LTF-C – Neural Network for Solving Classification Problems

Marcin Wojnarski

Institute of Computer Science, Jagiellonian University,
Nawojki 11, 30-072 Kraków, Poland
mwojnars@ns.onet.pl

Abstract. This paper presents a new model of an artificial neural network solving classification problems – Local Transfer Function Classifier (LTF-C). Its structure is very similar to this of the Radial Basis Function neural network (RBF), however it utilizes entirely different learning algorithms, including not only changing positions and sizes of neuron reception fields, but also inserting and removing neurons during the training. Applying this network to practical tasks, such as handwritten digit recognition, shows, that it is characterized by high accuracy, small size and high speed of functioning.

1 Introduction

The issue of constructing an automated classification system appears in many real-life situations. Let us consider tasks such as automatic recognition of handwritten characters, speech, web pages content, diagnosing of diseases or defects in various devices and so on. All of them can be seen as problems of creating a system which is able to recognise – when presented a pattern X ($X = [x_1, x_2, \ldots, x_n]$) – which class c ($c = 1, 2, \ldots, k$) this pattern belongs to.

Since we usually cannot find a complete analytical solution to a specified classification task, we would like to create a system which will learn the desired solution by itself. The artificial neural network is just such a system.

The neural network presented in this paper – Local Transfer Function Classifier (LTF-C) – is similar to one of the existing models (RBF), but it employs new training algorithms. The most noticeable difference is that the structure of LTF-C is not defined at the begining of the training, as in the case of most of other neural systems, but changes dynamically during the learning. Such an algorithm allows to fit better to the training set and guarantees that the network will be just as big, as it is really needed, hence it will be fast.

2 The Network Architecture

The only information the network utilizes to learn comes from a *training set*, composed of examples of correct classification in some number of particular cases. Therefore, the training set is composed of N pairs of the form: $(X^{(i)}, c^{(i)})$.

Vectors $X^{(i)}$ can be treated as points in n-dimensional space $\mathbf{X}$ (we can identify vectors with points, so for the simplicity of the notation these terms will be used interchangeably). Close neighborhood of the point $X^{(i)}$ should belong to the same class as $X^{(i)}$, therefore the space $\mathbf{X}$ can be divided into finite number of *decision regions* – areas of the same value of classification. The problem resolves then to the task of modeling complex figures in n-dimensional space. One of possible solutions of such a task is to model interiors of these regions – by filling them as tight as possible with figures of versatile shapes. This idea lays in the basis of LTF-C.

The network is composed of two layers of neurons. The first one retains the information about figures filling the decision regions. Each figure is represented by a neuron (forms its *reception field*): the neuron *weights* define the position of the figure centre, and the *radii* – its size. The neuron output, belonging to the range of $[0, 1]$, says how "much" the presented pattern lies in the interior of the figure. Formally, the response y_i of the i-th neuron on the pattern X is given as:

$$y_i = f\left(\sqrt{\sum_{j=1}^{n}\left(\frac{w_{ij} - x_j}{r_{ij}}\right)^2}\right) , \tag{1}$$

where $W_i = [w_{i1}, w_{i2}, \ldots, w_{in}]$ – weights of the i-th neuron, $R_i = [r_{i1}, r_{i2}, \ldots, r_{in}]$ – radii of the i-th neuron, f – an output function. With such a definition reception fields have the shapes of hyperellipses with axes parallel to the axes of the co-ordinate system.

Reception fields have to fill some – the most often bounded – region, so they have to be bounded figures themselves. Hence the output function should satisfy:

$$\lim_{d\to\infty} f(d) = 0 , \tag{2}$$

which guarantees that the transfer function will be *local* – neuron responses will vanish for points X lying far from W_i. The author used the Gaussian function as f in all experiments, obtaining the following value of the neuron response:

$$y_i = \exp\left(-\sum_{j=1}^{n}\left(\frac{w_{ij} - x_j}{r_{ij}}\right)^2\right) . \tag{3}$$

Every hidden neuron must remember what is the class of the decision region it fills. To this end, it uses weights of connections with output neurons (they are not being modified during the training). If the i-th hidden neuron belongs to the c-th class (fills the decision region of the c-th class) the weight w'_{ij} of its connection with the j-th output neuron equals:

$$w'_{ij} = \begin{cases} 0 & \text{for } j \neq c \\ 1 & \text{for } j = c \end{cases} . \tag{4}$$

The output layer just aggregates the information coming from the hidden one. It is composed of k neurons (k – the number of classes) – if the i-th neuron is the most activated one after the presentation of the pattern X, it means that the network has classified X to the i-th class. This layer is composed of simple linear units – a response y'_i of the i-th output neuron equals:

$$y'_i = \sum_{j=1}^{m} w'_{ji} y_j \ , \tag{5}$$

where m is the number of hidden neurons.

3 A Neural Network Training Process

3.1 Modifying Position of Reception Fields

The goal of a hidden neuron belonging to the c-th class is to position its reception field in such a way that it contains as much points from the c-th class and as little points from other classes as possible. For that reason, during the training phase the neuron should move its weights W towards the points X belonging to the c-th class and move away from the ones belonging to other classes. Moreover, the higher the neuron response on the presented pattern, the greater the influence of that pattern on modification of the neuron weights should be. Thus, a new value of weights of the i-th hidden neuron, belonging to the c_i-th class, after presentation of the pattern X from the c-th class, should be a weighted mean of their previous value and X:

$$W_i \leftarrow W_i + \eta y_i (X - W_i) \ , \tag{6}$$

$$\eta = \begin{cases} \eta^+ & \text{for } c_i = c \\ -\eta^- & \text{for } c_i \neq c \end{cases} \ , \tag{7}$$

where η^+ and η^- are constants $(0 < \eta^+ \leq \eta^- \leq 1)$. Usually $\eta^+ = \eta^- = 1$.

An interesting property of the learning process arises from (6). If ΔW_i is the increment of weights of the i-th neuron in a specified learning step, its expected value $E(\Delta W_i)$ equals:

$$E(\Delta W_i) = \frac{1}{N} \sum_{j=1}^{N} m_i^{(j)} (X^{(j)} - W_i) = \frac{\sum m_i^{(j)}}{N} \left(\frac{\sum m_i^{(j)} X^{(j)}}{\sum m_i^{(j)}} - W_i \right) , \tag{8}$$

where N is the number of samples in the training set and $m_i^{(j)} = \eta y_i^{(j)}$ is the weight of the component $X^{(j)} - W_i$ in (6). If we interpret the j-th pattern as a particle with $X^{(j)}$ as its position vector and $m_i^{(j)}$ as its mass, $M_i = \dfrac{\sum m_i^{(j)} X^{(j)}}{\sum m_i^{(j)}}$ will be the centre of mass of all particles represented by training patterns, and actually – by points lying in a reception field of the i-th neuron (only $m_i^{(j)}$ of such points is significantly nonzero). Since the vector $E(\Delta W_i)$ is – as it arises from the above equality – parallel to the vector $M_i - W_i$, weights of the neuron will move during the training towards the centre of mass of points from its reception field. This, in turn, guarantees the stability of the learning process and makes possible to understand what exactly happens during the training.

It is also worth noticing that (6) is similar to some well-known rules, such as Kohonen's, Hebbian or Hebbian with modifications [1, 2].

Learning according to (6) has some disadvantages. Hidden neurons are trained entirely independently, therefore they will gather after training in several regions of the input space – these ones, where the concentration of training points is the largest. In other regions of the input space there will not be any neurons. Another disadvantage of this formula is that differences in the difficulty of classification in various parts of the input space are not taken into account, while more neurons are needed in regions of more complicated decision border. To solve these problems the term of the *attractiveness* of the learning pattern was introduced. It defines how big influence on the modification of weights specified learning pattern should have. The worse (less correct) the network response on the pattern X, the bigger the attractiveness of this pattern should be.

Before giving the definition of the attractiveness we must say what we mean by less or more correct response. *Correctness* Δ of the network response on the pattern X from the c-th class was defined as follows:

$$\Delta = y'_c - \max\{\, y'_i : i \neq c \,\} \ . \tag{9}$$

The sign of Δ says whether the network answer was correct, and its absolute value says how sure the network was while giving this response. Certainly, the greater Δ the more correct the answer of the network is.

The attractiveness function $A(\Delta)$ must satisfy the following conditions:

1. $A(\Delta) \in [0, 1]$ – For the learning process to be stable.
2. $\lim\limits_{\Delta \to +\infty} A(\Delta) = 0$ – Well classified patterns do not influence learning then.
3. $\lim\limits_{\Delta \to -\infty} A(\Delta) = 0$ – It ensures that patterns mistakenly classified during the data acquisition will not have significant influence on the training. We can observe a similar property in the way people acquire knowledge – if the human get information completely unfitting his current knowledge, he does not believe in it, e.g. presuming he has misheard. Only if the same information come at him several times, he adjusts his opinion.

Taking into account above conditions, $A(\Delta)$ was defined as a Gaussian function with two slopes of separately chosen biases:

$$A(\Delta) = \begin{cases} \exp\left(-2\left(\dfrac{\Delta - \Delta_0}{\Delta_{\max} - \Delta_0}\right)^2\right) & \text{for } \Delta \geq \Delta_0 \\ \exp\left(-2\left(\dfrac{\Delta - \Delta_0}{\Delta_{\min} - \Delta_0}\right)^2\right) & \text{for } \Delta < \Delta_0 \end{cases} , \tag{10}$$

where Δ_0, $\Delta_{\min}$, $\Delta_{\max}$ are constants satisfying: $\Delta_{\min} < \Delta_0 \leq 0$ and $\Delta_0 < \Delta_{\max}$. The author used most often: $\Delta_0 = -0.5$, $\Delta_{\min} = -1.0$, $\Delta_{\max} = 0.5$.

There is still one more disadvantage of (6). The range of modifications of the reception field position does not depend on the number of learning steps to carry out. Thus, even just before the end of the training the modifications are large, preventing neurons from fitting well to the data. To correct this drawback a parameter $\tau_i^{(t)}$ was introduced, which denotes the velocity of the training of the i-th hidden neuron in the t-th training step:

$$\tau_i^{(t)} = 1 - \frac{t - t_i}{T - t_i} \ , \tag{11}$$

where t_i – a training step when the i-th neuron was created, T – the total number of training steps to carry out. The parameter τ_i is decreasing linearly from 1 at the moment of creating the i-th neuron to 0 in the last training step.

And the corrected formula for the weights modification has the form:

$$W_i \leftarrow W_i + \eta \tau_i^{(t)} A(\Delta) y_i (X - W_i) \ . \tag{12}$$

3.2 Modifying Size of Reception Fields

Size of the neuron reception field is defined by the vector of radii R_i (1), independently along each axis of the co-ordinate system. One of reasons for adjusting it adaptively is that regions of different size and difficulty of classification can exist in the input space simultaneously. There can exist, for instance, vast areas of univocal classification, very easy to model with only one huge reception field, and regions adjacent to decision borders, requiring high precision and, therefore, small reception fields. Another reason is that different attributes can be of unequal importance for classification – some of them can be insignificant, corresponding radii should be then large, while others can play the vital role in classification – corresponding radii ought to be quite short.

Change of the j-th radius of the i-th neuron after presentation of the sample (X, c) should depend on:

1. the response y_i of the neuron – in order to allow only patterns in the reception field to influence the training,
2. the attractiveness of the pattern – to enable difficult patterns to have bigger influence on the training (see Sect. 3.1),
3. the number of training steps to carry out – for neurons to fit well to the data at the end of the training (see Sect. 3.1),
4. the distance d_{ij} along the j-th axis between the pattern and the centre of the reception field:
$$d_{ij} = \left| \frac{x_j - w_{ij}}{r_{ij}} \right| . \tag{13}$$

The following formula satisfying given assumptions was devised:

$$r_{ij} \leftarrow r_{ij} \exp\left(\eta_{\mathrm{g}} \tau_i^{(t)} A(\Delta) y_i d_{ij} \right) \ , \tag{14}$$

$$\eta_{\mathrm{g}} = \begin{cases} \eta_{\mathrm{g}}^{+} & \text{for } c_i = c \\ -\eta_{\mathrm{g}}^{-} & \text{for } c_i \neq c \end{cases} \ , \tag{15}$$

where c_i is the number of a class which the i-th neuron belongs to, and η_{g}^{+} and η_{g}^{-} are constants $(0 < \eta_{\mathrm{g}}^{+} \leq \eta_{\mathrm{g}}^{-})$. The author used: $\eta_{\mathrm{g}}^{-} = 1$ and $\eta_{\mathrm{g}}^{+} = 0.8$.

3.3 Inserting Hidden Neurons

Before starting training neurons, they have to be created first, with weights and radii properly initialized. It is no that easy – when adaptive parameters are initialized randomly, nearly all reception fields land in regions with no training points. Initializing centres of reception fields with random points from the training set is not good, as well, as most of the neurons will be in regions, where many points lie, not where difficult classification requires more units. The best solution – applied in LTF-C – is adding neurons during the training, in regions where the network response is unsatisfactory.

In the t-th step of the training, after presentation of the sample (X, c), a neuron is inserted to the hidden layer with probability P, depending on $A(\Delta)$ (10), i.e. on how incorrect the network response has been, and $\tau_{\text{ins}}^{(t)}$, saying how intensive the process of creating new neurons should be in that learning step (compare with $\tau_i^{(t)}$ in Sect. 3.1):

$$P = \tau_{\text{ins}}^{(t)} p A(\Delta) \ , \tag{16}$$

where p is a positive constant (most often $p = 0.05$) and $\tau_{\text{ins}}^{(t)}$ is defined as:

$$\tau_{\text{ins}}^{(t)} = \begin{cases} 1 - \dfrac{t}{0.9T} & \text{for } t < 0.9T \\ 0 & \text{for } t \geq 0.9T \end{cases} \ . \tag{17}$$

In the last 10% of time $\tau_{\text{ins}}^{(t)} = 0$, since neurons created just before the end would not have enough time to learn.

Weights of the inserted neuron are initialized as follows (m – the number of hidden neurons existing till now):

$$W_{m+1} = X \ , \tag{18}$$

$$w'_{(m+1)i} = \begin{cases} 0, & \text{for } i \neq c \\ 1, & \text{for } i = c \end{cases} \ , \tag{19}$$

where $w'_{(m+1)i}$ is the weight of the connection with the i-th output neuron.

Initializing radii is more difficult. They should be rather long, as even one too small radius may result in excluding all the training points from the reception field. However, they should not be also too large either, since a new neuron could disturb the training process of other units too much. The following formula satisfies above conditions quite well:

$$r_{(m+1)i} = r_{\min} + Y(r_{\max} - r_{\min}) \ , \tag{20}$$

where:

$$r_{\min} = \min S \ , \quad r_{\max} = \max S \ , \tag{21}$$

$$S = \{ \, r_{ij} : 1 \leq i \leq m, 1 \leq j \leq n \, \} \cup \left\{ \frac{\sqrt{n}}{10} \right\} \tag{22}$$

(value of $\frac{\sqrt{n}}{10}$ was picked out under the assumption that components x_i of input vectors belong to $[0, 1]$), Y – a random variable of uniform distribution from the range of $[0, 1]$. Each component of the vector R_{m+1} is initialized individually.

3.4 Removing Hidden Neurons

Despite a sophisticated algorithm for creating neurons, many of them land in regions where they are useless or even harmful, only worsening the network performance. Thus, an algorithm for removing unnecessary hidden neurons is needed. The one used in LTF-C evaluates after each presentation of a pattern so-called *global usefulness* u_i of every hidden neuron. For this purpose it utilizes *instantaneous usefulness* v_i, saying how important the existence of the i-th neuron has been for reckoning a correct network response on only one pattern X. The instantaneous usefulness is computed only on the ground of the last presented sample (X, c), according to the formula:

$$v_i = A(\Delta_i) - A(\Delta) \ , \tag{23}$$

where A is the attractiveness function (10), Δ – correctness of the last response of the network, and Δ_i says how correct the response would have been if the i-th neuron had not existed (compare (9) and (5)):

$$\Delta_i = y_c^{(i)} - \max\left\{ y_k^{(i)} : k \neq c \right\} \ , \tag{24}$$

$$y_j^{(i)} = y'_j - w'_{ij} y_i \ . \tag{25}$$

The instantaneous usefulness v_i is positive if the i-th neuron has had beneficial contribution to reckoning the network response, and negative if the response would have been better after removing this neuron. Evaluating v_i for all neurons is not very expensive – complexity of this operation is proportional to the number of weights of the output layer.

The global usefulness u_i of the i-th neuron should be an average of values v_i computed for different training patterns. Arithmetic mean of v_i for all samples would be the best, but its use is impossible due to high memory requirements and computational complexity. Therefore, exponential mean was applied – only last values of u_i and v_i are necessary to calculate it. One has only to remember that patterns must be presented in each epoch in a different order, since this sequence influences the value of the usefulness. The formula for modification of u_i has the form:

$$u_i \leftarrow (1 - \eta_\mathrm{u}) u_i + \eta_\mathrm{u} v_i \ , \tag{26}$$

where η_u is a constant from the range of $[0, 1]$. The i-th neuron is removed when

$$u_i < U \ , \tag{27}$$

where the threshold U is a constant: $U \in [0, 1]$. Usually $U \approx \eta_\mathrm{u}$.

The requirement that an exponential mean with the parameter η_u should have similar properties as an arithmetic mean of N components (N – size of the training set) yields that η_u should be approximately $\frac{2}{N}$. And imagining what should happen with a neuron which reception field does not contain any training points yields that during neuron creation the usefulness u_i should be initialized with the value of $e^2 U \approx 8U$.

It should be mentioned that there exist other neural networks which structure changes during the training, e.g. Group Method of Data Handling [3].

4 Concluding Remarks

The presented neural network – LTF-C – was tested in solving three practical problems. Databases with patterns describing these problems were downloaded from Yann Le Cun's homepage [4] and from the UCI Repository [6]:

1. the MNIST database of handwritten digits [4, 5],
2. the Australian Credit Approval dataset [6, 7],
3. the Wisconsin Breast Cancer database [6].

In all experiments LTF-C's accuracy and effectiveness were similar or better than those of the most popular classification systems, such as Multi-Layer Perceptron (MLP), RBF neural network or k-Nearest Neighbour (k-NN) algorithm.

When tested in handwritten digit recognition, LTF-C with 424 hidden neurons obtained 2.6% error rate, compared with 5.0% of k-NN, 3.6% of RBF, $2.95 - 4.7\%$ of MLP, 1.1% of Soft Margin Classifier (SMC) and Tangent Distance Classifier (TDC), $0.7 - 1.1\%$ of LeNet [4, 5]. Systems which obtained lower error rates than LTF-C were either time-ineffective – SMC and TDC – or non-versatile (designed specifically for character recognition) – TDC and LeNet.

LTF-C can also, like every neural network, be implemented naturally as a parallel system. Responses of all neurons in a layer can be computed independently, so parallel implementation can speed up processing even by the factor of several hundreds. As other neural systems, LTF-C is resistant to noised data and the damage of some part of units. Furthermore, thanks to utilizing local transfer functions, it is free of a serious weakness of the most popular neural networks – MLPs – the problem of local minima.

Unfortunately, the presented system is not free of drawbacks. Large number of training parameters is the biggest one. It can make using LTF-C difficult and force to spend too much time on choosing the best values of the constants. Thus, further survey in this area is needed. The author is working also on creating a model similar to LTF-C, but able to approximate every given function.

References

1. Fiesler, E., Beale, R. (ed.): Handbook of neural computation. Oxford University Press, Oxford (1997)
2. Hebb, D.O.: The Organization of Behaviour. Wiley, New York (1949)
3. Pham, D.T., Xing, L.: Neural Networks for Identification, Prediction and Control. Springer Verlag, London (1995)
4. Le Cun, Y.: The MNIST database of handwritten digits. http://www.research.att.com/~yann/exdb/mnist/index.html
5. Le Cun, Y., et al.: Comparison of learning algorithms for handwritten digit recognition. In: Fogelman, F., Gallinari, P. (eds.): International Conference on Artificial Neural Networks. Paris (1995) 53–60
6. Mertz, C.J., Murphy, P.M.: UCI repository. http://www.ics.uci.edu/pub/machine-learning-databases
7. Michie, D., Spiegelhalter, D.J., Taylor, C.C.: Machine Learning, Neural and Statistical Classification. Elis Horwood, London (1994)