Application of MLP

Erignoux Laurent

April 4, 2012

0.1 Description of the program

The homework consisted to make a program in order to apply a back-propagation algorithm both in normal and batch mode to a multilayer neural network.

Execution :

The project needs a C++ compiler, the makefile is included in the archive. I made an input image usable in Bitmap according to the file given for the homework.

then the program can be used without any argument, just make sure the input image is in the folder.

0.1.1 algrothme choices

activation function :

$$\phi = 1,7159.tanh(x.2/3);$$

$$\frac{d\phi}{dx} = 1,7159.tanh(x.2/3);$$

Initializations : each weight (link or bias) is initialized randomly between -0,5 and 0,5

Scale :

- input data are normalized in [-1,1]
- output expectations are +1, -1

0.1.2 Main

The main class merely instantiate and execute these classes.

The program is based on 4 main classes :

0.1.3 Neuron

The neuron class : describe a Neuron which as a number of parents (links to his parents) and children (links to his children). It's characteristics are : an activation function : here a*tanh(b*x) and it's derivative.

Main Variables :

- parents links : the link to each parents
- children links : the links to each child
- nbParents : number of parents
- nbChildren : number of children
- bias : implement the bias of each neuron

Main functions :

- activationFunction : implementation of the activation function
- process the signal : apply the activation function to the local field
- addSignal : add a signal to the local field
- setGradient : calculate the gradient of the neuron

0.1.4 Link

The link class : link a parent and a child neuron. It is described by a random initialized weight

The link class is extended in three different class : Starting link, End link, and double link

1. starting link : doesn't have a father
2. end link : doesn't have any child, the signal is transmitted to the learning processor
3. double link : standard link linking two neurons

Main Variables :

- parentNeuron : the parent neuron of the link
- childNeuron : the child neuron of the link

Main functions :

- transmit signal : transmit the signal to it's child
- changeWeight : change the link weight

0.1.5 Layer

The layer class : is a set of a fixed number of neuron.

Main Variables :

- Neurons : list of the neurons in the layers
- nbNeurons : number of neurons in the layer

Main functions :

- linkSubLayer : link a subLayer to himself (the father is always initiator)
- process signal : process the signal of each Neuron in the layer

0.1.6 learning Processor

the learning Processor class, this class manage the learning process.

Main Variables :

- learning rate
- a stopping criteria (average error accepted)
- batchMode : describe the learning back-propagation mode.

Main functions :

- processResult : This function get the output signal of the network and propagate it backward in the layers to make the learning process.
- batchProcess : process the final error in batch mode

0.1.7 Data

the Data class : contain the data on an Double array format. Data can be read from a Bitmap image and written on a bitmap image.

It is instantiated with the Bitmap image filename that needs to be stored.

0.2 Working

0.2.1 creation of the network

The first step is the creation of the multi Layer Perceptron. we start by creating the input and output links which respectively get the input signal and return the output to the learning Processor. The each Layer is created according to the number of hidden layer asked and the number of Neuron in each layer.

Once created the layers are connected together : each possible link between consecutive layers is made.

Then the learning manager is started, it is basically linked to the last Neuron link to be able to get the output and treat the information to initiate and propagate the backward error signal.

0.2.2 transmitting signal

One the network created the signals can be sent to the input links. This is done in the main class, data is read and transmitted to input links. once stored the layers are processed successively to transmit the signal to the next layer. The signal reach the end and the learning process start

0.2.3 learning

Once an output signal is obtained, it is transmitted to the learning Processor. If learning is necessary, it calculate the error and transmit required parameters to the last neuron to adapt it's weights (his parents ones). then layers are processed backward to process the learning.

learning is pursued until an average error of 0.001 is achieved (or 100 iterations are processed to avoid infinite loops.)

Learning process :

normal mode

the learning processor get an output signal from the last neuron. it calculate it's gradient (currentError*currentNeuron-¿dActivationFunction(currentNeuron-¿getLocalField());)

then make the last links change their respective weight.

then it loops back the layers and ask to each neuron to set his gradient and then change it's parents weights.

once to the first layer, neurons are cleared of their last local fields and gradients

batch mode

A batch mode can also be asked when inputed. In that mode the data is processed in series of 20 inputs. At the end of the 20 input processing, the global error is calculated and transmitted to the learning processor to update the weights in the layers.

0.2.4 testing

Once the network is created and as learned a first batch of data, it is tested to create a mapping of a 2D picture. Basically every Image bit is transmitted to the network and the output is mapped in two colors depending if it is positive or negative. Thus a mapping of the whole image can be obtained.

The result is stocked in an Array and the converted to a bitmap Image to get a more accessible result

0.3 Nonworking

If the stopping criterion is too low, the program diverge and weights tend to infinity, this is an error I have been unable to solve in spite of much time spent and a comparison to the class presentation and on-line resources. It the learning process even if it shows taking account of th error and modifying the inputs doesn't seems to produce relevant results.

Indeed no separation occurs between the different links. I didn't found the cause of this problem in spite of much time spent on it and much documentation to check that my code was correct.

The program currently output a void image, both learning are unsuccessful, no matter the size of the hidden layers and the number of hidden layers.