Firstly, an equation of the line for a perceptron to separate the data in the provided graph, of the form: where w denotes one of the node’s weight values, x denotes one of the node’s input values, θ denotes the node’s threshold value, the subscript W denotes variables for the weight dimension, and the subscript L denotes variables for the length dimension, is:  
 . Upon rearrangement, one retrieves a more familiar equation of a line:   
. When xL = 0, xW = 10. When xW = 0, xL = 40. A line intersecting the axes at these point classifies setosa from NON-setosa data, by inspection. I am interpreting this question   
(Assignment 1 – Your Task, item 1) as meaning that I am simply to provide the equation here, not implement it in my program, as I implement one capable of learning.

Before any learning is done in my program, the input values are all normalized to the range of [0 to 1] by dividing them all by the largest value in that dimension. This preprocessing precludes the possibility of any dimension being more important for weight adjustment than any other, or worse, some input values increasing weights and others decreasing weights for the same error correction (occurs when one input dimension is greater than 1, and another is positive but less than 1).

The perceptron which learns to classify setosa from NON-setosa has one node with randomly initialized weights in the range [0 to 1], and a threshold of 0. Simple feedback learning is used to increase the weight value when the threshold ought to be surpassed but isn’t by (the input datum value times the learning rate, for each input dimension). These values are subtracted from the weights when the threshold is surpassed but shouldn’t be. With these weight values learned, classification of versicolor and virginica is possible.

The setosa – NON-setosa weight values are needed to separate versicolor from virginica so that we know when we have a setosa input or not, as it does no good to adjust versicolor – virginica weights using setosa data. The perceptron network used to classify versicolor and virginica data consists of four nodes, one to interpret each input dimension with an associated weight, plus one final node to interpret the activations of the four previous nodes.

Batch error correction was used to train the versicolor – virginica network. After each training epoch the average error value was calculated for each dimension, for both versicolor and virginica classification, where classification errors were made. The error for a single misclassification was calculated as the difference between the activation and the threshold. If versicolor classification experienced more error than virginica, for example, then the weights (initialized to random values in range [0 to 1]) were adjusted so as to reduce versicolor misclassification that epoch. The weights were adjusted by ((the class’ classification error – the other class’ classification error) times the learning rate). My reasoning for subtracting the other class’ error was so as to not make large changes if both errors were large, resulting in problems for the other class. When a node in the first layer’s activation surpassed its threshold of 1, it set its output to the final node to 1, and -1 when it didn’t surpass the threshold. It was then a simple matter of summing the inputs to the final node and comparing them to its threshold of 0 to make the final classification. For simplicity, the thresholds were not adjusted as adjusting the weights was entirely sufficient for accurate classification.

Both the setosa classification training and versicolor – virginica classification training were allowed up to 100 training epochs. The setosa training typically had 100% classification within 5-10 epochs, and the versicolor – virginica weight values converged within a similar number of epochs. As such, the performance of the algorithm was insensitive to the number of training epochs allowed as long as it was greater than this. Both the setosa classification and final versicolor – virginica classification thresholds were set to 0 since their weights were adjusted in a symmetric (about 0), bipolar fashion. The thresholds for the four nodes for each of the versicolor – virginica classification dimensions were set to 1, although performance seemed independent of this value, as long as it was above 0. Setting the learning rate was a balancing act. As the learning rate got too close to 1, the weights for versicolor – virginica classification became unstable; they would grow to (-/+) infinity. If it was too close to 0, the frequency with which   
NON-setosas were classified as setosas increased. This type of misclassification occurs on some runs of the program, albeit rarely with the proper choice of learning rate.

The equations for the nodes used in my program are in the ‘output.txt’ file, with classification rates, weight vector values, etc… as they may vary with each execution of the program. Occasionally the program misclassifies NON-setosa for setosa or vice-versa, but re-running the program usually results in 100% setosa – NON-setosa classification for the test data. The program can be run either by compiling and running the source file at /Asg1\_10006197/Asg1\_10006197/main.cpp, or by simply running the executable at /Asg1\_10006197/Debug/Asg1\_10006197.exe, both have the training and test data in the appropriate relative directories to reproduce ‘output.txt’. I have left one ‘output.txt’ file with the source code, but not with the executable, it is generated by running it. The source file should only require libraries which are standard with Visual Studio Community 2013.