20CS6037Fall2020

Assignment 4

Assigned 11/17/2020

Due on Canvas 11/28/2020 by 11:59PM

(30 points)

Problem 1 (10 points) Implement the **delta training** rule for a two-input linear unit. Train it to fit the target concept $x_1 + 2x_2 - 2 > 0$. You must generate your own examples as follows: generate random pairs (x_1, x_2) and assign them to the positive class if $x_1 + 2x_2 - 2 > 0$; otherwise assign them to the negative class.

(a) Plot the error E as a function of the number of training iterations/epochs.

(b) Plot the decision surface after 5, 10, 50, 100 iterations.

(c) Use different learning rates, analyze which works better and explain why.

(d) Now implement delta rule in an incremental fashion (as opposed to batch fashion when all the data are presented for training, the incremental approach updates the network after each example). For the same choice of other parameters (learning rate, etc.). Compare the two approaches in terms of **total execution time** and **number of weight updates (for example, use the MATLAB tic-toc combination)**.

Problem 2 (20 points) Consider now exactly the same problem as above (save the data you created, and the training and test sets you used) and implement variable learning rates as follows:

(a) **Decaying rates**. Start with a rate η and decrease after each iteration multiplying it by a number in (0, 1), for example, 0.8, so that after one iteration, your new learning rate will be $\mathbf{0.8\eta}$, after two iterations it will be $\mathbf{0.8^2\eta}$, and after k iteration it will be $\mathbf{0.8^k\eta}$. We know the tradeoff between the magnitude of the learning rate and speed of the algorithm: large rates tend to yield algorithms which are unstable, while small weights will result in a slow algorithm.

(b) **Adaptive rates**. Here the idea is to implement a procedure where the learning rate can increase or decrease as it may be needed. The idea is as follows:

1. Start with an initial learning rate. Calculate the initial network output and error.
2. Calculate new weights, biases, and the corresponding network output and error for each epoch using the current learning rate.
3. If the new error exceeds the previous error by a threshold **t** (which we decide in advance), then discard the calculated new weights and bias, and decrease the learning rate by multiplying it by a value **d**, in (0,1), preferably, close to 1;
4. Otherwise, if the new error is smaller than the previous error, then the weights and biases are kept, and the learning rate is increased by multiplying it by a value **D** (slightly) larger than 1.

Note: For example, starting with learning rate **η=0.5, t=1.03, d=0.9**, and **D=1.02**, a possible sequence of the learning rates could be η=0.5↓, 0.45↑, 0.4590↓, 0.4131↑, 0.4214↑, 0.4298↑, 0.4384↑ (note that I just generated these values w/o actually taking into account the errors and threshold **t**).