CPSC 420-500: Program 3, Perceptron and Backpropagation

Yoonsuck Choe
Department of Computer Science
Texas A&M University

October 31, 2008

1 Overview

You will implement perceptron learning from scratch (see section 3 for details), and train it on AND, OR, and XOR functions. Then, you will take an existing backpropagation code (see section 4), train it and test it under different conditions and report your findings. The same three boolean functions will be used for the backpropagation learning. For further details on perceptrons and backpropagation, see the lecture slides and also Hertz et al. (1991). Specific submission instruction will be given in section 5 and section 6.

2 Language and OS

You may use either C/C++, Java, Matlab (or Octave), or Lisp. The resulting code should be able to compile and run on the departmental unix host (unix.cs.tamu.edu). You may use a different language with a permission from the instructor, in which case you will be asked to do a demo in front of the TA.

To compile your programs other than Lisp (which you already know), see the following instructions.

- C program file: **perceptron.c**
  to compile: **cc -o perceptron perceptron.c -lm**
  to execute: **./perceptron**
- C++ program file: **perceptron.C**
  to compile: **c++ -o perceptron perceptron.C -lm**
  to execute: **./perceptron**

- Java program file: **perceptron.java**
  to compile: **javac perceptron.java**
  to execute: **java perceptron**

The full paths for the compilers are (on compute.cs.tamu.edu):

- /usr/bin/cc
- /opt/csw/gcc3/bin/g++
- /usr/jdk/latest/bin/java
- /usr/jdk/latest/bin/javac

3 Perceptron

Perceptron activation is defined as:

$$Output = step\left(\sum_{i=0}^{2} W[i] * INP[i]\right), \tag{1}$$

where $step(X) = 1$ if $X \geq 0$ and $step(X) = 0$ if $X < 0$ (see figure 1).

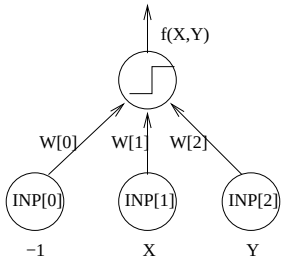

Figure 1: **Perceptron.** The input $INP[0]$ is the *bias unit*, fixed to -1, and has an associated weight $W[0]$, which is the threshold. The two inputs X and Y are given and the output $f(X, Y)$ will be calculating (or attempting to calculate) a boolean function, one of OR, AND, or XOR.

Implement a perceptron with two input units (three, including the bias unit that has a fixed input value -1) and one output unit. Your program should take three inputs from the command line.

1. maximum number of epochs to run (integer),
2. learning rate parameter α value (double),
3. function selection string (“and”, “or”, and “xor”).

For example, a typical run would go like this (`$` is the unix prompt):

```
$ ./perceptron 10000 0.0001 and
```

A pseudo code for perceptron learning is as follows:

1. Initialize weights to random numbers between 0.0 and 1.0. Note that you have just one set of 3 weights ($W[0], W[1], W[2]$). You will train this same set using the four input-target pairs.
2. Initialize epoch count to 0.
3. while sum of $(target - output)^2$ for all input patterns is not 0 do:
   - for each input-target pattern
     (a) present input and calculate output
     (b) calculate the $error = target - output$
     (c) update the weights $W[i]$ using the perceptron learning rule.
   - endfor
   - increment epoch count
   - if (epoch count $>$ max epochs), break from while loop
4. Print out the output for the inputs (0,0), (0,1), (1,0), and (1,1).

4 Backpropagation

For backpropagation, you will download the following file (make it one line):

```
http://faculty.cs.tamu.edu/choe/
src/backprop-1.6.tar.gz
```

and run it under different conditions. First, you need to unzip and untar it by running the following:

```
$ tar xzvf backprop-1.6.tar.gz
$ cd backprop
```

then read the `README` file to learn how to compile and run it.

Running the `bp` program (which is generated by compiling) will give you a huge dump on the screen. To selectively view the data you're more interested in, use the `grep` command. For example, to view the progression of error:

```
$ ./bp conf/xor.conf | grep ERR
```

and to view the actual output values for the inputs:

```
$ ./bp conf/xor.conf | grep OUT
```

5 Assignment

This section will detail what you actually have to do and have to submit. All projects should be turned in using the `csnet turnin`.

5.1 Perceptron

Implement perceptron learning algorithm as detailed in section 3, and with the program conduct the following experiments, and submit the required material, along with the code and the `README` file as usual.

Experiments:

1. Test AND, OR, and XOR for learning rates $\alpha = 0.001$, and 0.0001. For each Boolean function, run the experiment with different initial random weights (use the random number generator function to do this) two times. Discard all runs that ended in 1 epoch (you will see several of these: why would they occur?). The total number of trial will thus be 2 learning rates $\times$ 2 random initial weights $\times$ 3 functions to learn = 12. Set the max epoch to 10000 for all trials.
2. For each trial, report the following in the README file:
   (a) initial weights, and show the plot of the decision boundary (the straight line defined by the weights).
   (b) final weights, and show the plot of the decision boundary.
   (c) number of epochs taken to complete training if successful (let us call this n), and
   (d) for each epoch, the sum of squared error for all four input patterns.
3. Answer these questions in the README file:
   (a) Do you think perceptron will be able to learn the boolean function $f(X, Y) = \neg(X \vee Y)$? What do you think is the role of the *sign* of the weights in this case? Think about the geometric interpretation in that case.

Table 1: **Boolean Function $f(X, Y) = \neg(X \vee Y)$.**

X	Y	$\neg(X \vee Y)$
0	0	1
0	1	0
1	0	0
1	1	0

5.2 Backpropagation

With the provided code, conduct experiments on AND, OR, and XOR. Note that in the `bp.cc` code, learning rate α is a named `eta`, just in case you want to take a look inside the code.

Experiments:

1. Test AND, OR, and XOR for learning rates $\alpha = 0.01$ and 0.001, and plot the sum of squared error for each trial (a total of 6 trials). A total of 3 set of plots (for AND, OR, and XOR), with each set containing 2 curves (for two α's) is required. Save the plots in image files and name them `and1.jpg, and2.jpg, or1.jpg, or2.jpg, xor1.jpg, xor1.jpg ...`. Use `grep` to extract the error values (see below), and load that file (`dump.txt`) in a spreadsheet.

   ```
   $ ./bp conf/xor.conf | grep ERR > dump.txt
   ```

2. With learning rate $\alpha = 0.01$, test AND, OR, and XOR, with 1, 2, and 4 hidden units, Plot the sum of squared error for each trial. A total of 3 set of plots (for AND, OR, and XOR), with each set containing 3 curves (for three different number of hidden units) is required. Save the plots in image files and name them `bp-and1.jpg, bp-and2.jpg, bp-and3.jpg, ....`

3. For all trials above, count the number of epochs until the end is reached, and measure the time taken using the `timex` unix command (add user time and sys time):

   ```
   timex ./bp conf/xor.conf
   ```

   Report the number of epochs and time spent for each trial in the README file. How many number of hidden units was the best in your opinion and why?

6 Submission Details

The **due date** is 12/4/2008, 11:59pm. Grading criteria will be similar to the previous programming assignments. You must submit the following:

- source code,
- compiled executable binary,
- README file containing material detailed in section 5, and
- plots (image files; include a list of image files and a brief description of each in the README file).
- put everything in a single zip or tar.gz file.

Only electronic submission through CSNET will be accepted.

References

Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Reading, MA: Addison-Wesley.