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.
- nn_keras.py, containing your Python code for the programming part. Your Python code should run either on Google Colab or on the Python 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 train and test neural networks on UCI datasets using Keras.

Function Name and Arguments

File nn_keras_main.py contains incomplete code that aims to learn a neural network model given some training data, and then evaluate this model on test data.

To complete that code, you must create a file called `nn_keras.py`, where you implement a Python function called `nn_keras`. Your function should be invoked as follows:

```
nn_keras(directory, dataset, layers, units_per_layer, epochs)
```

- The first argument, `directory` is a string that specifies the folder where the dataset is stored. The directory argument can specify any folder stored on the local computer, or (if you are using Google Colab) accessible on your Colab environment.
- The second argument, `dataset` is a string that specifies the name of the dataset. For example, if the directory value is "../uci_data" and the dataset value is "pendigits_string", then your training data should be loaded from file "uci_data/pendigits_string_training.txt" and your test data should be loaded from file "../uci_data/pendigits_string_test.txt".

- 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, `epochs`, is the number of training rounds ("epochs" in Keras terminology) that you should use.

The training and test files will follow the same format and naming conventions 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, which may be a string.

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.
- Let Keras use its default method for initialization of all weights (i.e., your code should not address this issue at all).
- For the optimizer, use "adam" with default settings.
- All hidden layers should be fully connected ("dense", in Keras terminology). All hidden layers should use the sigmoid activation function.
- The output layer should also be fully connected and use the sigmoid activation function.
- The loss function should be `tf.keras.losses.SparseCategoricalCrossentropy()`.
- For any Keras option that is not explicitly discussed here (for example, the batch size), you should let Keras use its default values.

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 (i.e., D is the dimensionality of each input vector). 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. It is OK if Keras prints out various things per epoch.

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 Python in a uniform manner, use:

```
print('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:

```
print('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 4 layers, 40 units per hidden layer, 20 training rounds, sigmoid activation for the hidden layers.

Expected Classification Accuracy

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

- I ran my solution with these arguments 15 times:

  ```
  nn_keras("../uci_data", "pendigits_string", 4, 50, 20)
  ```

  The classification accuracy on the test set was between 87.34% and 89.14%.

- I ran my solution with these arguments 15 times:

  ```
  nn_keras("../uci_data", "yeast_string", 4, 50, 20)
  ```

  The classification accuracy on the test set was between 27.69% and 38.64%.

- I ran my solution with these arguments 15 times:

  ```
  nn_keras("../uci_data", "satellite_string", 4, 50, 20)
  ```

  The classification accuracy on the test set was between 80.65% and 81.75%.

Grading

- 25 points: correct implementation for 2-layer neural networks (no hidden layers).
- 25 points: correct implementation for more than two layers neural networks (networks with hidden layers).
- 10 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 function arguments (hyperparameters) achieving the best test accuracy for any of the three test datasets. These function 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 function 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 changing the activation function, loss function, type of layers that you use, etc. You can submit a Python file called nn_keras_opt, that implements your improved version of the `nn_keras` function. 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, and what arguments we should call your improved `nn_keras` function with in order to verify your results.

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).

Slide 16 of slides on perceptron training provides pseudocode for training a perceptron. Step 1 of that pseudocode is to initialize weights to small random values. What would happen if you initialized all weights to zero? How would classification accuracy be affected, compared to initializing weights randomly (would you expect it to be higher, lower, or about the same)? Justify your answer.
