BULETINUL Universităţii Petrol – Gaze din Ploieşti	Vol. LXI No. 3/2009	371 - 376	Seria Tehnică

A Robust Method for Object Classification Based on Back-Propagation Neural Networks

Bogdan Vasilescu

Petroleum – Gas University of Ploieşti, 39 Bucureşti Blvd., Ploieşti, ROMÂNIA
e-mail: vasilescu@gmail.com

Abstract

This paper will introduce the principles of training multi-layer perceptron neural networks and the application of such a topology to simple-object classification. The neural network is trained with the feed-forward Back-Propagation Network (BPN) algorithm. Finally, the paper presents the influence of the learning rate and hidden-layer neurons on the behaviour of the network for a specific classification problem.

Key words: *neural networks, back-propagation, object classification.*

Introduction

An artificial neural network (ANN) is a computational model inspired by the way biological nervous systems process information. It consists of a large number of highly interconnected artificial neurons working together to solve specific problems [8]. Moreover, neural networks can be regarded as non-linear statistical data modelling tools used to model complex relationships between inputs and outputs or to find patterns in data, through a learning process [6]. Similar to biological systems, learning in ANNs involves adjustments to the synaptic connections that exist between the neurons.

The tremendous interest in neural networks over the last years can be attributed to a few key factors [7]. First, ANNs are very powerful modelling techniques applicable to extremely complex functions. In particular, neural networks are nonlinear and therefore suitable for modelling domains where linear approximations are not valid. Second, neural networks are much easier to use than traditional nonlinear statistical methods since they learn by example. The neural network user gathers representative data, and then invokes training algorithms to automatically learn the structure of the data.

One of the most important aspects of working with neural networks is choosing an appropriate topology for a specific requirement [1]. For example, pattern recognition problems typically implement a multi-layer perceptron topology with back-propagation, which has been trained accordingly [3]. The training consists of feeding a set of inputs to the network and associating the output patterns to the inputs, through a repetitive learning process. When the network is used, it tries to identify an unknown input that has no output associated with it.

Consequently, this paper uses a three-layer back-propagation neural network in order to study the behaviour of such a topology for a simple-object classification problem.

The given problem and neural network topology

An important application of neural networks is pattern recognition [2]. The current paper focuses on a basic object classification problem, which is common in video surveillance applications that require motion detection, face recognition and object separation, based on the images captured through a video sensor. This is a very delicate issue because the objects to be classified could be situated at any distance from the fixed image sensor, and could also be randomly rotated with regard to it.

A typical solution consists of developing a software implementation of a multi-layer back-propagation neural network that can be trained with a set of input examples in order to determine, with enough certainty, if a given object is part of a specific class or not. The author of this paper chose to develop a Borland C++ Builder application in order to assess the impact of two training parameters on the behaviour of the network.

For demonstration purposes, two classes of simple objects have been chosen, namely circle-type objects (responsible being the first output neuron, neuron "0"), and square-type objects (responsible being the second output neuron, neuron "1"). The purpose of the designed neural network is to classify an input object into one of the two given categories.

In order to solve the proposed problem, the author chose a three-layer back-propagation neural network, having two neurons on the output layer, corresponding to the two possible classes the analyzed object could belong to.

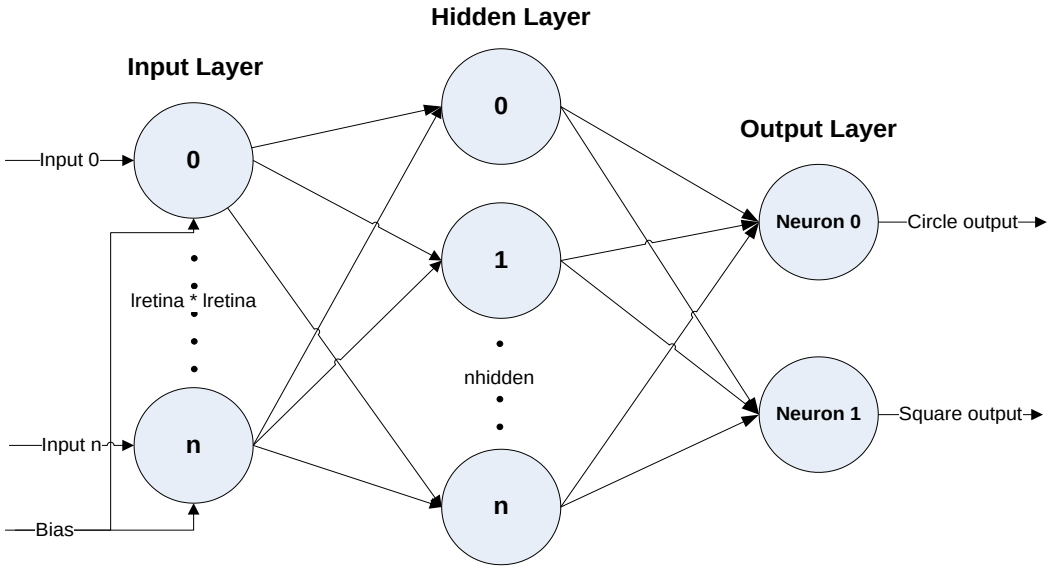


Fig. 1. The topology of the designed neural network.

The number of neurons from the input layer was determined by the size of the “retina”. In this case, for computational speed considerations, the retina is assumed to capture 50x50 pixel images, and therefore the number of input neurons is 2500.

In order for the perceptrons to function properly, a “bias” input was also incorporated, with a constant value of 1. The bias input is necessary because without it, in case all of the inputs are 0, the only output possible is a zero.

The sample training objects are represented by black and white images. However, colour sample images can also be used since the implemented training process automatically transforms the input images into greyscale.

Fig. 2. Sample images training database: a) Circle-type objects; b) Square-type objects.

The sample images training database consists of 50 images of circle-type objects and 50 images of square-type objects, such as the ones presented in Figure 2, covering the stretch/skew/rotate range as much as possible.

Training and fine-tuning the network

The neural network presented above is not of any interest because it has not been trained (so it is not able to solve any particular problem). The training methodology [9] consists of feeding the network with a set of input numbers. Consequently, the network will give a result in its output layer. Since the weights of the connections in the network are initially in a random state, this result will surely be unsatisfactory in the beginning, so adjustments are made to the weights of some of the connections in order to obtain better results.

Next, the input layer of the network is fed with other examples and the process of adjusting weights continues until eventually the desired output is obtained for each example. The entire set of training examples must be shown to the network many times in order to get a satisfactory result. After all this training, the network has learned to solve the problem and its knowledge is stored by all the different connection weights. For this particular example, the author chose the back-propagation training algorithm, well suited for pattern recognition problems because of its simplicity and reasonable speed.

In order to get the network up and running, the author performed the following operations when developing the Borland C++ application:

- Initialisation, which can be achieved with a C++ code sequence such as the following.

```
network = new bpnet(layers, lretina*lretina, nhidden, LabeledEdit6->
      Text.ToInt(), bias, CheckBox3->Checked, learning_rate);
```

- Providing a set of example input values and desired output values (to train the network), repeatedly cycling through the training data. Training is performed via the C++ `Train()` method, shown below.

```
void bpnet::Train(const double *inputs, const double *outputs)
{
      Evaluate(inputs);
      for (int i=nlayers-1; i>=0; i--) layers[i]->CalcDelta(outputs);
        if (adaptive_learning_rate) RecalcLearningRate();
      for (int i=nlayers-1; i>=0; i--) layers[i]->ReCalcG();
        trainings++;
      Error(outputs);
}
```

- When the training is finished, presenting a set of new input values should give the correct output.

```
double* bpnet::Evaluate(const double *inputs)
{
	for(int i=0; i<dimint; i++)
		layers[0]->outputs[i]=inputs[i];
	if (layers[0]->bias)
		layers[0]->outputs[dimint]=1;
	for(int i=1; i<nlayers; i++) layers[i]->Propagate();
	for(int i=0; i<dimout; i++)
		results[i]=layers[nlayers-1]->outputs[i];
	return results;
}
```

- Once a network has been created and trained, the weights may be saved to a file via the `Save()` method, having a character array string as its filename argument:

```
void bpnet::Save(const char *file_name)
{
	ofstream f(file_name);
	f<<nlayers<<"  "<<dimint<<"  "<<nhidden<<"  "<<dimout<<"  "<<bias<<"
"<<learning_rate<<" "<<trainings<<endl;
	for (int k=1; k<nlayers; k++)  {
		for (int i=0; i<layers[k-1]->neurons; i++){
		  for (int j=0; j<layers[k]->neurons-layers[k]->bias; j++)
		      f<<layers[k]->G[i][j]<<" ";
		}
	}
	f.close();
}
```

The back-propagation algorithm uses the gradient vector of the error surface [5] to point along the line of steepest descent from the current point. This way, when moving along this line for a short distance, the error will decrease. A minimum of some sort is eventually reached through a sequence of such moves. The "learning rate" parameter is responsible for a quick convergence when it takes large steps, but it may also overstep the solution or go off in the wrong direction if the error surface is very unconventional. In contrast, smaller steps require more iterations to complete, but may go in the correct direction.

In practice, the step size is proportional to the learning rate, which is very situation-dependent and is typically chosen by experiment. In the case of the proposed application, the author has conducted measurements in order to determine the optimal value for the learning rate.

The first test conducted kept all parameters constant and varied the learning rate between 0.95 and 0.05 with a step size of 0.1, for a training session between 5000 and 75000 repetitions. The author of the paper observed that the minimum error for the fully trained network corresponds to a learning rate value below 0.1 (Figure 3.a).

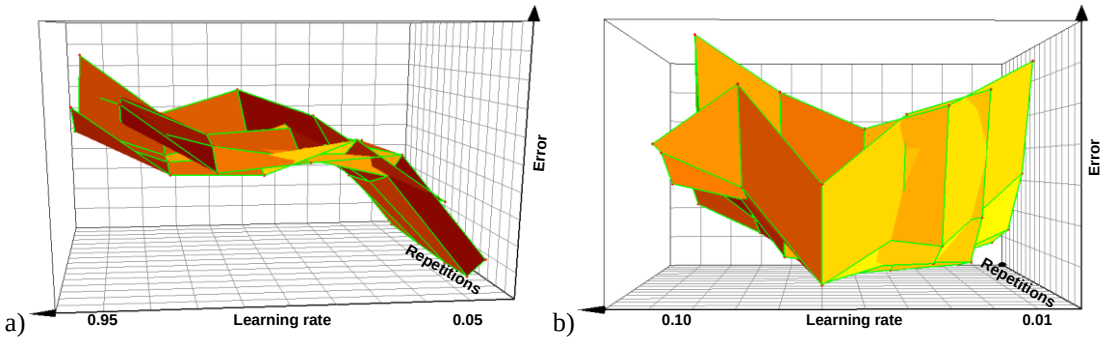


Fig. 3. Influence of the learning rate on the error vector a) Whole range; b) Below 0.1.

Moreover, in order to narrow down the interval, a second automatic test has been conducted with a varied learning rate between 0.1 and 0.01 with a step size of 0.01, for the same training session between 5000 and 75000 iterations. As a result, the author observed that the minimum error was reached for a learning rate value of 0.06 (Figure 3.b).

Another parameter taken into consideration by the author is the number of neurons from the hidden layer. Therefore, another set of automatic tests has been run in order to determine its weight.

For a fixed learning rate of 0.05, testing has revealed that the error decreases significantly when the number of neurons in the hidden layer increases above 5 (Figure 4.a). Moreover, as the number of neurons increases up to 75, the hidden layer performs better and the relative error decreases continuously (Figure 4.b). However, the plot shows that the network becomes saturated after a certain step and the only benefit of an increased number of neurons on the hidden layer is a lower initial error at the beginning of the trainings, or, in other words, a higher convergence speed.

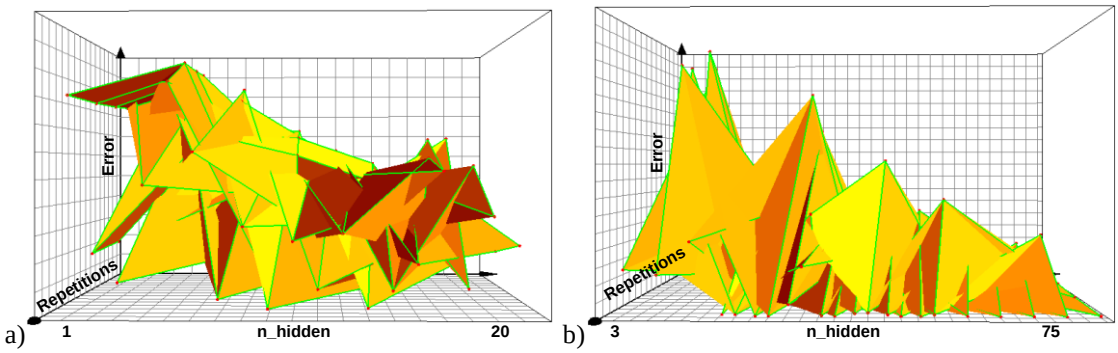


Fig. 4. Influence of the number of hidden neurons on the error vector a) Below 20; b) Whole range.

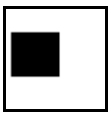
The algorithm can also be modified by including a momentum term [4] which encourages movement in a fixed direction. This adjustment allows it to pick up speed as several steps are taken in the same direction. As a consequence, the algorithm sometimes has the ability to escape local minima, and also to move rapidly over flat spots and plateaus.

Results

The following example represents an export from the developed application, for a three layer neural network with 30 neurons on the hidden layer and 2 neurons on the output layer. The training rate is 0.05 and the training cycle lasted for 100000 repetitions.

The network was able to identify the following test objects as being part of the circle-type and square-type object categories, respectively, with high probability.


94.19% circle-type


91.09% square-type

Conclusions

This paper discusses the application of a multi-layer perceptron neural network topology to a simple-object classification problem. For pattern recognition problems, the best training procedure is to assemble an extensive series of sample images that exhibit different

characteristics of interest. However, it cannot be overstressed that a neural network is only as good as the training data it receives. Therefore, grungy training data inexorably leads to an unreliable and erratic network.

Furthermore, the current study revealed that the behaviour of the network after the training session is greatly influenced by several parameters, such as the learning rate and the number of hidden-layer neurons. This paper presents the measurements of the influence of each on the proposed object classification application.

In addition, resizing the network by adding extra hidden-layer nodes or shuffling it by choosing a different initial point can both help to avoid trap situations where the back-propagation algorithm can't descend further because of local minima.

Other alternatives include introducing a momentum term, which essentially helps the algorithm pick up speed and avoid overstepping the solution, as well as using substitutes to the mean square error as a measure of how well the network is performing, both of which will constitute the object of a future study.

All in all, if carefully designed and trained, neural networks have a very broad applicability to real world business problems. Furthermore, they have already been successfully applied in many paradigms, such as function approximations, temporal series prediction, pattern recognition, voice recognition, retinal scanning, financial forecasting, classification and interpretation problems, texture analysis and 3D object recognition.

References

1. Fausett, L. – *Fundamentals of Neural Networks*. Prentice Hall, New York, 1994.
2. Haykin, S. – *Neural Networks: A Comprehensive Foundation*. Macmillan Publishing, New York, 1994.
3. Patterson, D. – *Artificial Neural Networks*. Prentice Hall, Singapore, 1996.
4. Bhagat, P.M. – *Pattern Recognition in Industry*. Elsevier, 2005.
5. Duda, R.O., Hart, P.E., Stork, D.G., - *Pattern classification* (2nd ed.). Wiley, 2001.
6. http://www.statsoft.com/textbook/stneunet.html.
7. http://en.wikipedia.org/wiki/Artificial_neural_network.
8. http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html.
9. * * * – *Back-Propagation Neural Network Tutorial*. http://ieee.uow.edu.au/~daniel/software/libneural/BPN_tutorial/BPN_English/BPN_English/

O metodă robustă de clasificare a obiectelor bazată pe reţele neuronale cu propagare înapoi

Rezumat

Lucrarea prezintă principiile de antrenare a reţelelor neuronale multi-strat şi aplicarea acestei topologii la o problemă de clasificare a obiectelor simple. Reţeaua neuronală este antrenată folosind algoritmul de propagare înapoi. În plus, lucrarea prezintă măsurătorile care au condus la determinarea influenţei ratei de învăţare şi a numărului de neuroni din stratul ascuns asupra comportamentului reţelei, pentru o problemă dată de clasificare a unor obiecte simple.