Neural Networks: Sigmoid Activation Function for continuous output variable

Okay, so I am in the middle of Andrew Ng's machine learning course on coursera and would like to adapt the neural network which was completed as part of assignment 4.

In particular, the neural network which I had completed correctly as part of the assignment was as follows:

- Sigmoid activation function: `g(z) = 1/(1+e^(-z))`
- 10 output units, each which could take 0 or 1
- 1 hidden layer
- Back-propagation method used to minimize cost function
- Cost function:

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

where `L=number of layers , s_l = number of units in layer l , m = number of training examples , K = number of output units`

Now I want to adjust the exercise so that there is one continuous output unit that takes any value between [0,1] and I am trying to work out what needs to change, so far I have

- Replaced the data with my own, i.e.,such that the output is continuous variable between 0 and 1
- Updated references to the number of output units
- Updated the cost function in the back-propagation algorithm to:

$$J = \frac{1}{2m}\Sigma_{i=1}^{m}(g(a_3) - y)^2 + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

where `a_3` is the value of the output unit determined from forward propagation.

I am certain that something else must change as the gradient checking method shows the gradient determined by back-propagation and that by the numerical approximation no longer match up. I did not change the sigmoid gradient; it is left at `f(z)*(1-f(z))` where `f(z)` is the sigmoid function `1/(1+e^(-z)))` nor did I update the numerical approximation of the derivative formula; simply `(J(theta+e) - J(theta-e))/(2e)` .

Can anyone advise of what other steps would be required?

Coded in Matlab as follows:

```
% FORWARD PROPAGATION
% input layer
a1 = [ones(m,1),X];
% hidden layer
z2 = a1*Theta1';
a2 = sigmoid(z2);
a2 = [ones(m,1),a2];
% output layer
z3 = a2*Theta2';
a3 = sigmoid(z3);

% BACKWARD PROPAGATION
delta3 = a3 - y;
delta2 = delta3*Theta2(:,2:end).*sigmoidGradient(z2);
Theta1_grad = (delta2'*a1)/m;
Theta2_grad = (delta3'*a2)/m;

% COST FUNCTION
J = 1/(2 * m) * sum( (a3-y).^2 );

% Implement regularization with the cost function and gradients.
Theta1_grad(:,2:end) = Theta1_grad(:,2:end) + Theta1(:,2:end)*lambda/m;
Theta2_grad(:,2:end) = Theta2_grad(:,2:end) + Theta2(:,2:end)*lambda/m;
J = J + lambda/(2*m)*( sum(sum(Theta1(:,2:end).^2)) + sum(sum(Theta2(:,2:end).^2))
```

I have since realised that this question is similar to that asked by @Mikhail Erofeev on StackOverflow, however in this case I wish the continuous variable to be between 0 and 1 and therefore use a sigmoid function.

matlab | machine-learning | neural-network

edited Dec 18 '13 at 22:49

asked Dec 18 '13 at 2:05

user1420372

204 1 8

add comment

1 Answer

First, your cost function should be:

```
J = 1/m * sum( (a3-y).^2 );
```

I think your `Theta2_grad = (delta3'*a2)/m;` is expected to match the numerical approximation after changed to `delta3 = 1/2 * (a3 - y);`).

Check this slide for more details.

EDIT: In case there is some minor discrepancy between our codes, I pasted my code below for your reference. The code has already been compared with numerical approximation function `checkNNGradients(lambda);` , the Relative Difference is less than `1e-4` (not meets the `1e-11` requirement by Dr.Andrew Ng though)

```
function [J grad] = nnCostFunctionRegression(nn_params, ...
                                    input_layer_size, ...
                                    hidden_layer_size, ...
                                    num_labels, ...
                                    X, y, lambda)

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))):en
                 num_labels, (hidden_layer_size + 1));

m = size(X, 1);
J = 0;
Theta1_grad = zeros(size(Theta1));
Theta2_grad = zeros(size(Theta2));


X = [ones(m, 1) X];
z1 = sigmoid(X * Theta1');
zs = z1;
z1 = [ones(m, 1) z1];
z2 = z1 * Theta2';
ht = sigmoid(z2);


y_recode = zeros(length(y),num_labels);
for i=1:length(y)
    y_recode(i,y(i))=1;
end
y = y_recode;


regularization=lambda/2/m*(sum(sum(Theta1(:,2:end).^2))+sum(sum(Theta2(:,2:end).
J=1/(m)*sum(sum((ht - y).^2))+regularization;
```

edited Dec 18 '13 at 22:50

answered Dec 18 '13 at 5:48

lennon310

10.4k 3 12 32

Thank-you for your suggestions; I tried both updating delta3 and delta2 as you suggested, but still the gradients don't match. – user1420372 Dec 18 '13 at 21:43

@user1420372 your cost function should be a3-y rather than sigmoid(a3)-y, see my update in answer. – lennon310 Dec 18 '13 at 22:25

Thanks! I had actually just noticed that - however the gradient is still incorrect - will edit code in question to fix. – user1420372 Dec 18 '13 at 22:48

I added my code in the answer for your reference. It matches the numercial approximation, although the relative difference is 1e-4, which is larger than 1e-11 – lennon310 Dec 18 '13 at 22:57

Thanks a lot for your help. I have also realised that by leaving the cost function as it was in the exercise, i.e., with the logs, the gradients match well and the output kind of looks like predicted (preliminary - still testing). – user1420372 Dec 19 '13 at 5:21

show **1** more comment

Not the answer you're looking for? Browse other questions tagged matlab machine-learning neural-network **or ask your own question.**