This tutorial uses the vectorized method. If you're using a for-loop over the training examples, you're doing it the hard way, and you're on your own.

A note on Errata: The cost and gradient equations in the ex4.pdf file are correct. There may be some errata in the video lectures. Check the Course Wiki to be sure.

I'll use the less-than-helpful greek letters and math notation from the video lectures in this tutorial, though I'll start off with a glossary so we can agree on what they are. I will also suggest some common variable names, so students can more easily get help on the Forum.

It is left to the reader to convert these descriptions into program statements. You will need to determine the correct order and transpositions for each matrix multiplication, so that the result has the correct size.

**Glossary**:

Each of these variables will have a subscript, noting which NN layer it is associated with.

Θ: A Theta matrix of weights to compute the inner values of the neural network. When we used a vector theta, it was noted with the lower-case theta character θ.

z: is the result of multiplying a data vector with a Θ matrix. A typical variable name would be "z2".

a: The "activation" output from a neural layer. This is always generated using a sigmoid function g() on a z value. A typical variable name would be "a2".

δ: lower-case delta is used for the "error" term in each layer. A typical variable name would be "d2".

Δ: upper-case delta is used to hold the sum of the product of a δ value with the previous layer's a value. In the vectorized solution, these sums are calculated automatically though the magic of matrix algebra. A typical variable name would be "Delta2".

Θ\_gradient: This is the thing we're solving for, the partial derivative of theta. There is one of these variables associated with each Δ. These values are returned by nnCostFunction(), so the variable names must be "Theta1\_grad" and "Theta2\_grad".

g() is the sigmoid function.

g′() is the sigmoid gradient function.

Tip: One handy method for excluding a column of bias units is to use the notation SomeMatrix(:,2:end). This selects all of the rows of a matrix, and omits the entire first column.

See the Appendix at the bottom of the tutorial for information on the sizes of the data objects.

A note regarding bias units, regularization, and back-propagation:

There are two methods for handing exclusion of the bias units in the Theta matrices in the back-propagation and gradient calculations. I've described only one of them here, it's the one that I understood the best. Both methods work, choose the one that makes sense to you and avoids dimension errors. It matters not a whit whether the bias unit is excluded before or after it is calculated - both methods give the same results, though the order of operations and transpositions required may be different. Those with contrary opinions are welcome to write their own tutorial.

Forward Propagation:

We'll start by outlining the forward propagation process. Though this was already accomplished once during Exercise 3, you'll need to duplicate some of that work because computing the gradients requires some of the intermediate results from forward propagation. Also, the y values in ex4 are a matrix, instead of a vector. This changes the method for computing the cost J.

1 - Expand the 'y' output values into a matrix of single values (see ex4.pdf Page 5). This is most easily done using an eye() matrix of size num\_labels, with vectorized indexing by 'y'. A useful variable name would be "y\_matrix", as this...

y\_matrix = eye(num\_labels)(y,:)

Note: For MATLAB users, this expression must be split into two lines, such as...

eye\_matrix = eye(num\_labels)

y\_matrix = eye\_matrix(y,:)

Discussions of other methods are available in the Course Wiki - Programming Exercises section.

2 - Perform the forward propagation:

a1 equals the X input matrix with a column of 1's added (bias units) as the first column.

z2 equals the product of a1 and Θ1

a2 is the result of passing z2 through g()

Then add a column of bias units to a2 (as the first column).

NOTE: Be sure you DON'T add the bias units as a new row of Theta.

z3 equals the product of a2 and Θ2

a3 is the result of passing z3 through g()

Cost Function, non-regularized:

3 - Compute the unregularized cost according to ex4.pdf (top of Page 5), using a3, your y\_matrix, and m (the number of training examples). Note that the 'h' argument inside the log() function is exactly a3. Cost should be a scalar value. Since y\_matrix and a3 are both matrices, you need to compute the double-sum.

Remember to use element-wise multiplication with the log() function. Also, we're using the natural log, not log10().

Now you can run ex4.m to check the unregularized cost is correct, then you can submit this portion to the grader.

Cost Regularization:

4 - Compute the regularized component of the cost according to ex4.pdf Page 6, using Θ1 and Θ2 (excluding the Theta columns for the bias units), along with λ, and m. The easiest method to do this is to compute the regularization terms separately, then add them to the unregularized cost from Step 3.

You can run ex4.m to check the regularized cost, then you can submit this portion to the grader.

-----------------------------------

Appendix:

Here are the sizes for the Ex4 character recognition example, using the method described in this tutorial.

NOTE: The submit grader (and the gradient checking process) uses a different test case; these sizes are NOT for the submit grader or for gradient checking.

a1: 5000x401

z2: 5000x25

a2: 5000x26

a3: 5000x10

d3: 5000x10

d2: 5000x25

Theta1, Delta1 and Theta1\_grad: 25x401

Theta2, Delta2 and Theta2\_grad: 10x26

=========

Here is a link to a test case you can use for forward propagation.

<https://www.coursera.org/learn/machine-learning/discussions/a8Kce_WxEeS16yIACyoj1Q/replies/-B1dvvW0EeS16yIACyoj1Q>

Once you're at the test case, follow the link it that post to get another link, which gives the internal variable values.