

Project 1

CS 4793 – Fall 2003
Tom Bylander, Instructor

assigned September 11, 2003
due October 16, 2003

This project is designed to be performed by groups from 1 to 5 people. On September 18, you will need to tell me what group you are in. Except for the initial source code distribution, no code sharing is permitted between groups. No collaboration is permitted between groups except for explaining what the initial source code is doing. Choosing a programming language other than C needs to be approved by me in advance.

The initial source code, which is written in C, can be downloaded from the web site. The README file explains the design of the code. The code performs the perceptron algorithm, and is designed so that other learning algorithms can be implemented by replacing `perceptron.c` while keeping all the other source files the same. The design is perhaps a little unusual. This is partly intentional to ensure some difficulty if you try to port an algorithm from some other source. Any deviation from the design framework needs to be approved by me in advance.

A group of n students will implement n of the following learning algorithms, which will be explained in more detail farther below:

1. Linear machines, which is perceptron learning of linear discriminant functions.
2. Perceptron learning using $+1$ and -1 thresholds instead of the 0 threshold.
3. Classification noise perceptron.
4. Adaline least-mean-square algorithm for classification (method 4 in the book).
5. Adaline version of a linear machine.

For each learning algorithm that you implement, also implement a batch version of the algorithm. Add a “-b” option to the command line to select batch learning. Batch learning is implemented by calling `update_neuron` only once at the end of each epoch. The learning rate should be divided by the number of examples (or, for somewhat faster convergence, the number of times that `backprop_neuron` is called).

For extra credit, when batch learning is not selected, you should randomize the order of the examples before each epoch.

In addition, a group of n students will download n classification datasets and test the algorithms on these datasets. The Neural Network FAQ linked from the class web site describes how to find additional datasets. The datasets will need to be formatted so that the `read_dataset` function can load them. The average size should be at least 1000 examples and at least 10 inputs. You will need multiclass (more than 2 classes) datasets to properly test linear machines.

Testing will be performed using the holdout method. The holdout program from the initial source code will partition a dataset into a training set and a test set. An option on the command line will run the learned network on the test set. Perform the testing using 100 epochs and 1000 epochs on both incremental and batch learning. Also include the `iris.data` dataset (available on the web site) in your testing.

Learning Algorithms

This describes the learning algorithms listed above in more detail. As in class, \mathbf{x} is a vector of inputs and \mathbf{d} is a vector of desired outputs. Let X be the maximum length of any input vector. There will be as many neurons as outputs. For a given neuron, u is $b + \sum_i w_i x_i$, $o = a(u)$ is the output where a is the activation function, and d is the desired output. I will assume d is either +1 or -1. For datasets that use 1 and 0 as outputs, the 0 should be treated the same as a -1.

Linear Machines

‘ To implement the idea of discriminant functions, the neuron with the maximum u value is assigned an output of 1, while all other neurons are assigned an output of -1. After this assignment, the perceptron learning rule is applied. I suggest defining a procedure `network_run` to call `run_neuron` on the neurons and then perform this assignment. If the dataset has only one output, this should behave the same as the perceptron algorithm.

Perceptrons with +1 and -1 Thresholds

For this type of perceptron, we want $u \geq 1$ when $d = -1$ and $u \leq -1$ when $d = 1$. To implement this, define an activation function called `ramp` such that:

$$o = \text{ramp}(u) = \begin{cases} 1 & \text{if } u \geq 1 \\ u & \text{if } -1 < u < 1 \\ -1 & \text{if } u \leq -1 \end{cases}$$

If $\text{ramp}(u) \neq d$, then `backprop_neuron` should be called with `dout=-d`. The loss is 0 if $\text{ramp}(u) = d$; otherwise, the loss is $(|d - u|)/2$. The 0-1 loss is $|(d - \text{ramp}(u))/2|$. The learning rate should be less than $1/X^2$.

Classification Noise Perceptron

Let \mathbf{z} be the “average example”, calculated as:

$$\mathbf{z} = \frac{1}{m} \sum_i d_i \mathbf{x}_i$$

You will also need the “average bias”:

$$\mu_b = \frac{1}{m} \sum_i d_i$$

Note that each neuron will have a different average example and average bias. The sigmoid activation function is used. The learning rule is:

```

if  $d \neq o$ 
then  $u \leftarrow b + \sum_i w_i x_i$ 
       $v \leftarrow \mu_b + \sum_i w_i z_i$ 
      if  $v \leq 0$ 
        then  $a \leftarrow 10$ 
      else if  $d * u \geq 0$ 
        then  $a \leftarrow 0$ 
      else  $a \leftarrow \min(10, -d * u/v)$ 
       $\mathbf{w} \leftarrow \mathbf{w} + \eta(dx + az)$ 
       $b \leftarrow b + \eta(d + a\mu_b)$ 

```

Adding in the average example and average bias is intended to counterbalance the noise in the examples. The factor 10 is an arbitrary value to avoid multiplying the average example times an overly large number.

LMS Algorithm for Classification

This is similar to Adaline Method 4 in the book, but in this case, if $d * o \geq 1$, then no changes will be made to the weights. Let the identity function $o = u$ be the default activation function. The learning rule is:

```

if  $d * o < 1$ 
  then  $\mathbf{w} \leftarrow \mathbf{w} + \eta(d - o)\mathbf{x}$ 
         $b \leftarrow b + \eta(d - o)$ 

```

The learning rate should be less than $1/X^2$. The loss should be calculated as 0 if $d * o \geq 1$ else $(d - o)^2$. The 0-1 loss can be calculated as $|(d - \text{ramp}(o))/2|$. See the definition of the ramp function above.

LMS Linear Machine

This is similar to the Perceptron linear machine described earlier, but the output will be defined in this way. Let o be the output of the desired class, i.e., where $d = 1$. Let o' be the highest output of the other neurons, all of which have $d = -1$. Now subtract $(o + o')/2$ from all the neurons' outputs. Apply the LMS learning rule above to the modified outputs.

Submission

To submit your project, email it to cs4793@ai.cs.utsa.edu. Your source code should be attached as a tar or zip file. Do not include object files or executable files in your submission. I should be able to use "make all" to compile and link your code. You should also include a report indicating what algorithms you implemented and displaying your test results. Describe any design decisions that you made and how you implemented them.