Implement multilayer perceptron with Python

Yueqi Li
yueqi.li@uky.edu

March 28, 2022

I. Introduction

The multilayer perceptron is an artificial neural network structure and is a nonparametric estimator that can be used for classification and regression. We implement the backpropagation algorithm to train a multilayer perceptron for classification of handwriting which also known as MNIST.

II. Methods

In this project, we implement the Multilayer perceptron with different activation function and also backpropagation.

i. *Pre-Process*

First, we turn the categorical data to the dummy data, for dataset 1, we use $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ to represent 0 and 1, and we use $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$ to represent 0, 1, 2, 3, 4.

We also dived all value by 255, which will avoid memory overflow when we use the sigmoid activation function

ii. *activation function*

For the activation function, we choose sigmoid, since it's differentiable which is helpful for us to do the backpropagation. The sigmiod function;it's first direviative is follow:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

We take the first derivative for backpropagation, which is:

$$\sigma'(x) = 1 - \sigma(x) = 1 - \frac{1}{1 + e^{-x}} \tag{2}$$

We also implement soft max for the last step to get the prediction, where the function is :

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_j^n e^{z_j}} \tag{3}$$

We also need the derivative of it, which is $\begin{cases} \sigma(z_i) \times (1 - \sigma(z_i)) & \text{if } i = j \\ \sigma(z_i) \times \sigma(z_j) & \text{if } i \neq j \end{cases}$

For the loss function, we choose to use Mean Categorical Cross Entropy:

$$L(y_i) = -\sum_i^n -y_i \cdot \log \hat{y_i} \tag{4}$$

We also need the derivative of it, which is:

$$L'(y_i) = \frac{y_i}{\hat{y}_i} \tag{5}$$

We use Stochastic gradient descent as our optimizer.

ii. *Architecture*

For this project, we use different architecture for those two dataset, for dataset 1, which only has 1 and 0. We turn 782 to 64 by sigmoid, then we use softmax to turn it to 2x2 matrix for the prediction; For dataset2, we have five number to classify, so we made the architecture more complicated, such that we add one more sigimoid function.

In the training, in order to save time, we choose to use mini batch to update the weight, the batch size we choose is 500.

III. Result

For the first 0 1 dataset, we have 99.43% training accuracy with 0.5 learning rate , and 99.15% for the testing accuracy. The Confusion Table is:

Training Confusion Table	
5892	31
39	6703
Testing Confusion Table	
974	6
12	1123

For the 0 to 4 dataset, we arrive 97.65% training accuracy with 5.0 learning rate , and 93.94% for the testing accuracy. The Confusion Table is:

Training Confusion Table				
5822	0	56	32	13
0	6681	34	19	8
43	21	5782	85	27
31	10	182	5883	25
14	21	67	28	5712
Testing Confusion Table				
939	3	23	11	4
0	1112	12	8	3
22	14	942	40	14
14	15	42	928	11
9	12	35	19	907