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

Due Date: Monday, December 16, 11:59pm *Late:* Tuesday, December 17, 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>-hw2.zip`). The individual files are:

1. A *.pdf file for the write-up, named `a2.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, and 2, named `q1a.py`, `q1b.py`, `q1c.py`, and `q2.py`.

Data: In this assignment we will use the MNIST datasets which contains 28×28 images of handwritten digits. The other datasets come from the last homework assignments (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 any machine-learning-specific library for your code, e.g., TensorFlow, PyTorch.

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. Finally, you will also apply and train your MLP to tackle the infamous MNIST dataset, which is a larger-scale, more challenging problem.

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

Problem #1a: The XOR Problem Revisted (30)

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 (such as any relating to the model's ability to fit the data) as well as your final accuracy. Code goes into a file you will name `q1a.py`. Contrast your observations of this model's function approximation ability with that of the maximum entropy model you fit in the last homework.

Problem #1b: The Spiral Problem Revisited (15)

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`.

Problem #1c: IRIS Revisited (15)

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 maximum entropy model to the IRIS dataset, `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(·)* 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? What is a way to control for this?

Problem #2: Image Categorization with MLPs (40 points)

Adapt your MLP's softmax output layer by making the outputs correspond to a probability distribution over the 10 object classes. Then, train the network using the MNIST training set. Use the cross-entropy loss metric, and use (mini-batch) stochastic gradient descent to train your network. You will need to experiment with different initializations and learning rates.

Training: Use a validation set: partition the training samples from each class into 80% training and 20% validation. The 80% will be used to tune network weights. The validation set will be used to check generalization at each epoch. You should select a fixed number of epochs to run. Record where error on the validation set increases, and then use the weights in the epoch before this overfitting occurs as your final network weights.

Once trained, run your network on the MNIST test set, and compute the recognition rate for each individual class, and for all classes using your trained classifier. In the write-up, provide the following:

- a learning curve plot showing the cross-entropy vs. epoch for both the training set and the validation set, and use a line to show the epoch where overfitting occurs
- a histogram showing the accuracy for all classes, and for each individual digit class
- a table containing 1 example of 1 correct and 1 incorrect test sample from each digit class.

Comment on the accuracies, success & failure examples & the learning curve; two paragraphs should be sufficient.
Code: Provide a main program q2.py, along with any supporting .py files in a zip, and a file containing the saved model weights (see https://pytorch.org/docs/stable/notes/serialization.html).