Neural Network implementation in octave/matlab

Asked 3 years, 11 months ago Modified 3 years, 11 months ago Viewed 574 times

0

I'm trying to make a simple neural network formed by three layers to resolve a binary classification problem. The first two layers have eight neurons (+ the bias units). I'm using `fminunc`. This is my cost function:

```
function [jVal, gradient] = cost2(thetaVec, X, y)
  Theta1 = reshape(thetaVec(1:72),8, 9); % my weights used for
  Theta2 = reshape(thetaVec(73:81),1, 9); %forward propagation
  Delta1 = 0; %Delta is divided in Delta1 and Delta2 for simplicity but
  Delta2 = 0; %they're combined to eventually calculate the gradient
  jVal   = 0; %the value of the costfunction
  m = length(y);
  for i = 1:m
    a1 = X(i, :); %X size: 3x9, a1 size: 1x9
     z2 = Theta1 * a1';
     a2 = 1 ./(1 + exp(-z2)); %a2 size: 8x1
     a2 = [ones(columns(a2), 1) a2']; % bias unit added to a2: a2 size:
1x9
     z3 = Theta2 * a2';
     a3 = 1 ./(1 + exp(-z3)); %a3 = h(x(i)) size: 1x1
     jVal   += (-1/m) * (y(i) * log(a3) + (1 - y(i)) * log(1 - a3));
     delta3 = a3 - y(i); %delta3 size: 1x1
     delta2 = Theta2' * delta3 .* a2 .* (1 - a2); %delta2 size: 9x9
     Delta2 += delta3 * a2'; %I use Delta1 and Delta2 as accumulators
     Delta1 += delta2 * a1'; %size Delta2: 9x1, size Delta1: 9x1
   endfor
   jVal = jVal/m; %avarage of jVal
   Delta = [Delta1;Delta2]; %Deltas are combined. Size Delta: 18x1
   gradient = (1/m) * Delta;% size gradient: 18x1
 endfunction
```

My main:

```
    %the values of the vector from which I derive my weights are chosen
randomly
    INIT_EPSILON = 0.1; %between thi interval
    Theta1 = rand(8, 9) * (2*INIT_EPSILON) - INIT_EPSILON;
    Theta2 = rand(1, 9) * (2*INIT_EPSILON) - INIT_EPSILON;
    thetaVec = [ Theta1(:); Theta2(:)];
    options = optimset('GradObj', 'on', 'MaxIter', 10000);
    [optTheta, functionVal, exitFlag] = fminunc(@(t) cost2(t, X, y), thetaVec,
options)
```

gradient should be a matrix 9x9, instead it is 18x1, so I can't use `fminunc`. Actually, I tried to modify the *backpropagation* part in my cost function several times to obtain a gradient 9x9 (in particular I used to change `delta2`). However, it never worked, the output was:

```
optTheta = %a vector of various values
functionVal =  0.71681 %or a similar value
exitFlag =  1
```

So, even if the exitflag was 1 it didn't converged. Where am I doing wrong?

matlab neural-network artificial-intelligence octave gradient-descent

Share

Improve this question

Follow

edited Dec 28, 2020 at 13:47

asked Dec 28, 2020 at 9:18

Shawn _

5 • 3

I think the problem is in the line where you defined 'Delta'. There should be comma instead of a semicolon. This would explain why you have a 18x1 vector instead of a 9x9 matrix. Also you should use '. *' so you calculate elementwise. Please let me know if this helped so that i can form an Answer. – S. Chris Dec 28, 2020 at 9:56

What is X? What are its dimensions? What is y? What are its dimensions? What is thetavec? What does the transformation to Theta1 and Theta2 achieve (is it rows of observations, or rows of features?). What is Delta1 and Delta2. What are their dimensions? What is a1. What is a2. What are their dimensions? What are they *supposed* to be? Don't get me wrong, I can follow the code and I know what you're *trying* to do, but the fact that you don't write your code in a way that makes it explicit is a recipe for disaster, let alone very difficult for an outsider to read. (continuing below) – Tasos Papastylianou Dec 28, 2020 at 11:41

1 As for where the error lies, just at a glance (and given the above comment), I believe it's the fact that you get an a2 which is the "wrong" orientation and you don't notice, and then when you manipulate it further you accidentally treat it as if it was a horizontal vector, when in fact it was a vertical one. Hence why you should be mindful of what dimensions your outputs are at all times, and preferably point these out in your code. – Tasos Papastylianou Dec 28, 2020 at 11:42

@S. Chris, thanks for the comment. I tried to use a comma but it gives me an error, and also if I use a ' the situation doesn't change. Moreover, in a 9x9 matrix there are 81 elements, instead in a 18x1 only 18, so I don't think the problem is in the definition of Delta because in any way I combine Delta 1 and Delta 2 there aren't enough elements. Let me know if there was a misunderstanding and I didn't understand what you meant. Thanks in advance – Shawn _ Dec 28, 2020 at 11:51

@Tasos Papastylianou I edited the code and tried to be clearer, let me know if I have to add something else. Regarding the orientation of a2, do you think I should transpose it to calculate delta2 or Delta2? (lines 17 and 18). Thanks in advance for your help and advices – Shawn _ Dec 28, 2020 at 14:00

1 Answer

Sorted by: Highest score (default)

You currently have the following code:

```
delta3  = a3 - y(i);                              % (1×1)
delta2  = Theta2' * delta3 .* a2 .* (1 - a2);     % (9×1).*(1×9) = (9×9)
Delta2 += delta3 * a2';                           % (9×9)* (9×1) = (9×1)
Delta1 += delta2 * a1';                           % (9×9)* (n×1) = (9×1)
```

I think instead it should be something like:

```
delta3  = a3 - y(i);                              % (1×1)
delta2  = Theta2 * delta3 .* a2 .* (1 - a2);      % (1×9).*(1×9) = (1×9)
Delta2 += delta3 * a2';                           % (9×9)* (9×1) = (9×1)
Delta1 += delta2.' * a1;                          % (9×1)* (1×9) = (9×9)
```

And then you discard the gradients of the bias in Delta1 at each step, ending up with an (8×9) matrix as your ongoing Delta1 component. (you may need to transpose Delta1 first, I haven't followed your transpositions closely).

Finally, the whole point of the vertical concatenation step at the end is to "implode" your matrices back into a single-column long vector form, so that they follow they same "specification" as the input "thetaVec", therefore you'd take your (8×9) and (1×9) Delta1 and Delta2 objects, and 'implode' them, i.e.:

```
[Delta1(:) ; Delta2(:)]
```

UPDATE

Continuing here the discussion in the comments above.

Consider what Delta1 and Delta2 are. This is the total error (over all observations) corresponding to the elements in Theta1 and Theta2 respectively. In other words, Delta1 should have the same size as Theta1, and Delta2 should have the same size as Theta2. Now that you have included the matrix sizes in your code, you can see instantly that this is not the case.

Furthermore, since each iteration adds to these matrices, the result of each iteration should be a Delta1 and Delta2 of the right size, such that you add the errors contributed at each iteration, to get the total error (per parameter) over all iterations.

Also, consider what delta2 and delta3 are. These are also errors, but they do not refer to errors in the parameters, they refer to errors in nodes. In other words, they show the contribution / responsibility of each node in a layer to the final error. Therefore their size needs to be a vector with the same number of elements as there are nodes

in the respective layer. You can already see that it makes no sense for delta2 to have a size of 9x9!

So the logic of the algorithm is this. Take the contribution of the nodes to the error, and backpropagate it, both to the previous nodes, and to the the parameters between them.

E.g. multiply each delta3 (in this case there's only one node) with each node in layer 2, to find how errors were distributed over each parameter element in Theta2.

Similarly, to obtain delta2, we considered the error in delta3, and distributed it to each node in layer 2, by following the parameter multiplication backwards. Then on top of that, we multiplied by the gradient (since this defines to what extent / rate that error would have been propagated forwards).

Now that we've dealt with layer 3 and its interaction with layer 2, we move on to 2 and its interaction with 1. So, similarly, multiply each delta2 (there are 9 nodes in layer 2, so delta2 should have 9 elements) with each node in layer 1. This gives a 9x9 matrix reflecting how each node in layer 1, through parameters Theta1, gave rise to the errors for each node of layer 2. However, because we do not care about the contributions to the 'bias' node of layer 2, we remove this part from Delta1, which leaves us with an 8x9 matrix, exactly the same size as Theta1.

In theory you could have also backpropagated delta2 to find a delta1, but since we have no use for it, we skip this step. After all, what we really care about is the error in the parameters, not in the nodes. (i.e. we only care about the errors in the nodes at each step because we need them to get the errors in the parameters from the layer before).

edited Dec 28, 2020 at 19:51

answered Dec 28, 2020 at 12:39

Tasos Papastylianou
22.2k ● 2 ● 34 ● 63