**Checklist:**

Completed:

Pseudocode YES

Well-documented YES

Code works YES

Learning parameters: YES

inputs, targets, nodeLayers, learning rate,

epochs, batchSize

Termination conditions: max epochs, correct classification YES

Mini-batch SGD YES

Outputs as requested (iris, MNIST, and XOR) YES

Incomplete, Details \_\_\_\_none to my knowledge\_\_\_\_\_\_\_\_\_

Not sure, Details \_\_\_\_ none to my knowledge \_\_\_\_\_\_\_\_\_\_\_

**Pseudocode:**

1. Create an empty cell to hold the weights of the network based on the nodelayers input
2. For each weighted layer in the network:
   1. Randomly assign a the weights to the cell with a mean of 0 and a s.d. of 1
3. Create an empty cell to hold the biases of the network
4. For each non-input layer
   1. Randomly assign initial biases to the cell with a mean of 0 and a s.d. of 1
5. Begin backpropagation - Initiate an epoch counter to one
6. While the epoch counter is less than or equal to the total number of epochs from the input
   1. Initialize correct\_guesses and MSE variables to zero – these will be used to calculate accuracy
   2. Create an empty activation cell to hold the activations of the network based on the nodelayers input
   3. Create an empty errors cell to hold the activations of the network based on the nodelayers input
   4. Create an empty weighted input (z) cell to hold the activations of the network based on the nodelayers input
   5. Begin running minibatch - Initiate minibatch\_start to 1 and minibatch\_end to the batchSize input
   6. While minibatch\_end is less than or equal to the total number of input records
      1. Define the initial activation matrix from the inputs based on minibatch\_start and minibatch\_end
      2. Add these activations to the first entry in the activations cell
      3. Feedforward - For each additional activation layer in the network
         1. Expand the bias vector for the current layer into a matrix with a size consistent with the number of entries in the minibatch (this is necessary for our matrix math to work)
         2. Calculate the weighted input for the layer
         3. Add the weighted input to the weighted input cell for that layer
         4. Calculate the activations for the layer
         5. Add the activations to the activation cell for that layer
      4. Compute the error at the output layer
      5. Add the errors at the output layer to the appropriate location in the errors cell
      6. Calculate the number of correct values for the minibatch
      7. Calculate the minibatch’s contribution to the MSE
      8. Backpropogate the error – for each layer in the network starting with the layer preceding the output layer and stepping backward to the first layer after the input layer
         1. Calculate the error for the layer
         2. Add the error for the layer to the appropriate location in the errors cell
      9. Gradient descent – for each layer starting with the output layer and stepping back to the layer following the input layer
         1. Update the weights
         2. Update the biases
      10. Update the minibatch\_start and minibatch\_end variables
   7. Calculate the MSE for the epoch
   8. Output the results
   9. Check for termination criteria
   10. Increase epoch counter

**Instructions:**

1. Ensure that all three datasets along with “net.m” and “call\_function.m” are saved in the present working directory
2. Open the “call\_function.m” program
3. Choose the dataset you would like to evaluate by uncommenting the corresponding block of code
4. Set the desired network input parameters
5. Execute “call\_function.m” to pass the parameters to the “net.m” function

**Description:**

The first part of the function – rows 1 through 13 – is geared toward setting up the objects that will be necessary to keep track of what’s happening in our network. Specifically, we initiate empty cells for the weight values and the biases.

Once we start the backpropogation algorithm in row 19, it’s first about passing the inputs all the way through the network using feedword – starting at row 40. As we calculate the activations through the layers of the network, we’re also keeping track of the weights, biases, and weighted inputs for each layer. The reason we keep track of these things is that they’ll be needed to calculate the error during backpropogation.

Once the activations have been passed through the network, we’ll next need to step back collecting the errors for all the layers, again storing these values in the cell that had been created. It’s only when we have the errors and activations for each layer, we can calculate the gradient and use this information to update our weights and biases – rows 84 through 89.

Once we’ve cycled through all of the minibatches in the epoch, the last step is the calculate the MSE for the epoch and output the error – rows 101 through 104.

**Code:**

Note – double-click into the object below to scroll through the code

****

**Analysis:**

I’m currently under the impression that everything in my code is working. One area where I had difficulty and perhaps maybe still have an issue in my code is in the calculation of MSE for the epoch. Particularly, I played around with difference ways to turn “y(x) - a” into a scalar value so that it could be contributed to the MSE. I think I currently have it correct but I suppose I will soon find out for sure.

Another area where I had some question was around passing the nodelayers variable into the function. Specifically, I wanted to be sure that the number of neurons in the input and output layers were appropriate for the dataset. For example, what would happen if you tried to use a network with one neuron in the input layer for the MNIST dataset? What I ended up deciding to do is to still “pass” the number of neurons in the input and output layers into the function as a variable, but these two values are determined based in the inputs and targets prior to passing the whole vector into the function.

**Ideas for enhancements:**

The current network can be improved upon by adding some of the enhancements we’ve discussed in class. Specifically, here are a few ways the network can be improved:

* Using a different cost function to eliminate saturation
* Setting more efficient initial weight values
* Using regularization to avoid overfitting

**Outputs:**

Note – double-click into the objects to scroll through the results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data set** | **Epochs** | **Hidden nodes** | **Batch size** | **eta** |
| iris.csv | 100 | 20 | 10 | .1 |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data set** | **Epochs** | **Hidden nodes** | **Batch size** | **eta** |
| MNIST | 30 | 30 | 10 | 3.0 |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data set** | **Epochs** | **Hidden nodes** | **Batch size** | **eta** |
| xor.csv | 10 | 2 | 4 | .1 |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data set** | **Epochs** | **Hidden nodes** | **Batch size** | **eta** |
| xor.csv | 10 | 2 | 1 | .1 |



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data set** | **Epochs** | **Hidden nodes** | **Batch size** | **eta** |
| xor.csv | 20 | [3 2] | 1 | .1 |

