CHARACTER RECOGNITION USING NEURAL NETWORKS

Eugen Ganea, **Marius Brezovan, Simona Ganea**

Department of Software Engineering, School of Automation, Computers and Electronics, University of Craiova, 13 A.I. Cuza Str., 1100 Craiova, Romania
Phone: (0251) 435724/ext.163, Fax: (0251) 438198, *Email: ganea_eugen@software.ucv.ro*

Abstract: Numerous advances have been made in developing intelligent systems, some inspired by biological networks. The paper discuses about then usefulness of neural networks, more specifically the motivations behind the development of neural networks, the outline network architectures and learning processes. We conclude with character recognition, a successful layered neural network application.

Keywords: neural network, training procedure, layered network, learning rule, perceptron, back propagation, character recognition.

1. INTRODUCTION

Neural networks are a powerful model to deal with many pattern recognition problems. In many applications a complete supervised approach is assumed: a supervisor provides a set of labeled examples specifying for each input pattern the target output for the neural network. Thus, this learning scheme requires knowing a priori the exact target values for the neural network outputs for each example in the learning set. Neural networks classifiers (NN) have been used extensively in character recognition [2, 3, 5, and 7]. Many recognition systems are based on multilayer perceptrons (MLPs) [3, 5, and 7]. Gader et al. [7] describe an algorithm for hand printed word recognition that uses four 27–output–4–layer back propagation networks to account for uppercase and lowercase characters.

In this paper we investigate the characters recognition using different number of classes and different classification strategies. Section 2 presents the fundamentals of neuronal networks. Section 3 presents some strategies to supervised learning and presented three main classes of learning procedure. The application is described in Section 4. Some conclusions are drawn in Section 5.

2. FUNDAMENTALS OF NEURAL NETWORKS

All neural networks have in common the following four attributes:

- a set of processing units;
- a set of connections;
- a computing procedure;
- a training procedure.

2.1 *The processing units*

Neural network consists of a huge number of very simple processing units, similar to neurons in brain. These units may work in a complete parallel fashion, operating simultaneous. The computations in this system are made without the coordination of any master processor, excepting the case in which the neural network is simulated on a conventional computer. The unit computes a scalar function of its input, and broadcast the result to the units connected to it. The result is called activation value.

The unit may be classified into:

- *input units,* which receive data from the environment;
- *hidden units,* which transform the internal data of the network;
- *output unit,* which represent the decision or control signals.

The state of the network represents the set of activation values of over all units.

2.2 *The Connections*

The units in a network are organized into a given topology by a set of connections, or weights shown as lines in the following diagrams. The connections are fundamental in determining the topology of the neural network. Common topologies are: unstructured, layered and modular. Each has its domain of applicability:

- *unstructured networks* are useful for pattern completion;
- *layered networks* are useful for pattern association;
- *modular networks* are useful for building complex systems from simple structures.

The connectivity between two groups of units may be complete (the most frequent case), random, or local (connecting one neighborhood to another). A completely connected network has the most degree of freedom, so it can learn more functions than the constraint networks. This has the drawback in the case of small training set in which the network will simply memorize the input vectors, not being able to generalize. The networks with a limited connectivity may improve generalization, and effectiveness of the system. The local connections may help in special problems as recognizing shapes in a visual processing system.

2.3 *Computation*

Computations begin by presenting the input vectors to the units from the input layer. Then the activation of the remaining units are computed, synchronously (all at once in a parallel system) or asynchronously (one at a time in a randomized order). In the layered networks this process is called forward propagation. In this type of networks the computation ends after the activations of the output units are computed. For a given unit the computation takes two stages:

- first it is computed the network input, or internal activation;
- compute the output activation as a function of network input.

$$x_j = \sum_i w_{ji} y_i \,, \quad (1)$$

where x_j is the internal activation, y_i is the output activation of an incoming unit, and w_{ji} is the weight from unit i to unit j.

After the computation of x_j, the second phase is to compute the output activation y_j as function of x_j. Usually, this function may have three forms: linear, threshold, sigmoidal. The linear case implies $y_j = x_j$, which is not powerful, because a number of hidden layers of linear units may be replaced by only one equivalent such layer. So to construct non linear functions, a network needs non linear units. The simplest form of non linearity is implemented by the threshold function:

$$y(x) = \begin{Bmatrix} 0, if x \le 0 \\ 1, if x > 0 \end{Bmatrix}, \quad (2)$$

Multi layered network with such units may compute any Boolean function. The problem with this function is then when training the network, finding the right weights takes exponential time. A practical learning rule exists only for networks on hidden la layer. There are many applications in which continuous outputs are desired. The most common function is the sigmoidal function:

$$y(x) = \frac{1}{1+e^{-x}}, \quad (3)$$

or

$$y(x) = \tanh(x), \quad (4)$$

Non local activation functions can be used to impose global constraint on the output of the units in certain layer, for example to some 1, just like probabilities. One of these functions is softmax:

$$y(j) = \frac{e^{x_j}}{\sum_i e^{x_i}}, \quad (5)$$

2.4 *Training*

In the most general sense training a network means adapting its connections so that the network can exhibit the desired computational behavior for all input patterns. The process usually involves modifying weights (moving the hyperplanes / hyper spheres); but sometimes it also involves modifying the actual topology of the network, adding or deleting connections from the network (adding or deleting

hyperplanes / hyperspheres). In a sense, weight modification is more general than topology modification, since a network with abundant connections can learn to set any of its eights to zero, which has the same effect as deleting its weights. However, topological changes can improve both generalization and the speed of learning, by constraining the class of functions that the network is capable of learning. Finding a set of weights that will enable a given network to compute a given function is usually a non trivial procedure. An analytical solution exists only in the simplest case of pattern association, when the network is linear and the goal is to map a set of orthogonal input vectors to output vectors.
In this case the weights are given by the following relation:

$$w_{ji} = \sum_p \frac{y_i t_j}{| y_p |}, \quad (6)$$

Here y is the input vector, t is the target vector and p is the pattern index. In general, networks are non linear and multilayered, and their weights can be trained only by an iterative procedure, such as gradient descent on a global performance measure. This requires multiple passes of training on the entire training set; each pass is called iteration or an epoch. More over, since the accumulated knowledge is distributed over all of the weights, the weights must be modified very gently so as not to destroy all the previous learning. A small constant called the learning rate (e) is thus used to control the magnitude of weight modification. Finding a good value for the learning rate is very important; if the value is too small, learning takes forever; but if the value is too large, learning disrupts the previous knowledge. Unfortunately, there is no analytical method for finding the optimal learning rate; it is usually optimized empirically, by just trying different values.

3. SUPERVISED LEARNING

There are three main classes of learning procedure:

- supervised learning, in which a teacher provides output targets for each input pattern and corrects the network‘s explicitly.
- semi-supervised (or reinforcement) learning, in which a teacher merely indicates whether the networks' response to a training pattern is "good" or "bad";
- unsupervised learning, in which there is no teacher and a network must find regularities in the training data by itself.

Most networks fall squarely into one of these categories, but there are also various anomalous networks, such as hybrid networks which straddle these categories, and dynamic networks whose architecture can go or shrink over time. In speech recognition are mainly used the Multi – layer Perceptrons (first class) and sometimes the Kohonen maps (last class).

3.1 *Introduction*

Supervised learning means that a "teacher" provides output target for each input pattern, and corrects the network's errors explicitly. This paradigm can be applied to many types of networks, both feed forward and recurrent in nature. We will discuss these two cases separately. Perceptrons are the simplest type of feed forward networks that use supervised learning. A perceptron is comprised of binary threshold units arranged in two layers. Because a perceptron's activations are binary, this general learning rule reduces to the Perceptron Learning Rule, which says that if an input is active and the output y is wrong then w should be either increased or decreased by a small amount μ, depending if the desired output is 1 or 0, respectively. This procedure is guaranteed to find a set of weights to correctly classify the patterns in any training set if the patterns are linearly separable, if they can be separated into two classes by a straight line. Most training sets, however, are not linearly separable; in these cases we require multiple layers. Multi layer perceptrons (MLPs) can theoretically learn any function, but they are more complex to training. MLPs may have any number of hidden layers although a single hidden layer is sufficient for many applications, and additional hidden layers tend to make training slower, as the terrain in weight space becomes more complicated. MLPs can also be architecturally constrained in various ways, for instance by limiting their connectivity to geometrically local areas, or by limiting the values of the weights or tying different weights together.

3.2 *Back propagation*

Back propagation is the most widely used supervised training algorithm for neural networks. We begin with a full derivation of the learning rule. Suppose we have a multi layered feed forward network of non linear (typically sigmoidal) units. We want to find value for the weights that will enable the network to compute a desired function from input vectors to output vectors. Because the units compute non linear functions we can not solve for the weights analytically; so we will instead use a gradient descendent procedure on some global error function E. Let us define i, j and k as arbitrary unit indices, O as the set of output units, p as training pattern indices (where each training pattern

contains an input vector and output target vector), as the net input to unit j for pattern p, as the output activation of unit j from pattern p, as the weight from unit I to unit j, as the target activation for unit j in pattern p (for j in O), as the global output error for training pattern p, and E as the global error for the entire training set. Assuming the most common type of networks, we have:

$$x_j = \sum_i w_{ji} y_i$$

$$y(x) = \frac{1}{1+e^{-x}} \quad , (7)$$

It is essential that this activation function be differentiable, as opposite to non differentiable as in a simple threshold function, because we will be computing its gradient in a moment. The choice of error function is somewhat arbitrary; let us assume the Sum Squared error function:

$$E = \frac{1}{2}\sum_j (y_j - t_j)^2 \quad , \text{j} \in \text{O}, \ (8)$$

We want to modify each weight in proportion to its influence on the error E, in the direction that will reduce E:

$$\Delta(w_{ji}) = -\mu \frac{\partial E}{\partial w_{ji}}, \ (9)$$

where μ is a small constant, called the learning rate. By the Chain Rule and from the previous equations, we can expand this as follows:

$$\Delta(\frac{E}{w_{ji}}) = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial x_j}\frac{\partial x_j}{\partial w_{ji}}, \ (10)$$

the first of these three terms, which introduces the shorthand definition, remains to be expanded. Exactly how it is expanded depends on whether j in an output unit or not. If j is an output unit we have:

$$\text{j} \in \text{O} \Rightarrow \frac{\partial E}{\partial y_j} = (\text{y}_j - \text{t}_j), \ (11)$$

But if j is not an output unit, then it directly affects a set of units and by the Chain Rule we obtain:

$$j \notin O \Rightarrow \frac{\partial E}{\partial y_j} = \sum_{k \in out(j)} \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial x_k}\frac{\partial x_k}{\partial w_{kj}} =$$

$$= \sum_{k \in out(j)} \gamma(k)\sigma(x_k) w_{kj}, \ (12)$$

The recursion in this equation, in which γ_j refers to γ_k, says that the γ's in each layer can be derived directly from the γ's in the next layer. Thus, we can derive all the γ's in a multilayer network by starting at the output layer and working our way backwards towards the input layer, one layer at a time. This learning procedure is called "back propagation" because the error terms (γ's) are propagated through the network in this backwards direction. Back propagation can take a long time for it to converge to an optimal set of weights. Learning may be accelerated by increasing the learning rate e, but only up to certain point, because when the learning rate becomes too large, weights become excessive, units become saturated, and learning becomes impossible. Thus, a number of other heuristics have been developed to accelerate learning. These techniques are generally motivated by an intuitive image of back propagation as a gradient descendent procedure. That is, if we envision a highly landscape representing the error function E over weight space, then back propagation tries to find a local minimum value of E by taking incremental steps down the current hill side in the direction. This image helps us see, for example, that if we take too large of a step, when run the risk of moving so far down the current hill side that we find ourselves shouting up some other nearby hill side, with possible a higher error than before. Bearing this image in mind, one common heuristic for accelerating the learning process is known as momentum, which tends to push the weights further along in the most recently useful direction:

$$\Delta w_{ji}(t) = (-\mu \frac{\partial E}{\partial w_{ji}}) + (\alpha \cdot \Delta w_{ji}(t-1)), \ (13)$$

where α is the momentum constant, usually between 0.50 and 0.95. This heuristic causes the step size to steadily increase as long as we keep moving down a long gentle valley, and also to recover from this behavior when the error surface forces us to change directions. Ordinarily the weights are updated after each training pattern (this is called online training). But sometimes it is more effective to update the weights only after accumulating the gradients over a whole batch of training patterns (this is called batch training), because by superimposing the error landscapes for many training patterns, we can find a direction to

move, which is best for the whole group of patterns, and then confidently take a larger step in that direction. Because back propagation is a simple gradient descend procedure, it is unfortunately susceptible to the problem of local minima, it may converge upon a set of weights that are locally optimal, but globally suboptimal. In any case, it is possible to deal with the problem of local minima by adding noise to the weight modifications.

3.3 *The algorithms*

The algorithms are a forward implementation of the discussed theories regarding MLP. Our choice for transfer functions is sigmoid function. Its derivative has the expression:

$$\frac{\partial}{\partial x}(f(x)) = f(x)(1-f(x)), \quad (14)$$

The propagation algorithm:

```
Propagate (input_vector, output_vector);
    *Copy the input_vector in the input activation Y0
        for (i=1;i<nLayers;i++)
            for(j=0;j<nNodesLayer[i];j++){
                    ba[i][j] = brutActivation(i,j);
                    y[i][j]=FT(ba[i][j]);
            }
      *Copy the output layer activations YN in the
      output_vector.
End.
```

And the training algorithm:

```
Train (vect_in, T);
      Propagate (vect_in, NULL);
      for (i=nLayers-1;i>0;i--)
          for (j=0;j<nNodesLayer[i];j++){
            if ( i is the output layer){
               delta[i][j]=(y[i][j] – t[j]);
               if(using negative penalty)
                      delta[i][j]*=b[j];
              energy+=(y[i][j]-t[j])*( y[i][j]-t[j]);
             delta[i][j]*=dFT(ba[i][j]);
            }
           else { //for hidden layers
              suma= (float)0.0;
             for (l=0;l<nNodesLayer;l++)
                 suma+=delta[i+1][1]*w[i+1][1][j];
            delta[i][j]=suma*dFT(ba[i][j]);
          }
          //updating weights
         for (k=0;k<nNodesLayer[i-1];k++)
           w[i][j][k]+= -- eta*delta[i][j]*y[i-1][k];
       }
    return energy/2;
End.
```

4. CHARACTER RECOGNITION USING JAVA

It is often useful to have a machine perform pattern recognition. A machine that reads banking checks can process many more checks than a human being in the same time. This kind of application saves time and money, and eliminates the requirement that a human perform such a repetitive task. We demonstrate how character recognition can be done with a back propagation network. A network is to be designed and trained to recognize the 26 letters of the alphabet. An imaging system that digitizes each letter centered in the system's field of vision is available. The result is that each letter is represented as a 5 by 7 grid of boolean values. We use the graphical printing characters of the IBM extended ASCII character set to show a grayscale output for each pixel (Boolean value). For example, if we want to represent the letter X we have the following values:

```
1 0 0 0 1
0 1 0 1 0
0 0 1 0 0
0 0 1 0 0
0 1 0 1 0
1 0 0 0 1
```

The imaging system is not perfect and the letters may suffer from noise.

Perfect classification of ideal input vectors is required and reasonably accurate classification of noisy vectors. The twenty – six 35 element input vector are defined using a matrix. Each target vector is a 26-element vector with a 1 in the position of the letter it represents, and 0's everywhere else. For example, the letter E is to be represented by a 1 in the fifth element (as E is the fifth letter of the alphabet), and 0's in the rest of the elements of the twenty-six vector. The network receives the 35 Boolean values as a 35-element input vector. It is then required to identify the letter by responding with a 26-element output vector. The 26 elements of the output vector each represent a letter. To operate correctly, the network should respond with a 1 in the position of the letter being presented to the network. All other values in the output vector should be 0. In addition, the network should be able to handle noise. In practice, the network does not receive a perfect Boolean vector as input. Specifically, the network should make as few mistakes as possible when classifying vectors with noise of mean 0 and standard deviation of 0.2 or less. The neural network needs 35 inputs and 26 neurons in its output layer to identify the letters. The network is a two-layer log-sigmoid/log-sigmoid network. The log-sigmoid transfer function was picked because its output range (0 to 1) is perfect for learning to output boolean values.

The hidden (first) layer has 10 neurons. If the network has trouble learning, then neurons can be added to this layer. The network is trained to output a 1 in the correct position of the output vector and to fill the rest of the output vector with 0's. However, noisy input vectors may result in the network not creating perfect 1's and 0's. The result of this post-processing is the output that is actually used. To create a network that can handle noisy input vectors it is best to train the network on both ideal and noisy vectors. To do this, the network is first trained on ideal vectors until it has a low sum-squared error. Then, the network is trained on 10 sets of ideal and noisy vectors. The network is trained on two copies of the noise-free alphabet at the same time as it is trained on noisy vectors. The two copies of the noise-free alphabet are used to maintain the network's ability to classify ideal input vectors. Unfortunately, after the training described above the network may have learned to classify some difficult noisy vectors at the expense of properly classifying a noise-free vector. Therefore, the network is again trained on just ideal vectors. This ensures that the network responds perfectly when presented with an ideal letter. All training is done using back propagation with both adaptive learning rate and momentum with the function *set_training().* The network is initially trained without noise for a maximum of 5000 epochs or until the network sum-squared error falls beneath 0.1. To obtain a network not sensitive to noise, we trained with two ideal copies and two noisy copies of the vectors in alphabet. The target vectors consist of four copies of the vectors in target. The noisy vectors have noise of mean 0.1 and 0.2 added to them. This forces the neuron to learn how to properly identify noisy letters, while requiring that it can still respond well to ideal vectors. To train with noise, the maximum number of epochs is reduced to 300 and the error goal is increased to 0.6, reflecting that higher error is expected because more vectors (including some with noise), are being presented. Once the network is trained with noise, it makes sense to train it without noise once more to ensure that ideal input vectors are always classified correctly. Therefore, the network is again trained with code identical to the first pass of the training. The reliability of the neural network pattern recognition system is measured by testing the network with hundreds of input vectors with varying quantities of noise. We test the network at various noise levels, and then graph the percentage of network errors versus noise. Noise with a mean of 0 and a standard deviation from 0 to 0.5 is added to input vectors. At each noise level, 100 presentations of different noisy versions of each letter are made and the network's output is calculated. The output is then passed through the competitive transfer function so that only one of the 26 outputs (representing the letters of the alphabet), has a value of 1. The number of erroneous classifications is then added and percentages are obtained.

5. CONCLUSION

This problem demonstrates how a simple pattern recognition system can be designed. Note that the training process did not consist of a single call to a training function. Instead, the network was trained several times on various input vectors. In this case, training a network on different sets of noisy vectors forced the network to learn how to deal with noise, a common problem in the real world. As we were able to see neural network present certain advantages, like: learn rapidly, highly parallel processing, distributed representations, are able to learn new concepts, they are robust with respect to input noise, node failure, can adapt to input stimulus, they are a tool for modeling and exploring brain functions, are successful in areas like vision and speech recognition. But they also present some drawbacks: neural networks can not model higher level cognitive mechanism (attention, symbols, focus of attention), a wrong level of abstraction for describing higher level processes (the problems are represented as a list of features, having numerical values), and a huge number of trying for training, sensitive to local minima.

REFERENCES

Aleksander, Igor, and Morton, Helen (1990), *An Introduction to Neural Computing*, Chapman and Hall, London.

Anzai, Yuichiro (1992), *Pattern Recognition and Machine Learning*, Academic Press, Englewood Cliffs, NJ.

Freeman, James A., and Skapura, David M. (1991), *Neural Networks Algorithms, Applications, and Programming Techniques*, Addison-Wesley, Reading, MA.

Hertz, John, Krogh, Anders, and Palmer, Richard (1991) , *Introduction to the Theory of Neural Computation*, Addison-Wesley, Reading, MA.

Rzempoluck, E. J. (1998), *Neural Network Data Analysis Using Simulne,* Springer, New-York.

Wasserman, Philip D. (1989), *Neural Computing*, Van Nostrand Reinhold, New York.

Gader, Paul, et al. (1992), *"Fuzzy and Crisp Handwritten Character Recognition Using Neural Networks,"* Conference Proceedings of the 1992 Artificial Neural Networks in Engineering Conference, V.3, pp. 421–424.