Felicia Rosell
4 april 2017

Report Lab 1

Search Engines DD2424

In this assignment, mini-batch gradient descent was used to classify images from CIFAR-10 into 10 classes. The network only had one layer and L2 regularization was used.
The main result of the report is that the network achieved an accuracy of 37 % in classifying images. This accuracy was achieved when using step size η at 0.01 and no regularization.

The analytical gradient was successfully computed. This was verified using max difference (function below), and by verifying that the gradient entries were not to small. If they were, a relative error measure would have been more appropriate.

$$|\max(g_a - g_n)| < 10^{-6}$$

The following graphs show the cost functions for different values of λ and η.

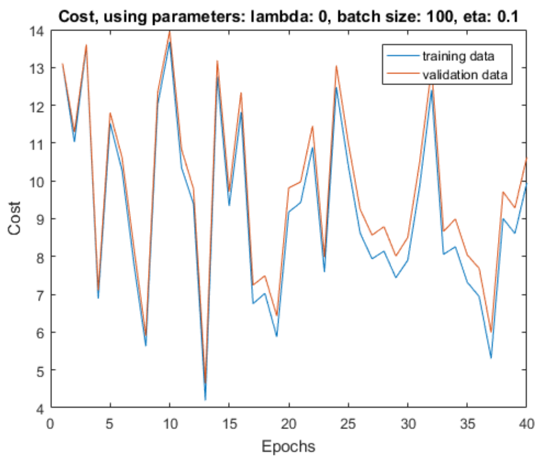

Figure 1.

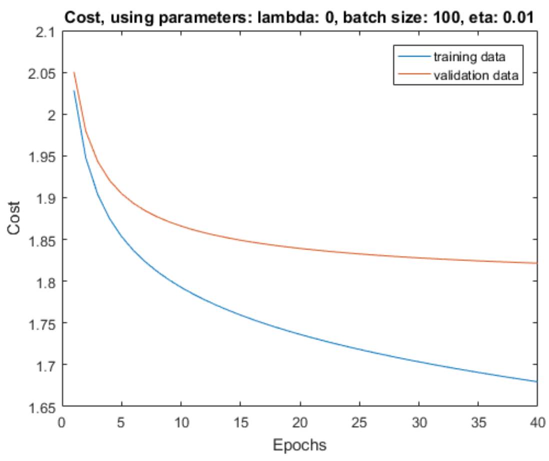

Figure 2.

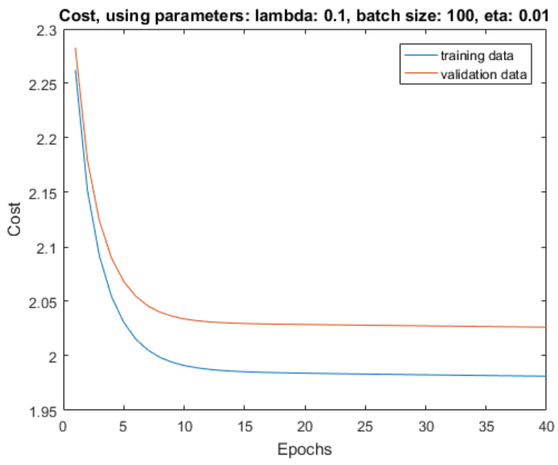

Figure 3.

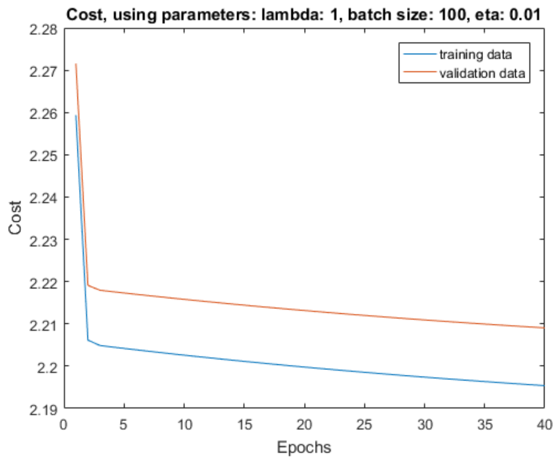

Figure 4.

If you compare *figure 1* to *figures 2-4*, you can draw the conclusion that if the step size η is too large, the method cannot find a local minimum. When comparing *figure 2, 3* and *4* you can see different results depending on how much regularization is used. If regularization is very large as in *figure 4*, the cost function seems to be pressed down very early. The fact that the network seems to perform worse with regularization, even for the validation data, could suggest that the method becomes underfitted with the regularization.

Table 1. Accuracy for different parameters

λ	**η**	**Accuracy**
0	0.1	0.1881
0	0.01	0.3678
0.1	0.01	0.3338
1	0.01	0.2192

This table indicates similar results as *figures 1-4*. When the step size is very small, the accuracy 19 % is only somewhat better than with random guessing (10 %). When decreasing the step size to 0.1 the results improve. In this setup, the accuracy is the highest when there is no regularization, and step size 0.01. The accuracy is almost as good when λ is 0.1, but when λ is 1 the accuracy is almost as bad as with the small step size of 0.1.

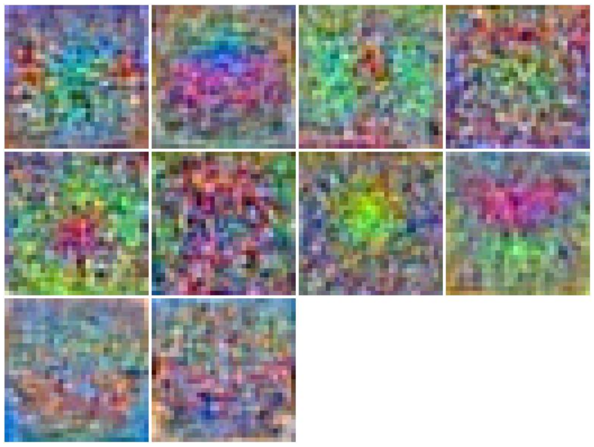
Figure 5. Weight matrix, λ is set to 0 and η to 0.1

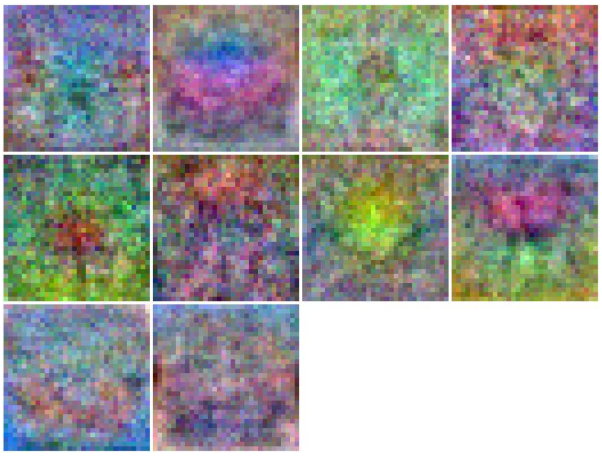
Figure 6. Weight matrix, λ is set to 0 and η to 0.01

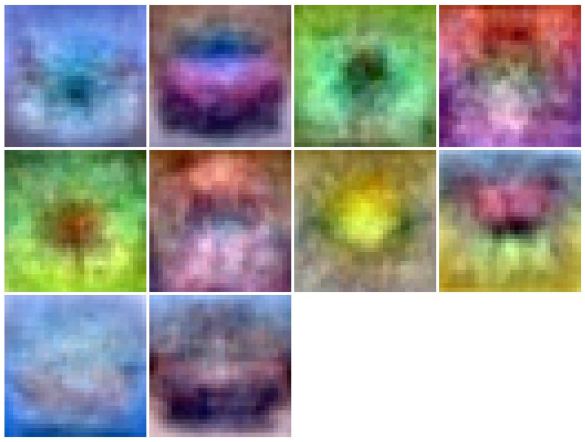
Figure 7. Weight matrix, λ: 0.1 and η to 0.01

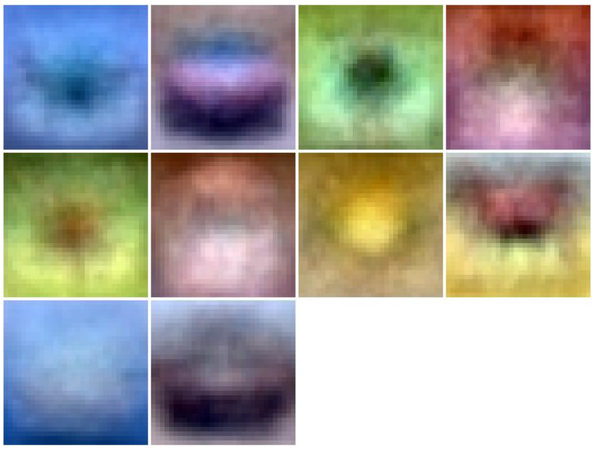
Figure 8. Weight matrix, λ is set to 1 and η to 0.01

Figures 5-8 show the weight matrix for different network parameters. If you compare *figure 6* to *figure 7* and *8* you can see that with increasing regularization the images do seem to generalize the images in a good way. In the second image, you can see the contour of a car, and in the 8th image, the contour of a cow. You can see that with regularization, lots of the noise is flattened out, creating a smooth model.

Felicia Rosell
4 april 2017

Code

```
[X,Y,y] = LoadBatch('data_batch_1.mat');

K = 10;
d = size(X,1);
N = size(X,2);

W = randn(K,d)*0.01;
b = randn(K,1)*0.01;

lambda = 1;
n_epochs = 40;
n_batch = 100;
eta = 0.01;

[Wstar, bstar] = MiniBatchGD(X, Y, n_batch, eta, n_epochs, W, b, lambda);

for i=1:10
    im = reshape(Wstar(i, :), 32, 32, 3);
    s_im{i} = (im - min(im(:))) / (max(im(:)) - min(im(:)));
    s_im{i} = permute(s_im{i}, [2, 1, 3]);
end

montage(s_im);

[Xtest,Ytest,ytest] = LoadBatch('test_batch.mat');
acc = ComputeAccuracy(Xtest, ytest, Wstar, bstar);
disp(['Accuracy: ', num2str(acc)]);


function acc = ComputeAccuracy(X, y, W, b)
%Calculate the accuracy scalar
%   that is the percentage of correctly classified
%   samples

    P = EvaluateClassifier(X, W, b);
    sumCorrect = 0;
    for sample=1:size(P,2)
        [~, class] = max(P(:,sample));
        if class == y(sample)
            sumCorrect = sumCorrect + 1;
        end
    end
    acc = sumCorrect / sample;
end
```

Felicia Rosell
4 april 2017

```
function J = ComputeCost(X, Y, W, b, lambda)
%Computes the cost
%   J is a scalar with the sum of the loss of the network's
%       predictions for the images in X relative
%       to the labels and regularization term on W


    s = 0;
    P = EvaluateClassifier(X, W, b);
    N = size(X,2);
    for i=1:N
        cross = -log(dot(Y(:,i)',P(:,i)));
        s = s + cross;
    end
    s = s / N;

    J = s + lambda*sum(diag(W'*W));
end

function [grad_W, grad_b] = ComputeGradients(X, Y, P, W, lambda)
%• each column of X corresponds to an image and it has size d×n.
%• each column of Y (K×n) is the one-hot ground truth label for the
corresponding
%   column of X.
%• each column of P contains the probability for each label for the image
%   in the corresponding column of X. P has size K×n.
%• grad_W is the gradient matrix of the cost J relative to W and has size
%   K×d.
%• grad_b is the gradient vector of the cost J relative to b and has size
%   K×1.
    n = size(X,2);

    sumW = 0;
    sumb = 0;

    for i=1:n

        y = Y(:,i);
        p = P(:,i);
        x = X(:,i);

        g = - (y'/(y'*p))*(diag(p)-p*p');
        dldW = g'*x'; %size c x d (10*3072)
        sumW = sumW + dldW;

        dldb = g;
        sumb = sumb + dldb;

    end

    grad_W = sumW/n + 2*lambda*W;
    grad_b = sumb'/n;
end
```

```
function P = EvaluateClassifier(X, W, b)
%Evaluates the classifier by calculating the score
%   and softmax
%   each column of P contains the probability of each label
%       for the image. P has size K*N
    K = size(W,1);
    N = size(X,2);
    P = zeros(K,N);
    for i=1:N
        s = W*X(:,i) + b;
        P(:,i) = exp(s)/dot(ones(K,1),exp(s));
    end
end

function [X, Y, y] = LoadBatch(filename)
%Function that reads the data from the file
%   X is a matrix containing image pixel data.
%       it has size d*N, N is number of
%       images = 10000, and d is dimensionality = 32*32*2=3072,
%       each column represents one image
%   Y contains on each column the one-hot represention of the label
%       for each image
%       and is the size N*K where K is #labels = 10
%   y is a row vector containing the label for each image, between 1 and 10
    batch = load(filename);
    X = double(batch.data')/255;
    y = batch.labels' + 1;
    N = size(X,2);
    K = 10;
    Y = zeros(K,N);
    for i=1:N
        Y(y(i),i) = 1;
    end
end
```

```
function [Wstar, bstar] = MiniBatchGD(X, Y, n_batch, eta, n_epochs, W, b,
lambda)
%Mini-batch learning function of W and b, with gradient descent
%   X training images
%   Y labels for training images
%   W and b initial values
%   lambda regularization factor in the cost function
%   GDparams contains n_batch, eta and n_epochs
N = size(X,2);

costTrain = zeros(1, n_epochs);
costVal = zeros(1, n_epochs);

[Xval,Yval,~] = LoadBatch('data_batch_2.mat');

for i=1:n_epochs

    for j=1:N/n_batch
        j_start = (j-1)*n_batch + 1;
        j_end = j*n_batch;
        Xbatch = X(:, j_start:j_end);
        Ybatch = Y(:, j_start:j_end);

        P = EvaluateClassifier(Xbatch, W, b);
        [grad_W, grad_b] = ComputeGradients(Xbatch, Ybatch, P, W, lambda);

        W = W - eta*grad_W;
        b = b - eta*grad_b;

    end

    costTrain(i) = ComputeCost(X, Y, W, b, lambda);
    costVal(i) = ComputeCost(Xval, Yval, W, b, lambda);
    disp(['epoch: ', num2str(i), '/', num2str(n_epochs), '     Cost: ',
num2str(costTrain(i))]);

end
Wstar = W;
bstar = b;

plot(1:n_epochs, costTrain, 1:n_epochs, costVal);
title(['Cost, using parameters: lambda: ', num2str(lambda), ', batch size:
', num2str(n_batch), ', eta: ', num2str(eta)]);
xlabel('Epochs')
ylabel('Cost')
legend('training data', 'validation data')
figure
end
```