provided in Week 3 for setting up training of a model.

2 Problem 2 (Theory)

Suppose you have data points $(x^1, y^1), \dots, (x^m, y^m)$. You have a classifier that output the score/logits $\hat{y}^i = (\hat{y}^i_1, \hat{y}^i_2) = F_w(x^i)$. These logits go into a loss function $L(\hat{y}^i, y^i)$ to give the loss for the datapoint (x^i, y^i). The overall loss is

$$L = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^i, y^i) = \frac{1}{m}\sum_{i=1}^{m} L(F_w(x^i), y^i)$$

The gradient of above w.r.t. weights w is

$$G = \nabla_w L = \frac{1}{m}\sum_{i=1}^{m} \nabla_w L(F_w(x^i), y^i)$$

In SGD, you sampled one datapoint at random, say $(\mathbf{x}, \mathbf{y})$, and compute the gradient $G(\mathbf{x}, \mathbf{y}) = \nabla_w L(F_w(\mathbf{x}), \mathbf{y})$ In mini-batch SGD you sample k such points at random $(\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_k, \mathbf{y}_k)$ and get the gradient as

$$G((\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_k, \mathbf{y}_k)) = \frac{1}{k}\sum_{j=1}^{k} \nabla_w L(F_w(\mathbf{x}_j), \mathbf{y}_j)$$

Show the following, with the working (meaning how you obtained the result):

1. $E[G(\mathbf{x}, \mathbf{y})] = G$
2. $E[G((\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_k, \mathbf{y}_k))] = G$
3. $Var[G(\mathbf{x}, \mathbf{y})] > Var[G((\mathbf{x}_1, \mathbf{y}_1), \dots, (\mathbf{x}_k, \mathbf{y}_k))]$