Homework 2 for CSI 531

Due: Oct 26, 11:59pm

Instructions: This homework is made of two parts: programming and written problems.

A separate blackboard assignment is created for each part.

For the programming part: You should submit your solutions in one single zip file, named "[yourFirstName-yourLastName].zip", to the homework1-code assignment through blackboard. The zip file should include one folder, named "code" and one jing.txt file that includes the link to your JING screencast video (look at JING instructions on blackboard). All the code for programming problems should be in the "code" folder. Make sure to mention the Python version you are using. Make sure you can import your .py files into Python without error and test your solutions before submitting.

For the written part: Submit a PDF file named "[yourFirstName-yourLastName].pdf" that includes your solution, typed in an editor like MS word or using LaTeX. Do **not** submit images of handwritten solutions.

1. Stochastic Gradient Ascent (SGA) for Logistic Regression. In the exercise, you will implement logistic regression algorithm using SGA, similar to the logistic regression algorithm that you have seen in class. You will work with the datasets attached to the assignment and complete the logisticRegression.py file to learn the coefficients and predict binary class labels. The data comes from breast cancer diagnosis where each sample (30 features) is labeled by a diagnose: either M (malignant) or B (benign) (recorded in the 31-st column in the datasets). Read the main code, check the configuration parameters, and make sure the data is loaded and augmented correctly. Do not use logistic regression packages.

   (a) [15 pts.] Complete the function predict(x, w), gradient(x, y, w), and cross_entropy(y_hat, y) functions according to the instructions in logisticRegression.py. These functions will be used in the main SGA algorithm (logisticRegression_SGA).

   (b) [15 pts.] Complete the logisticRegression_SGA(X, y, psi, epsilon, epochs) function. In class, we used a stopping criterion for repeat loop in the SGA algorithm for logistic regression: the loop continued until the norm of difference between w's in consecutive iterations were less than a predefined number ϵ (line 11 of the algorithm:

$\|\tilde{w}^t - \tilde{w}^{t-1}\| \leq \epsilon$). Here, in addition to this criterion, we would like to limit the number of epochs (iterations over the whole dataset, or t (step/iteration number) in the algorithms in the slides) to a predefined number (max_epochs). In the main function, max_epochs is initialized to 8.

(c) [15 pts.] Complete the rest of the main code to use the learned w to predict class labels for test datapoints (that has not been used for learning the w's) and to calculate and print the average cross-entropy error for training and testing data.

(d) [15 pts.] Run the code on the cancer dataset with different psi and epsilon values. Check the change in cross-entropy values across iterations (in the plot) and the average training and testing cross-entropy errors. What do you observe about the losses and number of iterations? What do you conclude?

2. Multi-Layer Perceptron (MLP). As you remember in class, the activation functions (f) used in Neural Networks can be of different kinds, such as Relu, Sigmoid, and Step functions. For MLP, we worked with Sigmoid activation function for input and hidden layers in class. That is how we achieved the net gradient for one weight between the input and hidden layer $\delta_j^h = z_j(1-z_j)\sum_{k=1}^{p}\delta_k^o.w_{jk}^o = z_j(1-z_j)W^o\delta^{\mathbf{o}}$ and the gradient for weights between hidden and output layer $\delta_k^o = (o_j - y_j).o_k.(1-o_k)$. We used these to calculate the gradients and updates in the backpropagation and gradient descent phases in the MLP training algorithm. In this exercise, we would like to explore other activation functions for a specific MLP. Consider the following MLP with 1 hidden layer, 2 input neurons and 1 output neuron:

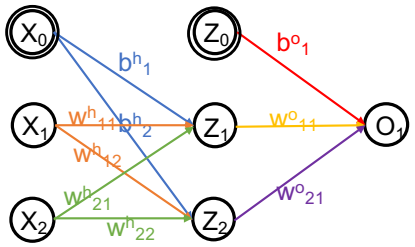

Figure 1: Simple MLP Architecture

(a) [25 pts.] Suppose that, instead of the sigmoid activation function, we use linear activation function in all neurons. This way, the output of unit j in the hidden layer is $z_j = c\times(\mathbf{w_j^h}^T\tilde{\mathbf{x}}) = c\times(\sum_{i=1}^{n} w_{ij}^h x_i + b_j^h x_0)$, where $\tilde{\mathbf{x}}$ is the augmented input vector for some fixed constant scalar c. Similarly, the output of unit j in the output layer is $o_j = c\times\sum_{i=1}^{n} w_{ij}^o z_i + b_j^o z_0$. Suppose that you are given all weight parameter

values (W^o and W^h are given). Re-design the MLP to compute the same function (the same output $\mathbf{o}$ based on given inputs $\mathbf{x}$) without using any hidden units (no hidden layer). Calculate the new weights of this network based on the old weights and the constant scalar c.

(b) [15 pts.] According to your solution to the previous part, is it always possible to represent a neural network that only uses linear neurons, without a hidden layer? Explain why.