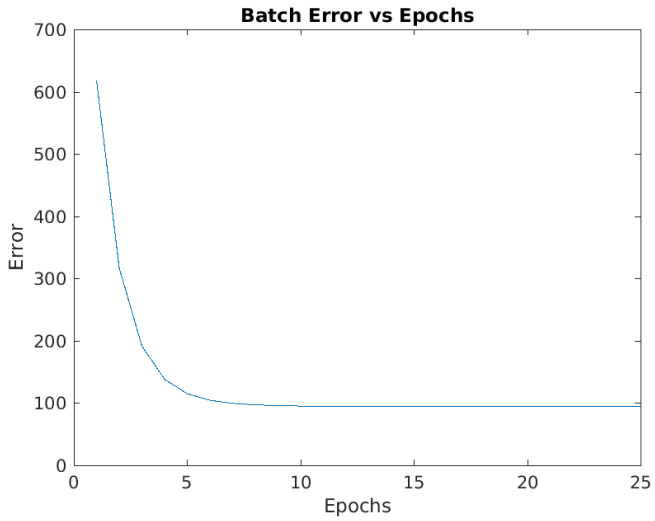
Machine Learning Assignment 2

M13471127 - Devarashetti, Akhil Kanna
M13500936 - Ghotra, Sandeep Singh
10/30/2019

Problem 1: Delta Training

We created 300 data-points for x1 and x2 ranging from -3 to 3 which is based on a normal distribution with mean close to 0 and standard deviation close to 1.

a. **Training Error**: From our experiments, it was clear that the error would saturate after 15-20 epochs in the batch mode.

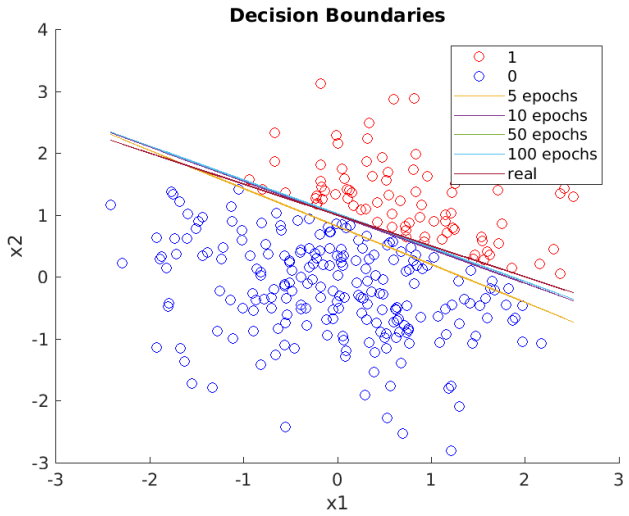


How to execute: Find the file named ‘batch.m’ in the zipped folder and execute it. The logic for all the delta rule algorithms used in this assignment reside in the ‘perceptron.m’ file.

Output:

```
>> batch
training time:
    0.0631

weights
    0.2934    0.5615   -0.6696

confusion_matrix =
   227     0
     8    65

Accuracy:
    0.9733
```

b. **Decision Surface**: The yellow line in the figure below gives the predicted decision boundary between the two classes.



How to execute: Find the file named 'batch.m' in the zipped folder and execute it.

c. **Different learning rates:**

The table below shows our results of training the model with 300 data-points.

Learning Rate	0.1	0.01	0.001	0.0001
Hit-rate (Accuracy)	0.63	0.67	0.97	0.97
Error	4 x 10^79	9 x 10^25	100	100
Epochs till saturation	25 (Not saturated)	25 (Not saturated)	20	100

Our exploration: We started off with 10 epochs and a very low learning rate of 0.0001 to not over-learn from the errors. We indeed saw a slow decrease in the error. We then increased the learning rate to 0.001 and epochs to 100. The error curve lowered dramatically within the first 20 epochs and flatlined after that. Increasing the learning rate to 0.01 and 0.1 led to exponential increase in the weights and the error. It is a clear indication of changing the weights too much from the error.

Suggestion: The learning rate of 0.001 seems to work best.

Reason: The change in weight should be very low between 0-1 for each epoch. But sometimes the delta can go high due to the accumulation of all the errors in the epoch, or due to the magnitude of the input. So, to reduce this, we will require a very low learning rate.
The reason for this value of the learning rate has to do with the order of magnitude of the inputs (x1, x2) and the error over the total number of data-points (300 x error).
From the experiments, we noticed that:

- Order of magnitude of learning rate is inversely proportional to the order of magnitude of the inputs.
- The learning rate is inversely proportional to the number of points in each epoch.

d. Incremental Learning:

We trained the model with 25 epochs over 300 data-points and the results are in the table below for an average over 9 trails.

Mode	Training time (avg)	No. of weight updates
Batch	64.1 milliseconds	25
Incremental	68.8 milliseconds	25 x 300 = 7500

The batch mode is 7.3 % faster than the incremental mode for the 25 epochs.

How to execute: Find the file named 'incremental.m' in the zipped folder and execute it.

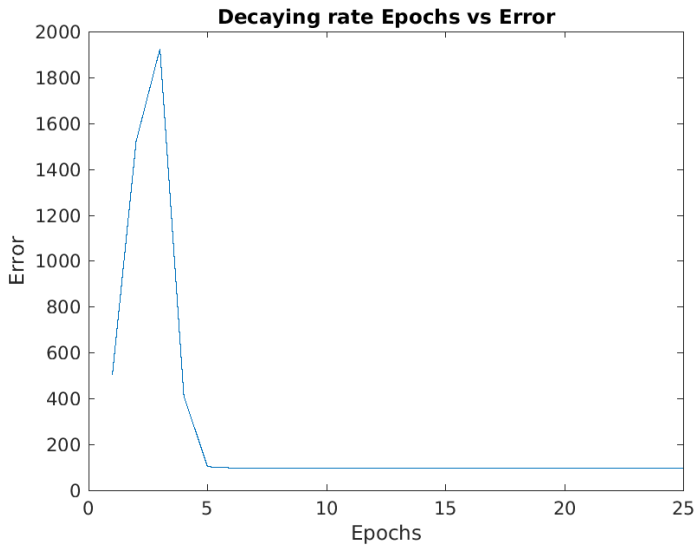
Output:

```
>> incremental
training time:
    0.0688

weights
    0.2773    0.5980   -0.6971

confusion_matrix =
   222     0
     9    69

Accuracy:
    0.9700
```

Problem 2: Variable Learning Rates

a. Decaying Rates:

The figure below depicts a typical behaviour of the decaying learning rates. The exact values of the hyperparameters used here are learning rate = 0.01 and decay = 0.6. A good combination of learning rate and decay would be using a high enough learning rate that does not over-learn, but fine-tunes the learning with each epoch. Although the error in the initial epochs was very high, the error quickly falls down to a minimum.



This rate of optimization is much higher than the constant learning rate seen in the problem 1a where the error reaches its minimum after 20 epochs while this approach reaches error-minimum in 6 epochs. This is 70% faster than constant learning rate method.

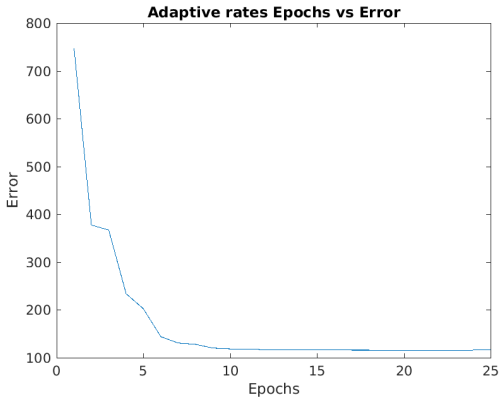
How to execute: Find the file named 'decaying_rates.m' in the zipped folder and execute it.

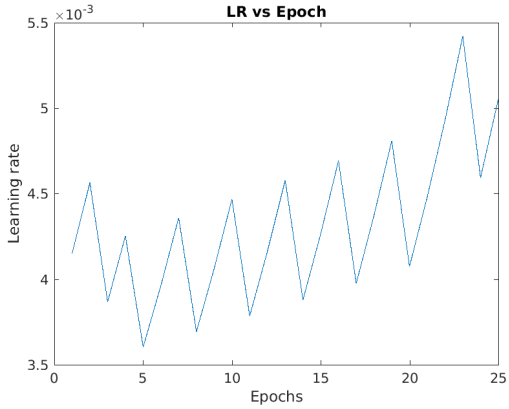
Output:

```
>> decaying_rates
weights
    0.3409    0.5849   -0.6573

confusion_matrix =
   209     1
     9    81

Accuracy:
    0.9667
```

b. Adaptive rates:

The graphs below are the results of our model with learning rate as 0.01, d=0.7, D=1.1 and t=0.1.





This approach optimizes the learning rate. This means that even when learning rate is too high or too low, the algorithm makes adjustments to it until better results are observed. One has to be careful while selecting the values of d and D. The reducing rate of learning rate must be greater than the increasing rate. This means that 1 - d > D - 1. This approach is highly useful when we do not know what value of learning rate would be good or when we do not know the number of data-points and magnitude of inputs.

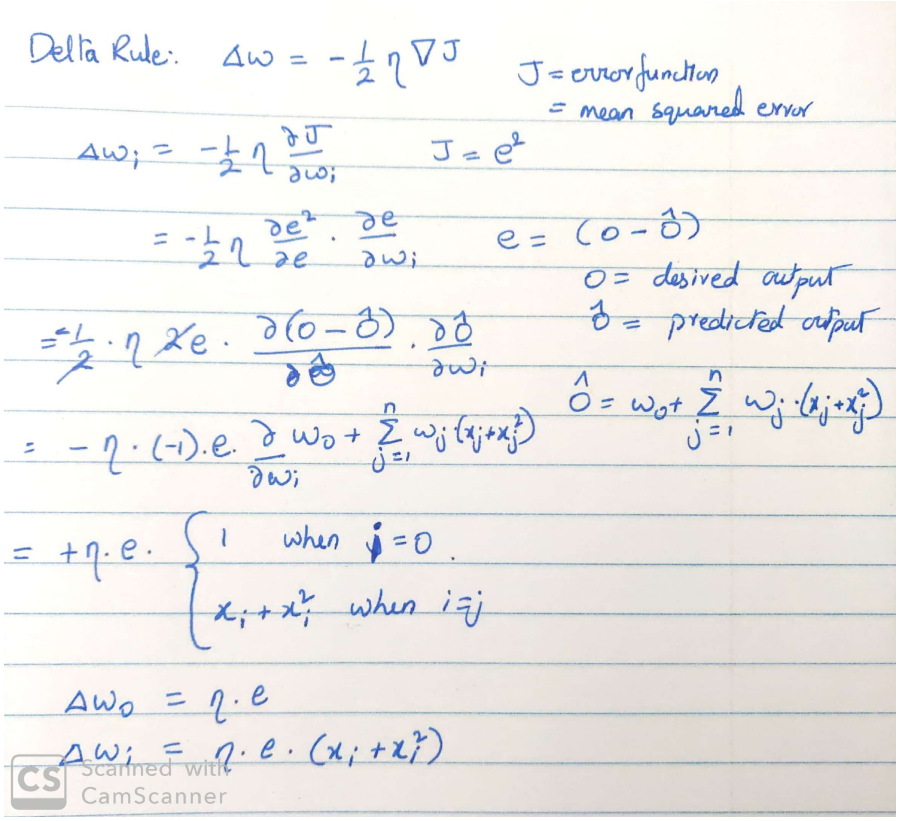
How to execute: Find the file named 'adaptive_rates.m' in the zipped folder and execute it.

Output:

```
>> adaptive_rates
weights
    0.2971    0.5262   -0.6334

confusion_matrix =
   218     0
    14    68

Accuracy:
    0.9533
```

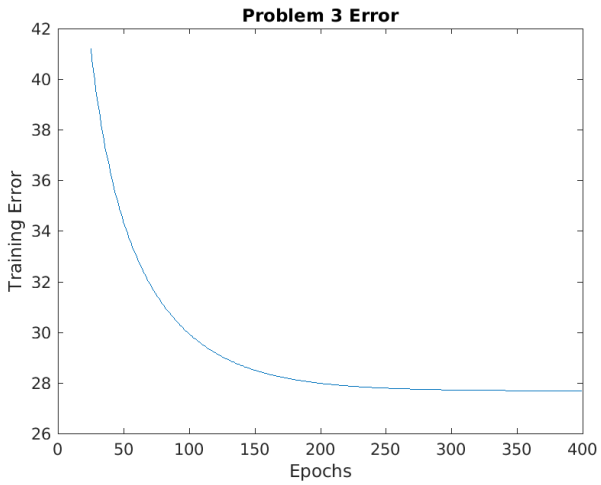
Problem 3: Gradient Descent Training

a. The gradient descent training rule for o:



b. For this algorithm, we used the given dataset named “hayes-roth.data”. We rescaled all the columns from 0 to 1 and the output classes from 1, 2, 3 to -1, 0, 1. Using the above derived learning algorithm with learning rate of 0.001 for 400 epochs, the accuracy we measured is 86%.

Error graph:



Output of the program:

```
>> problem3
error =
   27.6777

learned weights
   -0.9105
   -0.0536
   -0.0110
    0.5440
    0.5321
    0.4492

Confusion matrix =
     7     2     0
     0     8     1
     0     1     9

accuracy =
    0.8571
```