SCHOOL OF SOFTWARE ENGINEERING OF USTC

SECOND SEMESTER, 2012–2013
Campus: Hefei

DATA-MINING

Multi–Layer Perceptron

(Time allowed: TWO hours)

The objective of this experiment class is to implement a Multi–Layer Perceptron, using the primitives from the first experiment class We will implement the Multi–Layer Perceptron as one, single class.

You can do your work in *C++* or *Java*. You can also use *C*, but it will be probably more work, maybe more than you can handle. *Python*, *R* or *Matlab* are **NOT** allowed. You can work on either Linux or Windows, and whatever compiler you wish, as long as your code is portable. I expect clean, readable code : each class in separate files, for example.

You have to work alone, one student per computer, this is non-negociable. It is to help you to be focused and practice by yourself. If you are stuck after trying various ways, you can call for help. You might have to patient while the teacher is busy helping someone else.

1. Create a class *MLP*, with two parameters, N and K for the constructor. N is dimensions of the points to process. K is the number of hidden neurons.

We want the following class members

- V : K *Vector* instances, with N coefficients each.
- v_n : a *Vector* instance, with K coefficients.
- W : a *Vector* instance, with K coefficients.
- w_n : a double value.
- Y : a *Vector* instance, with K coefficients : the output for the K hyperplanes.
- Y_p : a *Vector* instance, with K coefficients : the derivative of the output for the K hyperplanes.
- z : a double value : the output of the MLP.
- z_p : a double value : the derivative of the output of the MLP.
- E : a *Vector* instance, with K coefficients : the e term from the slides, used by the backpropagation algorithm.
- *eta* : the learning rate for the stochastic gradient descent.

V and V_n are the $N + 1$ coefficients of the K hyperplanes. W and w_n are the $K + 1$ coefficients of the affine combinations of the K hyperplanes. You might have those class members private, and provide read-only accessors functions.

2. Implement a private member function *void set_hyperplane_weights(size_t i, Vector& C, const Vector& N)*, that set V and v_n for the i-th hyperplane, such as it represents an hyperplane of origin C and normal N.

3. Implement a private member function *void forward(const Vector& X) const*, that computes the *forward* pass of the backpropagation algorithm, with X as input. This function should update Y, Y_p, z, z_p.

CONTINUED

4. Implement a private member function *void backward(const Vector& X, double c) const*, that computes the *backward* pass of the backpropagation algorithm, with X as input and as desired output c. This function should update W, w_n, V, v_n, E.

5. Implement a public member function *bool classify(const Vector& X) const*, that returns *true* if the output of the MLP for X is positive, *false* if the output of the MLP for X is negative.

6. Implement a public member function *void update(const Vector& X, bool inClass)*, that updates W, w_n, V, v_n *update_weights* implements one iteration of the backpropagation algorithm. We assume that we use the tanh transfer function. The derivative of tanh is $1 - \tanh^2$.

7. Write a public member function *reset*, that initialize the classifier, by calling *set_hyperplane_weights* with C being a null vector, and N a random unit vector (one different for each hyperplane). Do not forget to initialize the weights W and w_n

8. Write a simple test function, to check that your code works properly. If you implemented correctly the previous steps, you should be able to classify

x	y	*label*
-1	1	*true*
1	-1	*true*
-1	-1	*false*
1	1	*false*

9. Note how much iterations you need to classify correctly the previous example, without changing the learning rate. Now try with this How should you modify the initialization to have the same learning speed as the

x	y	*label*
100	102	*true*
102	100	*true*
100	100	*false*
102	102	*false*

previous example ?

10. You are now ready to try your MLP on a real dataset. Write a main function that does the following things

(a) Load a dataset (use the same as previous experiments), with the function *read_data*

(b) Use *MLP* to classify it. The last element of each row of the dataset is the class of the point : 1 or 0.

(c) When the learning is completed, shows the percentage of correctly classified points.

(d) Output the dataset in a file. The last element of each row of the dataset is the class decided by your classifier.

How are you going to determine that the learning is completed ?