UMEÅ UNIVERSITY **February 14, 2010**
Institution of Computer Science
Lab report

Lab 3

Artificial Intelligence 7.5p

Happy, Sad, Mischevious or Mad?

Name	Eric Jönsson
E-mail	`ericj@cs.umu.se`
Path	`~ericj/edu/ai/lab3`

Graders
Michael Minock and Johan Granberg

Contents

1 Lab specification

Single-layered feed-forward neural networks (or *perceptron* for short) presents an interesting way of classifying information. For this assignment, the student was to write a perceptron that given a series of sketches of faces judge whether these faces looks happy, sad, mischevious or mad.

The complete specification can be downloaded from the following adress:

`http://www8.cs.umu.se/kurser/5DV019/HT09/happy.pdf`

2 File access and program manual

2.1 File overview

All files related to the assignment can be found at the following path:

`~ericj/edu/ai/lab3`

This directory contains the following files and directories:

- An `ant` build file, `build.xml`.
- Two shellscripts for internal testing, `check.sh` and `masscheck.sh`.
- Another shellscript for testing by the grader, `classify`.
- A training set, consisting of the files `training.dat` and `correct.dat`.
- A small test set taken from the training set, `test.dat`.

Furthermore, the source code is contained in the `src` subdirectory while this report is found in the `doc` directory.

2.2 Compilation

To compile the program, simply type `ant` (or `ant jar`), like this:

```
bilbo:~/edu/ai/lab3> ant
Buildfile: build.xml

compile:
    [mkdir] Created dir: /home/ericj/edu/ai/lab3/bin
    [javac] Compiling 6 source files to /home/ericj/edu/ai/lab3/bin

jar:
      [jar] Building jar: /home/ericj/edu/ai/lab3/Faces.jar

BUILD SUCCESSFUL
Total time: 1 second
bilbo:~/edu/ai/lab3>
```

This results in an executable Java archive, `Faces.jar`.

2.3 Running the program

There are two modes to this program: training and guessing. To train the perceptron on the default training set, type `ant train`:

```
bilbo:~/edu/ai/lab3> ant train
Buildfile: build.xml

compile:

jar:

train:
     [java] Perceptron trained. Bring it on!

BUILD SUCCESSFUL
Total time: 2 seconds
bilbo:~/edu/ai/lab3>
```

Alternatively, if you'd like to specify the training set yourself, run Java like this:

```
bilbo:~/edu/ai/lab3> java -jar Faces.jar train training.dat correct.dat
```

..where `training.dat` and `correct.dat` is your training set with corresponding solutions. This creates a file, `matrix.dat`, containing the weight matrix for the perceptron. To have the perceptron start guessing, run the `classify` shellscript, like this:

```
bilbo:~/edu/ai/lab3> ./classify test.dat
Image1 2
...
Image9 2
Image10 3
bilbo:~/edu/ai/lab3>
```

Remember to train the system before attempting classification: if the `matrix.dat` file doesn't exist, you'll get an error:

```
bilbo:~/edu/ai/lab3> ./classify test.dat
Failed reading file.
bilbo:~/edu/ai/lab3>
```

3 System description

This implementation reads the input files, extracts the relevant information and builds a perceptron with a sufficient amount of input- and output nodes[1]. This resulting neural net is a complete bipartite graph[2], and has therefore internally

---

[1] For this assignment, this means 400 input- and 4 output nodes.

[2] This basically means that every input node is connected to every output node. (and vice versa)

been represented by a $n * m$ matrix consisting of the link weights. This relation is illustrated in figure 1.

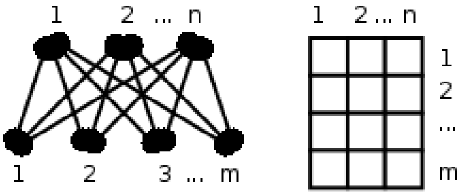

Figure 1: A $n * m$ complete bipartite graph and its corresponding weight matrix

When constructed, all link weights are initialized to 1. These are then changed in accordance to a supervised training algorithm based on gradient-descent: the input is pushed through the sigmoid (squashing) function $g(x) = \frac{1}{1+e^{-x}}$ which happens to have the convenient range $0 < g(x) < 1$. The link weights are then updated according to the relative error, based on the sigmoid functions derivative $g'(x) = g(1 - g(x))$ – the general idea being that the smaller the relative error becomes, the smaller the changes to the link weights. This procedure is then repeated until the perceptron starts guessing right, expressed as the sum of total link changes in each iteration.

4 Test runs

To accurately test the system, two shellscripts were used. The script `check.sh` compares the output of the program with the correct file and counts the number of errors. Below is a sample output.

```
bilbo:~/edu/ai/lab3> ./check.sh Faces.jar training.dat \
  correct.dat test.dat
Errors: 0/10
bilbo:~/edu/ai/lab3>
```

To get an idea of how well the system performs on a larger set of test data, the script `masscheck.sh` was used. It runs for a number of iterations, repeatedly training and re-training the perceptron, generating new weight matrices every time. Each time, the script checks the output of `check.sh` to count the number of errors, finally presenting an average value.

Below is parts of the script output using the original training set as test set and running for a thousand iterations.

```
bilbo:~/edu/ai/lab3> ./masscheck.sh Faces.jar training.dat \
  correct.dat training.dat 1000
Test 1/1000 complete
...
Test 998/1000 complete
Test 999/1000 complete
Test 1000/1000 complete
Average: 12.0/400
bilbo:~/edu/ai/lab3>
```

Producing on average twelve errors per four-hundred faces, the perceptron has an accuracy of about 97% on the given training set.

5 Issues

How well would the perceptron perform when given a face not included in the training set? Probably not very well. Emanuel Dohi, a graduate student of the CS department, reminded me of the dangers of over-training – tuning the perceptron to such a high degree that it doesn't only pick up on the obvious features of the faces, but the individual errors in the training set as well. The test runs in section 4 clearly demonstrate that the system is horribly, horribly over-trained.

To amend this and perhaps make the perceptron a little bit more suitable for real-world problems, only a single parameter in the source code needs to be modified – the limit for weight changes before the perceptron considers itself fully trained. Emanuel Dohi suggested that my perceptron should guess correctly no more often than 80% of the time on the training set to be able to react decently to other input as well, but I've chosen to ignore it for now, saving it for another exercise.