KTH Royal Institute of Technology

DD2424 Deep Learning in Data Science

Assignment 3 Bonus Report

k layer network with multiple outputs bonus part

Written by

Yutong Jiang

yutongj@kth.se

Date: Mar 29, 2022

1 introduction

In the first part, I intend to explore batch norm by applying batch normalization to the scores after the non-linear activation function.

In the second part, I intend to apply several methods to my deep network easier to optimize. The first method I did is replacing CLR with Adam and decay learning rate. Besides, I also tried to increasing the number of hidden nodes and replacing Relu with leakyRelu.

2 Batch normalization after non-linear activation function

In this part, I apply batch normalization to the scores after the non-linear activation function. The network I used is 9-layer network and the main change I made is the forward and backward pass.

For the forward pass, it has been changed to

$$s_i = W_i * H_i + b_i \tag{1}$$

$$relu_i = max(0, s_i) \tag{2}$$

$$\hat{s_i} = BatchNormalize(relu_i, mu_i, v_i) \tag{3}$$

$$H_{i+1} = gamma. * \hat{s_i} + beta; \tag{4}$$

The backward pass has been changes accordingly, which can be shown as follows.

$$G = G. * (gamma_i * ones(1, len)) \tag{5}$$

$$G = BatchNormPass(G, relu_i, mu_i, v_i) \tag{6}$$

$$G = G. * Ind(relu_i > 0) \tag{7}$$

$$gradW_i = \frac{1}{len} * G * Hi' + 2 * lambda * Wi \tag{8}$$

$$gradb_i = \frac{1}{len} * G * ones(len, 1) \tag{9}$$

if i > 1, propagate G_{batch} to the previous layer.

The other parameters stay the same and hidden nodes I choose is [50 30 20 20 10 10 10 10].

The final accuracy I could get is 0.5237, which is greater than the accuracy I get in basic part.

Hence, applying batch normalization after non-linear activation function is an efficient way to get higher accuracy.

3 Adam optimizer and learning rate decay

Just as what has been taught in class, the principle of updating could be shown as follows.

$$m^{t+1} = \beta_1 * m^t + (1 - \beta_1) * g_t \tag{10}$$

$$v^{t+1} = \beta_2 * v^t + (1 - \beta_2) * g_t . * g_t \tag{11}$$

Then considering the bias by seting

$$m^{t+1} = \frac{m^{t+1}}{1 - \beta_1} \tag{12}$$

$$v^{t+1} = \frac{v^{t+1}}{1 - \beta_2} \tag{13}$$

Then the Adam updata rule could be shown as

$$x^{t+1} = x^t - \frac{\eta}{\sqrt{v^{t+1}} + eps} \hat{m}^{t+1} \tag{14}$$

The default values has been set to $\beta_1 = 0.9, \beta_2 = 0.999, \beta_3 = 1e - 8$, the hidden nodes of hidden layers are set to [50 30 20 20 10 10] with decay learning rate. The initial learning rate is set to be 5e-3 and after each epoch it is set to eta = eta * 0.8. The total epoch I choose is 20. With the parameter setting, the accuracy I could achieve with Adam is 0.4688.

In order to compare with SGD, the paramter when training the neural network is set totally the same as above. Besides, I also changed the cyclical learning rate to decay learning rate to guarantee the single variable principle. Based on the setting above, the final accuracy I could achieve with gradient descent is 0.3313.

Based on the accuracy obtained above, I could find that Adam could converge faster than the stochastic gradient descent. Hence, Adam is an efficient way to update weight/bias and it could make the network converge faster than the traditional stochastic gradient descent.

4 increasing the number of hidden nodes at each layer

While if I change the hidden nodes to [200 100 50 50 20 20], the accuracy I could achieve is 0.5635 when the other parameters remain the same as the basic part.

Hence, it is obvious to notice the improvement when increasing the hidden nodes in each layer.

5 changing the activation function from relu to leaky relu

In the basic part, the activation function we explore is relu. While in the bonus part, I intend to explore the performance of leaky relu when comparing with traditional relu.

The code we need to change is the forward and backward algorithm in ComputeGradients.m and EvaluateClassifier.m. The leaky relu could be represented as

$$H(i+1) = max(a * \tilde{x}, \tilde{x}) \tag{15}$$

In the bonus part, a is set to 0.01, hidden nodes is set to [50 30 20 20 10 10], the other parameters keep the same as before.

Then for the backward algorithm, the derivative of leaky relu should be updated as

$$tmp = sign(H(i)) \tag{16}$$

$$tmp(tmp < 0) = a \tag{17}$$

By applying leaky-relu, the final accuracy I could achieve is 0.5302, while which is greater than the accuracy I achieved with relu, which is 0.5245. We could find that there is a slight improvement by changing the type of activation function. The reasons why leaky relu could have a better performance is that by applying leaky relu, the problem of dying neuron could be eased.