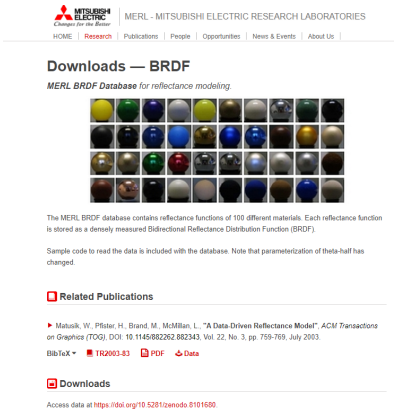
CSC 411(511), Final Project

Implementing a Fully-Connected Artificial Neural Network for Regression

Introduction

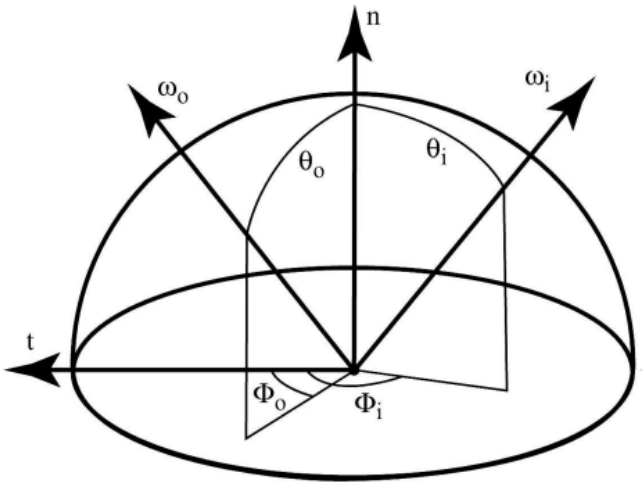
You will be using the MERL BRDF (Bi-Directional Reflectance Distribution Function) dataset for your Final Project. You may work in teams of up to five people. The dataset consists of measured reflectance from a variety of material surfaces.

The data and reader code can be obtained from https://www.merl.com/research/license/BRDF/.



The reader program is already set up to write out (standard output) a red, green, blue triplet of reflected radiance for a given set of incident angles (of irradiant light).[1]

The geometry of a BRDF looks like the following:



[1] More information on the dataset can be found in the accompanying paper: https://www.merl.com/publications/docs/TR2003-83.pdf

The variables include:

- $\boldsymbol{\theta}_i, \Phi_i$ - the angles of the incident vector (incoming light)
- $\boldsymbol{\theta}_o, \Phi_o$ - the angles of the existent vector (outgoing light)

ω_i and ω_o vectors each represent a differential ($d\omega$) beam, which you can think of as an infinitesimally narrow light beam. There is one for the incident beam, and the other, the reflected (outgoing) beam.

The following code from the main function in the reader program shows what is being printed.

```
lookup_brdf_val(brdf, theta_in, phi_in, theta_out, phi_out, red, green,
blue);
printf("%f %f %f\n", (float)red, (float)green, (float)blue);
```

The default resolution of the data is what you will use in your projects (i.e. 16 x 64 x 16 x 64 table). Note that $\boldsymbol{\theta}$ ranges between 0 and $\pi/2$ (in increments of 1/32 of π, or equivalently 1/16 of $\pi/2$).

Φ ranges between 0 and 2π, in 64 equal increments.

So, in your final data output, you will have all combinations of the input and output vectors at that resolution from the table.

How you manage the data input to your neural network (NN) is up to you, but the regression network you implement should accept as input the two vectors (in and out), and produce an intensity output.

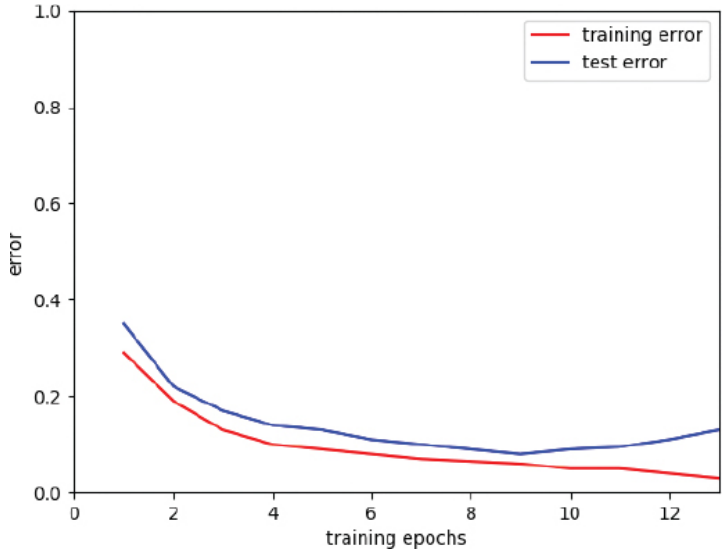
Requirements

How exactly you manage the intermediate data format read by your Python implementation of the NN is up to you. Hint: It would make sense to print this to a text file (in addition to the red, green, and blue as shown in lecture) using four more columns per row with the theta_in, phi_in, theta_out, and phi_out variables for their corresponding r, g, and b.

All students should randomize the order of your training set samples for every epoch using stochastic gradient descent in the backpropagation algorithm.

Make sure to select for your training set, samples distributed over the entire range of data output by the reader program.

You will need to plot the training error according to the examples in the textbook (per epoch), along the testing error. For example:



You should monitor the training of your NN and use "early stopping" to limit the amount of overfitting.

You will use the validation (sub)set only for adjusting hyperparameters. These include:

1. LEARNING_RATE
2. Number of fully connected neurons per the two required hidden-layers (for 511 projects).

Note, 411 students are only required to use a single hidden layer and may choose the number of neurons in that layer along with the LEARNING_RATE parameter. Using a validation set is not required for 411 projects.

Lastly, use of a NN framework is NOT permissible. You must implement the entire network using Python. Use numpy for the math, and matplotlib for the plotting of error data. Again, it is ***not*** permissible to use Tensorflow, etc.

See the code in Chapter 4 of Learning Deep Learning (LDL) for an example of what is required to implement a fully-connected network from scratch.[2]

Rubric

- Working NN code written from scratch in Python 50%
  - Must NOT use NN frameworks
  - **Requires only 1-hidden layer and one output (red)**
  - Use the proper activation functions (refer to lecture materials)
- Error plots rendered by matplotlib 25%

[2] Note that the code in Chapter 4 is for classification and uses its own appropriate activation functions as such. You must use the appropriate activation functions for regression networks (i.e., see Ch 5 & Ch 6 from LDL).

- Network weights printed out in a text file (only the final weights) 10%
- 5 minutes of a recorded video presentation of your work 15%
    - Include slides in your submission
- *Extra-credit: Try different LEARNING_RATE values and report on the results of each as above for a single LEARNING_RATE. (up to 5% extra)*
- Working NN code written from scratch in Python 50%
    - Must NOT use NN frameworks
    - **Requires only 2-hidden layer and three outputs (r, g, and b)**
    - Use the proper activation functions (refer to lecture materials)
- Error plots rendered by matplotlib 20%
- Justification (in presentation slides, etc.) for hyperparameters used 5%
    - LEARNING_RATE
    - Neuron counts
- Network weights printed out in a text file (only the final weights) 10%
- 10 minutes of a recorded video presentation of your work 15%
    - Include slides in your submission
- *Extra-credit: Try different hidden-layer activation functions and report clearly the quantitative results of using these with your validation set. (up to 5% extra)*