Aykut Aykut
0034039
October 31st, 2018

Comp421 Homework03

In this homework, I was asked to implement a multilayer perceptron with 1 hidden layer for multiclass discremination which can classify 20*16 pixel images to 5 distinct classes, A, B, C, D, E. In order to do that , I followed these 22 steps:

1) I read the x_data_set which contains all 320 (20*16) features of 195 images into memory and y_data_set which contains the true class labels of 195 images. In both files, each row corresponds to a image.
2) I changed the class labels with the following mapping in order to obtain one-hot encoding class label matrix using *dummy_cols* function of *fastDummies* library:
    a) A -> row vector of (1, 0, 0, 0, 0)
    b) B -> row vector of (0, 1, 0, 0, 0)
    c) C -> row vector of (0, 0, 1, 0, 0)
    d) D -> row vector of (0, 0, 0, 1, 0)
    e) E -> row vector of (0, 0, 0, 0, 1)
3) Train-test split: I splitted the x_data_set and y_data_set into 2 groups, training and test. First 25 images of each class in x_data_set and y_data_set are in training set and remaning 14 images of each class in x_data_set and y_data_set are in test set.
4) I merged training sets of each class into x_training_data_set with *rbind* function and reseted their index. After that in order to apply matrix multiplication, I converted it into data matrix. I applied the same procedure for x_test_data_set, y_training_labels and y_test_labels.
5) I removed useless variables which I used on the way preparing training and test data matrices.
6) I defined a *safelog* function in order to handle log(0) case and it is the same *safelog* function with the one we used in lectures.
    $safelog(x) = log(x + 10^{-100})$
7) I defined the *sigmoid* function using the following formula:
    $sigmoid(m) = \frac{1}{1+exp(-m)}$
8) I defined the softmax function using the following formula:
    $softmax(m_c^i) = \frac{exp(m_c^i)}{\sum_{c=1}^{K} exp(m_c^i)}$ where K is class size 5, c = 1, 2, 3, 4, 5 and i = 1, 2, …, 125
9) I defined *gradient_v* and *gradient_w* functions using the following formulas:
    $gradient\_v(z,\ y_{truth},\ y_{predicted}) =- t(z)\ \% * \%\ (y_{truth} - y_{preicted})$ where t is the transpose function.

For *gradient_w* function, I used the chain rule:

$$\frac{\partial Error}{\partial w_{hd}} = \sum_{i=1}^{N} \frac{\partial Error_i}{\partial y_{predicted_i}} * \frac{\partial y_{predicted_i}}{\partial z_{ih}} * \frac{\partial z_{ih}}{\partial w_{hd}}$$

$$gradient\_w(x,\ z,\ v,\ y_{truth},\ y_{predicted}) = (-\,t(x)\ \%*\%\ (((y_{truth} - y_{predicted})\ \%*\%\ t(v)) * z * (1 - z)))[,\ 2:h]$$

where h = *nrow(v)* = 21.

I needed to get column slice at the end in order to remove the first column which is all zero gradient for w_0 and dimensionality consistency between w and the gradient of w.

10) I defined *error* function using the following formula:

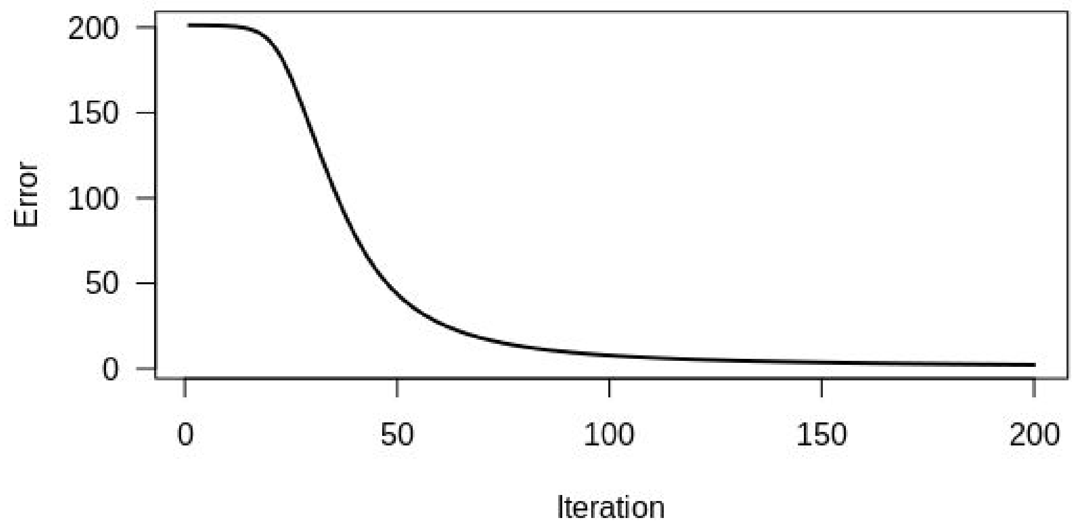
$$error(y_{truth},\ y_{predicted}) = -\,sum(y_{truth} * safelog(y_{predicted}))$$

11) I set the *eta, epsilon, H, max_iteration* and *seed* parameters with the same value as they are in the homework description.

    eta <- 0.005
    epsilon <- 1e-3
    H <- 20
    max_iteration <- 200
    set.seed(521)

12) I calculated the sample size N, the dimension of features D and number of classes K using *x_train* and *y_train_truth*.
13) I initialized w and v matrixes uniformly random between -0.01 and +0.01. Size of w is (D+1)*H and size of v is (H+1)*K.
14) I used backpropagation algorithm under batch learning scenario for training. *sigmoid* function is used as the activation function for the hidden layer and *softmax* function is used as the activation function for the output layer. Error is calculated by using *error* function and then it is recorded at each iteration.
15) During training, gradient descent algorithm is used as the update rule. First v is updated and then w is updated using the new updated v.
16) I continued training the model until either the absolute value of differences between current error and last error is smaller than *epsilon* or *iteration* count reached the *max_iteration* constant. At the end, it took 200 iterations and *iteration* count reached the *max_iteration* constant.

17) I plotted error value vs. iteration in a line plot.



18) I defined a *get_label* function which takes a row vector and returns column index of the element equals to 1 in input vector.
19) I defined *predict_labels* function which takes a matrix *y_predicted*, applies the *get_label* function to each row of *y_predicted* and returns the resulting matrix.
20) I found maximum element at each row (image) using *qlcMatrix* library and converted *y_train_predicted* matrix into a one-hot encoding matrix. Then I predicted a label for each training image using *predict_labels* function. I calculated the training confusion matrix and it is as follows:

```
> print(train_confusion_matrix)
           y_train
y_train_hat  1  2  3  4  5
          1 25  0  0  0  0
          2  0 25  0  0  0
          3  0  0 25  0  0
          4  0  0  0 25  0
          5  0  0  0  0 25
```

21) I predicted test images' labels using the trained model parameters *w* and *v*. Then I found maximum element at each row (image) using *qlcMatrix* library and converted *y_test_predicted* matrix into a one-hot encoding matrix. Then I predicted a label for each test image using *predict_labels* function. I calculated the test confusion matrix and it is as follows:

```
> print(test_confusion_matrix)
          y_test
y_test_hat  1  2  3  4  5
         1 13  1  0  0  0
         2  1 13  0  0  0
         3  0  0 14  0  1
         4  0  0  0 14  0
         5  0  0  0  0 13
```

22) I obtained the same results, error vs. iteration plot, training confusion matrix and test confusion matrix with the results given in the homework description.