CSE 4309 - Assignments - Assignment 4

List of assignment due dates.

The assignment should be submitted via Canvas. Submit a file called assignment4.zip, containing the following two files:

- answers.pdf, for your answers to the written tasks, and for the output that the programming task asks you to include. Only PDF files will be accepted. All text should be typed, and if any figures are present they should be computer-generated. Scans of handwriten answers will NOT be accepted.
- neural_network.m, or neural_network.py, containing your Matlab or Python code for the programming part. Your Matlab or Python code should run on the versions specified in the syllabus, unless permission is obtained via e-mail from the instructor or the teaching assistant. Also submit any other files that are needed in order to document or run your code (for example, additional source code files).

These naming conventions are mandatory, non-adherence to these specifications can incur a penalty of up to 20 points.

Your name and UTA ID number should appear on the top line of both documents.

Task 1 (60 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. Note that this number is not applicable either to the input layer or to the output layer.
- 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:

- For each dataset, for all training and test objects in that dataset, you should normalize all attribute values, by dividing them with the MAXIMUM ABSOLUTE 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 absolute 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, between -0.05 and 0.05. The probability of each random value should follow a uniform distribution between -0.05 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 3, 4, 5, 6 in the backpropagation summary, on slide 45 of the neural networks slides. If you want your code to correspond step-by-step to that summary, then in step 5 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 inputs, where D is the number of attributes. The attributes of the input object provide these D input values to each perceptron at layer 2.
- 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. 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.
- 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=%10s, true=%10s, 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, 10 training rounds.
- Training and testing on pendigits dataset, with 3 layers, 20 units per hidden layer, 20 training rounds.

Expected Classification Accuracy

You may get different classification accuracies when you run your code multiple times with the same input arguments. This is due to the fact that weights are initialized randomly. These are some results I got on the pendigits dataset with my implementation:

- I ran my code 10 times with 2 layers and 10 training rounds. Each time the classification accuracy was 0.8659. Each run took about 3 seconds on my computer.
- I ran my code 10 times with 3 layers, 20 units for the hidden layer, and 20 training rounds. The classification accuracy ranged between 0.9088 and 0.9297. Each run took about 7 seconds on my computer.

It is possible that sometimes you get results outside those ranges, but most of the time that you run your code with the parameters specified above you should be getting accuracies within those ranges.

Grading

- 25 points: correct implementation for 2-layer neural networks (no hidden layers).
- 20 points: correct implementation for more than two layers neural networks (networks with hidden layers).
- 15 points: handling non-numerical string labels. The UCI datasets include modified "string label" versions of the files, where class labels are non-numerical strings. For the pendigits dataset, the "string label" versions are stored at pendigits_string_training.txt and pendigits_string_test.txt.

Task 1b (Extra Credit, maximum 10 points).

A maximum of 10 extra credit points will be given to the submission or submissions that identify the command line arguments achieving the best test accuracy for any of the three test datasets. These command line arguments, and the attained accuracy, should be reported in answers.pdf, under a clear "Task 1b" heading. These

results should be achievable by calling the executable you submit for Task 1. You should achieve the reported accuracy at least five out of 10 times when you run your code.

Task 1c (Extra Credit, maximum 10 points).

In this task, you are free to change any implementation options that you are not free to change in Task 1. Examples of such options include choice of activation function, initial distribution of weights, learning rate, or any other implementation choices. You can submit a Matlab function or Python executable called neural_network_opt, that implements your modifications. A maximum of 10 points will be given to the submission or submissions that, according to the instructor and GTA, achieve the best improvements (on any of the three datasets) compared to the specifications in Task 1. In your answers.pdf document, under a clear "Task 1c" heading, explain:

- What modifications you made.
- What results you achieved. You should achieve the reported accuracy at least five out of 10 times when you run your code.
- What command line arguments to provide your program with in order to obtain those results.
- How long it took your program to run with those arguments.

Task 2 (10 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 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. You should NOT worry about what your perceptron does when the input values are not 0 or 1.

Task 3 (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 all edges connecting outputs of one layer to inputs in the next layer, and that also shows the values of all weights (including bias weights) of all perceptrons.

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.

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 all edges connecting outputs of one layer to inputs in the next layer, and that also shows the values of all weights (including bias weights) of all perceptrons.

Task 5 (10 points).

The programming part of this assignment requires that you initialize weights to small random values. What would go wrong if you initialized all weights to zero? How would classification accuracy be affected? Justify your answer.

CSE 4309 - Assignments - Assignment 4