CSCI 635: Introduction to Machine Learning
Homework 3: Nonlinear Prediction

Due Date: Monday, December 4, 11:59pm *Late:* Tuesday, December 5, 11:59pm

Instructions: The assignment is out of 100 points. Submit your assignment through the Dropbox in MyCourses as two files zipped into one container (named according to the convention <your_last_name>-hw3.zip). The individual files are:

1. A *.pdf file for the write-up, named <your_last_name>-hw3.pdf. Furthermore, while your Python scripts will allow you to save your plots to disk, please copy/insert the generated plots into your document again mapped to the correct problem number. For answers to various questions throughout, please copy the question(s) being asked and write your answer underneath (as well as the problem number).
2. The *.py files for your answers/implementations to Questions 1a, 1b, 1c, named `q1a.py`, `q1b.py`, `q1c.py`. Your code should be commented and it should be made clear to the grader how to run your files to reproduce your results (include this in a brief README.txt to help make this clear).

Data: In this assignment we will use the datasets that came from the last homework assignments (i.e., HW # 2).

Grade penalties will be applied (but not limited to) for:

- Not submitting the write-up as instructed above.
- Submitting code with incorrect file names.
- Using machine-learning-specific libraries, e.g., TensorFlow, PyTorch, Scikit-Learn, etc.

In this assignment, you will implement, in **raw numpy/scipy**, a nonlinear model, i.e., the multilayer perceptron (MLP), to tackle the problems you applied your Maximum Entropy and naïve Bayes models on in the last homework. Furthermore, you will design the MLP's parameter update procedure – backpropagation of errors – and optimize it with stochastic gradient descent (SGD). As in the last homework, your focus will be on multiclass classification.

Problem #1: Learning a Multilayer Perceptron Model (60 points)

Problem #1a: The XOR Problem Revisited (50)

You will start by first revisiting the classical XOR problem you attempted to solve in the last homework with maximum entropy. The data for XOR is in `xor.dat` (from last homework).

First implement a single hidden layer MLP (this means that Θ will have two sets of weights and two bias vectors, $\Theta = \{W_1, \mathbf{b}_1, W_2, \mathbf{b}_2\}$) and perform a gradient check it as you did in the last assignment. You should observe agreement between your analytic solution and the numerical solution, up to a reasonable amount of decimal points in order for this part of the problem to be counted correct. Make sure you check your gradients before you proceed with the rest of this problem. Make sure the program returns "CORRECT" for each parameter value, i.e., for any specific parameter θ_j in Θ, we return "CORRECT" if: $\sim \nabla_{\theta_j} < 1e-4$.

Once you have finished gradient-checking, fit a **single hidden layer** MLP to the XOR dataset. Record what tried for your training process and any observations as well as your final accuracy. Code goes into a file you will name `q1a.py`. Contrast your observations of this model's performance with that of the maximum entropy model you fit in the last homework.

Problem #1b: The Spiral Problem Revisited (25)

Now it is time to fit your MLP to a multi-class problem (generally defined as $k > 2$, or beyond binary classification). The spirals dataset, as mentioned in the last assignment, represents an excellent toy problem for observing nonlinearity.

Instead of the XOR data, you will now load in the spiral dataset file, `spiral_train.dat`. Train your MLP on the data and then plot its decision boundary (the process for this MLP would be the same as for the maximum entropy model, much as you did in the last assignment). Please save and update your answer document with the generated plot once you have fit the MLP model as best you can to the data. Make sure you comment on your observations and describe any insights/steps taken in tuning the model. You will certainly want to re-use the code as work through sub-problems to minimize implementation bugs/errors. Do NOT forget to report your accuracy. Code goes into a file you will name `q1b.py`. Contrast your observations of this model's decision boundaries/performance with that of the maximum entropy model you fit in the last homework.

Problem #1c: IRIS Revisited (25)

You will now fit the MLP to the IRIS dataset, which has a separate validation set in addition to the training set. You will have two quantities to track – the training loss and the validation loss.

Fit/tune your MLP to `iris_train.dat`. Note that since you are concerned with out-of-sample performance, you must estimate generalization accuracy by using the validation/development dataset, `/problems/HW2/data/iris_test.dat`. Code goes into a file you will name `q1c.py`.

Note that for this particular problem, you will want to make sure your parameter optimization **operates with mini-batches** instead of the full batch of data to calculate gradients (in case you did this for the previous sub-problems). As a result, you will have to write your own mini-batch creation function, i.e., a *create_mini_batch*($\cdot$) sort of routine, where you draw randomly **without replacement** M data vectors from $\mathbf{X}$ (and their respective labels from y). This will be necessary in order to write a useful/successful training loop.

Besides recording your accuracy for both your training and development sets, track your loss as a function of epoch. Create a plot with both of these curves superimposed. Furthermore, consider the following questions: What ultimately happens as you train your model for more epochs? What phenomenon are you observing and why does this happen?

Fight this phenomenon by implementing an L2 decay penalty. Re-fit your MLP (tune the regularization coefficient λ) to the data and include a plot of its training/validation loss curves as well as its final train and valid accuracies. Comment on the training and validation losses and accuracies with respect to your regularized MLP.