Task 1 (70 points, programming)

In this task you will implement neural networks, using the backpropagation algorithm described in the slides on neural networks. Regarding backpropagation, you should follow exactly the formulas and specifications on those slides.

Arguments

You must implement a Matlab function or a Python executable file called `neural_network`, that uses backpropagation to train a neural network, and then applies that neural network to classify test data. Your function should be invoked as follows:

```
neural_network(<training_file>, <test_file>, <layers>, <units_per_layer>, <rounds>)
```

If you use Python, just convert the Matlab function arguments shown above to command-line arguments. The arguments provide to the function the following information:

- The first argument, <training_file>, is the path name of the training file, where the training data is stored. The path name can specify any file stored on the local computer.
- The second argument, <test_file>, is the path name of the test file, where the test data is stored. The path name can specify any file stored on the local computer.
- The third argument, <layers>, specifies how many layers to use. Note that the input layer is layer 1, so the number of layers cannot be smaller than 2 (any neural network should have at least an input layer and an output layer).
- The fourth argument,<units_per_layer>, specifies how many perceptrons to place at each HIDDEN layer. This number excludes the bias input. So, each hidden layer should contain <units_per_layer> perceptrons and one bias input unit. Note that this number is not applicable to units in the input layer, since those units are not perceptrons but simply provide the values of the input object and the bias input. Also, note that the number of perceptrons in the output layer is equal to the number of classes, and thus this number is independent of the <units_per_layer> argument.
- The fifth argument, <rounds>, is the number of training rounds that you should use. Each training round consists of using the whole training set once (i.e., using each training example once to update the weights).

The training and test files will follow the same format as the text files in the UCI datasets directory. A description of the datasets and the file format can be found on this link. For each dataset, a training file and a test file are provided. The name of each

file indicates what dataset the file belongs to, and whether the file contains training or test data. Your code should also work with ANY OTHER training and test files using the same format as the files in the UCI datasets directory.

As the description states, **do NOT use data from the last column (i.e., the class labels) as features**. In these files, all columns except for the last one contain example inputs. The last column contains the class label.

Training

In your implementation, you should use these guidelines:

- Every perceptron should have a bias input, that is always set to 1. The weight of that bias input is changed during backpropagation, and is treated exactly the same way as any other weight.
- For each dataset, for all training and test objects in that dataset, you should normalize all attribute values, by dividing them with the MAXIMUM value over all attributes over all training objects for that dataset. This is a single MAXIMUM value, you should NOT use a different maximum value for each dimension. In other words, for each dataset, you need to find the single highest value across all dimensions and all training objects. Every value in every dimension of every training and test object should be divided by that highest value.
- The weights of each unit in the network should be initialized to random values, chosen between -.05 and 0.05.
- You should initialize your learning rate to 1 for the first training round, and then multiply it by 0.98 for each subsequent training round. So, the learning_rate used for training round r should be 0.98^{r-1}.
- Your stopping criterion should simply be the number of training rounds, which is specified as the fifth argument. The number of training rounds specifies how many times you iterate over the entire training set. A single training round consists of a single execution of steps 4, 5, 6 in the backpropagation summary, on slide 71 of the neural networks slides. If you want your code to correspond step-by-step to that summary, then in step 6 of that summary you should use a threshold equal to -1, so that the error is never used as the stopping criterion.

The number of layers in the neural network is specified by the <layers> argument. If we have L layers:

- Layer 1 is the input layer, that contains no perceptrons, it just specifies the inputs to the neural network. Thus, 2 is the minimum legal value for L.

- Each perceptron at layer 2 has D+1 inputs, where D is the number of attributes. The attributes of the input object provide the D non-bias inputs to each perceptron at this layer.
- Layer L is the output layer, containing as many perceptrons as the number of classes.
- If L > 2, then layers 2, ..., L-1 are the hidden layers. Each of these layers has as many perceptrons as specified in <units_per_layer>, the third argument. The bias input is a unit, but not a perceptron, so it is NOT counted in those <units_per_layer> perceptrons. If L = 2, then the fourth argument (<units_per_layer>) is ignored.
- If L > 2, each perceptron at layers 3, ..., L has as inputs the outputs of ALL perceptrons at the previous layer, in addition to the bias input.
- Note that each dataset contains more than two classes, so your output layer needs to contain a number of units (perceptrons) equal to the number of classes, as discussed in the slides.

There is no need to output anything for the training phase.

Classification

For each test object you should print a line containing the following info:

- Object ID. This is the line number where that object occurs in the test file. Start with 1 in numbering the objects, not with 0.
- Predicted class (the result of the classification). If your classification result is a tie among two or more classes, choose one of them randomly.
- True class (from the last column of the test file).
- Accuracy. This is defined as follows:
    - If there were no ties in your classification result, and the predicted class is correct, the accuracy is 1.
    - If there were no ties in your classification result, and the predicted class is incorrect, the accuracy is 0.
    - If there were ties in your classification result, and the correct class was one of the classes that tied for best, the accuracy is 1 divided by the number of classes that tied for best.
    - If there were ties in your classification result, and the correct class was NOT one of the classes that tied for best, the accuracy is 0.

To produce this output in a uniform manner, use:

```
fprintf('ID=%5d, predicted=%3d, true=%3d, accuracy=%4.2f\n',
        object_id, predicted_class, true_class, accuracy);
```

After you have printed the results for all test objects, you should print the overall classification accuracy, which is defined as the average of the classification accuracies you printed out for each test object. To print the classification accuracy in a uniform manner, use:

```
fprintf('classification accuracy=%6.4f\n', classification_accuracy);
```

In your answers.pdf document, please provide ONLY THE LAST LINE (the line printing the classification accuracy) of the output by the test stage, for the following invocations of your program:

- Training and testing on pendigits dataset, with 2 layers, 50 training rounds.
- Training and testing on pendigits dataset, with 3 layers, 20 units per hidden layer, 20 training rounds.

Task 2 (5 points).

Note: In this question you should assume that the activation function of a perceptron is the step function. More specifically, this function:

- outputs 0 if the weighted sum of inputs is LESS THAN 0 (not less than or equal).
- outputs 1 if the weighted sum of inputs is greater than or equal to 0.

Design a perceptron that takes (in addition to the bias input) three Boolean inputs (i.e., inputs that are equal to 0 for false, and 1 for true), and outputs: 1 if at least two of the three inputs are true, 0 otherwise.

Task 3 (5 points).

Note: In this question you should assume that the activation function of a perceptron is the same as for Task 2.

By (X => Y) we denote the boolean statement "If X then Y". Design a perceptron (i.e, an individual neuron) that takes in two Boolean inputs X and Y and outputs the Boolean value of (X => Y). As a reminder, 0 stands for "false" and 1 stands for "true". You should NOT worry about what your perceptron does when the input values are not 0 or 1

Task 4 (10 points).

Note: In this question you should assume that the activation function of a perceptron is the same as for Task 2.

Design a neural network that:

- takes two inputs, A and B.
- outputs 1 if $2A + 3B = 4$.
- outputs 0 otherwise.

Your solution should include a drawing of the network, that shows the weights of each input of each perceptron, and that also shows which perceptron outputs are linked to which perceptron inputs.

Task 5 (10 points).

Note: In this question you should assume that the activation function of a perceptron is the same as for Task 2.

Is it possible to design a neural network (which could be a single perceptron or a larger network), that satisfies these specs?

- Takes a single input called X, which can be any real number.
- If $X < 3$, the network outputs 0.
- If $3 < X < 7$, the network outputs 1.
- If $X > 7$, the network outputs 0.

We don't care what the network outputs when $X = 3$ or $X = 7$.

If your answer is no, explain why not. If your answer is yes, your solution should include a drawing of the network, that shows the weights of each input of each perceptron, and that also shows which perceptron outputs are linked to which perceptron inputs.