Assignment 04

In this assignment, you use Neural Networks with the insurance practice project described at

https://www.scriptedin.com/contests/view/6

The data sets can be downloaded, and are provided on HuskyCT.

The computeCost function takes the data set X, for each example/observation in X, does the following (Here L = 3 indicating 3 layers):

I. First the labels are converted:

“1”, which is “No” is converted to a vector of [1,0]; and “2”, which is “Yes” is converted to [0,1] using the identity matrix trick. The new labels are stored in the variable Y.

II. Forward propagation**:** For each example/observation x

1. $a^{(1)} = x$
2. Add bias value $a^{(1)}{}_0 = 1$
3. For *l* =2… L
    a. $z^{(l)} = \theta^{(l-1)} a^{(l-1)}$
    b. $a^{(l)} = g(z^{(l)})$
    c. Add bias value $a^{(l)}{}_0 = 1$
4. End
5. $h_\theta(x) = a^{(L)}$

III. Now compute the regularized cost function

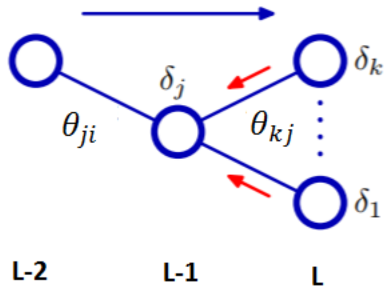
$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left(-y^{(i)}{}_k \log h_\theta\left(x^{(i)}\right)_k - \left(1 - y^{(i)}{}_k\right)\log\left(1 - h_\theta(x^{(i)})_k\right)\right)$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\theta^{(l)}{}_{ij}\right)^2$$

IV. Now the function starts back-propagating

The error $\delta_k{}^{(L)}$ at node k in the output layer L is the difference between the output value $a_k{}^{(L)}$ and the actual value y_k:

$$\delta_k{}^{(L)} = a_k{}^{(L)} - y_k$$



The error made by node j in the layer just before the output layer is

$$\delta_j{}^{(L-1)} = h'\left(z_j{}^{(L-1)}\right)\sum_{k=1}^{K} \theta_{kj}\delta_k{}^{(L)}$$

The sum above is the weighted sum of error for all output nodes, weighted by the weights between the output nodes and node j.

In the case of a tanh function, at layer l, the output:

$$a^{(l)}{}_j = h\left(z^{(l)}{}_j\right) = tanh\left(z^{(l)}{}_j\right) = \frac{e^{z^{(l)}{}_j} - e^{-z^{(l)}{}_j}}{e^{z^{(l)}{}_j} + e^{-z^{(l)}{}_j}}$$

Can prove that:

$$h\left(z^{(l)}{}_j\right) = 1 - tanh\left(z^{(l)}{}_j\right)^2 = 1 - a^{(l)}{}_j{}^2$$

so the error above becomes

$$\delta_j{}^{(L-1)} = \left(1 - a^{(L-1)}{}_j{}^2\right)\sum_{k=1}^{N_k} \theta_{kj}\delta_k{}^{(L)}$$

N_k is the total number of output nodes at the output layer L

In the case of a logistic function, at layer l, the output:

$$a^{(l)}{}_j = h\left(z^{(l)}{}_j\right) = sigmoid\left(z^{(l)}{}_j\right) = \frac{1}{1 + e^{-z^{(l)}{}_j}}$$

Can prove that:

$$h\left(z^{(l)}{}_j\right) = sigmoid\left(z^{(l)}{}_j\right)\left(1 - sigmoid\left(z^{(l)}{}_j\right)\right) = a^{(l)}{}_j\left(1 - a^{(l)}{}_j\right)$$

so the error above becomes

$$\delta_j{}^{(L-1)} = a^{(l)}{}_j\left(1 - a^{(L-1)}{}_j\right)\sum_{k=1}^{N_k} \theta_{kj}\delta_k{}^{(L)}$$

In general, going backward from layer l+1 to layer l, the error at node i in layer l is

$$\delta_i{}^{(l)} = a^{(l)}{}_i\left(1 - a^{(l)}{}_i\right)\sum_{j=1}^{N_j} \theta_{ji}\delta_j{}^{(l+1)}$$

It may be better to write the error in the vector form. The error $\delta^{(L)}$ for all output nodes, and their actual values

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

Vector $\delta^{(L)}$ has all the error $\delta_k{}^{(L)}$for layer L in it, and similarly $a^{(L)}$ has all the error $a^{(L)}{}_k$ for layer L in it, y has all the true values of 0/1 for layer L in it. For other layers, from layer to layer going backward, the error is

$$\delta^{(l)}{=}a^{(l)}\left(1 - a^{(l)}\right).*\,\theta^{(l)^T}\delta^{(l+1)}$$

The “.*” operation is element-by-element multiplication.

The algorithm pseudo code for computing errors at different layers would be, starting from the output layer L

1. $\delta^{(L)} = a^{(L)} - \mathrm{y}$
2. For *l*=L-1…2
    a. $\delta^{(l)}{=}a^{(l)}\left(1 - a^{(l)}\right).*\,\theta^{(l)^T}\delta^{(l+1)}$
3. End

V. The gradient of the cost function for the instance n is

$$\frac{\partial J_n}{\partial \theta_{ji}} = \frac{\partial J_n}{\partial z_j}\frac{\partial z_j}{\partial \theta_{ji}}$$

But

$\frac{\partial z_j}{\partial \theta_{ji}} = a_i$ (as $z_j = \sum_{i=1}^{N_i} a_i\,\theta_{ji}$, weighted sum of all the outputs from the previous layer i)

And

$$\frac{\partial J_n}{\partial z_j} = \delta_j$$

(The layer notation, can be any layer l, is dropped in this equation to make it less confusing), so

$$\frac{\partial J_n}{\partial \theta_{ji}} = \delta_j a_i$$

To compute the gradient of the cost function:

Repeat until convergence:

1. $\Delta^{(l)}$ = zeros(shape of $\theta^{(l)}$), for each layer *l*=1..L-1
2. For each instance in the training dataset
    a. Forward propagation to compute $a^{(l)}$, for each layer *l*=1..L
    b. Backpropagation to compute $\delta^{(l)}$, for each layer *l*=L..2
    c. Compute $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)} a^{(l)^T}$, for each layer *l*=1..L-1
3. End
4. Compute the regularized gradient $\frac{\partial J(\theta)}{\partial \theta^{(l)}} = \frac{1}{m}\Delta^{(l)} + \lambda\theta^{(l)}$, for each layer *l*=1..L-1

The shape/size of $\Delta^{(l)}$ is the same as the shape/size of $\theta^{(l)}$, which is a matrix. The for loop in step 2 updates the derivatives in the matrix $\Delta^{(l)}$.

The cost and the gradient of the cost are stored in the variables J and grad.

For each layer:

$A^{(l)}$: contains multiple a on its rows, each is corresponding to an example/observation. Note that the last $A^{(L)}$ is also the output H.

$Z^{(l)}$: contains multiple z on its rows, each is corresponding to an example/observation.

delta $\delta^{(l)}$ contains the error on each node at each layer l. BigDelta $\Delta^{(l)}$ contains gradient at layer l.

1. (1 point) Write the normalize() function that standardizes/normalizes variables in X

2. (1 point) Write a signmoid() function of z to return a sigmoid/tanh of z

3. (1.5 point) Write a gradient() function to return gradient of the nonlinearity

4. (1.5 point) Write the computeCost() function that returns the cost

5. (1.5 point) Write the computeGrad() function that returns the gradient

6. (1.5 point) Write a predict() function to provide prediction.

7. (2 point) Write a optimize() function that implements the gradient descent updates to get the optimal theta θ

The main code was provided, that calls functions above to train a model using the training data and return θ, then test the model using the test data, compute accuracy, confusion matrix, precision, and recall.

Note that in computeCost() and computeGrad() functions, Theta variables are flatten in theta to pass to these functions, then reshaped to the original later in the functions. In the main code, it looks like:

```
theta = np.concatenate(
    (Theta1.reshape(hidden_num * (1 + input_num), 1, order="F"),
     Theta2.reshape(label_num * (1 + hidden_num), 1, order="F")))
```

Where Theta1, Theta1 are the weight between the input layer and the hidden layer, and between the hidden layer and the output layer, respectively.

Submission on Husky includes a zip file of:

a. A notebook named as “group_xxx_assignment_yyy.ipynb”
b. A Word document/article with detailed explanation
c. All the other files

Also share the notebook and the article (only after the deadline) to the project site.