NOTE: post-implementation annotations are highlighted yellow

Final Specifications - Kim Soffen, Diane Yang, Lisa Wang, Amna Hashmi

Signatures/Interfaces

For the pre-processing segment of the project, we have created a ProcessedImage class, which is structured as follows:

ProcessedImage: takes as an input the image that will be converted into text.

- Attributes:
  - image: contains the image itself converted into pure black and white
  - pixels: contains a matrix of the pixels
  - width: width of the image in pixels
  - height: height of the image in pixels
- Methods:
  - get_lines: divides the text into separate lines and returns an array of the lines.
  - get_chars: divides each line into separate characters and returns a flat array of the characters.
  - resize: adjusts the dimensions of each character to be 30 x 30 pixels

We changed this interface a bit by adding an attribute called "space" which contains a blank white square image representing a space. We insert this every time we find enough consecutive columns of white pixels to detect a space.

The client must respect two invariants for this pre-processing component. First, the input image can contain only uppercase letters and must be in a PNG format.

All values and functions are, as of right now, exposed.

For the neural network portion:

Classes:

- Node:
  - Inputs (transformed value of nodes that input to node)
  - weights (how much each input affects node)
  - fixed weight (initial weights)
  - forward neighbors (nodes that the current node sends its transformed value to)
  - forward weights (how much node's output affects each of its forward neighbors)
  - raw value (value from combining inputs with corresponding weights)
  - transformed (value from putting raw value through sigmoid function)
  - used for "output nodes" which each correspond to one character and gives the probability that the input image is that node
  - also used for "hidden nodes"; we will begin by making 30 of these and will adjust

as necessary. These add another layer of weights so we can identify the proper character since the data is not linearly separable.

- Weight:
    - includes value attribute
- Input:
    - list of input values, formatted as an array; each value considered an ‘input node’
- Target:
    - list of expected return values from each output node, formatted as an array
- Neural Network:
    - defines layer of input nodes, hidden nodes, and output nodes (functions of each of these types of nodes described above) along with the initial weights
- Network Framework:
    - sets up methods for using the network:
        - encoding the labels for each character
        - getting the associated label to an input after feeding through the network
        - converting an image into an input instance
        - initializing random weights for the inputs
        - a performance test to find the number correctly labeled
        - a training method for the network

These interfaced did not change

As of right now, all neural network values and functions are exposed (none hidden). Invariants: Inputs must be a 30x30 pixel image We changed this to 20x20 (because we will set up everything to have 900 input nodes), be pure black and white (so pixels will be given a value of 0 or 1) This was changed to reflect monochromatic color: ie 0 = black, 255 = white and represent a capital letter (since that is the data set we will be training over) We expanded to include lowercase letters and periods. The letter must be centered and of a standard size within the 30x30 box (we haven’t yet decided what the best size is).

Neural network training pseudo code:

```
    Forward phase:
    Repeat over all nodes
        For node where all parents processed
            raw value = sum over parents of weight * transformed value
            transformed value = sigmoid (raw value)

    Backward phase:
    Repeat over all nodes
        For node where all children processed
            If node is output node, epsilon = expected value - actual value
            Else, epsilon = sum over all children: weight * delta value
            delta = sigmoid prime (raw value) * epsilon
```

Weight from parent to child = old weight + learning rate * parent transformed value * parent delta

Modules/Actual Code will be included in a separate file sent to you

Timeline

1 week from now:

- Finish neural network class functions
- Finish network framework class functions
- Set up small networks to test functions
- Be able to separate characters out within a line

2 weeks from now:

- Begin training neural network over data set
- Work on adjusting learning rate and number of hidden nodes to train neural network
- Have program be able to properly format characters (ie size) to be fed into neural network
- Add in other characters if possible (i.e. punctuation)

3 weeks from now:

- completely finish training and tweaking network
- put the two halves (neural network and character given document) together into one functional program
- code to output a .txt file
- make the program work with command line

Progress Report:

We are using the CS181 Homework 2 as the framework for our neural network. We are currently completing this Homework as it was designed for CS181, and then adapting it to work for our purposes

Data Set: http://archive.ics.uci.edu/ml/datasets/Letter+Recognition and http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/

We ended up making our own data set

Version control: git repository established within GitHub and functional