Assignment 5: Perceptron learning

Feb, 2019

1 Submission

- Due date:
- Deliverable: perceptron.cpp
- Submit on Canvas

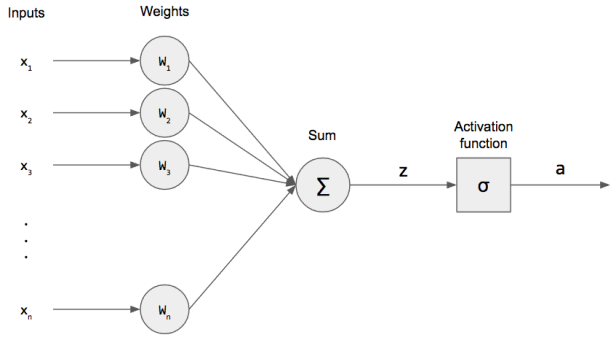
2 Goal

In this assignment, you are asked to implement a basic neural network, "Perceptron", to solve the linear classification problem.

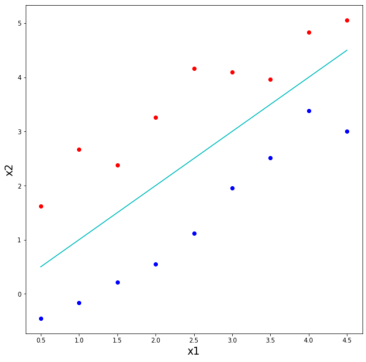
3 Problem Description

3.1 Perceptron

Perceptron is the most simplest feed-forward artificial neural network. Its structure only contains one node (fig 1a).



(a) Perceptron



(b) Linearly separate two types of data. Say the line function si $ax_1 + bx_2 + c = 0$

Perceptron is a linear classifier. It's like to use a line to separate two types of data (fig 1b). The core of it is a linear function:

$$y = \sum_{i=1}^{n} w_i x_i + b$$

Furthermore, if we add an extra $x_0 = 1$ term, we can use $w_0 = b$ and remove b from the function:

$$y = \sum_{i=0}^{n} w_i x_i$$

However, using this function will only produce a straght line (or a surface if the data is more than 2D). We need another function called "activation" function to transfer the result to binary output ("0" and "1"), for example using a step function:

$$f_{threshold}(\theta) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

Therefore, a perceptron can be written as the following function:

$$f(\sum_{i=0}^{n} w_i x_i) = \begin{cases} 1, & \sum_{i=0}^{n} w_i x_i > 0, \\ 0, & \text{otherwise} \end{cases}$$

while f is the activation function.

3.2 Perceptron Learning

Suppose the input $\vec{x}_j$ is an n-dimensional input vector in the dataset D, where:

$$D = \{(\vec{x}_0, y_0), (\vec{x}_1, y_1), ..., (\vec{x}_m, y_m)\}$$

$j = 0, ..., m$ say:

- D is the training set which size is m (m number of examples), where

  $$D = \{(\vec{x}_0, y_0), (\vec{x}_1, y_1), ..., (\vec{x}_m, y_m)\}$$

- $\vec{x}_j$ is the j-th example in D
- y_j is the label for the j-th example $\vec{x}_j$, and $y_j = 0$ or 1.
- $x_{i,j}$ is the i-th element in the input vector $\vec{x}_j$.
- w_i is the i-th weight.
- $N(\vec{x}_j)$ is the output (0 or 1) of the perceptron when receiving $\vec{x}_j$.
- α is the learning rate, and $0 < \alpha < 1$

The steps of the algorithm are:

1. Initialize the weights, as well as the bias (w_0). Can be zeros or some small random numbers.

2. For each example $\vec{x_j}$ in the training set D, Calculate the output of the network:

$$N(\vec{x}_j) = f(\mathbf{w}\dot{\vec{x}}_j) = f(\sum_{i=0}^{n} w_i x_{i,j})$$

   where f is the threshold function as shown above.

3. Update each weight by:

$$w_i \leftarrow w_i + \alpha(y_j - N(\vec{x}_j))x_{i,j} \quad (i = 0, ..., n)$$

   for $i = 0, ..., n$. Note that w_0 is the bias and x_0, j is 1.

4. We call it "one **epoch**" when using all the examples once in the same dataset D. Keep doing step 2 and 3 until the absolute error $\sum_{j=1}^{m} |y_j - N(\vec{x}_j)| = 0$, or your program runs the maximum number of epochs. In this assignment, the maximum number of epochs is 50.

3.2.1 Testing

After training, the perceptron can be used to separate the dataset. In this stage, it takes the input for a testing dataset, and produce 1 or 0 for each data point, but the weights will not be changed anymore. Then you can compare the output of your perceptron with the ground truth to see the accuracy.

3.2.2 Hint for the Implementation

In fact, a perceptron is a function. It takes $\vec{x}$ as input and produces 0 or 1. The weights $\mathbf{w} = \{w_0, w_1, ..., w_n\}$ are the parameters of this function. The learning algorithm will keep changing the weights until the error is zero (no weight needs to be updated).

3.3 Input and Output

3.3.1 Input

This is a neural network, so you will receive two inputs: one is for training, another is for testing. Your program reads the input from files by the following command:

```
$ ./perceptron train.txt test.txt
```

In the training input file, the first line will show the number of the input vectors. Following that number, each line is an input vector. Since the input vector is 4D and the label of each input is 0 or 1, each line will contains $4 + 1 = 5$ numbers. Here is an example:

```
4
0 6 5 0 1
5 2 2 9 1
3 5 0 0 0
9 4 1 5 0
```

The testing input file is similar, but no labels. For example:

```
4
1 2 3 4
2 2 3 3
1 1 4 4
4 4 8 9
```

For your convenience, **all the numbers in the both training and testing sets are integers.** However, you should use double for the weights. Your program will use the first input file for training, and the second input file for testing.

3.3.2 Output

The output of your program is a sequence of the labels corresponding to the test data points (lines), like this:

```
$ ./perceptron train.txt test.txt
0 0 1 1
```

That means the first data point in the testing set belongs to type "0", the second data point belongs to type "0", etc.

Again, all the inputs are integers. You will only use the first input file to train your perceptron. DO NOT USE BOTH FOR TRAINING! Or you will be deducted points for that.

4 Samples

You can find these two examples in `http://my.fit.edu/~hchang2014/Cpp/Assignment5/`. Right click the file link then choose "save as..." to get the input files.

I strongly suggest you to generate your own dataset to train your perceptron (to test your program). A simple way to generate similar dataset is:

1. Initial a linear function:

$$f(\vec{x}) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$$

   Choose any 5 numbers to be the weights **w**.

2. Generate some 4D vectors randomly, calculate the result by $w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4$. If the result is greater than zero, then it is a positive example, otherwise it is a negative example.

3. Use these data to train your perceptron and see if the weights are the same as you think.

4.1 Sample 1

```
./perceptron train1.txt test1.txt
0 1 1 0
```

4.2 Sample 2

```
./perceptron train2.txt test2.txt
0 1 0 1 0
```

5 Important Notes

- Submit your source code. No other files are needed for the assignment (or graded!).
- Make sure your code runs on our server (code01.fit.edu). Compile command:

```
g++ -std=c++11 perceptron.cpp -o perceptron
```

If you don't know how to compile and run your program on the server (code01.fit.edu), please ask.

6 Rubric

- **Logic(50%)**: Correct and satisfy our requirements ((use recursion to solve this problem).
- **Test cases(30%)**: Pass all the test cases. Some of them are public to all (Samples), but some of them are hidden. So test your program with more cases, not just the sample cases.
- **Compile/Runtime(10%)**: No compile or runtime errors. If your program cannot compile or run, you may lose all the points not only compile/runtime, but also the test cases.
- **Documentation(10%)**: Enough comments for the readability.