EEC 289Q Data Analytics for Computer Engineers
Homework 3

Ahmed Mahmoud

May, 7th 2018

Softmax Regression:

The following code shows the implementation of the softmax regression function

```
function [f,g] = softmax_regression(theta, X,y)
    m=size(X,2);
    n=size(X,1);
    theta=reshape(theta, n, []);
    num_classes=size(theta,2)+1;
    f = 0;
    g = zeros(size(theta));
%%% YOUR CODE HERE %%%
    theta_x = exp(theta'*X);
    sum_col = sum(theta_x);
    for I=1:m
        theta_x(:,I) = theta_x(:,I)/sum_col(I);
    end
    for I=1:m
         if y(I) <num_classes
             f = f - log(theta_x(y(I),I));
         end
    end
    %expand y to matrix to allow matrix multiply to obtain the gardient
    y_mat = full(sparse(y,1:m,1));
    g = -X * (y_mat(1:num_classes-1,:) - theta_x)';
    g=g(:); % make gradient a vector for minFunc
```

Using this code, we were able to achieve training accuracy of 87.2% and test accuracy of 87.6% while the optimization took 4.524218 seconds. We used the gradient checker on our implementation and the average absolute error was 0.0330 (of 10 tests).

Supervised Neural Networks:

The following code implements the cost function, forward propagation, and compute the gradients for multiple hidden layers neural network

```
function [ cost, grad, pred_prob] = supervised_dnn_cost( theta, ei, ...
    data, labels, pred_only)
    %SPNETCOSTSLAVE Slave cost function for simple phone net
    %   Does all the work of cost / gradient computation
    %   Returns cost broken into cross-entropy, weight norm, and prox reg
    %        components (ceCost, wCost, pCost)

    %% default values
    po = false;
    if exist('pred_only','var')
      po = pred_only;
    end

    %% reshape into network
    stack = params2stack(theta, ei);
    numHidden = numel(ei.layer_sizes) - 1;
    hAct = cell(numHidden+1, 1);
    gradStack = cell(numHidden+1, 1);
    %% forward prop
    %%% YOUR CODE HERE %%%
    for J=1:numHidden
        if J==1
            %input
            z = stack{J}.W*data;
        else
            %activiation
            z = stack{J}.W*hAct{J-1};
        end
        z = z + stack{J}.b;
        hAct{J}=sigmoid(z);
    end
    z = stack{numHidden+1}.W*hAct{numHidden};
    z = z + stack{numHidden+1}.b;
    E = exp(z);
    pred_prob = E./sum(E,1);
    hAct{numHidden+1} = pred_prob;

    %% return here if only predictions desired.
    if po
      cost = -1; ceCost = -1; wCost = -1; numCorrect = -1;
      grad = [];
      return;
    end
    %% compute cost
    %%% YOUR CODE HERE %%%
    c = log(pred_prob);
    ind =sub2ind(size(c), labels', 1:size(c,2));
    values = c(ind);
    ceCost = -sum(values);
    %% compute gradients using backpropagation
```

```
    %%% YOUR CODE HERE %%%
    d = zeros(size(pred_prob));
    d(ind)=1;
    error = (pred_prob-d);
    for l =numHidden+1:-1:1
        gradStack{l}.b = sum(error,2);
        if l ==1
            gradStack{l}.W = error*data';
            break;
        else
            gradStack{l}.W = error*hAct{l-1}';
        end
        error = (stack{l}.W)'*error.*hAct{l-1}.*(1-hAct{l-1});
    end
    %% compute weight penalty cost and gradient for non-bias terms
    %%% YOUR CODE HERE %%%
    wCost = 0;
    for l = 1:numHidden+1
        wCost = wCost + 0.5*ei.lambda*sum(stack{l}.W(:).^2);
    end
    cost = ceCost + wCost;
    for l=numHidden:-1:1
        gradStack{l}.w = gradStack{l}.W + ei.lambda*stack{l}.W;
    end
    %% reshape gradients into vector
    [grad] = stack2params(gradStack);
end
```

Using this code, we were able to get train accuracy of 100% and test accuracy of 0.9712% with weight decay value of 0.0. We can have a better test accuracy by changing the weight decay value to 0.25 which gave test accuracy of 0.973800% while the train accuracy lowered to 0.997717%. We think this is due to over-fitting with weight decay of 0. We used different values of weight decay with two and four hidden layers but not has shown superior performance. Also, changing/reduces the layer size always produced lower accuracy.

AlexNet:

First Layer: The input to AlexNet is a $[227 \times 227 \times 3]$ image (weight W). The first convolutional layer has receptive field (F) of 11, stride (S) of 4 and no zero-padding (P) with 96 kernels (K) (or convolutional layer depth of 96). Thus, the output is

$$\frac{W-F+2P}{S}+1=\frac{227-11+2*0}{4}+1=55$$

The number of neurons in this layer is $55*55*96=290,400$. Each of the 55×55 slice uses a unique $11*11*3=363$ weights and 1 bias. Thus, the total number of parameters in the first layer is $96*363+96=34,944$ parameters.

The operations for this layers are as follows: in order to obtain the one value in the $55\times55\times96$ output, the filter application will require 11×11multiply operations and 11 add operations to do the dot product. This will be done 3 times for the three color channels in the input and add up all the result together (3 multiply) in addition to 1 add for the bias. Thus, the number of operations for one entry is $((11*11+11)*3)+1=397$ operations. For the whole layer, the total number of operations is $(55*55*96)*397=115,288,800$.

Note: If we consider multiply and add as one operation (taking one cycle), then the number of operations per one entry is reduced to be $11*11*3+1=364$. This will make the total number of computation in this layer to be $(55*55*96)*364=105,705,600$.

Second Layer: After the max pooling in the first layer, the input to the second layer becomes $[27\times27\times96]$. Second convolutional layer has receptive field (F) of 5, stride (s) of 1 and zero-padding (P) of 2 with 256 kernels (K) . Thus, the output is

$$\frac{W-F+2P}{S}+1=\frac{27-5+2*2}{1}+1=27$$

The number of neurons in this layers is $27*27*256=186,624$. Each of the 27×27 slice uses a unique set of weight of size $5*5*96=2,400$ weights and 1 bias. Thus, the total number of parameters in the second layer is $256*2,400+256=614,656$ parameters.

Following the same computation we have done for the first layer, the total number of computation done in the second layer is $(27*27*256)*(((5*5+5)*96)+1)=537,663,744$.

Third Layer: After max pooling, the input to the third layer is $[13\times13\times256]$. The third convolutional layer has receptive field of (F) 3, stride (S) of 1 and zero-padding (P) of 1 with 384 kernels (K). Thus, the output is

$$\frac{W-F+2P}{S}+1=\frac{13-3+2*1}{1}+1=13$$

The number of neurons in this layer is $13*13*384=64,896$. Each of the 13×13 slice uses a unique set of weight of size $3*3*256=2,304$ weights and 1 bias. Thus, the total number of parameters in the third layer is $2,304*384+384=885,120$ parameters.

The total computation in the third layer is $(13*13*384)*(((3*3+3)*256)+1)=199,425,408$.

Fourth Layer: The output of third layer goes straight to fourth layer. The fourth convolutional layer has receptive field of (F) 3, stride (S) of 1 and zero-padding (P) of 1 with 384 kernels (K). The output size is (similar to third layer) 13.

The number of neurons is $13 * 13 * 384 = 64,896$. Each of this 13×13 slice uses a unique set of weights of size $3 * 3 * 384 = 3,456$ weights and 1 bias. Thus, the total number of parameters in the fourth layer is $3,456 * 384 + 384 = 1,327,488$.

The total computation in the forth layer is
$(13 * 13 * 384) * (((3 * 3 + 3) * 384) + 1) = 299,105,664$.

Fifth Layer: The output of fourth layers goes straight to the fifth layer. The fifth convolutional layer has receptive field of (F) 3, stride (S) of 1 and zero-padding (P) of 1 with 256 kernels (K). The output size is 13.

The number of neurons is $13 * 13 * 256 = 43,264$. Each of this 13×13 slice uses a unique set of weight of size $3 * 3 * 384 = 3,456$ weights and 1 bias. Thus, the total number of parameters in the fifth layer is $3,456 * 256 + 256 = 884,992$.

The total computation in the fifth layer is $(13*13*256)*(((3 * 3 + 3) * 384) + 1) = 199,403,776$.

Sixth Layer: After max pooling in the fifth layer, the input to the sixth fully connect (FC) layer is $[6 \times 6 \times 256]$. The total number of neurons of the sixth layer is 4096. Since it is fully connected, the number of parameters is $6 * 6 * 256 * 4096 = 37,748,736$. The number of operations for one neuron is $6 * 6 * 256$ multiply followed by $6 * 6 * 256$ accumulate/sum. Thus, the total number of operations are $(6 * 6 * 256) * 2 * 4096 = 75,497,472$.

Seventh Layer: The output of the sixth layer goes to the seventh layer i.e., the input to the seventh layer is $[4096]$. The total number of neurons of the seventh layer is 4096. Since it is fully connected, the number of parameters is $4096*4096 = 16,777,216$. The total number of operations are $4096 * 2 * 4096 = 33,554,432$.

Eighth Layer: The output of the seventh layer goes to the eighth layer i.e., the input to the eighth layer is $[4096]$. The total number of neurons of the eighth layer is 1000. Since it is fully connected, the number of parameters is $4096 * 1000 = 4,096,000$. The total number of operations are $4096 * 2 * 1000 = 8,192,000$.

Table 1 shows the number of neurons, parameters, and operations for all layers. We notice that 94% of the weights are in the fully connected layers 89% of the computation (and 98% of the neurons) are withing the convolutional layers.

Layer	Neurons	Parameters	Operations
#1	290,400(44.0481%)	34,944 (0.0560%)	115,288,800(7.8527%)
#2	186,624(28.3071%)	614,656 (0.9855%)	537,663,744(36.6223%)
#3	64,896 (9.8435%)	885,120 (1.4191 %)	199,425,408(13.5836%)
#4	64,896 (9.8435%)	1,327,488 (2.1284 %)	299,105,664(20.3732%)
#5	43,264 (6.5623%)	884,992 (1.41895%)	199,403,776(13.5821%)
#6	4096 (0.6212%)	37,748,736 (60.5246%)	75,497,472 (5.1424%)
#7	4096 (0.6212%)	16,777,216 (26.8998%)	33,554,432 (2.2855%)
#8	1000 (0.1516%)	4,096,000 (6.5673 %)	8,192,000 (0.5579%)
Total	659,272	62,369,152	1,468,131,296

Table 1: AlexNet absolute number of neurons, parameters, and operations for the convolutional and fully connected layers (and percentage).