Introduction to Machine Learning
Homework 6: Neural Networks

Prof. Linda Sellie

1. Given a neural network with 3 layers, where the *input* layer has 3 neurons, the *hidden* layer has 3 neurons, the *output* layer has 2 neurons, and

$$W^{(1)} = \begin{pmatrix} 4 & 4 & 1 \\ -4 & -4 & 1 \\ 1 & 1 & -5 \end{pmatrix}, \; W^{(2)} = \begin{pmatrix} 4 & 4 & 1 \\ 1 & 1 & 2 \end{pmatrix}, \; b^{(1)} = \begin{pmatrix} -2 \\ 6 \\ 1 \end{pmatrix}, \; b^{(2)} = \begin{pmatrix} -6 \\ 2 \end{pmatrix}$$

   (a) Draw the neural network
   (b) What is $h_{W,b}(x)$ when $x^T = (1, 2, 3)$
   (c) For the example $x^T = (1, 2, 3)$ and $y = (1, 0)$
      - What is $\delta_1^{(3)}$
      - What is $\frac{\partial J(W,b;x,y)}{\partial W_{11}^{(2)}}$
      - What is $\delta_1^{(2)}$
      - What is $\frac{\partial J(W,b;x,y)}{\partial W_{11}^{(1)}}$

2. Given a neural network with 3 layers, where the *input* layer has 2 neurons, the *hidden* layer has 2 neurons, the *output* layer has 1 neuron, and

$$W^{(1)} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \; W^{(2)} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \; b^{(1)} = \begin{pmatrix} 1 & -1 \end{pmatrix}, \; b^{(2)} = \begin{pmatrix} 1 \end{pmatrix}$$

   Suppose you had the following training set: $((1, 0), 1), ((0, 1), 0)$. Perform 1 step of gradient descent where the learning rate is 0.2.

3. Consider training a neural network. Would overfitting be more of a problem when the training set is small or large? Would overfitting be more or less of a problem when the number of parameters to learn (e.g. the number of weights in a neural network) is small or large?

4. Modify the neural network implementation we discussed in class to see if you can improve the performance by trying the following:

   (a) Add a regularization term to the cost function $\frac{\partial J(W,b)}{\partial W_{ij}^{(\ell)}} = \frac{1}{n}\left[\sum_{i=1}^{n} \frac{\partial J(W,b,x^i,y^i)}{\partial W_{ij}^{(\ell)}}\right] + \frac{\lambda}{2} W_{ij}^{(\ell)}$ where x^i, y^i are the ith training example. See section 1.2 in `http://adventuresinmachinelearning.com/improve-neural-networks-part-1/`

   (b) Try using the *ReLU* activation function, $f(z) = \max(0, z)$. You will notice it is not differentiable at 0, but you can use: $f'(z) = 0$ if $z < 0$ and $f'(z) = 1$ if $z \geq 0$. (You can also try using the *leaky ReLU* activation function.) For more information see
   `https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning`

   (c) Try using the *tanh* activation function, $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$. The derivative of *tanh* is $f'(z) = 1 - (f(z))^2$. For more information see `http://ufldl.stanford.edu/wiki/index.php/Neural_Networks`

   (d) Try using the one of the other activation functions

   (e) Try changing the number of iterations

   (f) Try changing the number of hidden layers

   What gave the best performance?