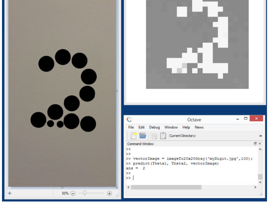
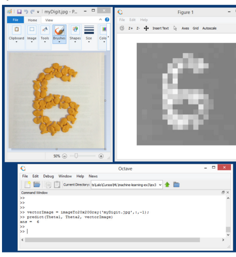
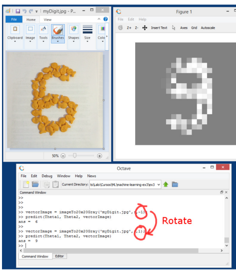


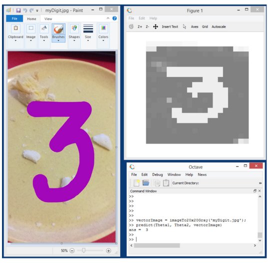
Digit 6



Digit 6 inverted is digit 9. This is the same photo of a six but rotated. Also, changed the contrast multiplier from 5 to 20. You can note that the gray background is smoother.



Digit 3



Explanation of Derivatives Used in Backpropagation

- We know that for a logistic regression classifier (which is what all of the output neurons in a neural network are), we use the cost function, $J(\theta) = -ylog(h_\theta(x)) - (1-y)log(1-h_\theta(x))$, and apply this over the K output neurons, and for all m examples.
- The equation to compute the partial derivatives of the theta terms in the output neurons:
- $\frac{\partial J(\theta)}{\partial \theta^{(L-1)}} = \frac{\partial J(\theta)}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial \theta^{(L-1)}}$
- And the equation to compute partial derivatives of the theta terms in the [last] hidden layer neurons (layer L-1):
- $\frac{\partial J(\theta)}{\partial \theta^{(L-2)}} = \frac{\partial J(\theta)}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L-1)}}{\partial \theta^{(L-2)}}$
- Clearly they share some pieces in common, so a delta term ($\delta^{(L)}$) can be used for the common pieces between the output layer and the hidden layer immediately before it (with the possibility that there could be many hidden layers if we wanted):
- $\delta^{(L)} = \frac{\partial J(\theta)}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}}$
- And we can go ahead and use another delta term ($\delta^{(L-1)}$) for the pieces that would be shared by the final hidden layer and a hidden layer before that, if we had one. Regardless, this delta term will still serve to make the math and implementation more concise.
- $\delta^{(L-1)} = \frac{\partial J(\theta)}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}}$
- $\delta^{(L-1)} = \delta^{(L)} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}}$
- With these delta terms, our equations become:
- $\frac{\partial J(\theta)}{\partial \theta^{(L-1)}} = \delta^{(L)} \frac{\partial z^{(L)}}{\partial \theta^{(L-1)}}$
- $\frac{\partial J(\theta)}{\partial \theta^{(L-2)}} = \delta^{(L-1)} \frac{\partial z^{(L-1)}}{\partial \theta^{(L-2)}}$
- Now, time to evaluate these derivatives:
- Let's start with the output layer:
- $\frac{\partial J(\theta)}{\partial \theta^{(L-1)}} = \delta^{(L)} \frac{\partial z^{(L)}}{\partial \theta^{(L-1)}}$
- Using $\delta^{(L)} = \frac{\partial J(\theta)}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}}$, we need to evaluate both partial derivatives.
- Given $J(\theta) = -ylog(a^{(L)}) - (1-y)log(1-a^{(L)})$, where $a^{(L)} = h_\theta(x)$), the partial derivative is:
- $\frac{\partial J(\theta)}{\partial a^{(L)}} = \frac{1-y}{1-a^{(L)}} - \frac{y}{a^{(L)}}$
- And given a=g(z), where $g = \frac{1}{1+e^{-z}}$, the partial derivative is:
- $\frac{\partial a^{(L)}}{\partial z^{(L)}} = a^{(L)}(1-a^{(L)})$
- So, let's substitute these in for $\delta^{(L)}$:

$\delta^{(L)} = \frac{\partial J(\theta)}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}}$

$\delta^{(L)} = (\frac{1-y}{1-a^{(L)}} - \frac{y}{a^{(L)}})(a^{(L)}(1-a^{(L)}))$

$\delta^{(L)} = a^{(L)} - y$

- So, for a 3-layer network (L=3),

$\delta^{(3)} = a^{(3)} - y$

- Note that this is the correct equation, as given in our notes.
- Now, given z=θ∗input, and in layer L the input is $a^{(L-1)}$, the partial derivative is:

$\frac{\partial z^{(L)}}{\partial \theta^{(L-1)}} = a^{(L-1)}$

- **Put it together for the output layer**:

$\frac{\partial J(\theta)}{\partial \theta^{(L-1)}} = \delta^{(L)} \frac{\partial z^{(L)}}{\partial \theta^{(L-1)}}$

$\frac{\partial J(\theta)}{\partial \theta^{(L-1)}} = (a^{(L)} - y)(a^{(L-1)})$

- Let's continue on for the hidden layer (let's assume we only have 1 hidden layer):

$\frac{\partial J(\theta)}{\partial \theta^{(L-2)}} = \delta^{(L-1)} \frac{\partial z^{(L-1)}}{\partial \theta^{(L-2)}}$

- Let's figure out $\delta(L-1)$.
- Once again, given z=θ∗input, the partial derivative is:

$\frac{\partial z^{(L)}}{\partial a^{(L-1)}} = \theta^{(L-1)}$

- And: $\frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} = a^{(L-1)}(1-a^{(L-1)})$
- So, let's substitute these in for $\delta^{(L-1)}$:

$\delta^{(L-1)} = \delta^{(L)} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}}$

$\delta^{(L-1)} = \delta^{(L)}(\theta^{(L-1)})(a^{(L-1)}(1-a^{(L-1)}))$

$\delta^{(L-1)} = \delta^{(L)}\theta^{(L-1)}a^{(L-1)}(1-a^{(L-1)})$

- So, for a 3-layer network,

$\delta^{(2)} = \delta^{(3)}\theta^{(2)}a^{(2)}(1-a^{(2)})$

- **Put it together for the [last] hidden layer**:

$\frac{\partial J(\theta)}{\partial \theta^{(L-2)}} = \delta^{(L-1)} \frac{\partial z^{(L-1)}}{\partial \theta^{(L-2)}}$

$\frac{\partial J(\theta)}{\partial \theta^{(L-2)}} = (\delta^{(L)} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}})(a^{(L-2)})$

$\frac{\partial J(\theta)}{\partial \theta^{(L-2)}} = ((a^{(L)} - y)(\theta^{(L-1)})(a^{(L-1)}(1-a^{(L-1)})))(a^{(L-2)})$

NN for linear systems

Introduction

The NN we created for classification can easily be modified to have a linear output. First solve the 4th programming exercise. You can create a new function script, nnCostFunctionLinear.m, with the following characteristics

- There is only one output node, so you do not need the 'num_labels' parameter.
- Since there is one linear output, you do not need to convert y into a logical matrix.
- You still need a non-linear function in the hidden layer.
- The non-linear function is often the tanh() function - it has an output range from -1 to +1, and its gradient is easily implemented. Let g(z)=tanh(z).
- The gradient of tanh is $g'(z) = 1 - g(z)^2$. Use this in backpropagation in place of the sigmoid gradient.
- Remove the sigmoid function from the output layer (i.e. calculate a3 without using a sigmoid function), since we want a linear output.
- Cost computation: Use the linear cost function for J (from ex1 and ex5) for the unregularized portion. For the regularized portion, use the same method as ex4.
- Where reshape() is used to form the Theta matrices, replace 'num_labels' with '1'.

You still need to randomly initialize the Theta values, just as with any NN. You will want to experiment with different epsilon values. You will also need to create a predictLinear() function, using the tanh() function in the hidden layer, and a linear output.

Testing your linear NN

Here is a test case for your nnCostFunctionLinear()

```
% inputs
nn_params = [31 16 15 -29 -13 -8 -7 13 54 -17 -11 -9 16]'/ 10;
il = 1;
hl = 4;
X = [1; 2; 3];
y = [1; 4; 9];
lambda = 0.01;

% command
[j g] = nnCostFunctionLinear(nn_params, il, hl, X, y, lambda)

% results
j =  0.020815
g =
    -0.0131002
    -0.0110085
    -0.0070569
     0.0189212
    -0.0189639
    -0.0192539
    -0.0102291
     0.0344732
     0.0024947
     0.0080624
     0.0021964
     0.0031675
    -0.0064244

```

Now create a script that uses the 'ex5data1.mat' from ex5, but without creating the polynomial terms. With 8 units in the hidden layer and MaxIter set to 200, you should be able to get a final cost value of 0.3 to 0.4. The results will vary a bit due to the random Theta initialization. If you plot the training set and the predicted values for the training set (using your predictLinear() function), you should have a good match.

Deriving the Sigmoid Gradient Function

We let the sigmoid function be $\sigma(x) = \frac{1}{1+e^{-x}}$

Deriving the equation above yields to $(\frac{1}{1+e^{-x}})^2 \frac{d}{ds} \frac{1}{1+e^{-x}}$

Which is equal to $(\frac{1}{1+e^{-x}})^2 e^{-x}(-1)$

$(\frac{1}{1+e^{-x}})(\frac{1}{1+e^{-x}})(-e^{-x})$

$(\frac{1}{1+e^{-x}})(\frac{-e^{-x}}{1+e^{-x}})$

$\sigma(x)(1-\sigma(x))$

Additional Resources for Backpropagation

- Very thorough conceptual [example] (https://web.archive.org/web/20150317210621/https://www4.rgu.ac.uk/files/chapter3%20-%20bp.pdf)
- Short derivation of the backpropagation algorithm: http://pandamatak.com/people/anand/771/html/node37.html
- Stanford University Deep Learning notes: http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm
- Very thorough explanation and proof: http://neuralnetworksanddeeplearning.com/chap2.html