ECE-GY 6143 / Spring 2020

Homework 4

Please scan and upload your assignments on or before April 2, 2020.

- You are encouraged to discuss ideas with each other; but
- you **must acknowledge** your collaborator, and
- you **must compose your own** writeup and/or code independently.
- We **strongly** encourage answers to theory questions in Latex, and answers to coding questions in Python (Jupyter notebooks).
- Please upload your solutions in the form of a single .pdf or .zip file on NYUClasses.
- Maximum score: 50 points.

---

1. **(10 points)** In class, we discussed how to represent XOR-like functions using quadratic features, since standard linear classifiers (such as perceptrons) are insufficient for this task. However, here we show that XOR-like functions can indeed be simulated using *multi-layer* networks of perceptrons. This example shows a glimpse of the expressive power of "deep neural networks": merely increasing the depth from 1 to 2 layers can help reproduce nonlinear decision boundaries.

   a. Consider a standard two-variable XOR function, where we have 2-dimensional inputs $x_1, x_2 = \pm 1$, and output $y = x_1 (XOR) x_2 = \begin{cases} -1 & \text{if} \quad x_1 = x_2 \\ 1 & \text{otherwise} \end{cases}$.

   Geometrically argue why a single perceptron cannot be used to simulate the above function.

   b. Graphically depict, and write down the equation for, the optimal decision region for the following logical functions:
   (i) $x_1 (AND)(NOT(x_2))$
   (ii) $(NOT(x_1))(AND)x_2$
   (iii) $x_1 (OR) x_2$ Make note of the weights learned corresponding to the optimal decision boundary for each function.

   c. Using the above information, simulate a multi-layer perceptron *network* for the XOR operation with the learned weights from Part (b).

2. **(10 points)** In this problem, we will implement the Perceptron algorithm discussed in class on synthetic training data.

   a. Suppose that the data dimension d equals 2. Generate two clusters of data points with 100 points each, by sampling from Gaussian distributions centered at $(0.5, 0.5)$ and $(-0.5, -0.5)$. Choose the variance of the Gaussian to be small enough so that the data points are sufficiently well separated. Plot the data points on the 2D plane to confirm that this is the case.

   b. Implement the Perceptron algorithm as discussed in class. Choose the initial weights to be zero and the maximum number of epochs as $T = 100$, and the learning rate $\alpha = 1$. How quickly does your implementation converge?

c. Now, repeat the above experiment with a second synthetic dataset; this time, increase the variance (radius) of the two Gaussians such that the generated data points from different classes now overlap. What happens to the behavior of the algorithm? Does it converge? Show the classification regions obtained at the end of T epochs.

3. **(10 points)** In Lectures 2 and 5, we derived a closed form expression for solving linear and ridge regression problems. This is great for finding linear behavior in data; however, if the data is nonlinear, just as in the classification case, we have to resort to the *kernel trick*, i.e., replace all dot products in the data space with kernel inner products. In this problem, we theoretically derive kernel regression. Suppose we are given training data $\{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)$ where each response y_i is a scalar, and each data point x_i is a vector in d dimensions.

   a. Assume that all data have been mapped into a higher dimensional space using the feature mapping $x \mapsto \phi(x)$, write down an expression for the squared error function using a linear predictor w in the high-dimensional space.

   b. Let Φ be the matrix with n rows, where row i consists of the feature mapping $\phi(x_i)$. Write down a closed form expression for the optimal linear predictor w as a function of Φ and y.

   c. For a new query data point z, the predicted value is given by $f(z) = \langle w, \phi(z) \rangle$. Plug in the closed form expression for w from the previous sub-problem to get an expression for $f(z)$.

   d. Suppose you are given black-box access to a kernel dot product function K where $K(x, x') = \langle \phi(x), \phi(x') \rangle$. Mathematically show that all the calculations in (b) and (c) can be performed by invoking the kernel dot product alone *without explicitly writing down* $\phi(x)$ *ever*. You may want to use the Sherman-Morrison-Woodbury identity for matrices:
   $$(A^{-1} + B^T C^{-1} B)^{-1} B^T C^{-1} = A B^T (B A B^T + C)^{-1}.$$

4. **(20 points)** The *Fashion MNIST* dataset is a database of (low-resolution) clothing images that is similar to MNIST but somewhat more challenging. You can load it in Colab using the Python code below.

   a. Load the dataset and display 10 representative images from each class.

   b. Implement the following classification methods: k-NN, logistic regression, and support vector machines (with linear and rbf kernels). You can use sklearn.

   c. Report best possible test-error performances by tuning hyperparameters in each of your methods.

   d. Report train- and test-running time of each of your methods in the form of a table, and comment on the relative tradeoffs across the different methods.

```
import tensorflow as tf
from tensorflow import keras
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels),
(test_images, test_labels) = fashion_mnist.load_data()
```