Machine Learning Project 3

Project Report

Fully Connected Neural Network Design from Scratch for CIFAR-10 dataset

By
Qazi Umer Jamil
RIME 2019
NUST Registration No: 317920

NUST School of Mechanical and Manufacturing Engineering
Sector H-12, Islamabad, Pakistan

Dated: 21-12-2019

Contents

1 Introduction **1**
1.1 Environment . . . 1
1.2 Files . . . 1
1.3 Data Set . . . 2
1.3.1 Data Loading . . . 2
1.3.2 Data Normalization . . . 2
1.3.3 Data Manipulation . . . 2

2 Neural Network Architecture and Design **3**
2.1 Hyper parameters . . . 3
2.2 Parameters Initialization . . . 3
2.2.1 Bias Initialization . . . 3
2.2.2 Weights Initialization . . . 3
2.3 NN Training . . . 3
2.3.1 Forward Propagation . . . 3
2.3.2 Back Propagation . . . 4
2.3.3 Gradient Descent Algorithm . . . 5
2.3.4 Cost Function Graph . . . 6
2.4 Accuracy and Prediction . . . 7
2.4.1 Accuracy . . . 7
2.4.2 Prediction Function . . . 8

Bibliography **9**

List of Figures

2.1 Architecture 01- Cost Function Values vs. Number of iterations 7
2.2 Architecture 02- Cost Function Values vs. Number of iterations 7

List of Tables

2.1 NN Accuracy 8

1. Introduction

The project aim is to design a fully connected neural network with hidden layers from scratch for classification of images in the CIFAR-10 dataset.

1.1 Environment

MATLAB R2018a is used for model implementation and training.

1.2 Files

The attached zip file contains the following files:

- `main.m`
  Main file of the program
- `initW.m`
  To initiliaze weight matrices
- `GD.m`
  Implementation of mini batch gradient descent
- `accCal.m`
  To calculate accuracy
- `computeGradients.m`
  Implementation of forward and backward propagation
- `prediction.m`
  Implementation of prediction function
- `sigDerivative.m`
  Derivative of sigmoid function
- `sigmoid.m`
  Implementation of sigmoid formulae

1.3 Data Set

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The default CIFAR-10 dataset does not contains the Cross Validation dataset. Therefore, I used 20% of given training set as cross validation set. To do that, I combined all the 05 training batches into 01 single batch. Then I chose 80% of this new batch for training set and the remaining 20% as the CV set. So now, the training set contains 40,000 training examples, and the test and CV set contains 10,000 examples each.

The CIFAR-10 dataset for matlab is provided in .mat file which is of type Struct. Each .mat file struct contains the following two elements:

- **Data:** a 10000x3072 numpy array of uint8s. Each row of the array stores a 32x32 colour image. The first 1024 entries contain the red channel values, the next 1024 the green, and the final 1024 the blue. The image is stored in row-major order, so that the first 32 entries of the array are the red channel values of the first row of the image.
- **Labels:** a list of 10000 numbers in the range 0-9. The number at index i indicates the label of the ith image in the array data.

1.3.1 Data Loading

I load the each training and the test batch using MATLAB builtin load() function. After that, I combiner all the training batches into a single batch. I randomly separated 20% of training set for the Cross Validation Set and the remaining 80% for the training.

1.3.2 Data Normalization

In the CIFAR-10 dataset, each pixel in an image range from 0-255. To normalize the pixel values in an image, each pixel is divided by 255. This brings the each pixel value to 0-1. Normalizing the dataset helps in smoother convergence of gradient descent.

1.3.3 Data Manipulation

In the CIFAR-10 dataset, the label vectors range from 0-9. (There are total 10 labels). Since in matlab, matrices indexing starts from 1, I manually replaced all the '0' labels by '10. So now the new label vectors range from 1-10.

Further, since we are dealing with a multi class problem (number of classes are 10), the training, testing and the CV label vectors are also one hot encoded.

2. Neural Network Architecture and Design

I used a 4-layers neural network with 02 hidden layers. The activation unit used is Sigmoid.

2.1 Hyper parameters

I used different combinations of number of neurons in hidden layers and epochs for NN training. Some of these are as follows

- **Architecture 1:** [3072 x 2048 x 1024 x 10], Learning Rate =0.9, No. of epochs: 12
- **Architecture 2:** [3072 x 265 x 64 x 10], Learning Rate = 0.9, No. of epochs: 320

2.2 Parameters Initialization

2.2.1 Bias Initialization

All bias units are initialized to 0 [1].

2.2.2 Weights Initialization

All weight matrices are initialized with random values between 0.1 and -0.1, according to the following equation:

$$-\varepsilon \leq \Theta_{ij}^{(l)} \leq \varepsilon \tag{2.1}$$

2.3 NN Training

The modified training set contains 40,000 images. The training set is further divided into 08 mini batches of size 5000 images and mini batch gradient descent is applied.

2.3.1 Forward Propagation

Forward propagation is the process of computing values of activation units in the output layers based on the input image(feature vector). The code implementation for forward-propagation is as follows:

```
m = size(X_train, 1);
%% Forward propogation
```

```
    % Layer 1, input layer
    a1 = [(ones(m, 1)*bias_L1) X_train];

    %Layer 2
    z2 = a1 * Theta1';
    a2 = [(ones(m, 1)*bias_L2) sigmoid(z2)];

    %Layer 3
    z3 = a2 * Theta2';
    a3 = [(ones(m, 1)*bias_L3) sigmoid(z3)];

    %Layer 4
    z4 = a3 * Theta3';
    a4 = sigmoid(z4);

    %Cost Function
    J_train = -1*sum(sum(y_oht.*log(a4)+(1-y_oht) .* log(1-a4(y_oht))));
    J_train = (J_train/m) + lambda*(sum(sum(Theta1(:,2:end).^2))...
                                +sum(sum(Theta2(:,2:end).^2))...
                                +sum(sum(Theta3(:,2:end).^2)))/(2*m);
```

As it can be seen in the above implementation, I used cross entropy cost function for calculating the cost of each iteration.

In above, m is the total number of training examples. a_4 is the vector of output layer (cal also be called as predicted vector), calculated using forward propagation. y_oht is one hot encoded label vector of our training examples. lambda term is the regularization term.

2.3.2 Back Propagation

In back-propagation, we essentially back propagate the error from the output layer to the input layer by computing cost gradient of each activation unit.
These gradients values are then used in the gradient descent update rules to adjust the weights accordingly. The code implementation for back-propagation is as follows:

```
%% Backpropogation
for trainExample = 1:m
    % Layer 4, output layer
    z4b = z4(trainExample,:)';
    a4b = a4(trainExample,:)';

    % Layer 3
    z3b = z3(trainExample,:)';
    a3b = a3(trainExample,:)';

    % Layer 2
    z2b = z2(trainExample,:)';
    a2b = a2(trainExample,:)';
```

```
    % Layer 1, Input layer
    a1b = a1(trainExample,:)';

    weights_L4_delta = a4b - y_oht(trainExample,:)';

    Theta3'*(weights_L4_delta).*[ones(size(z3,1),1)  sigDerivative(z3)]';

    delta_L3 = (Theta3'*weights_L4_delta).*[1;sigDerivative(z3b)];
    bias_L3_delta = delta_L3(1:1);
    weights_L3_delta = delta_L3(2:end);

    delta_L2 = (Theta2'*weights_L3_delta).*[1;sigDerivative(z2b)];
    bias_L2_delta = delta_L2(1:1);
    weights_L2_delta = delta_L2(2:end);

    W1_gradients = W1_gradients + weights_L2_delta*a1b';
W2_gradients = W2_gradients + weights_L3_delta*a2b';
    W3_gradients = W3_gradients + weights_L4_delta*a3b';

    Bias_L2_gradient = Bias_L2_gradient + bias_L2_delta;
    Bias_L3_gradient = Bias_L3_gradient + bias_L3_delta;
end
```

2.3.3 Gradient Descent Algorithm

Gradient Descent is an iterative optimization algorithm. For each layer $l = L, L1, \ldots, 2$ [2], we update the weights according to the rule:

$$w_l = w_l - \alpha \frac{1}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T \tag{2.2}$$

And the biases according to the rule:

$$b_l = b_l - \alpha \frac{1}{m} \sum_x \delta^{x,l} \tag{2.3}$$

The code implementation for the gradient descent is as follows:

```
m = size(X, 1); % number of training examples
% Implementation of mini batch gradient descent
    for iter = 1: epochs

        for batchNum = 1: 12
         a = 1 + (5000*(batchNum-1));
         b = 5000*batchNum;
         X_train = X(a:b,:);
         y_train = y(a:b,:);
```

```
        % Forward and back propogation is done in the computeGradients.m file
        [W1_gradients, W2_gradients, W3_gradients,...
               Bias2_gradient, Bias3_gradient, J] = computeGradients(Theta1,...
               Theta2,Theta3, bias_L1, bias_L2, bias_L3, ...
                                       X_train, y_train, lambda);

           % Theta Update Rule
           Theta1 = Theta1 - (alpha*W1_gradients/m);
           Theta2 = Theta2 - (alpha*W2_gradients/m);
           Theta3 = Theta3 - (alpha*W3_gradients/m);

           % Bias Update Rule
           bias_L2 = bias_L2 - (alpha*Bias2_gradient/m);
           bias_L3 = bias_L3 - (alpha*Bias3_gradient/m);

           % Log the cost func values
           J_train(iter) = J;

            fprintf('Current iter: %i,  batchNum: %i, CostFuncVal: %f\n', iter, batchNum,J_train(iter));

            if isinf(J_train(iter))
               fprintf('Cost Function Value INCREASING. Please re-adjust learning rate parameter...\n');
               fprintf('Current Learning Rate is %f...\n', alpha);
               pause;
            end
    % save('try6_23_dec_weights', 'Theta1', 'Theta2', 'Theta3', 'bias_L2', 'bias_L3')
        end

    end
end
```

2.3.4 Cost Function Graph

Figure 2.2 shows the graph of Cost Function Values vs. Number of iterations for Architecture 01 and Architecture 02 (Please see section 2.1).

Figure 2.1: Architecture 01- Cost Function Values vs. Number of iterations

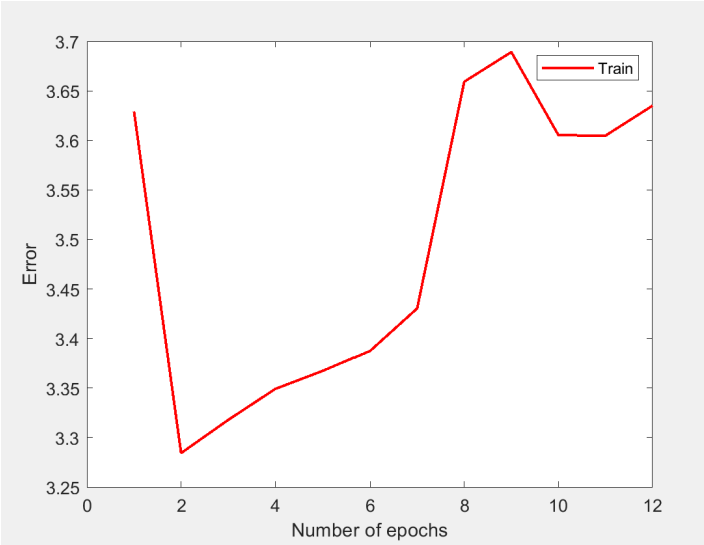
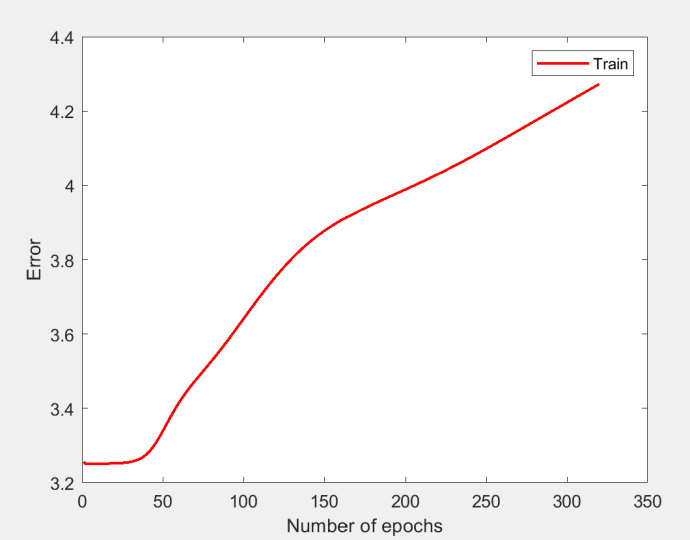


Figure 2.2: Architecture 02- Cost Function Values vs. Number of iterations



2.4 Accuracy and Prediction

2.4.1 Accuracy

The accuracy function takes in the test/train/CV examples, their labels, the obtained parameters and calculates the predicted label using forward propagation.

The calculated predicted label for each example basically contains the probabilities for each class. In our case, this vector is $10x1$. We pick the maximum probability in each predicted output and assign it to 1 and the other remaining class probabilities to 0. (We can also define a threshold for this purpose). For example, if, for an image the predicted vector was $[0.10; 0.20; 0.23; 0.34; 0.54; 0.35; 0.24; 0.32; 0.39; 0.385]$, It will pick 0.39 as it is the maximum probability in the predicted vector and get it index(using a matlab command) and produce a new label vector with a value of 1 at that index and all values equal to zero$[0; 0; 0; 0; 0; 0; 0; 0; 1; 0]$.

We will compare this newly produced label vector with our given labels and count the total number of right prediction and then take the mean. Multiply it by 100 and we will get percentage accuracy.

Table 2.1: NN Accuracy

NN Architecture	**Training Accuracy**	**Test Accuracy**	**CV Accuracy**
3072, 2048, 1024, 10	28.2275	28.4100	28.2800
3072, 256, 64, 10	37.4275	37.1800	37.3000

2.4.2 Prediction Function

Prediction function takes an image feature vector, normalize it, and then predict its label using forward propagation.

Bibliography

[1] http://cs231n.github.io/neural-networks-2/, last accessed on 18 dec, 2019.

[2] http://neuralnetworksanddeeplearning.com/chap2.html, last accessed on 15 dec, 2019.