COSC 522 Machine Learning (Fall 2024)

Project 2 - MNIST Digit Recognition Using Neural Networks (Due 10/09)

Objective:

The objective of this project is to have a thorough understanding of the traditional multi-layer perceptron network and backpropagation.

Reading:

I hope you've enjoyed reading Nielsen's "Neural Network and Deep Learning" as much as I do. Majority of this project is based on Chapters 1-3 of this book. So read carefully!

Data Sets:

MNIST and XOR

Tasks:

- (15 pts) Task 1: The answer to each question should take less than a third of a page, assuming single space and a font size of 12 Times.
    - (2) Explain the differences between batch processing and online processing. What is the size of the mini-batch in each case?
    - (2) Explain the differences between gradient descent and stochastic gradient descent.
    - (2) Explain the differences between perceptron and sigmoid neurons. What's the advantage of a sigmoid neuron as compared to perceptron?
    - (2) Explain the differences between feedforward vs. backpropagation. What is back propagated and what is forwarded?
    - (2) Why do we need to introduce "bias" when training a neural network?
    - (2) What are the differences among training set, validation set, and testing set?
    - (3) We have learned the cost function for Perceptron and MLP (using sigmoid). Why do they employ different functions? What are fundamentally different between these two cost functions?
- (15 pts) Task 2: Write a notebook with just MNIST loader. Apply the four Baysian classifiers you implemented in Project 1. Use "training_data" to train and use "testing_data" to test. Report the overall accuracy assuming equal prior probability, and k = 15 for kNN.
- (25 pts) Task 3: Download the code set from GitHub, debug, and make it run locally (or colab). The code is written for Python 2 and several functions are not compatible in Python 3.

- (10) Task 3.1: Nielsen was able to use a simple structure to train a network to recognize the handwritten digits (MNIST). Use the same setup as he used and plot the convergence curve using MNIST. Note that the convergence curve is a plot of classification accuracy (or error) vs. epoch. Nielsen's code has already output the accuracy at the end of each epoch. You just need to plot it on Figure 1.
- (15) Task 3.2: Play around the hyperparameter setups and draw three figures showing the effect of changing each hyperparameter to the convergence curve.
    - (5) Effect of network structure (Figure 2). On the same figure, show at least five convergence paths using Nielsen's baseline, different number of hidden layers and different number of hidden nodes in each layer. Keep learning rate at 1, mini-batch size at 15, but extend the number of epochs to 100 (or whichever constant you and your computer have patience with). One of the convergent curves need to have 0 hidden layer (as baseline). Provide comments on why different performance is observed if any at all. Identify the one structure that has the best performance.
    - (5) Effect of mini-batch size (Figure 3). Use the network structure with the best performance and modify mini-batch size. Show at least 4 convergence paths with different mini-batch sizes. Comment on the different performances obtained using the different mini-batch size. Identify the size with the best performance.
    - (5) Effect of learning rate (Figure 4). Use the network structure and the mini-batch size with the best performance and change learning rate from 1, 0.1, 0.01, to 0.001. Show the convergence curves. Comment on the different performances obtained using the different learning rate. Identify the rate with the best performance.
- (+15) Task 3.3 (Bonus): In Chapter 2 of Nielson's book, he explained in thorough detail how to realize backpropagation using matrix operations.
    - (10) Why is backpropagation a faster algorithm? The page limit for this question is 1 page. You need to thoroughly understand chapter 2 in order to be able to answer this question in less than a page.
    - (5) Compare the performance of this implementation as compared to the ones in Chapter one. Use a fixed set of hyperparameters so that it's apple-to-apple comparison. Report the results.

- (45) Task 4: Solving the XOR problem. The four training samples are [[-1, -1], [-1, 1], [1, -1], [1, 1]], with the corresponding label being [-1, 1, 1, -1]. You have experienced in HW2 that

single-layer perceptron cannot realize XOR since the decision boundary is not linear in the 2-d space.

- (20) Task 4.1: Use the network library you developed in Task 3 to implement the XOR gate. Going through the practices you had in Task 3.2. Fix mini-batch size at 1. Show effect of network structure, different learning rate, and different initialization strategy. You probably spent more time on XOR than on MNIST. Comment on why it is more difficult to train a smaller network than a larger one.
- (25) Task 4.2: Use kernel method for XOR. Here, we use the kernel trick to solve this problem. Kernel tricks use a kernel function to project the data from the original space (in this case, 2-d space) to a higher-dimensional space where a linear boundary can be found to perfectly separate the samples.
    - (5) Suppose the kernel function is a 2nd-degree polynomial, i.e., $K(\mathbf{x}, \mathbf{y}) = (x_1y_1 + x_2y_2 + C)^2$, where $\mathbf{x} = [x_1, x_2]^T$, $\mathbf{y} = [y_1, y_2]^T$. Derive the basis functions $\phi(.)$, that is, $K(\mathbf{x}, \mathbf{y}) = \phi^T(\mathbf{x}). \phi(\mathbf{y})$
    - (5) Using the derived basis function, what is the higher-dimensional space that the 2-d sample should be mapped to? Provide the higher-dimensional counterpart to the four 2-d samples.
    - (10) Apply Perceptron on the higher-dimensional samples and see if you can find a linear decision boundary using the higher-dimensional samples. Output the weights learned. Project the learned hyperplane onto 2-D space and show the decision boundary.
    - (5) Comment on the two implementations of XOR (Task 4.1 and Task 4.2). Which one do you prefer and why?