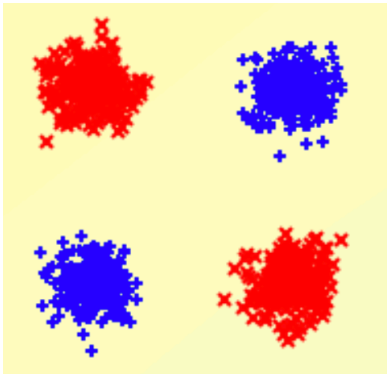
Introduction to Machine Learning (CS 135)
Assignment 04 (30 points)
Due on Gradescope by 11:59 PM Eastern, Saturday, 24 July 2021

For this assignment, you will be exploring the use of some neural network models—also known as multi-layer perceptron (MLP) models—on some supplied data. The data is laid out in what is known as the XOR pattern, with classes that cannot be separated by any technique like logistic regression using solely linear functions on the data, but which can be handled using a neural network classifier with a single hidden layer:*



Along with this document, you are provided with:

- A Python notebook in which to do all your work; you will submit this notebook, along with the usual collaborators file, and you will also submit a PDF generated from the final version of your notebook.
- Some Python utility files that will be called upon by your notebook to do some of the visualization work, and provide a better interface to some model internals (place them in the same directory as the notebook when working on your code).
- Data-files in the usual CSV format for inputs/outputs of a training and test set. Each input data-point has two features, x_1 and x_2, and output labels are either `0` or `1`.

Your code will examine different optimization algorithms for learning neural network weights (designated by the `solver` parameter in `sklearn` models), as well as different activation functions (`activation` in `sklearn`).

*This image shows two classes of data as crosses (red in original) and plus-signs (blue). If you have any sort of visual impairment that makes this hard to parse, the point of the diagram is simple, namely that we have two "blobs" of data of one class at upper-left and lower-right, and two "blobs" of another class at upper-right and lower-left. This means that no linear classification boundary is going to work—we will need some sort of hyperbolic form to separate the data into four "corners," two of each class.

Optimization: You will compare:

- **SGD** (stochastic gradient descent): a standard form of weight propagation using the first derivative of the network loss, with the gradient approximated each iteration using a subset of the data-set.
- **L-BFGS** (limited-memory Broyden-Fletcher-Goldfarb-Shanno): a more complex method using both the first and second derivatives.[†]

Activation: You will compare two functions, both seen in class:

- **ReLU** (rectified linear unit): a simple piece-wise linear activation function, popular in many modern applications.
- **Sigmoid**: the traditional logistic Sigmoid activation function, historically one of the most popular (and the same non-linear function as used for logistic regression, but put to quite a different use in the neural network context).

Code completion (26 points)

1. (*4 pts.*) Use the provided class `MLPClassifierWithSolverLBFGS`, which is a wrapper around the `sklearn` `MLPClassifier` model, but which provides more convenient access to some of its internal features. The class is fully described in its source-code. You will be building models with two hidden nodes that use the *ReLU* activation, and optimizing using *L-BFGS*.

   Train 16 separate such models, each one using a different random initialization-state; to do so, each model should be built with `random_state` set to $0, 1, 2, \ldots, 15$, in order. (The supplied code already builds the first of these, you just need to modify it to create and store the remaining models.)

   (a) Plot a 2D visualization of the learned decision boundaries for each of the 16 models.[‡]

   (b) What fraction of the runs achieve 0 error? What happens in other cases, and why do you suppose this is? How long, on average, does it take the models to converge?

2. (*4 pts.*) Repeat the process to build 16 randomized models, each of which combines the *logistic Sigmoid* activation with the *L-BFGS* solver. Again, plot each of the decision boundaries, and analyze the performance just as before.

[†]This method is explained in some detail at the following link (understanding all the details is beyond the scope of this class, and not necessary for completion of the assignment, but the information is provided for those who are interested): http://aria42.com/blog/2014/12/understanding-lbfgs.

[‡]The supplied code in `viz_tools_for_binary_classifier.py` will do this for you. By default, the code uses a red/blue color-map. If for any reason you find this color-map hard to process, you can substitute other color-maps for it to help you visualize the data better and answer the analysis questions. If you want to change the color-map, you can do so by changing the value of `cmap` at line `80` in the source-code just referenced. You can find a list of available color-maps at: https://matplotlib.org/stable/tutorials/colors/colormaps.html. You may also want to modify the markers/colors used to plot the data itself; you can do this by modifying lines `46`–`47` of the source.

3. (*6 pts*) Use the default `sklearn MLPClassifier` model, with the *ReLU* activation, but trained using *SGD*. Again, you should build 16 models with random seeds in $0, 1, \ldots, 15$, setting up the models to do stochastic gradient descent as follows:

   - `solver='sgd'`
   - `batch_size=10`
   - `max_iter=400`
   - `tol=1e-8`
   - `learning_rate='adaptive'`
   - `learning_rate_init=0.1`
   - `momentum=0.0`

   That is, your models use 10 training examples per weight update step, for 400 total passes through the data, stopping early only if loss values change by no more than the tolerance (`tol`), and modifying the learning rate as it goes.[§]

   Again, plot the decision boundaries learned by each of the 16 models, and analyze the performance rates as before. In addition, discuss the difference between the results you see here and those from the first set of models, which also used the ReLU activation, but optimized differently—what are the most noticeable differences between the runs, and why do you suppose this happens?

4. (*6 pts.*) Repeat the process of the last step, again using *SGD*, but using the *logistic* for activation. Again, plot the results, analyze them, and then compare these results to the ones using L-BFGS with the logistic.

5. (*6 pts.*) You will now analyze the loss curves of the various model runs already completed. Each such curve can be accessed via the `loss_curve` attribute of an MLP classifier model for analysis and plotting.

   (a) Plot the loss curves for each of the four groups of models in a (2×2) grid, so each panel will contain the curves for all 16 of the relevant models.

   (b) Based upon these plots, and the ones you made in prior steps, which activation function seems easier to optimize? Which requires the most iterations in general?

   (c) Does it appear convincing that one of the activation functions is simply easier to optimize than the other? What are three more experiments we could conduct that would help us establish if this is generally true?

[§]Fuller discussion of SGD can be found at https://scikit-learn.org/stable/modules/sgd.html.

Code submission (4 points)

1. (*2 pts.*) Submit the source code (`hw04.ipynb`) to Gradescope.
2. (*2 pts.*) Along with your code, submit a completed version of the `COLLABORATORS.txt` file. An example has been provided, which you should edit appropriately to include:
   - Your name.
   - The time it took you to compete the assignment.
   - Any resources you used to complete the assignment, including discussions with the instructor, TA's, or fellow students, and any online or offline resources consulted. If you did not need to consult any outside resources, you can say so.
   - A brief description of what parts, if any, of the assignment caused you to seek help.

Submission details: Your code must work properly with the versions of Python and other libraries that are part of the CS 135 standard environment. The class Canvas site contains instructions for installing and using this environment. You can write the Python code using whatever tools you like, but you should ensure that the code executes properly.