20CS6037
Fall 2021
Instructor: Anca Ralescu

Assignment 4

Assigned on Oct 14, 2021
Due on Canvas Oct 28, 2021, at 11:59PM
(50 points)

Instructions:

- **In a comment section at the top of your code list all the team members. And include the names in your report**
- **There are three opportunities to submit your program. However, only the last submitted version will be graded**
- **You are to actually write your own implementation. Please do NOT USE any pre-built packages from python or MATLAB**
- **If you use python, you can use numpy, matplotlib, pandas, seaborn and train_test_split from sklearn**
- **For full points, your submission must have proper results, analysis and all your graphs must be labelled (with axis titles, legend, plot title)**
- **If you have any general questions regarding the assignment, please bring it up as a discussion on canvas for all students to benefit**
- **You can also email the TAs and cc to the professor if you have any specific questions/clarifications**
- **Turn In your report (analysis + graphs: .pdf) and your source code (.zip) as two separate attachments**
- **Late Penalty: You will be assessed a penalty of 5% for each late day (max of 2). Email the TA and cc to the professor if you know that you will not meet the deadline due to any reason**
- **CITE your reference. You can refer to any literature/material, but make sure to cite it**
- **DO NOT copy code from online sources. This includes any websites/blogs/github. We expect you to write your own code**
- **DO NOT share your work. Assignments are meant to test your knowledge and understanding. This includes code and report. You can discuss your approach, observations with your peers, but do not share your work**
- **No points will be awarded if we find concrete evidence of cheating**

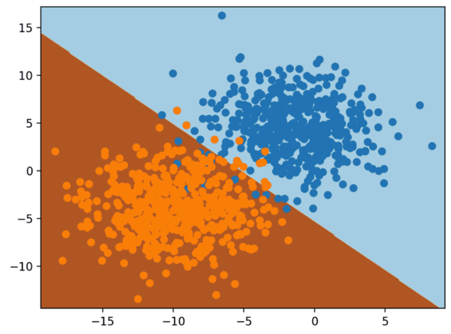
In this assignment you will be implementing a perceptron (specifically two-input linear unit). They are binary classifiers which makes its predictions based on a linear predictor function

Dataset:
You will be generating your own dataset as follows:

- For this assignment we will be using a random two feature dataset
- Generate random 200 pairs (x1, x2) selected uniformly from [-40, 40] (python users: np.random.randint
- Assign them to the classes by using this rule
    - $Y = \begin{cases} 1, & x_1 + 3x2 - 2 > 0 \\ -1, & x_1 + 3x_2 - 2 \le 0 \end{cases}$
- Thus, you will have a two feature and binary classification dataset. Save this and use as the dataset
- **NOTE: Make sure to have a balanced set of points between the positive and negative classes (~50%). You can keep generating random numbers till you meet this criterion**

Problem 1 (20 points total) Implement the **delta training (consult notes, also see https://en.wikipedia.org/wiki/Delta_rule)** rule for a two-input linear unit. Train it to fit our target concept.

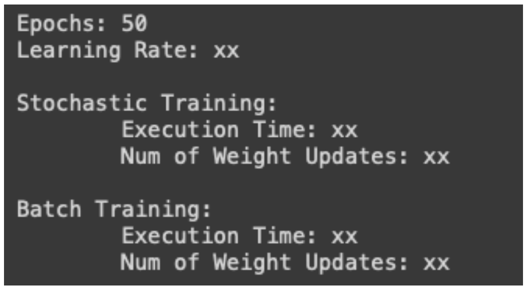
(a) **(4 points)** Plot the training error E vs number of training iterations/epochs (25 epochs)
(b) **(4 points)** Plot the decision surface after 5, 10, 50, 100 iterations. Plot all the 4 as subplots in one plot



Note: This is just a sample decision surface, you answer will not look like this.

(c) **(4 points)** Use different learning rates (0.1, 0.01, 0.001, 0.0001) and plot the training error E vs epochs (50 epochs). Plot the 4 as subplots in one plot. Briefly write up an analysis on which works better and explain why

(d) **(8 points)** Now implement delta rule in an incremental fashion (stochastic): as opposed to batch fashion when all the data are presented for training, the incremental approach updates the network after each example.
Train for 50 epochs and pick the best learning rate according to your analysis from (c)
Compare the two approaches in terms of **total execution time** and **number of weight updates (use the `timeit` from python or `tic-toc` from MATLAB).** Report the results using the following template:



```
Epochs: 50
Learning Rate: xx

Stochastic Training:
        Execution Time: xx
        Num of Weight Updates: xx

Batch Training:
        Execution Time: xx
        Num of Weight Updates: xx
```

Problem 2 (30 points total) Consider now the same problem as above and implement variable learning rates as follows:

(a) **(10 points) Decaying rates**: Start with a high learning 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 0.8η, after two iterations it will be $0.8^2\eta$, and after k iteration it will be $0.8^k\eta$. 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 slow algorithms.

Now train for 25 epochs and plot the following:

- Train error vs epochs (decaying rates)
- Train error vs epochs (non-decaying/constant rates)

NOTE: Both curves should be on the same plot. And both should have the same starting learning rate

Report your starting learning rate. Write up an analysis comparing your results and explain which works better and why.

(b) **(20 points) 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, say 0.3. 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=0.03, 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 considering the actual errors and threshold **t**)

Similar to (a) train for 25 epochs and plot the following:

- Train error vs epochs (Adaptive rates)
- Train error vs epochs (non-adaptive/constant rate)

NOTE: Both curves should be on the same plot. And both should have the same starting learning rate

Report your starting learning rate, t value, d value, D value. Write up an analysis comparing your results and explain which works better and why.

SUBMISSION:

1. Report (.pdf):
   a. Include group member names
   b. Include proper titles and sections
   c. Attach screenshots of output/plots
2. Source code (.zip)

NOTE: Please turn in the report and source code as two separate attachments on canvas