CSE 4392 - Assignments - Assignment 6

List of assignment due dates.

The assignment should be submitted via Canvas. Submit a file called assignment6.zip, containing all source code files needed to run your solutions for the programming tasks. Your Python code should run on Google Colab, unless permission is obtained via e-mail from the instructor or the teaching assistant.

All specified naming conventions for files and function names 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 answers.pdf and a ll source code files.

Task 1 (100 points, programming)

File month_main.py contains incomplete code that trains and evaluates two neural network models (a fully-connected model and an RNN model) that predict the month of the year that a specific moment belongs to, based on a time series of weather observations from the week before and the week after that moment. The Jena Climate dataset is used for training and testing the models. You can download the Jena Climate dataset as a CSV file from here: jena_climate_2009_2016.csv.

To complete that code, you must create a file called month_solution.py, where you implement the following Python functions:

- `normalized_data = data_normalization(raw_data, train_start, train_end)`
- `(inputs, targets) = make_inputs_and_targets(data, months, size, sampling)`
- `history = build_and_train_dense(train_inputs, train_targets, val_inputs, val_targets, filename)`
- `history = build_and_train_rnn(train_inputs, train_targets, val_inputs, val_targets, filename)`
- `test_acc = test_model(filename, test_inputs, test_targets)`
- `conf_matrix = confusion_matrix(filename, test_inputs, test_targets)`

Here is a description of what each function should do:

- The `data_normalization` function normalizes the data so that the mean for each feature is 0 and the standard deviation for each feature is 1. The function takes as inputs:
  - `data`: the time series of weather observations that the code reads from the jena_climate_2009_2016.csv file.
  - `train_start, train_end`: these specify, respectively, the start and end of the segment of `raw_data` that should be used for training.

  The `data_normalization` function returns a time series `normalized_data`, that is obtained by normalizing `raw_data` so that, for each feature, the mean value over the training segment is 0, and the standard deviation over the training segment is 1.

- The `make_inputs_and_targets` function creates a a set of input objects and target values, that can be used as training, validation or test set. The function takes as inputs:
  - `data`: a time series which in our code is a segment (training, validation, or test segment) of the `normalized_data` time series.
  - `months`: this is a time series of target values for `data`, so that `months[i]` is the correct month for the moment in time in which `data[i]` was recorded. Numbers are assigned to months following the usual convention (1 for January, 2 for February, etc.), except that 0 (and not 12) is assigned to December. This makes it easier later to train a model, as we will have class labels that start from 0.

  - `size`: this specifies the size of the resulting set of inputs and targets. For example, if `size == 10`, then the function extracts and returns 10,000 input vectors and 10,000 target values.
  - `sampling`: this specifies how to sample the values in `data`. For example, if `sampling == 6`, then we sample one out of every six time steps from the `data` time series. This reduces the length of each input vector by a factor equal to `sampling`.

  The `make_inputs_and_targets` function returns two values:
  - `inputs`: This is a three-dimensionaly numpy array. The first dimension is equal to the argument `size`. `inputs[i]` is a 2D matrix containing two weeks of consecutive weather observations extracted from `data`, using the appropriate sampling rate. To determine the number of observations (number of time steps) that inputs[i] should have to cover two weeks of data, consider that the Jena Climate dataset records one observation every ten minutes, and consider the sampling rate (with a sampling rate of 6, inputs[i] should only contain one observation per hour, which gives a total of 336 observations). As a reminder, each observation is a 14-dimensional vector.
  - `targets`: These are the target values for the inputs. `targets[i]` should be the month corresponding to the moment at which the mid-point of inputs[i] was recorded. For example, if the sampling rate is 6, inputs[i] contains 336 observations, and targets[i] should be the month corresponding to the moment when inputs[i][168] was recorded.

- The `build_and_train_dense` function trains a fully-connected model using the given training inputs and training labels. You should train for 10 epochs, use the "Adam" optimizer, and use a default batch size. Note that one of its arguments is a filename. Your function should save in that filename the best model, according to classification accuracy on the validation set. The function returns the training history, which your function can obtain from its call to the model.fit() function.

  The first layers of the fully-connected model that you create should be as specified below:

```
model = keras.Sequential([keras.Input(shape=input_shape),
                          keras.layers.Flatten(),
                          keras.layers.Dense(64, activation="tanh"),
                          # from here on you decide what to do, there are multiple correct options
                         ])
```

- The `build_and_train_rnn` function trains an RNN model using the given training inputs and training labels. For our purposes, a model is an RNN model if it contains at least one recurrent layer. You should train for 10 epochs, use the "Adam" optimizer, and use a default batch size. As before, your function should save in the given filename the best model, according to classification accuracy on the validation set. The function returns the training history, which your function can obtain from its call to the model.fit() function.

  The first layers of the RNN model that you create should be as specified below:

```
model = keras.Sequential([keras.Input(shape=input_shape),
                          keras.layers.Bidirectional(keras.layers.LSTM(32)),
                          # from here on you decide what to do, there are multiple correct options
                         ])
```

- The `test_model` function computes the classification accuracy of the model saved on the given `filename`, using the given `test inputs` and `test_targets`. Your function can call the Keras `model.evaluate` to do the main work. The return value is the test accuracy, represented as a number between 0 and 1.

- The `confusion_matrix` function computes the confusion matrix of the model saved on the given `filename`, using the given `test inputs` and `test_targets`. The confusion matrix is a 12x12 matrix, where at row i and column j (with row and column indices starting at 0, Python-style) it stores the number of test objects that had true class label i and were classified by the model as having class label j. You can see how these functions are used in month_main.py, to verify that you understand what arguments they take and what they are supposed to do. When grading, we reserve the right to test your functions with different code (instead of month_main.py), so your functions should not assume the existence of any global variables defined in month_main.py.

Some Results

Here is some information to help you test your code.

- A complete output of month_main.py using my solution is saved at file output10.txt.
- I ran month_main.py using my solution 10 times. I wrote a small matlab script, script1.m, to summarize my results. This is the summary of the results:

```
dense val acc:  min = 36.5%, max = 41.3%, mean = 38.8%, median = 38.5%
  rnn val acc:  min = 38.6%, max = 52.3%, mean = 46.2%, median = 46.1%
dense test acc: min = 40.4%, max = 46.7%, mean = 44.0%, median = 44.5%
  rnn test acc: min = 39.9%, max = 53.4%, mean = 47.6%, median = 48.8%
```

CSE 4392 - Assignments - Assignment 6