Biostatistics 615 Mastery Assignment #7 (10 pts)

Due by November 5th 2024 (Tuesday) 11:59pm. Use Gradescope (via Canvas) to submit an R file.

- Your submission should only contain one `R` file named `neuralNetworkLBFGSB.R` that contains a function named `neuralNetworkLBFGSB(p, df)`.
- Your code will be evaluated in Gradescope using 10 different test cases using an automated script. Full credit will be given if your code passes all test cases.
- You are allowed to submit multiple times before the deadline, but only the last submission will be graded. Automated feedback will be provided for each submission.
- You need to implement the function to work with arbitrary (valid) input values beyond the 10 cases tested. If you tweak your implementation so that your functions works specifically for the test cases, you will not receive any credit.
- You may test the function in the Google Colab page at https://bit.ly/615hw7xtra to test your code on a subset of test cases.
- Implement your function as efficient as you can. If your program does not finish after running for 3 seconds, you will lose the points for those test cases. Note that the official solution finishes much faster for any test case, so this should be a reasonable time limit. Also, note that the running time includes the time for reading the input files. The time reported in the Google Colab test page will be much faster because the input files are already loaded in the memory.
- Make sure that your implementation retains sufficient numerical precision.

Problem 1 - Fit a Neural Network Model using L-BFGS-B (10 pts)

`neuralNetworkLBFGSB(p, df)` to fit a neural network model using the L-BGFS-B quasi-Newton algorithm. Suppose $\{X_i, Y_i\}_{i=1}^n$ are the training dataset where X_i and Y_i are both scalars. To model the relationship between Y_i and X_i, we consider a one-layer neural network model with p nodes: for $i = 1, \ldots, n (100 \leq n \leq 200)$:

$$Y_i = \alpha_0 + \sum_{j=1}^{p} \alpha_j \text{GeLu}(\alpha_{p+j} + \alpha_{2p+j} X_i) + \epsilon_i$$

where $\epsilon_i \sim \text{N}(0, \sigma^2)$ and

$$\text{GeLu}(x) = x\Phi(x) = \frac{x}{2}\left[1 + \text{erf}(x/\sqrt{2})\right]$$

where $\Phi(x)$ is the standard normal cumulative distribution function and

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

To fit this neural network model, we consider the least squares estimates by minimizing the following objective function with respective to $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_{3p})^T$:

$$F(\boldsymbol{\alpha}) = \frac{1}{n}\sum_{i=1}^{n}\left\{Y_i - \alpha_0 - \sum_{j=1}^{p}\alpha_j \mathrm{GeLu}(\alpha_{p+j} + \alpha_{2p+j}X_i)\right\}^2 .$$

To solve this problem, use the L-BFGS-B algorithm in Topic 6. Instead of implementing the L-BFGS-B algorithm from scratch, in this homework, you are expected to use the `optim()` function built-in R, specifying `method="L-BFGS-B"` as an argument. Run `help(optim)` to see the detailed usage of the `optim()` function.

You need to implement your code to fit the following requirements.

- You need to restrict the range of each parameter α_i to $[-10, 10]$ for $i = 0, 1, \dots, 3p$.
- You need to specify sufficiently large number of iterations to ensure the convergence.
- Unlike Nelder-Mead algorithm, the initial values of $\boldsymbol{\alpha}$ should be randomly generated (e.g. $U[-1, 1]$) rather than setting them to zero.
- The input arguments of the `neuralNetworkLBFGSB(p, df)` function are the number of nodes p (integer) and a data frame `df` containing two vectors: `X` and `Y`.
- The function should return the list object that `optim()` function returns without any modification.

Example output of running the test code is given below:

```
> df = read.table('test.1.tsv',header=TRUE)
> head(df)
          X         Y
1 -3.141593 0.9706502
2 -3.103742 0.9392261
3 -3.065892 0.8200233
4 -3.028041 0.9437523
5 -2.990191 0.7037756
6 -2.952340 0.9143716
> neuralNetworkLBFGSB(10, df)
$par
 [1] -0.39004516  0.04952741 -0.21205026 -0.61917312  0.97290527  0.68635024
 [7]  0.86576028  0.04945598  0.30469199  0.09090632 -0.09539931 -0.95907045
[13] -0.65165837 -0.24545129  1.08811175 -0.48354285  0.28139998 -0.80127677
[19]  0.79220174  0.42870074  0.84195003 -0.21723009 -0.60768677  0.27421402
[25] -0.10027359  0.17147457  0.27139138 -0.05076553 -0.51119148  0.60451699
[31] -0.68691238

$value
[1] 0.008908137

$counts
function gradient
      42       42
```

```
$convergence
[1] 0

$message
[1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

Please note that your output may not be exactly the same as the example output above, but as long as the algorithm converges the results are similar to the solution within a predetermined relative error, you will receive a full credit.

The example input files `test.1.tsv` is provided within the file `hw7_xtra_examples.tar.gz` accessible in https://bit.ly/615hw7xtra.

The actual examples that will be used for grading will contain larger input, so be sure to implement your function to work with arbitrary valid input values.

Note that you are NOT allowed to use any functions outside the `base` and `stats` package in your implementation. Use `help(...)` to check whether a function belongs to the `base` or `stats` package or not.

The cloud machine that will be used for grading has a very small (0.5GB) memory, so make sure that you do not use too much memory in your implementation. If you code runs out of memory, ”Killed” error messages will appear in the user output.

No error handling for malformed arguments is needed.