CSE 4392 - 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 (if any) that each 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.
- All source code files needed to run your solutions for the programming tasks. Your 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 (30 points, programming)

In this task you will train and test neural networks on UCI datasets using Keras.

Arguments

You must implement a Python executable file called `nn_keras.py`, that should be invoked as follows:

```
nn_keras <training_file> <test_file> <layers> <units_per_layer> <rounds> <hidden_activation>
```

- 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 ("epochs" in Keras terminology) that you should use.
- The sixth argument, <hidden_activation>, is a string that can be "sigmoid", "tanh", or "relu". Your network should use the specified activation function in all hidden layers. If your network has no hidden layers (i.e., if the number of layers is 2) then this argument gets ignored.

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, 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).
- The output layer should also be fully connected, and it should use the softmax activation function.
- The loss function should be Categorical Cross Entropy (it is up to you if you use the sparse version or not, but make sure that the class labels that you pass to the fit() method are appropriate for your choice of loss function).
- 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, tanh activation for the hidden layers.

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 with my implementation:

- I ran my solution with this command line 15 times:

  ```
  nn_keras pendigits_string_training.txt pendigits_string_test.txt 4 50 20 tanh
  ```

The classification accuracy on the test set was between 95.74% and 96.66%.

- I ran my solution with this command line 15 times:

  ```
  nn_keras yeast_string_training.txt yeast_string_test.txt 4 50 20 tanh
  ```

  The classification accuracy on the test set was between 54.13% and 58.68%.

- I ran my solution with this command line 15 times:

  ```
  nn_keras satellite_string_training.txt satellite_string_test.txt 4 50 20 tanh
  ```

  The classification accuracy on the test set was between 81.60% and 84.55%.

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 different optimizers and settings for those optimizers, different types of layers, different batch sizes, etc. You can submit a Python executable called nn_keras_opt.py, 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 (30 points, programming)

This task is identical to the previous task, except that the model will be trained and tested on the MNIST dataset.

Arguments

You must implement a Python executable file called `nn_dense_mnist.py`, that should be invoked as follows:

```
nn_dense_mnist <layers> <units_per_layer> <rounds> <hidden_activation>
```

- Unlike the previous task, here we do not specify any dataset on the command line. Instead, your code should use the MNIST dataset.
- Argument <layers> should behave exactly the same way as in the previous task, with the additional specification that "Flatten" layers do not count. You can implement this task with or without using "Flatten" layers. Either way, the model should have [<layers> - 2] fully connected hidden layers.
- Arguments <units_per_layer>, <rounds>, <hidden_activation> should behave exactly the same way as in the previous task.

Training

In your implementation, you should use the same guidelines as in the previous task. In addition:

- Image values should be normalized so that they are between 0 and 1 (actually, this is mathematically equivalent to normalizing values following the previous task's instructions).

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 same information, and in the same format, as in the previous task. As in the previous task, 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:

- 2 layers, 10 epochs.
- 4 layers, 40 units per hidden layer, 20 epochs, tanh activation for the hidden layers.

Task 2b (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. These command line arguments, and the attained accuracy, should be reported in answers.pdf, under a clear "Task 2b" heading. These results should be achievable by calling the executable you submit for Task 2. You should achieve the reported accuracy at least five out of 10 times when you run your code.

Task 2c (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 2. Examples of such options include different optimizers and settings for those optimizers, different types of layers, different batch sizes, etc. You can submit a Python executable called nn_dense_mnist_opt.py, 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 compared to the specifications in Task 2. In your answers.pdf document, under a clear "Task 2c" 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 3 (40 points, programming)

This task is similar to the previous task, except here you will use a convolutional neural network.

Arguments

You must implement a Python executable file called `cnn_mnist.py`, that should be invoked as follows:

```
cnn_mnist <blocks> <filter_size> <filter_number>  <region_size>  <rounds> <cnn_activation>
```

- Argument <blocks> specifies how many convolutional layers your model will have. Every convolutional layer must be followed by a max pool layer. The total number of layers should be 2*blocks + 2, to account for the input layer, and the output layer. The output layer should be fully connected and use the softmax activation function. Except for the output layer, there should be no other fully connected layers.
- Argument <filter_size> specifies the number of rows of each 2D convolutional filter. The number of columns should be equal to the number of rows, so it is also specified by <filter_size>. For example, if <filter_size> = 3, each 2D filter has 3 rows and 3 columns.
- Argument <filter_number> specifies the number of 2D convolutional filters used at each convolutional layer. For example, if <filter_number> = 5, each convolutional layer applies 5 2D filters.
- Argument <region_size> specifies the size of the region for the max pool layer. For example, if <region_size> = 2, each output of a max pool layer should be the max value of a 2x2 region (i.e., 2 rows and 2 columns).
- Argument <rounds> specifies the number of training epochs, as in the previous tasks.
- Argument <cnn_activation> specifies the activation that should be used in convolutional layers. As was the case with the <hidden_activation> argument in previous tasks, this argument is a string that can be "sigmoid", "tanh", or "relu"

Training

In your implementation, you should use the same guidelines as in the previous task. In addition:

- Image values should be normalized so that they are between 0 and 1 (actually, this is mathematically

equivalent to normalizing values following the previous task's instructions).

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 same information, and in the same format, as in the previous task. As in the previous task, 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:

- 1 block, 10 epochs, filter_size = 3, filter_number = 5, region_size = 2, relu activation
- 2 blocks, 20 epochs, filter_size = 3, filter_number = 5, region_size = 2, relu activation

---

Task 3b (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. These command line arguments, and the attained accuracy, should be reported in answers.pdf, under a clear "Task 3b" heading. These results should be achievable by calling the executable you submit for Task 3. You should achieve the reported accuracy at least five out of 10 times when you run your code.

Task 3c (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 3. Examples of such options include different optimizers and settings for those optimizers, different types of layers, different batch sizes, etc. You can submit a Python executable called cnn_mnist_opt.py, 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 compared to the specifications in Task 3. In your answers.pdf document, under a clear "Task 3c" 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.

CSE 4392 - Assignments - Assignment 4