Fundamenta Informaticae XX (2003) 1–17 1
IOS Press

LTF-C: Architecture, Training Algorithm and Applications of New Neural Classifier

Marcin Wojnarski
Faculty of Mathematics, Informatics and Mechanics
Warsaw University
ul. Banacha 2, 02-097 Warszawa, Poland
mwojnars@ns.onet.pl

Abstract. This paper presents a new model of an artificial neural network solving classification problems, called Local Transfer Function Classifier (LTF-C). Its architecture is very similar to this of the Radial Basis Function neural network (RBF), however it utilizes an entirely different learning algorithm. This algorithm is composed of four main parts: changing positions of reception fields, changing their sizes, insertion of new hidden neurons and removal of unnecessary ones during the training.

The paper presents also results of LTF-C application to three real-life tasks: handwritten digit recognition, credit approval and cancer diagnosis. LTF-C was able to solve each of these problems with better accuracy than most popular classification systems. Moreover, LTF-C was relatively small and fast.

Keywords: Neural network, classification, recognition, RBF

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 are *classification problems*, that is problems of creating a system which can recognize – when presented with a pattern – which predefined class 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. Therefore, application of neural networks should be considered, as they are well known to be able to learn themselves effectivelly

and to achieve high accuracy. This paper presents the architecture, training algorithms and results of tests of a new RBF-like neural network solving classification problems, called Local Transfer Function Classifier (LTF-C).

LTF-C has virtually the same architecture as the Radial Basis Function (RBF) neural network, but it utilizes entirely different learning algorithm. This algorithm is composed of four main parts: changing positions of reception fields, changing their sizes, insertion of new hidden neurons and removal of unnecessary ones during the training.

Reception fields in LTF-C have shapes of hyperellipses. Their sizes can change during the training independently for each neuron and each axis of the coordinate system. The algorithm for changing the position of reception fields is similar to the Kohonen rule, however it is used in a supervised learning and is more sophisticated. Particularly, it enables patterns to have different influence on the training, according to their significance.

The algorithm for dynamic modification of the network structure is based on evaluation of the usefulness of every hidden neuron after each presentation of a pattern. This algorithm is very effective and does not slow down the training process significantly.

The paper presents also results of applying LTF-C to three real-life tasks: handwritten digit recognition, credit approval and cancer diagnosis. LTF-C was able to solve each of these problems with better accuracy than most popular classification systems. Moreover, LTF-C was relatively small and fast.

The algorithms presented in this paper were implemented in LTF-Cimulator program [11], which can be used to train and test LTF-C neural networks. LTF-Cimulator is a Microsoft Windows application and is freely available for non-commercial purposes.

2. The network architecture

Let the training set be composed of N pairs of the form: $\left(X^{(i)}, c^{(i)}\right)$, where $X^{(i)} = \left[x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}\right]$ is the i-th input pattern belonging to the $c^{(i)}$-th class $\left(c^{(i)} = 1, 2, \ldots, k\right)$. Vectors $X^{(i)}$ can be treated as points in the 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 decision regions, which are complex figures in the n-dimensional space.

Putting at present aside the algorithm creating the model of such a figure, we must consider the way of representing this model. In the case of a 2-dimensional figure we can remember its rough shape by memorizing the coordinates of certain points on its border along with their order on the circumference. Such a representation cannot be, however, generalized to multidimensional figures. We can also take a set of straight lines tangent to the figure in many different points and for each of them memorize which side of it the figure lies on. In the n-dimensional space straight lines would turn into hyperplanes – such a representation is used in a Multi-Layer Perceptron (MLP). Still, there might be problems with nonconvex figures. Furthermore, the model obtained is always sharp-edged, so it is necessary to take many hyperplanes to model spherical shapes.

We can model, however, not the border of the region, but its interior – by filling it as tight as possible with basic figures of versatile shapes. This idea lays in the basis of the proposed neural network.

The network is composed of two layers of neurons (Fig. 1). The first one retains the information

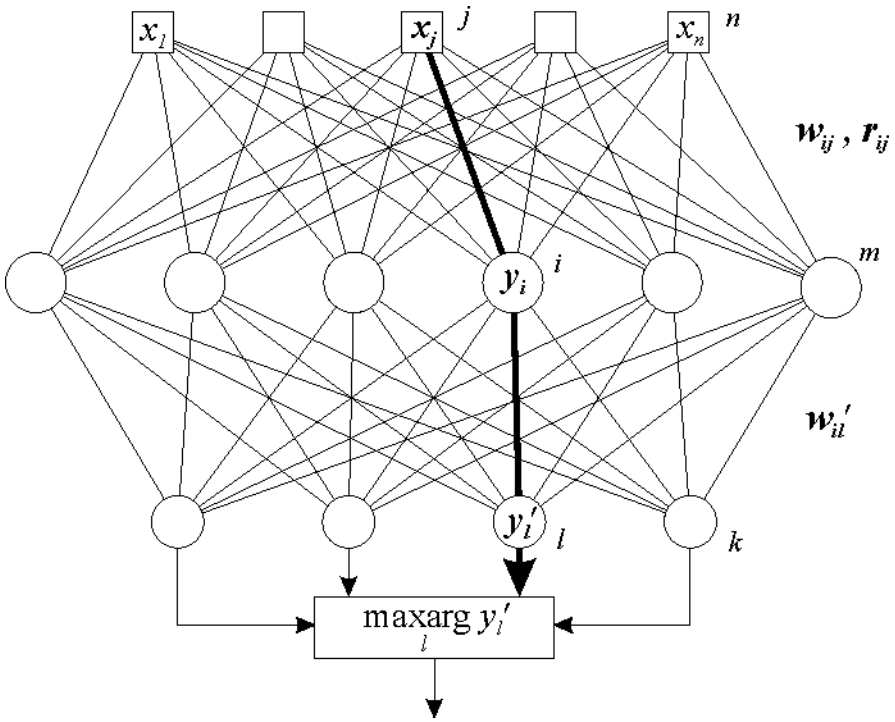

Figure 1. The architecture of the LTF-C neural network. Bolded symbols denote: signals flowing through the corresponding bolded connections (x_j, y_i, y'_l) and adaptive parameters (weights w_{ij}, w'_{il}, radius r_{ij}) attached to these connections

about basic figures filling the decision regions. Each figure is represented by a neuron (constitutes its *reception field*): the neuron weights define the position of the figure centre, and the radii – its size. The neuron output belongs to the range of $[0, 1]$. It defines how "much" the presented pattern lies in the interior of the figure (analogically to fuzzy rules we can use not only binary values for the neuron output). Formally, the response y_i of the i-th neuron on the input pattern X can be formulated as follows:

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

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.

Reception fields of neurons have to fill some – usually bounded – region, therefore they also ought to be bounded figures. Hence, the output function should satisfy the following condition:

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

which guarantees the *locality* of the transfer function – neuron responses will vanish for points X lying far from W_i.

The Gaussian function is used in all experiments:

$$f(d) = e^{-d^2}\,, \tag{3}$$

which yields the following form of the neuron response:

$$y_i = \exp\left(-\sum_{j=1}^{n}\left(\frac{w_{ij}-x_j}{r_{ij}}\right)^2\right). \tag{4}$$

With such a transfer function reception fields have shapes of hyperellipses with axes parallel to the axes of the coordinate system. Substituting the Euclidean norm in (1) with the Minkowsky's norm with the exponent both greater and smaller than 2 did not improve the results of the network. However, it does not necessarily mean that the Gaussian function is the best – the point is mainly to obtain reception fields of the most varied shapes [3].

Each 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. If the i-th hidden neuron belongs to the c-th class (fills the decision region of the c-th class) the weight w'_{il} of its connection with the l-th output neuron equals:

$$w'_{il} = \begin{cases} 0, & \text{for } l \neq c \\ 1, & \text{for } l = c \end{cases}. \tag{5}$$

These weights are set while a hidden neuron is created and do not change during the training.

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

$$y'_l = \sum_{i=1}^{m} w'_{il} y_i \,, \tag{6}$$

where y'_l is the response of the l-th output neuron and m is the number of hidden neurons.

As we can see, LTF-C has virtually the same architecture as the RBF neural network. The only difference is that the output layer in LTF-C has binary weights and does not undergo training.

One can also find a difference in the genesis of these systems. In the case of RBF the feature of the nonlinear projection of patterns onto a higher-dimensional space by the hidden layer is pointed out [9] (after such a projection the classification problem is more likely to be linearly separable, the Cover's theorem [1]), while the basis of LTF-C is an idea of modeling decision regions by filling their interiors.

3. A neural network training process

3.1. Changing position of reception fields

The goal of the hidden neuron belonging to the c-th class is to position its reception field in such a way that it contains as many points from the c-th class and as few points from other classes as possible. For that reason, during the training phase the neuron should move its weights W towards points from 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. It leads to a conclusion that a new value of weights of the i-th hidden neuron, belonging to

the c_i-th class, after presentation of a pattern X from the c-th class, should be a weighted average of their previous value and X:

$$W_i \leftarrow (1 - \eta y_i) W_i + \eta y_i X \ , \tag{7}$$

or equivalently:

$$W_i \leftarrow W_i + \eta y_i (X - W_i) \ , \tag{8}$$

$$\eta = \begin{cases} \eta^+, & \text{for } c_i = c \\ -\eta^-, & \text{for } c_i \neq c \end{cases} \ , \tag{9}$$

where η^+ and η^- are learning parameters satisfying: $0 < \eta^+ \leq \eta^- \leq 1$.

An interesting property of the learning process arises from the above formula. 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:

$$\begin{aligned} E(\Delta W_i) &= \frac{1}{N} \sum_{p=1}^{N} m_i^{(p)} (X^{(p)} - W_i) \\ &= \frac{\sum m_i^{(p)}}{N} \left(\frac{\sum m_i^{(p)} X^{(p)}}{\sum m_i^{(p)}} - W_i \right) \ , \end{aligned} \tag{10}$$

where N is the number of samples in the training set and $m_i^{(p)} = \eta y_i^{(p)}$ is the weight of the component $X^{(p)} - W_i$ in (8). If we interpret the p-th pattern as a particle with $X^{(p)}$ as its position vector and $m_i^{(p)}$ as its mass, $M_i = \dfrac{\sum m_i^{(p)} X^{(p)}}{\sum m_i^{(p)}}$ 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 $m_i^{(p)}$ 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 lying in its reception field. This, in turn, guarantees the stability of the learning process and makes it possible to understand what exactly happens during the training.

It should be added that masses $m_i^{(p)}$ can be negative, which occurs when a specified training pattern belongs to a different class than the neuron. Nonetheless, thanks to the locality of the neuron transfer function, eq. (2), this does not lessen the stability of the training.

It is also worth noticing that the rule (7) is similar to some well-known ones, such as Kohonen's, Hebbian or Hebbian with modifications (see [9, 5, 4]).

Learning according to (7) has some disadvantages. Hidden neurons are trained entirely independently, therefore they will gather after training in several regions of the input space, where the concentration of points X from the training set 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 a 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 a 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.

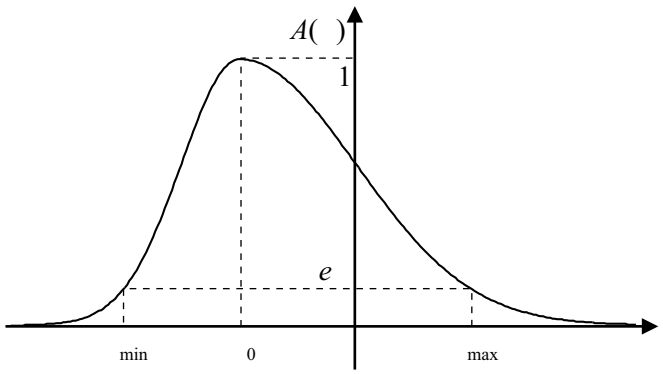

Figure 2. A graph of the attractiveness function

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

$$\Delta = y'_c - \max\left\{ y'_l : l \neq c \right\} . \tag{11}$$

With such a definition the sign of Δ says whether the network answer has been correct and its absolute value says how sure the network has been while giving that 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. $\lim_{\Delta \to +\infty} A(\Delta) = 0$ – So that well classified patterns do not influence learning.
3. $\lim_{\Delta \to -\infty} A(\Delta) = 0$ – This condition ensures that patterns which were, for instance, misclassified during the data acquisition, will not have significant influence on the training. We can notice a similar property in the way people acquire knowledge – if the human gets information completely unfitting his current knowledge, he does not believe in it, e.g., by presuming he has misheard. Only if the same information comes to him several times, he might adjust his opinion.

Taking into account above conditions, $A(\Delta)$ was defined as follows (see Fig. 2):

$$A(\Delta) = \begin{cases} \exp\left(-2\left(\frac{\Delta - \Delta_0}{\Delta_{\max} - \Delta_0}\right)^2\right), & \text{for } \Delta \geq \Delta_0 \\ \exp\left(-2\left(\frac{\Delta - \Delta_0}{\Delta_{\min} - \Delta_0}\right)^2\right), & \text{for } \Delta < \Delta_0 \end{cases} , \tag{12}$$

where Δ_0, $\Delta_{\min}$, $\Delta_{\max}$ are constants satisfying: $\Delta_{\min} < \Delta_0 \leq 0 < \Delta_{\max}$.

And after including the attractiveness the formula (8) for the weights modification turns into:

$$W_i \leftarrow W_i + \eta A(\Delta) y_i (X - W_i) . \tag{13}$$

3.2. Changing size of reception fields

A size of the neuron reception field is defined by the vector of radii R_i (eq. (1)), independently along each axis of the coordinate system. One of the reasons for adjusting it adaptively is that regions of different sizes 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 distance d_{ij} along the j-th axis between the pattern and the centre of the reception field:

$$d_{ij} = \left| \frac{x_j - w_{ij}}{r_{ij}} \right| . \tag{14}$$

The following formula satisfying given assumptions was devised:

$$r_{ij} \leftarrow r_{ij} \exp\left(\eta_{\mathrm{g}} A(\Delta) y_i d_{ij}\right) , \tag{15}$$

where

$$\eta_{\mathrm{g}} = \begin{cases} \eta_{\mathrm{g}}^{+}, & \text{for } c_i = c \\ -\eta_{\mathrm{g}}^{-}, & \text{for } c_i \neq c \end{cases} , \tag{16}$$

c_i is the number of a class which the i-th neuron belongs to, η_{g}^{+} and η_{g}^{-} are training parameters ($0 < \eta_{\mathrm{g}}^{+} \leq \eta_{\mathrm{g}}^{-}$).

3.3. Inserting hidden neurons

Before starting training neurons, first they have to be created, with weights and radii properly initialized. It is not that easy – when adaptive parameters are initialized randomly, nearly all reception fields land in regions with no training points, while regions full of learning patterns stay empty. Initializing field centers with points from the training set is not good, as well. In this case most of the neurons land in regions where many points lie, not where difficult classification requires more units. The best solution – applied in LTF-C – is to add neurons during the training in regions where network responses are unsatisfactory.

After presentation of the sample (X, c) a neuron is inserted to the hidden layer with probability P, depending on $A(\Delta)$ (eq. (12)), i.e. on how incorrect the network response has been:

$$P = pA(\Delta) , \tag{17}$$

where p is a positive constant. Weights of the inserted neuron are initialized as follows (m – the number of hidden neurons existing till now):

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

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

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

Initializing radii is more difficult. They should be rather long, as even one excessively small radius may result in excluding all the training points from the reception field. They should not be too large either, since a single presentation of a pattern just after the neuron creation could move the reception field to an entirely different region of the input space. The new neuron could also disturb too much the training process of other units. The following formula satisfies above conditions quite well:

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

where:

$$r_{\min} = \min Z \ , \tag{21}$$

$$r_{\max} = \max Z \ , \tag{22}$$

$$Z = \{ \, r_{ij} : 1 \leq i \leq m, 1 \leq j \leq n \, \} \cup \left\{ \frac{\sqrt{n}}{5} \right\} \ , \tag{23}$$

ρ – a random variable of uniform distribution on $[0, 1]$. The value of $\frac{\sqrt{n}}{5}$ was picked out under the assumption that the dispersion of values of attributes x_j is normalized, e.g. the attributes have unit variance.

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. For instance, many new neurons with very small reception fields, containing no points from the training set can be created near decision borders. They decrease not only the network speed, but also its accuracy (by lowering generalization). That is why an algorithm for removing unnecessary hidden neurons is needed.

The algorithm used in LTF-C evaluates after each presentation of a pattern so called *global usefulness* u_i of each hidden neuron. For that purpose it utilizes *instantaneous usefulnesses* v_i, saying how important the existence of the i-th neuron has been for reckoning a correct network response on the pattern X. The instantaneous usefulness is computed only on the ground of the last presented sample (X, c), according to the formula:

$$v_i = A(\Delta_i) - A(\Delta) \ , \tag{24}$$

where A is the attractiveness function (eq. (12)), Δ – the correctness of the last response of the network (eq. (11)), and Δ_i says how correct the response would have been if the i-th neuron had not existed (compare eq. (11) and (6)):

$$\Delta_i = y_c^{(i)} - \max \left\{ \, y_l^{(i)} : l \neq c \, \right\} \ , \tag{25}$$

$$y_l^{(i)} = y'_l - w'_{il} y_i \ . \tag{26}$$

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 – the complexity of this operation is proportional to the number of weights of the output layer, so it is lower than the complexity of computing the network response.

The global usefulness u_i of the i-th neuron should be an average of values v_i computed for different training patterns. Arithmetic average of v_i for all samples would be the best, but due to computational complexity it is not feasible to test the network on the whole training set after each learning cycle. 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 learning patterns must be presented in each epoch in a different order, since this sequence influences the value of the usefulness.

Suitable formula for the modification of u_i after the presentation of a pattern has the form:

$$u_i \leftarrow (1-\eta_\mathrm{u})u_i + \eta_\mathrm{u} v_i \ . \tag{27}$$

The i-th neuron is removed when

$$u_i < U \ . \tag{28}$$

Constants η_u and U belong to the range of $[0,1]$.

Choosing a proper value of η_u can be troublesome, as it is not known precisely how many recent values v_i have significant influence on the u_i. The goal is to obtain an exponential average with the parameter η_u, which has similar properties as the arithmetic mean of N components (N – the number of training patterns). Suppose we want to make variances of the both averages equal:

$$V_\mathrm{A} = V\left(\frac{1}{N}\sum_{i=1}^{N} X_i\right) \ , \tag{29}$$

$$V_\mathrm{E} = V\left(\eta_\mathrm{u}\sum_{i=1}^{\infty}(1-\eta_\mathrm{u})^{i-1} X_i\right) \ , \tag{30}$$

where $X_1, X_2, \ldots, X_N, \ldots$ are independent random variables of the same distribution as v_i (so $V(X_1) = V(X_2) = \ldots$). Then

$$V_\mathrm{A} = \frac{1}{N^2}\sum_{i=1}^{N} V(X_i) = \frac{1}{N} V(X_1) \ , \tag{31}$$

$$\begin{aligned} V_\mathrm{E} &= \eta_\mathrm{u}^2 \sum_{i=1}^{\infty}(1-\eta_\mathrm{u})^{2(i-1)} V(X_i) \\ &= \frac{\eta_\mathrm{u}^2 V(X_1)}{1-(1-\eta_\mathrm{u})^2} = \frac{\eta_\mathrm{u}}{2-\eta_\mathrm{u}} V(X_1) \ . \end{aligned} \tag{32}$$

Thus, for $V_\mathrm{A} = V_\mathrm{E}$ must hold:

$$\frac{1}{N} V(X_1) = \frac{\eta_\mathrm{u}}{2-\eta_\mathrm{u}} V(X_1) \ , \tag{33}$$

whence

$$\eta_\mathrm{u} = \frac{2}{N+1} \approx \frac{2}{N} \ . \tag{34}$$

There is yet the problem of initializing the usefulness u_i. Its initial value should not be too small, since a new neuron must have a chance to adapt to the surroundings before it is removed. It should not be also too big, as the algorithm for neuron removal would be inefficient.

Let us denote the initial value of u_i by u_0 . To estimate an appropriate value of u_0 we can imagine what should happen with a neuron which does not have any training points in its reception field, i.e. for which v_i is always 0. Such a neuron should be removed after N steps of the training, so if $u_i^{(N)}$ is its usefulness after N steps ($v_i^{(t)} = 0$ – instantaneous usefulness in the t-th step):

$$\begin{aligned} u_i^{(N)} &= (1-\eta_\mathrm{u})^N u_0 + \eta_\mathrm{u} \sum_{t=1}^{N} (1-\eta_\mathrm{u})^{t-1} v_i^{(t)} \\ &= \left(1 - \frac{2}{N}\right)^N u_0 \approx e^{-2} u_0 \ , \end{aligned} \tag{35}$$

the following condition should be satisfied:

$$u_i^{(N)} = U \ , \tag{36}$$

whence:

$$e^{-2} u_0 \approx U \ , \tag{37}$$

$$u_0 \approx 8U \ . \tag{38}$$

It is worth mentioning that the algorithm of weights and radii modification (eq. (13) and (15)), combined with dynamical modification of the network structure, displays some similarities with phenomena known from the evolution theory. Carrying out experiments on data with two-element input vectors (thus easy to visualize) the author noticed some kind of a competition between neurons with neighboring reception fields. It was favouring neurons with larger fields and forcing smaller ones to move to regions of the input space too narrow for bigger ones in order not to be removed. This phenomenon is similar to rivalry between species, which eliminate worse-adapted individuals or force them to find a free ecological niche. In the case of LTF-C this competition is still rather weak. However, this rivalry is probably worth strengthening, as applying evolutionary rules to computations has given excellent results so far.

3.5. Reducing training parameters during learning

Parameters defining the velocity of the training: $\eta^+, \eta^-, \eta_\mathrm{g}^+$ and η_g^- should be large at the beginning – for the learning to proceed quickly, and small at the end – for neurons to fit well to training data. For that reason they ought to be decreasing as the time goes by, preferably independently for each neuron – so that even lately created neurons are able to learn and older ones, taught quite well by then, are less mobile than younger ones. Thus, a parameter $\tau_i^{(t)}$ was introduced, which determine the ratio of values of parameters of the i-th neuron in the t-th training step to their initial values:

$$\tau_i^{(t)} = 1 - \frac{t - t_i}{T - t_i} \ , \tag{39}$$

where t_i – the number of a learning step when the i-th neuron was created, T – the total number of learning steps to carry out. At the moment of neuron creation τ_i equals 1 and is decreasing linearly as the time goes by, reaching 0 in the last step of the training.

Modified formulae (13) and (15) have the form:

$$W_i \leftarrow W_i + \eta\tau_i^{(t)} A(\Delta) y_i (X - W_i) \ , \tag{40}$$

$$r_{ij} \leftarrow r_{ij} \exp\left(\eta_{\mathrm{g}}\tau_i^{(t)} A(\Delta) y_i d_{ij}\right) \ . \tag{41}$$

Another parameter must also be decreasing during the training – the one determining the probability of creating a new neuron, p (eq. (17)). Inserting a new unit cause violent momentary perturbations in the training of existing neurons, making impossible for them to fit well to the data. For that reason a parameter $\tau_{\mathrm{ins}}^{(t)}$ was introduced, which says how intensive the process of creating new neurons should be in the t-th learning step:

$$\tau_{\mathrm{ins}}^{(t)} = \begin{cases} 1 - \dfrac{t}{0.9T}, & \text{for } t < 0.9T \\ 0, & \text{for } t \geq 0.9T \end{cases} \ . \tag{42}$$

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

Modified eq. (17) has the form:

$$P = \tau_{\mathrm{ins}}^{(t)} p A(\Delta) \ . \tag{43}$$

3.6. Choosing training parameters

The presented training algorithm is relatively complex. An unavoidable consequence of this complexity is the large number of training parameters which must be set by the experimenter before the start of the training. Fortunately, some general rules for finding the proper values can be devised:

- $\eta^+ = \eta^- = \eta_{\mathrm{g}}^- = 1.0$; eq. (7) and (15).
- $\eta_{\mathrm{g}}^+ = \max(0.1, \min(0.9, 2 * (1.0 - \alpha)))$; eq. (15). Here, α denotes the fraction of training patterns belonging to the most frequent class. η_{g}^+ depends linearly on α, but cannot be lower than 0.1 nor bigger than 0.9.
- $\Delta_0 = -0.5$, $\Delta_{\min} = -1.0$, $\Delta_{\max} = 0.5$; eq. (12).
- $p = 0.05$; eq. (17).
- $\eta_{\mathrm{u}} = \frac{2}{N}$; eq. (27). Here, N is the size of the training set. Finding the proper value of η_{u} is the most troublesome task, since the rule given here is not always valid. Namely, when the network after training contains only few hidden neurons and its accuracy is very poor, this indicates that η_{u} should be decreased significantly (even by the factor of 10).
- $U = \eta_{\mathrm{u}}$; eq. (28).
- The number of training epochs: 20.

The above rules are used by LTF-Cimulator [11] for choosing default values of training parameters. They were also used in the tests described in the next section.

Table 1. Characteristic of the databases used to test LTF-C

	MNIST	Australian Credit	Wisconsin Breast Cancer
size of the training set	huge	small	small
the number of classes	large	small	small
distribution of patterns among classes	uniform	uniform	irregular
the number of attributes	large	small	small
continuous attributes	+	+	–
categorical attributes with small number of values	–	+	–
categorical attributes with large number of values	–	+	+
missing values	–	+	+

Figure 3. Exemplary characters from the MNIST test set

4. Applications and results of tests

The presented neural network was tested in solving three problems of practical importance. These were handwritten digit recognition, credit approval and cancer diagnosis. Results were compared with these of other systems, tested by other scientists. It is worth emphasizing that each task was of a different type and required different features of a classification system (tab. 1).

4.1. Handwritten digit recognition

In this experiment the MNIST database of handwritten digits was used, which was downloaded from Yann Le Cun's homepage [6]. It is composed of two datasets: the training one, containing 60000 characters from 250 writers, and the test one, containing 10000 patterns. Digits from each set were written by different people. The database consists of 28x28-pixel images of scanned handwritten digits containing grey levels [6, 7]. Exemplary images from the test set are shown in fig. 3.

The only preprocessing performed on the data from the MNIST database (as well as from databases

decribed in the next sections) was normalization of input values, being originally integers from 0 to 255, to the range of $[0, 1]$. This was necessary in order to make the algorithm of initializing neuron radii work properly (see the remark by the formula (21)).

The best LTF-C neural network achieved 2.6% error rate on the test set and 1.5% on the training set. It contained 424 hidden neurons. It took 7 hours to train this network on a PC with AMD Duron 700 MHz CPU and 128 MB RAM.

The results of LTF-C are very good compared with these of systems tested by Le Cun (fig. 4). In particular, LTF-C performed much better than the most popular classification systems: MLP, RBF and k-Nearest Neighbor (k-NN) algorithm, when rough (neither deskewed nor distorted) training images were used. There were indeed systems which achieved much better accuracy than LTF-C, but all of them had also some serious drawbacks:

1. MLP and k-NN were able to achieve good results only when data preprocessing such as deskewing or adding distortions [6, 7] was applied. The error rate of k-NN decreased from 5.0% to 2.4% after deskewing; the error rate of 2-layer MLP with 300 neurons decreased from 4.7% to 3.6% after adding distortions or to 1.6% after deskewing. Such an immense decrease in error rates suggests that the accuracy of LTF-C would also improve after deskewing, which simplifies the problem, or after adding distortions, which artificially augments the training set and enables a system to fit better to the training data without decreasing generalization.

2. Soft Margin Classifier, Tangent Distance Classifier and k-NN require huge amount of memory and very long time for computing a response on a presented pattern [7] (of the order of the number of training patterns), which excludes them from most of applications (also character recognition).

3. Tangent Distance Classifier and LeNets are designed specifically for recognizing characters and utilize information specific only for this task. They achieved indeed the best results, however they are the least versatile. Moreover, training time of LeNets is very long [7].

LTF-C was trained on neither deskewed nor distorted images. This network is also free of drawbacks mentioned above.

4.2. Credit approval

In this experiment, LTF-C was tested on the Australian Credit Approval database from the UCI repository [8]. The task was to assess applications for credit cards and decide whether to give a card or not. The database consisted of 690 patterns containing 14 components (6 continous and 8 categorical). There were two classes with 45% and 55% of patterns respectively. Missing values in 5% of patterns were replaced by medians or means.

Results were obtained by 10-fold cross validation repeated 10 times. The average error rate of LTF-C in a single cross validation experiment varied between 12.8% and 14.2% (13.6% on average) with average number of neurons between 12 and 15. It took very short time to perform a 10-fold cross validation: only 5 seconds on AMD Duron 700 MHz.

Over 20 other classification systems were tested on the Australian Credit database in the StatLog project [9]. Their error rates in 10-fold cross validation were between 13.1% and 20.7%, e.g., the error rate of MLP was 15.4%, RBF – 14.5% and k-NN – 18.1% (fig. 5).

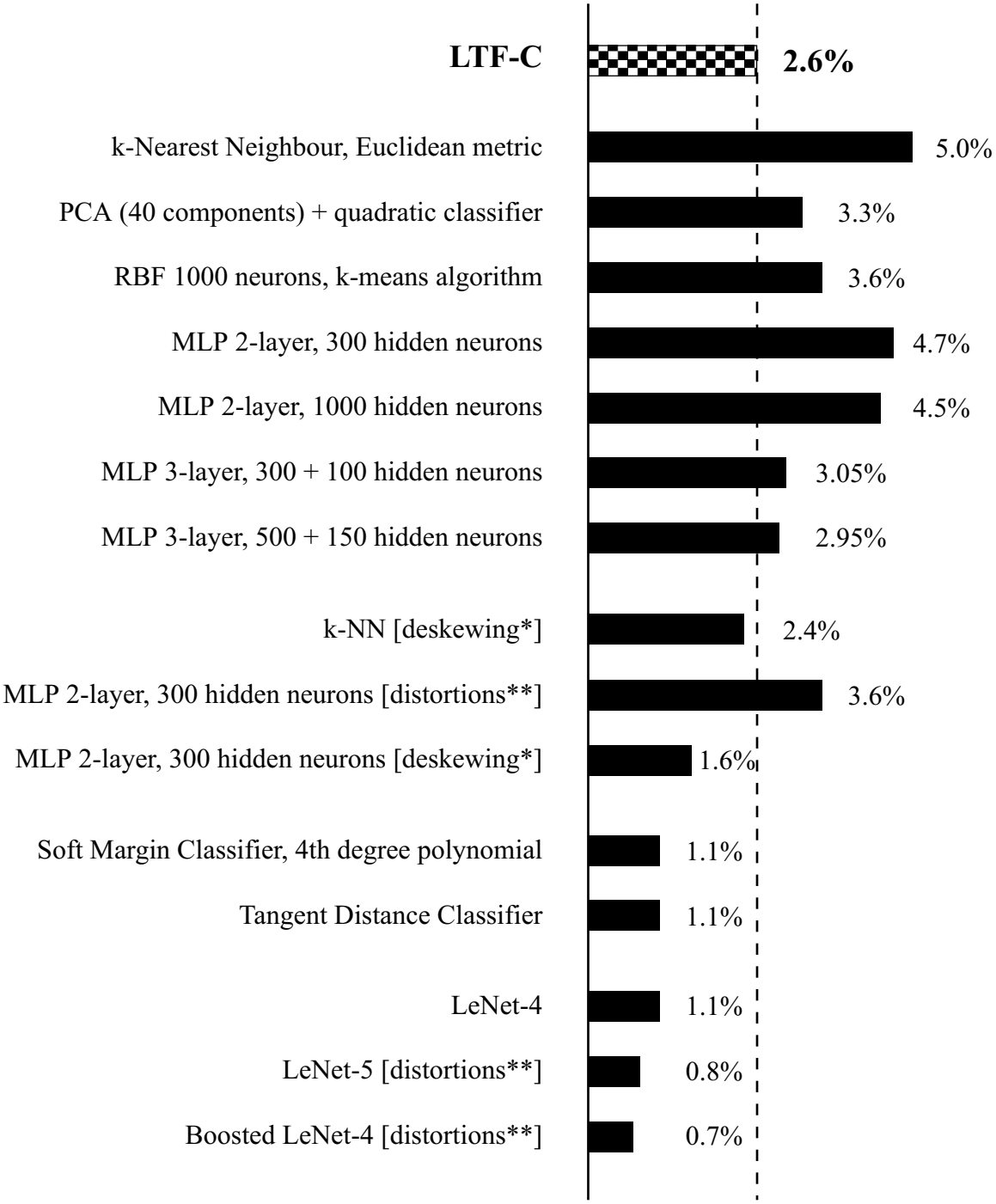

Figure 4. Error rates of systems tested on the MNIST database. *Deskewing* – all images were deskewed, so as to make them vertical; *distortions* – the training set was augmented with artificially distorted versions of the original training patterns

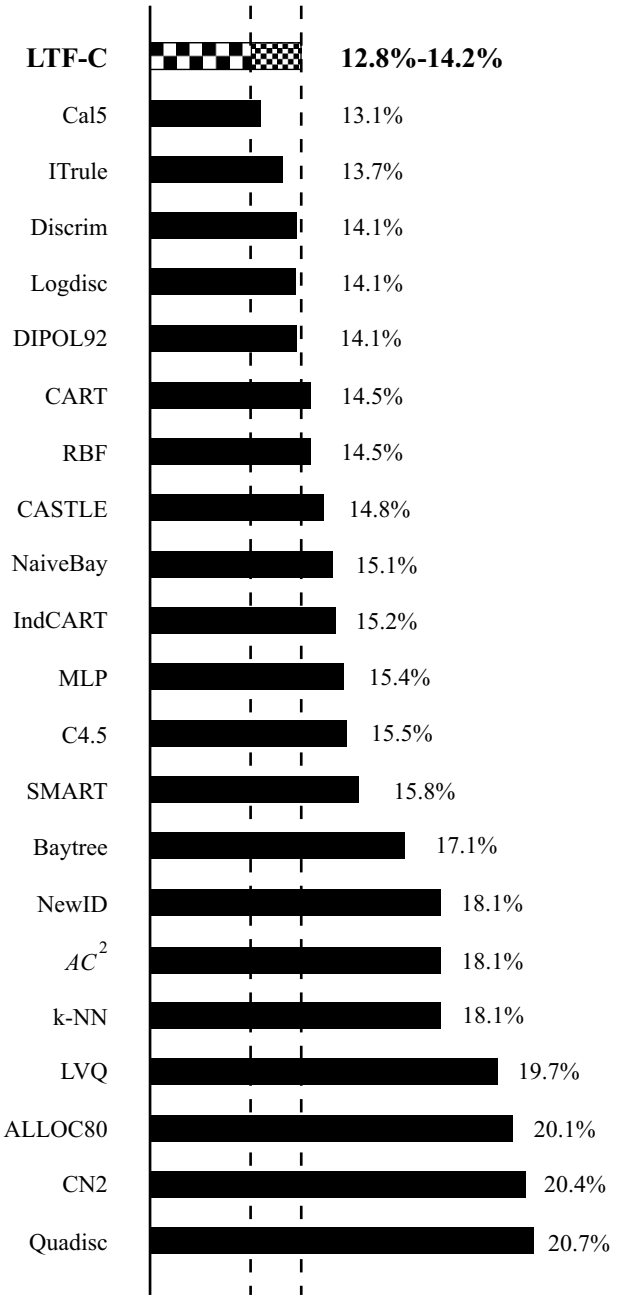

Figure 5. Error rates of systems tested on the Australian Credit Approval database

4.3. Cancer diagnosis

The last experiment was carried out with the use of the Wisconsin Breast Cancer database from the UCI repository [8]. Originally this database consisted 699 patterns belonging to one of two classes: benign (65.5%) or malignant (34.5%). However, 16 patterns which contained missing values were removed, so the database used in the experiment was composed of 683 patterns. Input vectors contained 9 attributes describing some characteristics of examined tissue. The attributes took integer values from 1 to 10.

In 10-fold cross validation repeated 10 times LTF-C obtained error rate between 2.5% and 3.5% (2.95% on average). In 5-fold cross validation repeated 10 times the error rate was between 2.6% and 3.1% (2.85% on average). The network was composed of 6 – 7 neurons. It took less than 5 seconds to perform a 10-fold cross validation on AMD Duron 700 MHz.

The results of LTF-C can be compared with error rates of other systems, given in [2]. Only 4 systems out of 37 obtained error rate below 3.0%. The error of the best one was 2.5%. And the worst systems had error rates of 6.6%, 13.3% and even 65.5%.

5. Concluding remarks

New model of an RBF-like neural network solving classification problems – Local Transfer Function Classifier – was presented in this paper. Not only the architecture and training algorithms of this model were described, but also results of tests were presented.

These results show that LTF-C can compete with the best classification systems, like the Multi-Layer Perceptron or the Radial Basis Function network. Along with very high accuracy, effectiveness and versatility LTF-C has many advantages characteristic for neural networks, such as natural possibility of implementing it as a parallel system, or resistance on 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.

However, there are still many elements of this model that can be improved. The most promising directions of further research are given below.

- Weights of output neurons must equal either 0 or 1. This condition could be relaxed, so that a non-zero output weight could be any positive real number. Such a modification would require a simple extension of the training algorithm, so that proper values of the output weights could be found. The tests performed so far show that this modification allows to create much smaller networks (even by 50%) with the same accuracy.

- Reception fields in LTF-C have shapes of ellipses with axes parallel to the axes of the coordinate system. The possibility of utilizing rotated ellipses is worth investigating, as this would allow the network to fit much better to the training data. The main problem is that a simple algorithm for performing rotations has quadratic complexity in respect to the size of the input vector.

- Large number of training parameters is a serious drawback of LTF-C. The existing rules for choosing their values not always give the best results. Perhaps, better rules could be found with an evolutionary algorithm.

Apart from the issues listed above, the author is working also on creating a neural network similar to LTF-C, but able to approximate every given function.

Acknowledgements

This work was partially supported by KBN grant 8T11C02519.

References

[1] Cover, T. M.: Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition, *IEEE Trans. Elect. Comp.*, **14**, 1965, 326–334.

[2] Duch, W.: Datasets used for classification: comparison of results,
Online: http://www.phys.uni.torun.pl/kmk/projects/datasets.html.

[3] Duch, W., Jankowski, N.: Survey of neural transfer functions, 1999.

[4] Fiesler, E., Beale, R., Eds.: *Handbook of Neural Computation*, The Computational Intelligence Library, Oxford University Press, New York, 1997.

[5] Hebb, D. O.: *The Organization of Behaviour*, Wiley, New York, 1949.

[6] Le Cun, Y.: The MNIST database of handwritten digits,
Online: http://yann.lecun.com/exdb/mnist/index.html.

[7] Le Cun, Y., et al.: Comparison of learning algorithms for handwritten digit recognition, 1995.

[8] Mertz, C. J., Murphy, P. M.: UCI repository,
Online: http://www.ics.uci.edu/pub/machine-learning-databases.

[9] Michie, D., Spiegelhalter, D. J., Taylor, C. C.: *Machine Learning, Neural and Statistical Classification*, Elis Horwood, London, 1994.

[10] Ripley, B. D.: *Pattern recognition and neural networks*, Cambridge University Press, Cambridge, 1996.

[11] Wojnarski, M.: LTF-Cimulator,
Online: http://rainbow.mimuw.edu.pl/∼mwojnar/ltfcim.