coursera

Discussion Forums

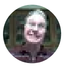
Week 5

SUBFORUMS

All

Assignment: Neural Network Learning

Week 5



Computing the NN cost J using the matrix product

Tom Mosher Mentor · 10 days ago · Edited

Students often ask why they can't use matrix multiplication to compute the cost value J in the Neural Network cost function. This post explains why.

Short answer: You **can** use matrix multiplication, but it is tricky.

Here is the equation for the unregularized cost J:

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

Notice the double-sum. 'i' ranges over the training examples 'm', and 'k' ranges over the output labels 'K'. The cost has two parts - the first involves the product of 'y' and log(h), and the second involves the product of (1-y) and log(1-h). Note that 'y' and 'h' are both matrices of size (m x K), and the multiplication is a scalar product.

Recall that for linear and logistic regression, 'y' and 'h' were both vectors, so we could compute the sum of their products easily using vector multiplication. After transposing one of the vectors, we get a result of size (1 x m) * (m x 1). That's a scalar value. So that worked fine, as long as 'y' and 'h' are vectors.

But the when 'h' and 'y' are matrices, the same trick does not work as easily. Here's why.

Let's first show the math using the element-wise product of two matrices A and B. For simplicity, let's use m= 3 and K=2.

$$A = \begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix}, B = \begin{bmatrix} m & n \\ o & p \\ q & r \end{bmatrix}$$

The sum over the rows and columns of the element-wise product is:

$$\sum\sum A.*B = am + bn + co + dp + eq + fr$$

Now let's detail the math for this using a matrix product. Since A and B are the same size, but the number of rows and columns are not the same, we must transpose one of the matrices before we compute the product. Let's transpose the 'A' matrix, so the product matrix will be size (K x K). We could of course invert the 'B' matrix, but then the product matrix would be size (m x m). The (m x m) matrix is probably a lot larger than (K x K).

It turns out (and is left for the reader to prove) that both the (m x m) and (K x K) matrices will give the same results for the cost J.

$$A' * B = \begin{bmatrix} a & c & e \\ b & d & f \end{bmatrix} * \begin{bmatrix} m & n \\ o & p \\ q & r \end{bmatrix}$$

After the matrix product, we get:

$$A' * B = \begin{bmatrix} (am + co + eq) & (an + cp + er) \\ (bm + do + fq) & (bn + dp + fr) \end{bmatrix}$$

So this is a size (K x K) result, as expected. Note that the terms which lie on the main diagonal are the same terms that result from the double-sum of the element-wise product. The next step is to compute the sum of the diagonal elements using the "trace()" command, or by sum(sum(...)) after element-wise multiplying by an identity matrix of size (K x K).

The sum-of-product terms that are NOT on the main diagonal are unwanted - they are not part of the cost calculation. So simply using sum(sum(...)) over the matrix product will include these terms, and you will get an incorrect cost value.

The performance of each of these methods - double-sum of the element-wise product, or the matrix product with either trace() or the sum of the diagonal elements - should be evaluated, and the best one used for a given data set.

3 Upvote · Follow 2 · Reply to Tom Mosher

Earliest **Top** **Most Recent**

Karlis Vigulis · 10 days ago · Edited

Thank you for the hint Tom. It helped me to solve the problem with my unregularized nnCostFunction. I tried summing the diagonal after matrix multiplication using trace() and it gave the right result now. But I still did not understand why there is a difference from using nested *for loops* to do those sums. Which is pretty much like if you would calculate them by hand, on paper. In theory, they should give identical results. Is it due to floating point inaccuracy that there's some variance? Again, I used to get J value of 0.497854 with *for loops*, instead of the expected 0.287629.

0 Upvote · Hide 6 Replies

Tom Mosher Mentor · 10 days ago

That's way too big a difference to be a roundoff or precision error.

I suspect your for-loop code has some defects.

0 Upvote

Karlis Vigulis · 10 days ago · Edited

Not entirely sure, if it's not against the rules to copy Matlab code here, but as it is not correct code I hope it is fine.

Here's the code I used previously:

... Deleted

Really struggling to figure out why it is wrong..

0 Upvote

Tom Mosher Mentor · 10 days ago · Edited

Sorry, the course Honor Code says you can't share your code for the programming exercises. And I can't comment on your code, either.

But in a complex equation like the cost function, I would focus on whether the parenthesis are all in the right places.

I recommend you read the tutorial for this exercise, from the Resources menu. It gives the vectorized method, so you can lose the for-loops and all of the index values. That's where students commit most of the programming errors.

0 Upvote

Tom Mosher Mentor · 10 days ago · Edited

(edited)

0 Upvote

Karlis Vigulis · 10 days ago · Edited

Thanks a lot Tom.

I will delete the code in the above post then...

0 Upvote

Tom Mosher Mentor · 10 days ago

That's good news.

0 Upvote

Reply

Reply

1

Reply

Reply