Optimization and of Parameters for Complex Data:

Experimental Project and Design Exercise

Project Overview:

- Machine Learning methods are commonly employed on medical data to model complex relationships between factors leading to or presented in disease. Cancer data has been a long standing target of high quality detection and identification. We'll use the dataset linked below to train two different network types.

https://www.kaggle.com/uciml/breast-cancer-wisconsin-data

- Feature **selection** is an important part of designing an ANN model. Once again, we'll use some medical data to do a few things with MLP models, and see if we can determine an optimal subset of features, or maybe train using all the given data.

https://www.kaggle.com/ronitf/heart-disease-uci

Objectives:

- Generate scripts to model and train a Multi-Layer Perceptron and RBF networks using Matlab built in functions.
- Utilize an iterative approach to train models with pre-selected and generative parameters
- Train multiple instances of selected network models, and preserve model parameters for future use.
- Rank model performance using Confusion Matrix, ROC and performance metrics, and provide justification for selected model performance.

Requirements/Task(s):

Setup: You'll need to access the websites for each data set and look at the statistics for the vector components in each sample. The goal of the two models we'll be aiming for will be to construct optimally scaled designs that generate reliable performance.

Tips and Tricks: A single layer may not provide enough complexity to generate good results. There are a wide variety of parameters that can be set using the script options provided by Matlab's 'fitnet' function.
The code below can be used as a generic framework. The model below has 2 layers with 10 nodes per layer.

```
trainFcn = 'trainlm';  % Levenberg-Marquardt backpropagation.
% Create a Fitting Network
hiddenLayer1Size = 10;
hiddenLayer2Size = 10;
net = fitnet([hiddenLayer1Size hiddenLayer2Size],  trainFcn);
```

You can adjust the nodes or add a layer by adding a new 'hiddenLayer*(x)*Size' parameter to the *array* being passed to the fitnet function.
You will be asked to justify your hyper-parameter selection (number of layers & nodes per layer) as well as the learning rate and data division perimeters you select.

TASKS:
For tasks 1&2 the data has been reformatted for MATLAB. You'll find it attached to the assignment as a zip file. The input matrix is called P, each row of which corresponds to different *measurements* of a patient's tumor cell sample. The corresponding element in the output vector T is -1 if the cell was determined to be benign, and +1 otherwise.
(If you need a binary version of T, add one to all samples and divide the result by 2.)

Task 1
UCI Wisconsin Breast Cancer dataset:

Kernels are a great part of the Kaggle database system. They show work products of others on the same data, and often give insights that would require us to do tons of work.
https://www.kaggle.com/kanncaa1/feature-selection-and-data-visualization
Take a look at the graphical representations of the data as given in this kernel and see what you notice in terms of decision boundary complexity for the given features.
Now take a look at the plots and performance as recorded in section 5-4 of this kernel:
https://www.kaggle.com/mirichoi0218/classification-breast-cancer-or-not-with-15-ml

Select the parameters for the model, such as data division and network topology, and construct a network with an ROC AUC (area under the curve) that fits the performance of the models shown. Perfect performance is not expected, however, I would like you to treat this project as if it were a commissioned project – you want to sell me this model to use on my data in the future so make sure it's *competitive*!

If you want to adjust parameters of you 'net' – you can do so before you use the 'train' function. The available parameters that are set by default when using the 'trainlm' function are shown here:
https://www.mathworks.com/help/deeplearning/ref/trainlm.html
The values can be updated using the *dot* method. You may want to adjust these parameters as you make changes to the model topology, defaults might work for some smaller networks, but deeper structures (>2 layers) will need to be adjusted.

You may need to generate several hundred networks to find one with the right convergence parameters. I suggest you train at least 10 itterations of any given network (topology/ parameter configuration) to see what happens (an 'on average' assessment might let you see which network might produce better results if given the right starting point – telling you which one would be best to train again with new random parameters).

Be sure to **save** your 'net' results for the best networks, you might need to reproduce your results if you run into issues.

Task 2
Creating RBF neural nets:
First load P and T into MATLAB
divide it into training, validation, and test subsets (60%, 20%, and 20% ratios) using the following commands:

```
[trainP,valP,testP,trainInd,valInd,testInd] = dividerand(P,0.6,0.2,0.2);
[trainT,valT,testT] = divideind(T,trainInd,valInd,testInd);
```

Now create **two** RBF neural nets, exact and regular, using

```
net1 = newrbe(trainP,trainT,SPREAD1);
net2 = newrb(trainP,trainT,GOAL,SPREAD2);
```

```
% Please read MATLAB's help documentation
% https://www.mathworks.com/help/deeplearning/ref/newrbe.html
% https://www.mathworks.com/help/deeplearning/ref/newrb.html
```

adjust the values of SPREAD and GOAL until you get a **validation** MSE of **>0.75**
→ [mse1v & mse2tr] when running the **sim**(ulation).

```
y1v = sim(net1,valP);
mse1v=mse(y1v-valT);
```

% Or for near training:

```
y2tr= sim(net2,trainP);
mse2tr=mse(y2tr-trainT);
```

Note: you have to choose a SPREAD and GOAL, for newrb -- you may want to start with larger spreads, e.g. 10 or more, and use a loop for next values
Report on your observations.
Specify the network (rb or rbe, SPREAD, size, and GOAL) that got you the results you were aiming for.
Report the **test** MSE of your final model.
Generate the training, validation, and test confusion matrices (see the function here):
https://www.mathworks.com/help/stats/confusionmat.html
(note:).
Plot the training, validation, and test ROC curves for the above network.
https://www.mathworks.com/help/deeplearning/ref/roc.html

Task 3

UCI Heart Disease Data:

Download the data from Kaggle and import the .csv file and select the parameters necessary to transform it for machine learning.

https://www.mathworks.com/help/matlab/import_export/import-data-interactively.html

Use the dataset from Task 1 as an example of how to format the data. Apply transforms or selection of parameters to get the selection or subset you're looking for.

Knowing how to generate usable data for ML tasks is a big part of making your skills usble in real world applications.

Find a kernel in the Kaggle link that contains some processing results and use those results as a baseline for the performance of your classifier.

The dataset is small, be sure to use enough data for training but still leave enough examples to test your model's performance reliably.

This part is more or less on your own, read about the data and select whatever features you like. You can try to train the model with just 2 of the features or you can train with all of them. You can use multiple layers, adjust learning rates, adjust learning functions, or remove validation and set a performance goal with a large number of epochs – its up to you!

These projects will take more time train than to set up if you do it right.

Make your decisions about the parameters you want to use, implement those parameters, take a look at your results, and adjust the next set of parameters until you feel comfortable with your results.

Be **reasonable** about the time you spend, but don't expect to do this the day before its due – **automate** whatever processes you can by generating loops to train multiple instances of each model or multiple sets of parameters.

Deliverable:

Generate a report of your project based on the general form we've used so far. Report the performance of your network in terms of ROC AUC and whatever other metric you like. Generate a graph of your top performing networks (top 5 or top 10) to compare their AUC, outputs, loss or whatever other value you feel helps prove the models are useful for each case.

This project is intended to showcase everything you've learned and done so far in terms of practical skill. Include any custom code you might develop, and show how you automated the process of training multiple versions of each network configuration.

This project should be set up like a proposal to a perspective client, don't confuse your results with 20 pages of individual experimental outputs, duplications of code or performance data, don't use network training screenshots or snipped versions of graphs/charts/plots/ROC

curves showing menu bars.

Don't show 200 epochs of a network that stopped training at epoch 15, don't plot 20 different configurations on one plot if the results are too small to see or have tons of overlap.
Save you plots as files in case your need to adjust them.

If you didn't spend the time to learn how to plot multiple sequences of data on a single plot, or how to save results from net structures, or how to adjust Matlab scripts to automate your work, **now is the time to learn**.

Submit your project report online by the deadline set on Canvas with this submission link. ***No late work will be accepted*** for this project **for any reason** – plan accordingly.