There are three ways of using the source code: using an already trained MLP to output new predictions on novel input data (run code as is and enter “test”); creating a new MLP and training it with customizable MLP hyper-parameters and input/output values (run code as is and enter “train”), and editing the source code for a more customizable approach (main function has the process used in our paper commented out). We used C++17.

**Using the Code**

The testing code reads trained MLP’s from .csv files and input data from either .csv files or .txt files. The .csv files are used to input multiple spectra in one file, and the formatting is detailed below. The .txt files are an alternative that can be used to read in a single spectrum, where the first line is a header and subsequent lines are the wavelength followed by a ‘ ‘ space character and the intensity at that wavelength. An example .csv file is provided with the code, and needed to read in .txt files as the example file specifies the wavelength range and resolution in it. The .txt file is read in, and interpolated to the wavelength range and resolution in the example file.

The user need not concern themselves with the formatting of the MLP .csv files as they are auto-generated by the program, or given as presented in our paper. However, MLP .csv headers are written for easy reading if the user wants to view one of them.

The user must know the formatting for the data .csv files, if wanting to input bulk spectra. Below is a table showing an example of the formatting. The first row in the .csv file is a header file: the first column in the header is simply “name”; subsequent columns in the header are numerical wavelength values; subsequent columns are the names of the labels (these must contain at least one character that is not a number). Each row after the header row is a new spectrum with according name, wavelength intensities, and label values.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | 0.503 | 0.5058 | Metallicity | Temperature [K] |
| Jupiter | 0.476937 | 0.478186 | 0.6 | 128 |
| Saturn | 0.424938 | 0.427929 | 1.0 | 95 |

The Testing code is set-up as a terminal UI that will prompt the user for the path to the data file followed by a path to the MLP file. The paths should be full paths to where the files are saved on the user’s computer, use either ‘/’ or ‘\\’ when specifying paths and do not enter the file extension; for example, “C:/Data/Albedo\_Spectra” or “C:/MLPs/Greedy0”.

If the user wants to add random gaussian noise to their spectra, they can alternatively enter “noise” into the terminal when it prompts for the data file path, and secret sub menu will be launched that will prompt the user to enter the level of noise. The level of noise is x% of the average intensity value for each spectrum, which will be the standard deviation of gaussian noise to be added to each spectrum.

Results will be output in the same folder that the data file is in.

For training, the code will first prompt the user to enter three data file paths: training, validation, and testing. The code will auto detect the number of inputs and outputs, assuming the formatting was followed correctly. Then it will prompt the user if they wish to build their own MLP or use the default one. The default one has the same hyper-parameters as the Greedy0 MLP presented in our paper. If the user selects to build their own, the code will launch a sub menu that will prompt the user to input each hyper-parameter (such as learning rate, hidden layers, etc). When the user builds their own MLP, the code will prompt the user if the MLP is a regressor, binary classifier, or multiple classifier (see end of README for details).

Next, training will automatically proceed by training the MLP on the training data specified, one epoch at a time. After each epoch, the MLP will be used to predict values on both the validation and testing set. Results will be output for each epoch in a subfolder created in the same parent folder as the training data. One epoch will be selected that had the lowest validation RMSE and will be used to output a .csv file with predictions on the test set. The best MLP will also be written, along with the MLP after each epoch in the subfolders corresponding to that epoch. There will also be a “results” .csv file written that shows the accuracy/RMSE for each epoch.

There are built in methods to create training, validation, and testing sets though the user will have to edit the c++ code for this.

REGRESSOR, BINARY CLASSIFIER, or, MULTIPLE CLASSIFIER:

A regressor outputs continuous values for each label specified in the data files. The classifiers assume label values are either 0 or 1. A 0 means the spectrum is negative for that class, and 1 is positive. For the binary classifier, each spectrum will be classified a one class (which ever had label has the highest output value from the trained MLP). For the multiple classifier, each spectrum will be classified as several classes (whichever classes had output values, from the trained MLP, greater than or equal to a given threshold). The threshold for the multiple classifier must be provided by the user, and the code will prompt the user to enter this value (typically this is 0.5).