АРХІТЕКТУРА ТА КОМПОНЕНТИ КОМП'ЮТЕРНИХ СИСТЕМ

УДК 681.14

V. Kotsovsky[1], F. Geche[1], A. Batyuk[2], A. Mitsa[1]
[1]Uzhgorod National University
[2]Lviv National University "Львівська політехніка"

BACKPROPAGATION ALGORITHM FOR COMPLEX NEURAL NETWORKS

Розглянуто комплексні штучні нейронні мережі, функції активації яких є комплексними аналогами раціональної сигмоїди. Наведено алгоритм навчання цих мереж, заснований на методі зворотного поширення похибки.

Ключові слова: штучний нейрон, штучні нейронні мережі, комплексні нейронні мережі, алгоритм зворотного поширення помилки.

Neural networks with complex weights and continuously differentiable activation function have been studied in the paper. Learning algorithm based on the backpropagation method for rational sigmoid function has been given in the paper.

Key Words: artificial neuron, artificial neural networks, complex neural networks, learning algorithms, backpropagation.

Introduction

Neural networks are the effective means of solving the task of function approximation, forecasting the dynamic systems behaviour, multiple attribute set classification, pattern recognition, associative search and lot of other tasks. At present many types of architecture of neural networks with real weights have been developed in the information science. The variety of architectures is conditioned by different relation types between neurons, various activation functions (continuous or discontinuous (threshold type)) and different functioning mode of the neural networks. In connection with it many learning algorithms have been offered for neural networks. We introduce the notion of the complex neuron with continuously differentiable function and consider neural networks being built of these neurons. We also describe the modification of the well-known algorithm of backpropagation [1] for complex networks. Complex neural networks can be used for both solving the same tasks as real networks (with possible reduction of the number of neuron of input and output layers) and specific problem solution with complex initial data (for example the approximation of functions of complex variable).

Complex neural networks

The complex neuron is the functional element with n inputs $z_1, \ldots, z_n$ and one output y, which is calculated thus:

$$y = f\left(\sum_{j=1}^{n} w_j z_j + w_0\right),$$

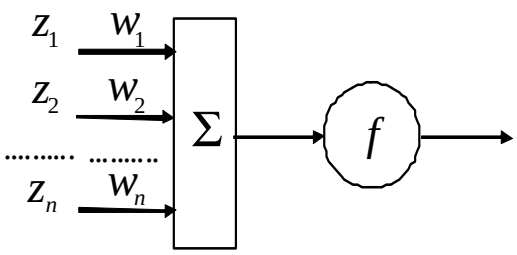

Figure 1. The complex neuron

where complex numbers $z_1,\ldots,z_n$ are input signals, $w_0, w_1,\ldots,w_n$, – complex weight coefficients (similarly to [2-3] we can term w_0 as the threshold of neuron element), $f: C \to C$ – nonlinear function, continuous with its partial derivative which we call the function of activation.

Complex neurons permit the different mode of connection in neural networks. We confine ourselves to studying the multilayer feed-forward neural networks that is the networks satisfying the following condition: the neurons of each layer are connected with the neurons of previous or next layers by the rule "each to each". The first layer is called the input layer, internal layers are called hidden ones and the last layer is named the output layer. The proceeding of neural network can be described with a following formula:

$$y_{kl} = f\left(\sum_j w_{jkl} z_{jkl} + w_{0kl}\right),\ x_{kj,l+1} = y_{kl},$$

where the index *j* denotes the number of input neuron, *k* is the number of output neuron, *l* is the layer index, $z_{jkl} = x_{jkl} + i\,y_{jkl}$ is the value of the *j* input signal of *k* neuron in *l* layer, $w_{jkl} = u_{jkl} + i\,v_{jkl}$ is the value of the *j* weight coefficient of *k* neuron in *l* layer.

Learning algorithm

Multilayer neural network calculates output vector $F(\boldsymbol{z})$ on the base of input vector $\boldsymbol{z}$. We mean the learning algorithm of instruction the selection of network parameters (weight coefficients w_{jkl}) thus that network puts in correspondence output vectors from the set $\{\boldsymbol{d}^1,\ldots,\boldsymbol{d}^m\}$ for input vectors from the set $\{\boldsymbol{z}^1,\ldots,\boldsymbol{z}^m\}$. The collection of the pairs $\{(\boldsymbol{z}^1,\boldsymbol{d}^1),\ldots,(\boldsymbol{z}^m,\boldsymbol{d}^m)\}$ is called the learning sample. Let f_k^t be the value of output signal of *k* neuron in the last output layer *l* in the case, then the network input vector is equal to $\boldsymbol{z}^t$. Let us introduce the important variable that will be named the network error *E*.

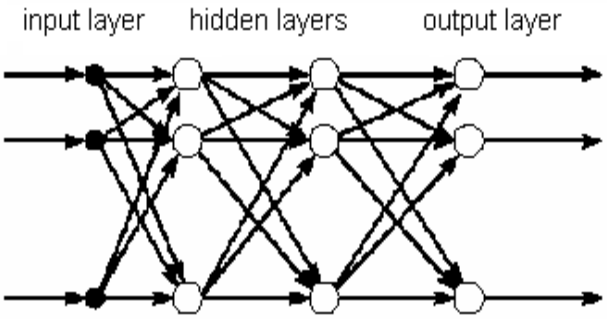

Figure 2. The multilayer feed forward complex network

$$E = \frac{1}{2}\sum_k \sum_t \left| f_k^t - d_k^t \right|^2 . \qquad (1)$$

We shall suppose that $E = E(\boldsymbol{W}) = E(\boldsymbol{U},\boldsymbol{V})$, where $\boldsymbol{U}$ is the vector components of which are the real parts of all coefficients of our neural network, $\boldsymbol{V}$ is the vector components of which are the imaginary parts

of coefficients of the network. During learning we shall change the weight vector in direction of antigradient of E on every iteration:

$$\Delta \boldsymbol{W}^r = -\eta_r \operatorname{grad} E\left(\boldsymbol{U}^r, \boldsymbol{V}^r\right), \boldsymbol{W}^{r+1} = \boldsymbol{W}^r + \Delta\, \boldsymbol{W}^r, \tag{2}$$

where r is the number of iteration.

Let

$$s_{kl}^t = \sum_j w_{jkl}^t z_{jkl}^t + w_{0kl}^t,\ a_{kl}^t = \operatorname{Re} s_{kl}^t,\ b_{kl}^t = \operatorname{Im} s_{kl}^t,\ f(z) = g(x,y) + i\, h(x,y).$$

Let us put down the components of gradient calculated by applying the last layer weigts

$$\frac{\partial E}{\partial u_{jkl}} = \sum_t \left(\frac{\partial E}{\partial g_{kl}^t} \left(\frac{\partial g_{kl}^t}{\partial a_{kl}^t} \frac{\partial a_{kl}^t}{\partial u_{jkl}} + \frac{\partial g_{kl}^t}{\partial b_{kl}^t} \frac{\partial b_{kl}^t}{\partial u_{jkl}} \right) + \frac{\partial E}{\partial h_{kl}^t} \left(\frac{\partial h_{kl}^t}{\partial a_{kl}^t} \frac{\partial a_{kl}^t}{\partial u_{jkl}} + \frac{\partial h_{kl}^t}{\partial b_{kl}^t} \frac{\partial b_{kl}^t}{\partial u_{jkl}} \right) \right), \tag{3}$$

$$\frac{\partial E}{\partial v_{jkl}} = \sum_t \left(\frac{\partial E}{\partial g_{kl}^t} \left(\frac{\partial g_{kl}^t}{\partial a_{kl}^t} \frac{\partial a_{kl}^t}{\partial v_{jkl}} + \frac{\partial g_{kl}^t}{\partial b_{kl}^t} \frac{\partial b_{kl}^t}{\partial v_{jkl}} \right) + \frac{\partial E}{\partial h_{kl}^t} \left(\frac{\partial h_{kl}^t}{\partial a_{kl}^t} \frac{\partial a_{kl}^t}{\partial v_{jkl}} + \frac{\partial h_{kl}^t}{\partial b_{kl}^t} \frac{\partial b_{kl}^t}{\partial v_{jkl}} \right) \right). \tag{4}$$

Let us adduce calculating formulas for partial derivatives in (3)-(4) (we shall miss the index t for the simplification of notation):

$$\frac{\partial \mathrm{E}}{\partial g_{kl}} = g_{kl} - \operatorname{Re} d_k,\ \frac{\partial \mathrm{E}}{\partial h_{kl}} = h_{kl} - \operatorname{Im} d_k, \tag{5}$$

$$\frac{\partial a_{kl}}{\partial u_{jkl}} = x_{jkl},\ \frac{\partial a_{kl}}{\partial u_{0kl}} = 1,\ \frac{\partial b_{kl}}{\partial u_{jkl}} = y_{jkl},\ \frac{\partial a_{kl}}{\partial u_{0kl}} = 0, \tag{6}$$

$$\frac{\partial a_{kl}}{\partial v_{jkl}} = -y_{jkl},\ \frac{\partial a_{kl}}{\partial v_{0kl}} = 0,\ \frac{\partial b_{kl}}{\partial v_{jkl}} = x_{jkl},\ \frac{\partial a_{kl}}{\partial v_{0kl}} = 1. \tag{7}$$

Then we set to selection of the activation function. The most popular activation function for real neural networks are logistic sigmoid curve $f(x) = \dfrac{1}{1+e^{-x}}$ or hyperbolic tangent $\tanh x$ (sometimes with some additional parameters). Unfortunally, the above mentioned functions are discontinuous as functions of complex variable. Therefore, they can't be applied in learning algorithms for complex networks which use the value of the gradient vector. The rational sigmoid

$$f(z) = \frac{z}{|z|+1}$$

is stripped of these disadvantages. For rational sigmoide we can write

$$f(z) = g(x,y) + i\, h(x,y),$$

where

$$g(x,y) = \frac{x}{\sqrt{x^2+y^2}+1},\ h(x,y) = \frac{y}{\sqrt{x^2+y^2}+1}.$$

It is necessary to notice than the rational sigmoide possesses the values that lay in unit disk centered at coordinate origin. In addition, the rational sigmoide compresses proportionally the real and imaginary parts of its input argument and has the property of reinforcing "weak" input signals and decreasing "strong" input signals.

Using the rational sigmoide curve we can easily obtain the following expressions for partial derivatives:

$$\frac{\partial g_{kl}}{\partial a_{kl}} = \frac{b_{kl}^2 + |s_{kl}|}{(|s_{kl}|+1)^3},\ \frac{\partial h_{kl}}{\partial b_{kl}} = \frac{a_{kl}^2 + |s_{kl}|}{(|s_{kl}|+1)^3},\ \frac{\partial h_{kl}}{\partial a_{kl}} = \frac{\partial g_{kl}}{\partial b_{kl}} = -\frac{a_{kl} b_{kl}}{|s_{kl}|(|s_{kl}|+1)^2}. \tag{8}$$

The values of derivatives $\frac{\partial E}{\partial u_{jkl}}$ and $\frac{\partial E}{\partial v_{jkl}}$, calculated according to the formulas (3)-(8) let compute the corrections Δu_{jkl} i Δv_{jkl} for neurons of the last (output) layer. Let us show, how we can calculate the corrections of weight coefficients of other layers of our neural network by the instrumentality of the value of partial derivatives $\frac{\partial E}{\partial u_{jkl}}$ and $\frac{\partial E}{\partial v_{jkl}}$. For the last layer we have:

$$\frac{\partial E}{\partial x_{jkl}} = \sum_t \left(\frac{\partial E}{\partial g_{kl}^t} \left(\frac{\partial g_{kl}^t}{\partial a_{kl}^t} \frac{\partial a_{kl}^t}{\partial x_{jkl}} + \frac{\partial g_{kl}^t}{\partial b_{kl}^t} \frac{\partial b_{kl}^t}{\partial x_{jkl}} \right) + \frac{\partial E}{\partial h_{kl}^t} \left(\frac{\partial h_{kl}^t}{\partial a_{kl}^t} \frac{\partial a_{kl}^t}{\partial x_{jkl}} + \frac{\partial h_{kl}^t}{\partial b_{kl}^t} \frac{\partial b_{kl}^t}{\partial x_{jkl}} \right) \right)$$

$$\frac{\partial E}{\partial y_{jkl}} = \sum_t \left(\frac{\partial E}{\partial g_{kl}^t} \left(\frac{\partial g_{kl}^t}{\partial a_{kl}^t} \frac{\partial a_{kl}^t}{\partial y_{jkl}} + \frac{\partial g_{kl}^t}{\partial b_{kl}^t} \frac{\partial b_{kl}^t}{\partial y_{jkl}} \right) + \frac{\partial E}{\partial h_{kl}^t} \left(\frac{\partial h_{kl}^t}{\partial a_{kl}^t} \frac{\partial a_{kl}^t}{\partial v_{jkl}} + \frac{\partial h_{kl}^t}{\partial b_{kl}^t} \frac{\partial b_{kl}^t}{\partial y_{jkl}} \right) \right)$$

In the last two formules the partial derivatives $\frac{\partial E}{\partial g_{kl}}$, $\frac{\partial E}{\partial h_{kl}}$, $\frac{\partial g_{kl}}{\partial a_{kl}}$, $\frac{\partial g_{kl}}{\partial b_{kl}}$, $\frac{\partial h_{kl}}{\partial a_{kl}}$, $\frac{\partial h_{kl}}{\partial b_{kl}}$ are already calculated by the formules (6)-(8). The other partial derivatives are equal:

$$\frac{\partial a_{kl}}{\partial x_{jkl}} = u_{jkl},\ \frac{\partial b_{kl}}{\partial x_{jkl}} = v_{jkl},\ \frac{\partial a_{kl}}{\partial y_{jkl}} = -v_{jkl},\ \frac{\partial b_{kl}}{\partial y_{jkl}} = u_{jkl}.$$

But the partial derivatives of *E* with respect of the value of input values x_{jkl} and y_{jkl} for the output layer coincide by implicity with the derivatives of the function of network error with respect of the real and imaginary parts of respective output values of neurons of previous layer. Therefore

$$\frac{\partial E}{\partial g_{j,l-1}} = \sum_k \frac{\partial E}{\partial x_{jkl}},\ \frac{\partial E}{\partial h_{j,l-1}} = \sum_k \frac{\partial E}{\partial y_{jkl}}. \tag{9}$$

The formulas (9) is similar to (5) for the previous layers and provide the passage from the calculation of the coordinates of the current layer gradient to the calculation of the respective coordinates of the previous layer gradient (the method of quick gradient calculation). The received algorithm of the correction of weight coefficients in according to formules (2)-(9) is the complex modification of well-known back propagation error algorithm, described in [1].

The question of the choice of the value of η_r (coefficient of the speed of the learning) in (2) is a very important one in connection with the application of complex weight neural networks. The traditional forthright approaches of searching η_r as the solution of the task of one-dimensional optimization are unacceptable because they require the multiple calculations of the network error *E* that is very difficult for networks with the bundle of neurons. Therefore we can set $\eta_r = \eta$, where η is any preassigned number from the segment $[0{,}01;1]$. In addition, for selectioning of the value of η_r we can offer the same approach that can be found in [3].

The learning of the neural network with the error function of the form (1) needs the consumption of considerable volume of the additional memory (one complex number for each parameter of the network).

Therefore for the complex neural networks with the great number of complex neurons it is possible to feed input vectors in random order and limit ourselves to calculating the gradient of the network error with respect of the only current element $(\boldsymbol{z}^t, \boldsymbol{d}^t)$ of the learning sample. In this case we can simplify the formulas (3)-(4):

$$\frac{\partial E}{\partial u_{jkl}} = \frac{\partial E}{\partial g_{kl}}\left(\frac{\partial g_{kl}}{\partial a_{kl}}\frac{\partial a_{kl}}{\partial u_{jkl}} + \frac{\partial g_{kl}}{\partial b_{kl}}\frac{\partial b_{kl}}{\partial u_{jkl}}\right) + \frac{\partial E}{\partial h_{kl}}\left(\frac{\partial h_{kl}}{\partial a_{kl}}\frac{\partial a_{kl}}{\partial u_{jkl}} + \frac{\partial h_{kl}}{\partial b_{kl}}\frac{\partial b_{kl}}{\partial u_{jkl}}\right),$$

$$\frac{\partial E}{\partial v_{jkl}} = \frac{\partial E}{\partial g_{kl}}\left(\frac{\partial g_{kl}}{\partial a_{kl}}\frac{\partial a_{kl}}{\partial v_{jkl}} + \frac{\partial g_{kl}}{\partial b_{kl}}\frac{\partial b_{kl}}{\partial v_{jkl}}\right) + \frac{\partial E}{\partial h_{kl}}\left(\frac{\partial h_{kl}}{\partial a_{kl}}\frac{\partial a_{kl}}{\partial v_{jkl}} + \frac{\partial h_{kl}}{\partial b_{kl}}\frac{\partial b_{kl}}{\partial v_{jkl}}\right)$$

It will be observed that it is possible to use the various modification of back propagation algorithm similar to the algorithms detailed in publication [2-5] changing its properly for use in learning of complex neural networks.

Conclusion

Artificial neural networks with complex weights is enough simple and powerful architecture. This architecture is extension of real weights network with continuously differentiable activation function and provides high precision approximation of nonlinear functions. We used the complex activation function similar to the well-known rational sigmoid and elaborated earning algorithm based on the backpropagation of the error of our network. Our method can be applied to networks with different activation functions. The batch-learning is also possible for our nets.

1. Rumelhart, D.E. Learning internal representations by error propagation / D. E. Rumelhart, G. E. Hinton, R. J. Williams // Parallel distributed processing, vol. 1, Cambridge, MA: MIT Press, 1986. p 318-362. 2. Уоссермен, Ф. Нейрокомпьютерная техника: теория и практика / Ф. Уоссерман – М.: Мир,1992. – 240 с. 3. Haykin, S. Neural networks, a comprehensive foundation / S. Haykin. – N.Y.: MacMillan College Publishing Company, 1994. – 1104 с. 4. Горбань, А. Н, Нейронные сети на персональном компьютере / А. Н. Горбань, Д.А. Россиев. – Новосибирск: Наука, 1996. – 225 с. 5. Anthony, M. Discrete Mathematics of Neural Networks / M. Anthony. – Philadelphia: SIAM, 2001. – 132 с.