Homework Assignment 7: Neural Networks

Due Sunday, April 12, 2020 at 11:59 pm EST

Description and dataset instructions

In class, we studied how to use a neural network to make a prediction through forward propagation. Also, we learned about the training of network parameters using backpropagation. In this assignment, we are going to train and use a neural network for hand-written digits (from 0 to 9; i.e., 10 classes) recognition.

You are given a data set in HW7_data1.mat of handwritten digits. The file contains a training matrix X and labels vector y. These matrices can be read directly into your program by using the `load` command. Each row in X corresponds to one training example. There are 5000 training examples in HW7_data1.mat, where each training example is a 20 pixel by 20 pixel grayscale image of the digit. Each pixel is represented by a floating point number indicating the grayscale intensity at that location. The 20 by 20 grid of pixels is "**unrolled**" into a 400-dimensional vector. Each of these training examples becomes a single row in our data matrix X. This gives us a 5000 by 400 matrix X where every row is a training example for a handwritten digit image. The second part of the training set is a 5000-dimensional vector y that contains labels for the training set. To make things more compatible with Matlab indexing, where there is no zero index, we have mapped the digit zero to the value ten. Therefore, a **"0" digit is labeled as "10"**, while the digits "1" to "9" are labeled as "1" to "9" in their natural order.

What to submit

Download and unzip the template ps7_matlab_template.zip. Rename it to `ps7_LastName_FirstName` and add in your solutions:

`ps7_LastName_FirstName/`

- input/ - input images, videos or other data supplied with the problem set
- output/ - directory containing output images and other files your code generates
- ps7.m - code for completing each part, esp. function calls; all functions themselves must be defined in individual function files with filename same as function name, as indicated
- *.m Matlab/Octave function files (one function per file), or any utility code
- ps7_LastName_FirstName_debugging.m – one m-file that has all of your codes from all the files you wrote for this assignment. It should be a concatenation of your main script and all of your functions in one file (simply copy all the codes and paste them in this file). In fact, this file in itself can be executed and you can regenerate all of your outputs using it.
- ps7_report.pdf - a PDF file with all output images and text responses

Zip it as `ps7_LastName_FirstName.zip`, and submit on Canvas.

Guidelines

1. Include all the required images in the report to avoid penalty.
2. Include all the textual responses, outputs and data structure values (if asked) in the report.
3. Make sure you submit the correct (and working) version of the code.

4. Include your name and ID on the report.
5. Comment your code appropriately.
6. Please avoid late submission. Late submission is not acceptable.
7. Plagiarism is prohibited as outlined in the Pitt Guidelines on Academic Integrity.

Questions

1. **Forward Propagation**

For this part, you will be using parameters from a neural network that we have already trained. Your goal is to implement the forward propagation algorithm using the provided weights (HW7_weights_1.mat) to make predictions. The neural network architecture is shown in the figure below.
You have been provided with a set of network parameters $\Theta^{(1)}$ *and* $\Theta^{(2)}$ that are previously trained. These are stored in HW7_weights_1.mat (A file that contains two matrices Theta1 and Theta2). The parameters have dimensions that are sized for a neural network with 25 units in the second layer and 10 output units (corresponding to the 10 digit classes). Thus, Theta1 has size 25 x 401 and Theta2 has size 10 x 26.

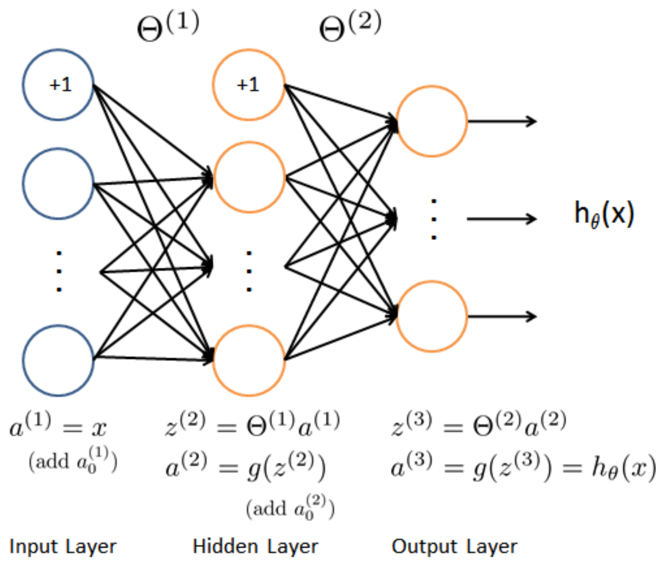

Figure 1. Neural Network Model

a. Load the training set and the network parameters into your workspace. Verify the dimensions of X, y, Theta1, and Theta2.

b. Randomly pick 16 rows from the matrix X. Reshape each row into a 20x20 matrix and visualize them, as images, in a 4x4 grid (subplots) figure. The commands `reshape` and `imagesc` are helpful.

Output:

- An image, ps7-1-b.png that contain 16 randomly picked digit samples.

c. Now, you should implement the forward propagation computation that computes $h_\theta(x^{(i)})$ for every example i and returns the associated predictions. Similar to the one-vs-all classification strategy, the prediction from the neural network should be the label that has the largest output $h_\theta(x^{(i)})_k$. **Write the function** `p = predict(Theta1, Theta2, X)` to return the neural network's prediction p for each sample (row) in X (a class label between 1 to 10). Please use the provided sigmoid function to compute the activations $g(z^{(l)})$.

Once you are done, call your `predict` function using the loaded set of parameters for Theta1 and Theta2 and training data X, and compute the prediction accuracy of the entire training set. For debugging, you should expect an accuracy larger than 96%.

Function file: `predict.m` containing function `predict.`

Output: (`textual response`):
- Report the accuracy of your prediction

Implementation Note: The matrix X contains the examples in rows. When you complete the code in predict.m, you will need to add the column of 1's to the matrix (the bias). The matrices Theta1 and Theta2 contain the parameters for each unit in rows. Specifically, the first row of Theta1corresponds to the first hidden unit in the second layer. In MATLAB, when you compute $z^{(2)} = \Theta^{(1)}a^{(1)}$, be ure that you index (and if necessary, transpose) X correctly so that you get $a^{(l)}$ as a column vector.

2. Sigmoid gradient

Gradient for the sigmoid function can be computed as

$$g'(z) = \frac{d}{dz}g(z) = g(z)(1 - g(z))$$

where

$$\text{sigmoid}(z) = g(z) = \frac{1}{1 + e^{-z}}.$$

Write a function `function g_prime = sigmoidGradient(z)` to compute the gradient of the sigmoid function. Your code should also work with vectors. For a vector input, your function should perform the sigmoid gradient function on every element. Hint: you can use the provided sigmoid function.
Test your function for the input z = [-10, 0, 10]'; the output should be close to be g_prime = [0, 0.25, 0]'.

Function file: `sigmoidGradient.m` containing function `sigmoidGradient.`

Output:

- `The sigmoid gradient when` z = [-10, 0, 10]’

3. Cost function and Backpropagation for gradient of cost function

In this part, you will implement the backpropagation algorithm for neural networks and apply it to the task of hand-written digit recognition. We are going to use the same network architecture in part 1.

The main outcome is to compute the cost function and gradient. For this purpose, you need to complete the code in the provided nnCostFunction.m to return the cost and the gradient of the cost function. Once you have computed the gradient, you will be able to train the neural network by minimizing the cost function $J(\Theta)$ using an advanced optimizer such as `fmincg` (provided).

a. First, complete the code in nnCostFunction.m to return the cost. The cost function for neural networks with regularization is given by

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] +$$
$$\frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}(\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\Theta_{j,k}^{(2)})^2\right].$$

where $h_\theta(x^{(i)})$ is computed as shown in figure 1 (and as you implemented in part 1) and $K = 10$ is the total number of possible labels. Note that $h_\theta(x^{(i)})_k = a_k^{(3)}$ is the activation (output value) of the k-th output unit. Also, recall that whereas the original labels (in the variable y) were 1, 2, ..., 10, for the purpose of training a neural network, we need to recode the labels as vectors containing only values 0 or 1, so that

$$y = \begin{bmatrix}1\\0\\0\\\vdots\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\\vdots\\0\end{bmatrix}, \quad \ldots \quad \text{or} \quad \begin{bmatrix}0\\0\\0\\\vdots\\1\end{bmatrix}$$

You should implement the forward computation that computes $h_\theta(x^{(i)})$ for every example i and sum the cost over all examples. You can assume that the neural network will only have 3 layers - an input layer, a hidden layer and an output layer. **However**, your code should work for any number of input units, hidden units and outputs units. While we have explicitly listed the indices above for $\Theta^{(1)}$ and $\Theta^{(2)}$ for clarity, do note that **your code should in general work with $\Theta^{(1)}$ and $\Theta^{(2)}$ of any size**. Note that you should not be regularizing the terms that correspond to the bias. For the matrices Theta1 and Theta2, this corresponds to the first column of each matrix.

Once you are done, you need to test your code. For testing, use the provided weights Theta1 and Theta2 from HW7_weights_1.mat, data (X and y) from part 1, and call your nnCostFunction in your main script. Note that, the input `nn_params` is the unrolled Theta vector. Therefore, before calling nnCostFunction, you will need to unroll your parameters. You can use this line `nn_params = [Theta1(:) ; Theta2(:)];` to unroll. If you implemented the cost correctly, when $\lambda = 0$, the cost J must be less than 0.3, and when $\lambda = 1$, the cost must be less than 0.4. Please note that you need to properly initialize `input_layer_size, hidden_layer_size, num_labels` to reflect our network architecture.

Output: (`textual response`):

- The value of `J` when $\lambda = 0$ *and* 1, respectavily.

Implementation Note: The matrix `X` contains the examples in rows (i.e., `X(i,:)'` is the i-th training example $x^{(i)}$, expressed as a $n \times 1$ vector.) When you complete the code in `nnCostFunction.m`, you will need to add the column of 1's to the `X` matrix. The parameters for each unit in the neural network is represented in `Theta1` and `Theta2` as one row. Specifically, the first row of `Theta1` corresponds to the first hidden unit in the second layer. You can use a `for`-loop over the examples to compute the cost.

b. Now, you will implement the backpropagation algorithm to compute the gradient for the neural network cost function. You will need to complete the nnCostFunction.m so that it returns an appropriate value for `grad`. Recall that the intuition behind the backpropagation algorithm is as follows. Given a training example $(x^{(t)}; y^{(t)})$, we will first run a "forward pass" to compute all the activations throughout the network, including the output value of the hypothesis $h_\theta(x)$. Then, for each node j in layer l, we would like to compute an "error term" $\delta_j^{(l)}$ that measures how much that node was "responsible" for any errors in our output.
For an output node, we can directly measure the difference between the network's activation and the true target value, and use that to define $\delta_j^{(3)}$ (since layer 3 is the output layer). For the hidden units, you will compute $\delta_j^{(l)}$ based on a weighted average of the error terms of the nodes in layer $(l + 1)$.
The backpropagation is depicted in figure 2 below. **Please refer to HW7_suuplement.pdf** for more detailed steps. You should implement steps 1 to 4 in a loop that processes one example at a time. Concretely, you should implement a for-loop `for t = 1:m` and place steps 1-4, from HW7_suuplement.pdf, inside the for-loop with the t^{th} iteration performing the calculation on the t^{th} training example $(x^{(t)}; y^{(t)})$. Step 5 will divide the accumulated gradients by m to obtain the gradients for the neural network cost function.

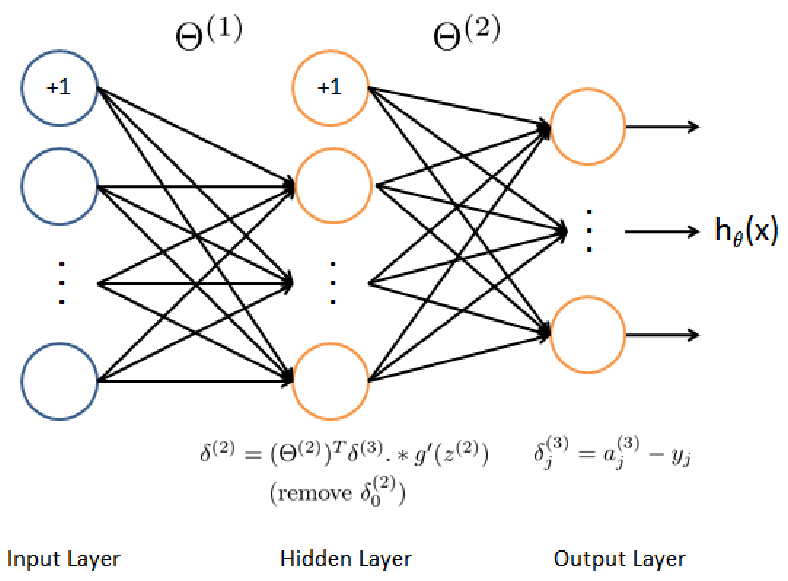
$\Theta^{(1)}$
$\Theta^{(2)}$
+1
+1
$h_\theta(x)$
$\delta^{(2)} = (\Theta^{(2)})^T\delta^{(3)} .* g'(z^{(2)})$
(remove $\delta_0^{(2)}$)
$\delta_j^{(3)} = a_j^{(3)} - y_j$
Input Layer
Hidden Layer
Output Layer

Figure 2. Backpropagation updates.

To **test** you code, you need to compare your gradient computed by backpropagation to the numerically computed gradient as we discussed in the lecture. We implemented the numerical gradient for you! Run this command in your main script:

```
checkNNGradients;
```

checkNNGradients.m will create a small neural network and dataset that will be used for checking your gradients. If your backpropagation implementation is correct, you should see a relative di erence that is less than 1e-9. Once your cost function passes the gradient check for the (unregularized) neural network cost function, you should be able to move to the next point.

Output:

- Screenshot from what you get after running `checkNNGradients`.

c. After you have successfully implemented the backpropagation algorithm, you will add regularization to the gradient. To account for regularization, **it turns out that you can add this as an additional term after computing the gradients using backpropagation**. Specifically, after you have computed $\Delta_{ij}^{(l)}$ using backpropagation, you should update/modify your `grad` in nnCostFunction to account for regularization using:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} \qquad \text{for } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} + \frac{\lambda}{m}\Theta_{ij}^{(l)} \qquad \text{for } j \geq 1$$

Note that you should not be regularizing the first column of $\Theta^{(l)}$ which is used for the bias term.

To test your implementation, run this code in your main script:

```
lambda = 3;
checkNNGradients(lambda);
```

If your code is correct, you should expect to see a relative difference that is less than 1e-9.

Output:

- Screenshot from what you get after running `checkNNGradients(lambda)`

d. After you have successfully implemented the neural network cost function and gradient computation, the next step to start with randomly initialized parameter (we did the renadom initialization for you) and use `fmincg` (advanced optimization) to train your network and learn a good set parameters. Please consider adding the following code to your main script to train your neural network. Please note that you need to define the network layer sizes, `input_layer_size, hidden_layer_size, num_labels`, before using this code. Also use X and y from part 1.

```
%  Initialize parameters (randInitializeWeights.m)
initial_Theta1 = randInitializeWeights(input_layer_size, hidden_layer_size);

initial_Theta2 = randInitializeWeights(hidden_layer_size, num_labels);

% Unroll parameters
initial_nn_params = [initial_Theta1(:) ; initial_Theta2(:)];
% Train the network
options = optimset('MaxIter', 50);
 lambda = 1;
 % Create "short hand" for the cost function to be minimized
costFunction = @(p) nnCostFunction(p, ...
                                   input_layer_size, ...
                                   hidden_layer_size, ...
                                   num_labels, X, y, lambda);

% Now, costFunction is a function that takes in only one argument (the
% neural network parameters)
[nn_params, cost] = fmincg(costFunction, initial_nn_params, options);

% Obtain Theta1 and Theta2 back from nn_params
Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
                 hidden_layer_size, (input_layer_size + 1));

Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), ...
                 num_labels, (hidden_layer_size + 1));
```

Now, once you have the trained Thetas (Theta1 and Theta2), you can proceed and make predictions (using you predict function from part 1), and compute the training accuracy using the training data from part 1. Run you training code for different values of lambda and different number of iterations, and report your prediction accuracy under each combination. Specifically, you need to fill in the table below with the prediction accuracies at each (lambda, #iteration) combinations.

Table 1. Training accuracy at different values for lambda and number of iterations

	`MaxIter = 50`	`MaxIter = 100`	`MaxIter = 200`	`MaxIter = 400`
$\lambda = 0$				
$\lambda = 1$				
$\lambda = 2$				
$\lambda = 4$				

Output:

- Complete table 1 and add it to your code. MATLAB table format is also fine.
- Comment on your results