Homework Assignment 7: Neural Networks

Due Friday, April 7, 2023 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 the standard iris flower classification problem. This is perhaps the best-known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.

This dataset is well studied and is a good problem for practicing on neural networks because all of the 4 input variables are numeric and have the same scale in centimeters. Each instance describes the properties of an observed flower measurements and the output variable is specific iris species. You are given the data set in HW7_Data.mat of iris flowers' sepal and petal measurements (features) and the corresponding class labels. The file contains a feature matrix X and labels vector y. You will need to load the data in your code. Each row in X corresponds to one training example. There are 150 examples in HW7_Data.mat, where each training example is a 4x1 vector that contains measurements/features. This gives us a 150 by 4 matrix X where every row is a training example for a handwritten digit image. The second part of the data set is a 150-dimensional vector y that contains labels for the data set. Lablels 1, 2, and 3 correspond to Iris Setosa, Iris Versicolour, and Iris Virginica, respectively.

For this this multi-class classification problem, we train a 3-layer network with 4 input units, 8 hidden units, and 3 output units. This allows us to discriminate between the three different classes for this problem

What to submit

Download and unzip the template ps7_matlab_template.zip. Rename it to `ps7_xxxx_LastName_FirstName` and add in your solutions:

`ps7_xxxx_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 or ps7.py - code for completing each part, esp. function calls.
- *.m or *.py Matlab/Python function files, or any utility code
- ps7_report.pdf - a PDF file with all output images and text responses

Zip it as `ps7_xxxx_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

0. Read Data

Load the dataset from HW7_Data.mat and verify the dimensions of your feature matrix X and labels vector y.

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_2.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_2.mat (A file that contains two matrices Theta1 and Theta2). The parameters have dimensions that are sized for a neural network with 8 units in the second layer and 3 output units (corresponding to the three flower classes). Thus, Theta1 has size 8 x 5 and Theta2 has size 3 x 9.

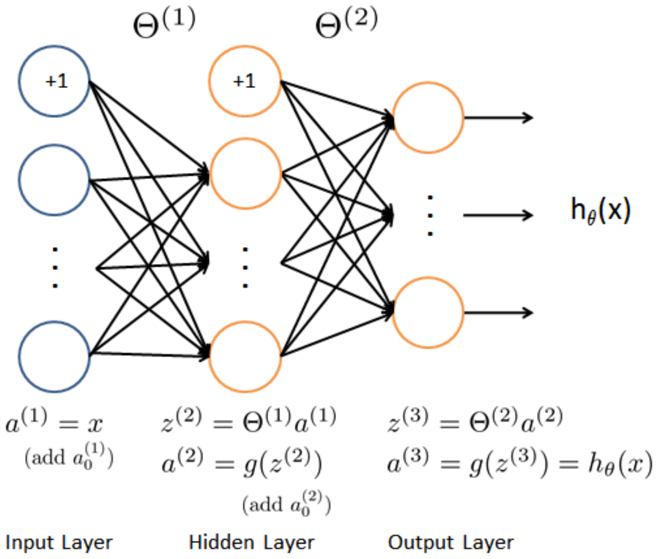

Figure 1. Neural Network Model

a. Implement the forward propagation computation that computes $h_\theta\left(x^{(q)}\right)$ for every example q 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 $h_\theta\left(x^{(q)}\right)_k$. **Write the function** `[p h_x] = predict(Theta1, Theta2, X)` to return the neural network's prediction p (a class label between 1 to 3) and the corresponding output hypothesis, h_x, for each sample (row) in X. For this problem, since we have three possible classes, the size of h_x should be #rows(X) x 3. Please use the provided sigmoid function to compute the activations $g\left(z^{(l)}\right)$.

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

> **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 Theta1 corresponds to the first hidden unit in the second layer. In MATLAB, when you compute $z^{(2)} = \Theta^{(1)} \mathrm{a}^{(1)}$, be sure that you index (and if necessary, transpose) X correctly so that you get $\mathrm{a}^{(l)}$ as a column vector.

b. To test your function, load the pre-trained weight matrices Theta1 and Theta2 from HW7_weights_2.mat into your workspace. Call your `predict` function using the loaded set of parameters for Theta1 and Theta2 and feature data X, and compute the prediction accuracy of the entire training set. For debugging, you should expect an accuracy larger than 98%.

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

2. Cost function

a. Write a MATLAB function, `function J = nnCost(Theta1, Theta2, X, y, K, lambda)` that computes the cost function (J) given the weight matrices, feature matrix, labels vector, number of classes (K), and the regularization parameter (λ). The cost function for neural networks with regularization is given by

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\left[\sum_{m=1}^{8}\sum_{n=1}^{4}\left(\Theta_{m,n}^{(1)}\right)^2 + \sum_{m=1}^{3}\sum_{n=1}^{8}\left(\Theta_{m,n}^{(2)}\right)^2\right]$$

where $h_\theta\left(x^{(i)}\right)$ is computed as shown in figure 1 (and as you implemented in part 1) and $K = 3$ is the total number of possible labels. Note that $h_\theta\left(x^{(i)}\right)_k = a_k^{(3)}$ 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, and 3, for the purpose of training a neural network, we need to recode the labels as vectors containing only values 0 or 1, so that

$$y = \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, or \begin{bmatrix}0\\0\\1\end{bmatrix}$$

You should implement the forward computation that computes $h_\theta\left(x^{(i)}\right)$ 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.

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

b. Once you are done, you need to test your code. For testing, use the provided weights Theta1 and Theta2 from HW7_weights_2.mat, and the entire data (X and y) from part 1, and call your nnCost function in your main script. 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 1.1.

Output: (`textual response`):

- The value of `J` when $\lambda = 0, 1, and\ 2$ respectavily.

3. Derivative of the activation function: 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]'

4. Backpropagation for gradient of cost function and stochastic gradient descent

In this part, you will implement the backpropagation algorithm for neural networks to learn the network weights and apply it to the task of iris flower classification. We are going to use the same network architecture in part 1. To achieve this goal, you'll need to develop this function to learn the network weights:

```
function [Theta1, Theta2] = sGD(input_layer_size,
hidden_layer_size, num_labels, X_train, y_train, lambda, alpha,
MaxEpochs)
```

Inputs:

- `input_layer_size`: is the size of the input layer, i.e., the number of units in the input layer,
- `hidden_layer_size`: is the size of the hidden layer,
- `num_labels`: is the number of the different labels (classes). The number of output units equals to this value if we have more than two classes.
- `X_train`: is the training matrix, where each row corresponds to one training example of features.
- `y_train`: is the corresponding class label for each training feature
- `lambda`: regularization parameter.
- `alpha`: gradient descent learning rate
- `MaxEpochs`: is the max number of epochs (network sees all training data once) to run your algorithm.

Outputs:

- `Theta1`: the trained weights $\Theta^{(1)}$
- `Theta2`: the trained weights $\Theta^{(2)}$

You can always assume that you have a three-layer network. However, your function must be robust to any network size. Below are the steps to train your network using backpropagation and stochastic gradient descent:

a. Define `Theta1` and `Theta2`, and randomly initialize them. The values in each matrix **must** belong to a uniform distribution from -0.1 to 0.1 (Hint: MATLAB `rand` function may be helpful.) Then you'll loop over each epoch, and within each epoch you'll perform backpropagation and gradient descent on each training example sequentially.

b. Now, you will implement the backpropagation algorithm to compute the gradient for the neural network cost function, and use the stochastic gradient descent to update the weights. Recall that the intuition behind the backpropagation algorithm is as follows. Given a training example $(x^{(q)}; y^{(q)})$, 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). Remember to follow the same convention for the true output as indicated in section 2 above. 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 lecture slides** for more detailed steps. Once you have the gradient you can update the weights of the network $\Theta^{(1)}$ and $\Theta^{(2)}$. Note, we don't compute the error for the bias unit $\delta_0^{(2)}$.

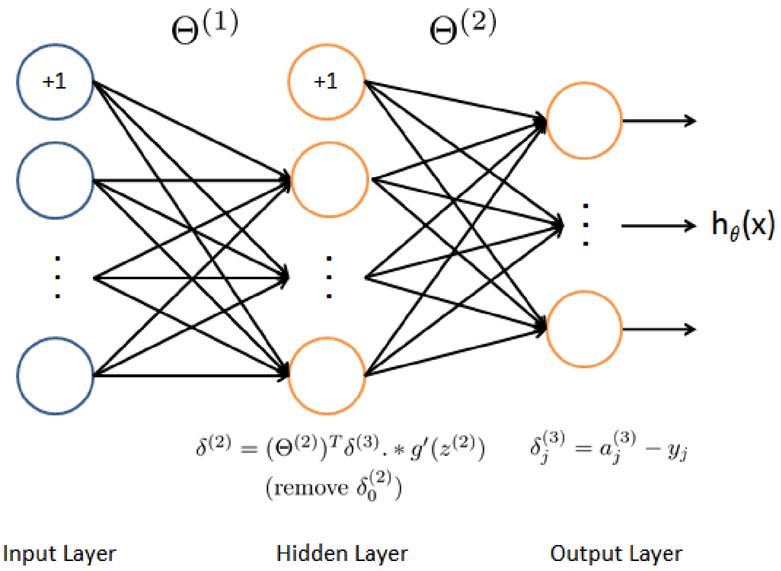
$\Theta^{(1)}$
$\Theta^{(2)}$
+1
+1
$h_\theta(x)$
$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$
(remove $\delta_0^{(2)}$)
$\delta_j^{(3)} = a_j^{(3)} - y_j$
Input Layer
Hidden Layer
Output Layer

Figure 2. Backpropagation updates.

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**. Note that you should not be regularizing the first column of $\Theta^{(l)}$, which is used for the bias term. Specifically, after you have computed $\Delta_{ji}^{(l)}$ using backpropagation, using:

$$D_{ji}^{(l)} = \Delta_{ji}^{(l)} + \lambda\Theta_{ji}^{(l)} \quad \text{if } i \neq 0$$

$$D_{ji}^{(l)} = \Delta_{ji}^{(l)} \quad \text{if } i = 0$$

d. After you have successfully implemented the gradient computation, you are ready to update the weight matrices `Theta1` and `Theta2`.

$$\Theta_{ji}^{(l)} = \Theta_{ji}^{(l)} - \alpha D_{ji}^{(l)}$$

Please remember that stochastic gradient descent updates the parameters once you estimated the gradient using only one training example. In addition, you need to choose a proper learning rate α. Please follow the guidelines we studied in weeks 2 and 3 to pick a proper value for α.

Output: (`textual response`):

- Report the value of α you used

e. After you finish one epoch of data, compute the cost using the entire training set (X_train, y_train). Your program should continue until you reach the max number of epochs, `MaxEpochs`. Please remember that, in each epoch, you perform backpropagation and stochastich gradient descent on each training example sequentially.
Output: A figure that shows the training cost vs iteration number as ps7-4-e-1.png

5. Testing the network

Now, once you have the function to train Thetas (Theta1 and Theta2), you can train you network and make predictions (using you predict function from part 1). **Randomly** split the dataset in HW7_Data.mat into training set (X_train, y_train) and testing set (X_test, y_test), with 85% of the samples for training and 15% for testing. Run you training code for different values of lambda and different number of epochs, and report your prediction accuracy on both the training and testing sets under each combination. Specifically, you need to fill in the table below with the prediction accuracies at each (lambda, #iteration) combinations. Also, compute the cost associated with each case in table 1.

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

	MaxEpochs = 50		MaxEpochs = 100	
	Training data accuracy	Testing data accuracy	Training data accuracy	Testing data accuracy
$\lambda = 0$				
$\lambda = 0.01$				
$\lambda = 0.1$				
$\lambda = 1$				

Output:

- Complete table 1 and add it to your report.
- The cost for each case in table 1. Format your answer in a table and add it to your report.
- Discuss your results.