VESTNIK UDMURTSKOGO UNIVERSITETA. MATEMATIKA. MEKHANIKA. KOMP'YUTERNYE NAUKI
COMPUTER SCIENCE 2018. Vol. 28. Issue 2

MSC2010: 68T05, 62M86

© *M. N. Nazarov*

NEURAL NETWORKS WITH DYNAMICAL COEFFICIENTS AND ADJUSTABLE CONNECTIONS ON THE BASIS OF INTEGRATED BACKPROPAGATION

We consider artificial neurons which will update their weight coefficients with an internal rule based on backpropagation, rather than using it as an external training procedure. To achieve this we include the backpropagation error estimate as a separate entity in all the neuron models and perform its exchange along the synaptic connections. In addition to this we add some special type of neurons with reference inputs, which will serve as a base source of error estimates for the whole network. Finally, we introduce a training control signal for all the neurons, which can enable the correction of weights and the exchange of error estimates. For recurrent neural networks we also demonstrate how to integrate backpropagation through time into their formalism with the help of some stack memory for reference inputs and external data inputs of neurons. Also, for widely used neural networks, such as long short-term memory, radial basis function networks, multilayer perceptrons and convolutional neural networks, we demonstrate their alternative description within the framework of our new formalism. As a useful consequence, our approach enables us to introduce neural networks with the adjustment of synaptic connections, tied to the integrated backpropagation.

Keywords: artificial neurons, backpropagation, adaptive connection adjustment, recurrent neural networks.

DOI: 10.20537/vm180212

Introduction

Backpropagation is one of the most successful and widely used algorithms for the training of neural networks. It has been adapted for such diverse models as multi-layer perceptrons, radial basis function networks and convolutional neural networks [1–3]. Moreover, its modification of backpropagation through time (BPTT) has been successfully applied for the training of specialized recurrent neural networks, such as long short-term memory [4,5]. The range of applied tasks that can be solved by these models is also quite diverse. For example, convolutional networks are used [6, 7] for image recognition, networks of radial basis functions are used for time series prediction and control systems construction [8], networks of long short-term memory are used for a handwritten text recognition and generation [9, 10], machine translation [11], speech synthesis and recognition [12, 13], and for a video processing in conjunction with convolutional networks [14].

However, the implementation of backpropagation is an external training procedure in relation to the models considered. Therefore, if we want to build a network with dynamical coefficients (see examples in [15]) on the basis of this algorithm, we will need to include it directly into the core formalism of standard models of neurons. This entails the introduction of backpropagation error estimates $\Delta(t)$ as some separate entities, as well as special neurons N_e with reference inputs $e(t)$ and a training control signal $a(t)$ for our network. In the case of recurrent networks we will have to add stack memory S_x for external data inputs and S_e for reference inputs of neurons.

As a result, our networks could be viewed as a special type of reprogrammable finite automata. The first consequence is an ability to construct hierarchical networks, which will control the training process for one another in ascending order. As a simple example one can consider two networks: the first is trained to spot some special stimuli in the input data to activate the training of a much bigger second one and control which parts of data will be sent to its data inputs and which to its reference inputs. Another important consequence would be the ability to introduce neural networks with the adjustment of synaptic connections (see the review in [16]) on the basis of integrated

backpropagation. In theory the ideal connection adjustment algorithm should prevent the overfitting of data by deleting all the unused connections and creating new links only when necessary.

§ 1. The description of basic models

A neuron number j from layer number i will be denoted as $N_{\dots}^{ij}$. Subscripts for $N_{\dots}^{ij}$ will be variable-length strings: $\varphi, \mathbf{r}, \mathbf{c}, \mathbf{e}$, where φ is an activation function, $\mathbf{r}$ denotes a recurrent mode, $\mathbf{c}$ identifies a mode with connection adjustment, and $\mathbf{e}$ denotes the presence of reference input for that neuron. In the case of a non-recurrent neuron with static connections and without the reference input, only φ will remain in this string. For example, the notation N_{σ}^{ij} will specify an ordinary neuron with a sigmoid activation function. In the general case we will introduce neurons $N_{\dots}^{ij}$ in our models as:

$$N_{\dots e}^{ij}(t) = \left(\overline{c}^{ij}(t), \overline{x}^{ij}(t), \overline{\omega}^{ij}(t), b^{ij}(t), \psi^{ij}, \varphi^{ij}, \overline{y}^{ij}(t), a^{ij}(t), \overline{\Delta}^{ij}(t), p^{ij}(t), \xi^{ij}(t), \overline{e}^{ij}(t)\right).$$

- $\overline{c}^{ij}(t) = (c_1^{ij}(t), \dots, c_n^{ij}(t))$ — connections to other layers and external inputs.
  1. If the input k is not connected to anything, then $c_k^{ij}(t) = (0, 0, 0)$.
  2. If the input k is connected to the external input $X_m(t)$, then $c_k^{ij}(t) = (0, 0, m)$.
  3. If the input k is connected to the output r of a neuron $N_{\dots}^{lm}$, then $c_k^{ij}(t) = (l, m, r)$.
- $\overline{x}^{ij}(t) = (x_1^{ij}(t), \dots, x_n^{ij}(t))$ — data input values of $N_{\dots e}^{ij}(t)$.
- $\overline{\omega}^{ij}(t) = (\omega_1^{ij}(t), \dots, \omega_n^{ij}(t))$ — weight coefficients of $N_{\dots e}^{ij}(t)$.
- $b^{ij}(t)$ — bias of neuron $N_{\dots e}^{ij}(t)$.
- ψ^{ij} — aggregation function of $N_{\dots e}^{ij}(t)$, for example $\psi(\overline{\omega}, \overline{x}) = \sum \omega_k \cdot x_k + b$.
- φ^{ij} — activation function of $N_{\dots e}^{ij}(t)$, for example $\varphi(z) = \mathrm{th}(z)$.
- $\overline{y}^{ij}(t) = (y_1^{ij}(t), \dots, y_k^{ij}(t))$ — output values $\overline{y}^{ij}(t) = \varphi(\psi(\overline{\omega}^{ij}(t), \overline{x}^{ij}(t)))$.
- $a^{ij}(t)$ — input signal of training activation for $N_{\dots e}^{ij}(t)$.
- $\overline{\Delta}^{ij}(t) = (\Delta_1^{ij}(t), \dots, \Delta_n^{ij}(t))$ — coefficients for backpropagation from $N_{\dots e}^{ij}(t)$.
- $p^{ij}(t)$ — paralysis indicator for weights of $N_{\dots e}^{ij}(t)$.
- $\xi^{ij}(t)$ — local minimum indicator of $N_{\dots e}^{ij}(t)$.
- $\overline{e}^{ij}(t) = (e_1^{ij}(t), \dots, e_k^{ij}(t))$ — optional reference inputs for $N_{\dots e}^{ij}(t)$.

Remark 1. For data inputs of neurons, four modes of operation are allowed:

1) when $c_k^{ij}(t) = (0, 0, 0)$, we will have the zero input $x_k^{ij}(t) = 0$;

2) when $c_k^{ij}(t) = (0, 0, m)$, we will have external connection $x_k^{ij}(t) = X_m(t)$;

3) for $c_k^{ij}(t) = (l, m, r)$ and $l < i$, we will have an ordinary link $x_k^{ij}(t) = y_r^{lm}(t)$;

4) for $c_k^{ij}(t) = (l, m, r)$ and $l \geq i$, we will have a recurrent one $x_k^{ij}(t) = y_r^{lm}(t-1)$.

Model 1. Neurons N_σ^{ij} and $N_{\sigma e}^{ij}$ with sigmoid activation $\varphi^{ij}(z) = 1/(1+e^{-2\alpha z})$. The derivative of this function is: $\partial\varphi/\partial z = 2\alpha\varphi(z)(1-\varphi(z))$. As an aggregation function we will use a weighted summation with the bias $\psi^{ij}(\overline{\omega},\overline{x}) = \sum \omega_k \cdot x_k + b$, which as a result gives us a standard formula for $y^{ij}(t) = \varphi^{ij}\left(\sum \omega_k^{ij}(t)\cdot x_k^{ij}(t) + b^{ij}(t)\right)$. Finally, general correction factors $\delta^{ij}(t)$ will be calculated:

$$\delta^{ij}(t) = \begin{cases} (y^{ij}(t) - e^{ij}(t)), & \text{for neurons with } e^{ij};\ \text{denote them by } N_{\sigma e}^{ij}; \\ \sum\limits_{\substack{l,p,k:\\ c_k^{lp}(t)=(i,j,1)}} \Delta_k^{lp}(t), & \text{for neurons without } e^{ij};\ \text{denote them by } N_\sigma^{ij}. \end{cases} \tag{1.1}$$

The application of formula (1.1) implies the explicit inclusion of weights $\omega_k^{ij}(t)$ into all of the back-propagation coefficients $\Delta_k^{ij}(t)$, which yields:

$$\Delta_k^{ij}(t) = 2\alpha\, y^{ij}(t)\,\left(1 - y^{ij}(t)\right)\,\delta^{ij}(t)\,\omega_k^{ij}(t)\,\sigma(a^{ij}(t)). \tag{1.2}$$

The activation of training will be applied with a positive training signal $a^{ij}(t) > 0$. For the adjustment of weights we will use a standard formula with an added $\sigma(a^{ij}(t))$:

$$\omega_k^{ij}(t+1) = \omega_k^{ij}(t) - 2\mu\,\alpha\, y^{ij}(t)\,\left(1 - y^{ij}(t)\right)\,\delta^{ij}(t)\,x_k^{ij}(t)\,\sigma(a^{ij}(t)). \tag{1.3}$$

Assuming $x_k^{ij}(t) = 1$, we will get a formula for bias adjustment $b^{ij}(t+1)$ from (1.3).
We assume that a paralysis of weights $\omega_k^{ij}(t)$ occurs when 70 % of them reach a threshold value $\omega_{\max}$:

$$p^{ij}(t) = \begin{cases} 1, & \text{if } \sum\limits_{k=\overline{1,n}} |\omega_k^{ij}(t)| > 0.7\cdot\omega_{\max}\cdot n, \\ 0, & \text{if otherwise.} \end{cases} \tag{1.4}$$

In this expression, n is the number of neuron inputs $x_1^{ij},\ldots,x_n^{ij}$. Formulas for detecting a local minimum of $\omega_k^{ij}(t)$ will also use this number, but the main criteria for them will be a low amplitude oscillation of $\Delta\omega_k^{ij}(t) = \omega_k^{ij}(t+1) - \omega_k^{ij}(t)$:

$$\xi^{ij}(t) = \begin{cases} 1, & \text{if } \sum\limits_{k=\overline{1,n}} \left| \sum\limits_{\tau=\overline{t-t_\xi,t}} \omega_k^{ij}(\tau+1) - \omega_k^{ij}(\tau) \right| < \omega_{\min}\cdot n \cdot \prod\limits_{\tau=\overline{t-t_\xi,t}} \sigma(a^{ij}(\tau)), \\ 0, & \text{if otherwise.} \end{cases} \tag{1.5}$$

As a result, our basic model of sigmoid neuron will have only six parameters:

- n — number of data inputs;
- $\omega_{\max}$ — maximum absolute values of weights ω_k^{ij};
- $\omega_{\min}$ — minimum absolute values of weights ω_k^{ij};
- t_ξ — local minimum detection time;
- μ — training rate of neuron;
- α — sigmoid stiffness ($\alpha \geq 1$).

Model 2. Neurons N_{th}^{ij} and $N_{\text{th}\,e}^{ij}$ with hyperbolic tangent as an activation function $\varphi^{ij}(z) = \text{th}(z)$. We will use a weighted summation with the bias $\psi^{ij}(\overline{\omega},\overline{x}) = \sum \omega_k\cdot x_k + b$ as an aggregation function

ψ^{ij}, just as in model 1. The derivative of the activation function will be $\partial\varphi/\partial z = (1-\varphi^2(z))$ which leads to the replacement of formulas (1.2) and (1.3) by

$$\Delta_k^{ij}(t) = \left(1-(y^{ij}(t))^2\right)\,\delta^{ij}(t)\,\omega_k^{ij}(t)\,\sigma(a^{ij}(t)), \tag{1.6}$$

$$\omega_k^{ij}(t+1) = \omega_k^{ij}(t) - \mu\,\left(1-(y^{ij}(t))^2\right)\,\delta^{ij}(t)\,x_k^{ij}(t)\,\sigma(a^{ij}(t)). \tag{1.7}$$

Bias update $b^{ij}(t+1)$ is a special case of (1.7) and can be obtained by a simple substitution $x_k^{ij}(t)=1$. Moreover, the formulas for calculating $\delta^{ij}(t)$, $p^{ij}(t)$, and $\xi^{ij}(t)$ are completely analogous to (1.1), (1.4), and (1.5).

Model 3. Neurons N_{id}^{ij} and $N_{\mathrm{id}\,e}^{ij}$ with a linear activation function $\varphi^{ij}(z)=z$. Just as in the first two models, we will use the standard aggregation function $\psi^{ij}(\overline{\omega},\overline{x}) = \sum\omega_k\cdot x_k + b$. Formulas for $\delta^{ij}(t)$, $p^{ij}(t)$, and $\xi^{ij}(t)$ will be analogous to (1.1), (1.4), and (1.5). In turn, an expression for $\Delta_k^{ij}(t)$ and $\omega_k^{ij}(t+1)$ considering the linear φ^{ij} will be replaced by

$$\Delta_k^{ij}(t) = \delta^{ij}(t)\,\omega_k^{ij}(t)\,\sigma(a^{ij}(t)),$$

$$\omega_k^{ij}(t+1) = \omega_k^{ij}(t) - \mu\,\delta^{ij}(t)\,x_k^{ij}(t)\,\sigma(a^{ij}(t)).$$

Model 4. Neurons N_{Ed}^{ij} and $N_{\mathrm{Ed}\,e}^{ij}$ for the calculation of Euclidean distance, which use $\varphi^{ij}(z)=\sqrt{z}$ as an activation function and $\psi^{ij}(\overline{\omega},\overline{x}) = \sum(\omega_k - x_k)^2$ as an aggregation function. As a result, an output value for them is $y^{ij}(t) = \sqrt{\sum\left(\omega_k^{ij}(t) - x_k^{ij}(t)\right)^2}$. General correction factors $\delta^{ij}(t)$ will be:

$$\delta^{ij}(t) = \begin{cases} \dfrac{1}{2}(y^{ij}(t) - e^{ij}(t)), & \text{for neurons with } e^{ij};\ \text{denote them by } N_{\mathrm{Ed}\,e}^{ij};\\ \displaystyle\sum_{\substack{l,p,k:\\ c_k^{lp}(t)=(i,j,1)}} \Delta_k^{lp}(t), & \text{for neurons without } e^{ij};\ \text{denote them by } N_{\mathrm{Ed}}^{ij}. \end{cases}$$

Taking into account a special aggregation function, we will have the following formula for backpropagation coefficients $\Delta_k^{ij}(t)$:

$$\Delta_k^{ij}(t) = \delta^{ij}(t)\cdot\left(\omega_k^{ij}(t) - x_k^{ij}(t)\right)\cdot\sigma(a^{ij}(t))/y^{ij}(t).$$

Finally, a formula for weight coefficients $\omega_k^{ij}(t+1)$ will be updated as follow:

$$\omega_k^{ij}(t+1) = \omega_k^{ij}(t) - 2\mu\cdot\Delta_k^{ij}(t).$$

Model 5. Convolutional neurons N_{Conv}^{ij} and $N_{\mathrm{Conv}\,e}^{ij}$ with linear activation $\varphi^{ij}(z)=z$, matrix input $\overline{x}^{ij}(t) = \left(x_{11}^{ij}(t),\dots,x_{nm}^{ij}(t)\right)$, vector output $\overline{y}^{ij}(t) = \left(y_1^{ij}(t),\dots,y_m^{ij}(t)\right)$ and weighted summation as an aggregation $\psi^{ij}(\overline{\omega},\overline{x}) = (\sum\omega_k\cdot x_{k1},\dots,\sum\omega_k\cdot x_{km})$. This variant will yield as its output the dot product of input data with the kernel of the weight coefficients $\overline{\omega}^{ij}(t) = (\omega_1^{ij}(t),\dots,\omega_n^{ij}(t))$. Thus, the final output values would be

$$\overline{y}^{ij}(t) = \left(y_1^{ij}(t),\dots,y_m^{ij}(t)\right) = \left(\sum_{k=\overline{1,n}}\omega_k^{ij}(t)x_{k1}^{ij}(t),\dots,\sum_{k=\overline{1,n}}\omega_k^{ij}(t)x_{km}^{ij}(t)\right).$$

As a result, general correction factors $\delta^{ij}(t)$ will be:

$$\delta^{ij}(t) = \begin{cases} \displaystyle\sum_r (y_r^{ij}(t) - e_r^{ij}(t)), & \text{for neurons with } \overline{e}^{ij};\ \text{denote them by } N_{\mathrm{Conv}\,e}^{ij};\\ \displaystyle\sum_r\sum_{\substack{l,p,k:\\ c_k^{lp}(t)=(i,j,r)}} \Delta_k^{lp}(t) & \text{for neurons without } \overline{e}^{ij};\ \text{denote them by } N_{\mathrm{Conv}}^{ij}. \end{cases}$$

We have used a double range of indices for the inputs $\overline{x}^{ij}(t) = \left(x_{11}^{ij}(t), \ldots, x_{nm}^{ij}(t)\right)$, which made it quite convenient to describe the aggregation operation. However, the application of a similar scheme for $\overline{c}^{ij}(t)$ and $\overline{\Delta}^{ij}(t)$ would break the compatibility with other neuron layers, which use the notation of models 1–4. As a result, we will represent them as vectors $\overline{c}^{ij}(t) = (c_1^{ij}(t), \ldots, c_{n\cdot m}^{ij}(t))$ and $\overline{\Delta}^{ij}(t) = (\Delta_1^{ij}(t), \ldots, \Delta_{n\cdot m}^{ij}(t))$, while binding them with inputs $\overline{x}^{ij}(t)$ ($k_1 = \overline{1,n}$ and $k_2 = \overline{1,m}$):

- when in the disconnected mode $c_{(k_1-1)m+k_2}^{ij} = (0,0,0) \longrightarrow x_{k_1k_2}^{ij} = 0$;
- for the external input $c_{(k_1-1)m+k_2}^{ij} = (0,0,r) \longrightarrow x_{k_1k_2}^{ij}(t) = X_r(t)$;
- in the standard mode $c_{(k_1-1)m+k_2}^{ij} = (l,p,r) \longrightarrow x_{k_1k_2}^{ij}(t) = y_r^{lp}(t)$ $(l < i)$;
- in the recurrent mode $c_{(k_1-1)m+k_2}^{ij} = (l,p,r) \longrightarrow x_{k_1k_2}^{ij}(t) = y_r^{lp}(t-1)$ $(l \geq i)$.

For backpropagation coefficients, we will have the corresponding formula:

$$\Delta_{(k_1-1)m+k_2}^{ij}(t) = \delta^{ij}(t) \cdot \omega_{k_1}^{ij}(t) \cdot \sigma(a^{ij}(t)), \quad \text{where } k_1 = \overline{1,n} \text{ and } k_2 = \overline{1,m}.$$

We will not be able to use general correction factors $\delta^{ij}(t)$ in the pure form for weights coefficients, as a result our formula for their update would be quite complicated:

$$\omega_k^{ij}(t+1) = \omega_k^{ij}(t) - \mu \sum_r x_{kr}^{ij} \begin{cases} (y_r^{ij}(t) - e_r^{ij}(t)), & \text{for } N_{\text{Conv}\,e}^{ij}; \\ \sum\limits_{\substack{l_1,l_2,p: \\ c_p^{l_1\,l_2}(t)=(i,j,r)}} \Delta_p^{l_1\,l_2}(t), & \text{for } N_{\text{Conv}}^{ij}. \end{cases} \tag{1.8}$$

All other formulas for $p^{ij}(t)$ and $\xi^{ij}(t)$ would be the same as in model 1.

Model 6. Linear rectification blocks B_{ReLu}^{ij} and $B_{\text{ReLu}\,e}^{ij}$ with an activation function $\varphi^{ij}(x^{ij}(t)) = \max(0, x^{ij}(t))$. We do not call this blocks of artificial neurons, because they do not have weight coefficients $\omega^{ij}(t)$ and an aggregation function. As a smooth approximation of a $\max(0,z)$ one can take $\varphi^{ij}(z) \approx (1/2\alpha)\log(1+e^{2\alpha z})$, which has the following derivative: $\dfrac{\partial\varphi}{\partial z} \approx \dfrac{1}{1+e^{-2\alpha z}}$. General correction coefficients $\delta^{ij}(t)$ are calculated by a formula similar to (1.1), while the backpropagation coefficient $\Delta_1^{ij}(t)$ is

$$\Delta_1^{ij}(t) = \delta^{ij}(t) \cdot \sigma(a^{ij}(t)) \,/\, \left(1 + e^{-2\alpha x^{ij}(t)}\right).$$

Model 7. Pooling layers B_{Pool}^{ij} and $B_{\text{Pool}\,e}^{ij}$ with a linear activation function $\varphi(z) = z$ and subsampling as an aggregation function $\psi(\overline{x^{ij}(t)}) = \max(x_1^{ij}(t), \ldots, x_n^{ij}(t))$. General correction coefficients will be calculated similarly to formula (1.1), while for the backpropagation coefficients $\Delta_k^{ij}(t)$ we will have

$$\Delta_k^{ij}(t) = \begin{cases} \delta^{ij}(t) \cdot \sigma(a^{ij}(t)), & \text{if } \psi(\overline{x^{ij}(t)}) = x_k^{ij}(t), \\ 0, & \text{if otherwise.} \end{cases}$$

Model 8. Gaussian blocks B_{norm}^{ij} and $B_{\text{norm}\,e}^{ij}$ with the normal activation function $\varphi^{ij}(z) = e^{-\beta z^2}$ and only one single data input. The derivative of this function is $\dfrac{\partial\varphi}{\partial z} = \varphi(z)(-2\beta\sqrt{\log(\beta) - \log(\varphi(z))})$. As a result, for a single backpropagation coefficient $\Delta_1^{ij}(t)$ we will have the following formula:

$$\Delta_1^{ij}(t) = -2\beta y^{ij}(t)\sqrt{\log(\beta) - \log(y^{ij}(t))}\,\delta^{ij}(t)\,\sigma(a^{ij}(t)).$$

General correction coefficients $\delta^{ij}(t)$ will use the same formula as in (1.1).

Model 9. Multiplication blocks B_*^{ij} and B_{*e}^{ij} with two data inputs x_1^{ij} and x_2^{ij}, a linear activation $\varphi(z) = z$ and an aggregation function $\psi(x_1, x_2) = x_1 \cdot x_2$. General correction coefficients $\delta^{ij}(t)$ will use formula analogous to (1.1) and the backpropagation coefficients:

$$\Delta_1^{ij}(t) = \delta^{ij}(t) \cdot x_2^{ij}(t) \cdot \sigma(a^{ij}(t)), \quad \Delta_2^{ij}(t) = \delta^{ij}(t) \cdot x_1^{ij}(t) \cdot \sigma(a^{ij}(t)).$$

Model 10. Summation blocks B_+^{ij} and B_{+e}^{ij} with two data inputs x_1^{ij} and x_2^{ij}, a linear activation $\varphi(z) = z$ and an aggregation $\psi(x_1, x_2) = x_1 + x_2$. General correction coefficients $\delta^{ij}(t)$ will use a formula analogous to (1.1) and the backpropagation coefficients

$$\Delta_k^{ij}(t) = \sigma(a^{ij}(t))\delta^{ij}(t).$$

Model 11. Tangent activator blocks B_{th}^{ij} and $B_{\text{th}\,e}^{ij}$ with a single input and hyperbolic tangent as an activation function $\varphi(z) = \tanh(z)$. General correction coefficients $\delta^{ij}(t)$ will use a formula analogous to (1.1) and the backpropagation coefficients

$$\Delta_k^{ij}(t) = \left(1 - (y^{ij}(t))^2\right) \sigma(a^{ij}(t))\, \delta^{ij}(t).$$

§ 2. Recurrent neurons with an integrated stack memory

For recurrent neurons without reference input $\overline{e}^{ij}(t)$, we will use the following scheme:

$$N_{\dots r}^{ij}(t) = \left(\overline{c}^{ij}(t), \overline{x}^{ij}(t), \overline{S_x}^{ij}(t), \overline{\omega}^{ij}(t), b^{ij}(t), \psi^{ij}, \varphi^{ij}, \overline{y}^{ij}(t), a^{ij}(t), \overline{\Delta}^{ij}(t), p^{ij}(t), \xi^{ij}(t)\right).$$

In this expression we introduce a stack memory $\overline{S_x}^{ij}(t)$ for those data inputs of neurons, which are connected to an external input source: $c_k^{ij}(t) = (0, 0, r)$, $x_k^{ij}(t) = X_r(t)$. The stack memory $\overline{S_x}^{ij}(t)$ will be a function according to a standard algorithm.

1. When $\sigma(a^{ij}(t)) = 0$, we make writing to $S_{x_k}^{ij}$ for all k, if $c_k^{ij}(t) = (0, 0, r)$ and $r \neq 0$:

$$\forall m = \overline{1, MaxM} \quad S_{x_k}^{ij}(m, t+1) = S_{x_k}^{ij}(m-1, t), \quad S_{x_k}^{ij}(0, t+1) = x_k^{ij}(t). \tag{2.1}$$

2. When $\sigma(a^{ij}(t)) = 1$, we make reading from $S_{x_k}^{ij}$ for all k, if $c_k^{ij}(t) = (0, 0, r)$ and $r \neq 0$:

$$\forall m = \overline{1, MaxM} \quad S_{x_k}^{ij}(m-1, t+1) = S_{x_k}^{ij}(m, t), \quad S_{x_k}^{ij}(MaxM, t+1) = 0. \tag{2.2}$$

After the inclusion of the stack memory, two formulas from standard neuron models should be updated with the highest priority: a formula for $y^{ij}(t)$ and $\delta^{ij}(t)$. We will consider their change with the example of the model 2 with hyperbolic tangent.

$$\delta^{ij}(t) = \left\{ \sum_{\substack{l,p,k:\ l>i \\ c_k^{lp}(t)=(i,j,1)}} \Delta_k^{lp}(t) + \sum_{\substack{l,p,k:\ l\leq i \\ c_k^{lp}(t)=(i,j,1)}} \Delta_k^{lp}(t-1) \right\}. \tag{2.3}$$

In general, we can guarantee that such scheme will implement the standard algorithm of backpropagation through time if we replace a formula for $y^{ij}(t)$ by this one:

$$y^{ij}(t) = \begin{cases} \varphi^{ij}\left(\sum_k \omega_k^{ij}(t) x_k^{ij}(t)\right), & \text{when } \sigma(a^{ij}(t)) = 0, \\ \varphi^{ij}\left(\sum\limits_{k:\, c_k^{ij}\neq(0,0,r)} \omega_k^{ij}(t) x_k^{ij}(t) + \sum\limits_{k:\, c_k^{ij}=(0,0,r)} \omega_k^{ij}(t) S_{x_k}^{ij}(0,t)\right), & \text{when } \sigma(a^{ij}(t)) = 1. \end{cases} \tag{2.4}$$

Taking into account the use of the stack memory, a transfer of the value $\Delta_k^{lp}(t-1)$ with a unit delay with $a^{ij}(t) = a^{ij}(t-1) > 0$ will be exactly the transfer of training data from the future (one stack of higher level), rather then from the past. Completely by analogy, we will change a formula for the weight coefficients $\omega_k^{ij}(t+1)$ to update:

$$\omega_k^{ij}(t+1) = \omega_k^{ij}(t) - \mu\left(1-(y^{ij}(t))^2\right)\delta^{ij}(t)\sigma(a^{ij}(t))\begin{cases} x_k^{ij}(t), & \text{if } c_k^{ij} \neq (0,0,r); \\ S_{x_k}^{ij}(0,t), & \text{if otherwise.} \end{cases} \tag{2.5}$$

At the same time, a formula for the backpropagation coefficients $\Delta_k^{ij}(t)$ will be (1.6), the same as in standard model, as well as a formula for bias update $b^{ij}(t+1)$. For the neurons $N_{\mathrm{th}\,re}^{ij}$ with reference inputs e^{ij}, we will also add the stack memory S_e^{ij}.

1. When $\sigma(a^{ij}(t)) = 0$, we make writing to S_e^{ij}:

$$\forall m = \overline{1, MaxM} \quad S_e^{ij}(m, t+1) = S_e^{ij}(m-1, t), \quad S_e^{ij}(0, t+1) = e^{ij}(t). \tag{2.6}$$

2. When $\sigma(a^{ij}(t)) = 1$, we make reading from S_e^{ij}:

$$\forall m = \overline{1, MaxM} \quad S_e^{ij}(m-1, t+1) = S_e^{ij}(m, t), \quad S_e^{ij}(MaxM, t+1) = 0. \tag{2.7}$$

Formulas for correction of coefficients $\delta^{ij}(t)$ of neurons $N_{\mathrm{th}\,re}^{ij}$ will be replaced by the following ones:

$$\delta^{ij}(t) = \left\{ (y^{ij}(t) - S_e^{ij}(0,t)) + \sum_{\substack{l,p,k:\ l \le i \\ c_k^{lp}(t) = (i,j,1)}} \Delta_k^{lp}(t-1) \right\}. \tag{2.8}$$

If a neuron $N_{\mathrm{th}\,re}^{ij}$ is allowed to have connections to external data inputs $X_r(t)$, then it will use formulas (2.4) and (2.5), while otherwise standard formulas from model 2 are used.

As a result, all recurrent neurons will have only one additional parameter:

- $MaxM$ — depth of stack memory.

Without any fundamental differences a recurrent mode could be introduced to all the other standard neuron models with inclusion of the stack memory. To prepare our recurrent network for training on $m \le MaxM$ etalon values, one will have to supply this data values $\overline{X}(t), \ldots, \overline{X}(t+m)$ with corresponding reference values $e(t), \ldots, e(t+m)$, while holding the $a^{ij}(t) = \ldots = a^{ij}(t+m) = 0$. To complete one full cycle of training, we will need to turn the training signal high and hold it for an additional m time steps $a^{ij}(t+m+1) = \ldots = a^{ij}(t+2m) = 1$.

§ 3. Neurons with adjustable connections

The general idea of our neural link adjustment is to remove those connections that are almost out of use and create new connections with the most active neurons of the previous layers, provided the training with current connections can lead to a paralysis of weight coefficients or to a fluctuation of their values near the local minimum.

Algorithm of a new link creation. On each iteration t of the neuron $N_{\ldots c}^{ij}$ we make the following steps:

Step 1. Check if a neuron is unable to fix $\delta^{ij}(t)$ with current $x_k^{ij}(t)$, even if it raises all the weight coefficients almost to $\omega_{\max}$ (we choose $0.7\,\omega_{\max}$ as a control value):

$$C_{new_1}^{ij}(t) = 1, \text{ if } \left| y(t) - \varphi\left(b^{ij}(t) - \mathrm{sign}(\delta^{ij}(t)) \sum_k x_k^{ij}(t)\, 0.7\,\omega_{\max} \right) \right| < \left|\delta^{ij}(t)\right|.$$

If this event is detected ($C^{ij}_{new_1}(t) = 1$) and the training signal is active ($a^{ij}(t) = 1$), then we will go to step 2. Otherwise, we need to verify additionally that a sign of $\delta^{ij}(t)$ and $\delta^{ij}(t-1)$ is different, while $x^{ij}_k(t)$ and $x^{ij}_k(t-1)$ are almost identical:

$$C^{ij}_{new_2}(t) = 1, \quad \text{if } \delta^{ij}(t-1)\,\delta^{ij}(t) < 0 \quad \text{and} \quad \sum_k |x^{ij}_k(t) - x^{ij}_k(t-1)| < 0.1\,n\,x_{\max}. \tag{3.1}$$

If this event is detected ($C^{ij}_{new_2}(t) = 1$) and the training signal is active ($a^{ij}(t) = 1$), then we will go to step 2. Otherwise, we restart the algorithm and wait for a next iteration.

Step 2. Among all the connections of our neuron $N^{ij}_{\dots c}$, we are looking for an empty one: $c^{ij}_k(t) = (0,0,0)$. If we managed to find some suitable k, then we will proceed to step 3. Otherwise, we restart the algorithm and wait for a next iteration.

Step 3. With a probability P_{deep1} we will go to step 4. If a transition to step 4 was not carried out, then we will search for a suitable candidate $y^{i-1\,p}_r$ for a new connection from previous $i-1$ layer of a neural net. We will carry out this selection according to the following conditions:

- there is no current connection to $y^{i-1\,p}_r$ from $N^{ij}_{\dots c}$: $\nexists\, \acute{k} : c^{ij}_{\acute{k}}(t) = (i-1, p, r)$;
- among all the admissible candidates, we select the maximum modulo: $\max |y^{i-1\,p}_r(t)|$;
- if more than one $y^{i-1\,p}_r$ was found, then we will choose any random one of them.

If we managed to find some suitable p and r, then we will go directly to the final step 5; otherwise, we will go to step 4 first.

Step 4. If our neuron is $N^{ij}_{\dots r\,c}$ from a recurrent layer, then with a probability P_{rec} we will try to create a recurrent connection; otherwise, we will try to create a deep connection with some distant neural layers. Among all the y^{lp}_r ($l < i-1$ for a direct one and $l \geq i$ for a recurrent one), we choose such one that the following conditions hold:

- there is no current connection from $N^{ij}_{\dots c}$: $\nexists\, \acute{k} : c^{ij}_{\acute{k}}(t) = (l, p, r)$ and $(l, p) \neq (i, j)$;
- we select the maximum modulo: $\max |y^{lp}_r(t-1)|$ for a recurrent one; $\max |y^{lp}_r(t) \cdot 2^{-|l-i|}|$ for a direct[1] one;
- if more than one y^{lp}_r was found, then we will choose any random one of them.

If we managed to find some suitable l, p, r, then we will go to the final step 5.

Step 5. For the chosen k and y^{lp}_r, we assume $c^{ij}_k(t+1) = (l, p, r)$ and perform an initialization: $\omega^{ij}_k(t+1) = \omega_{\min}$, if $\delta^{ij}(t) \leq 0$, and $\omega^{ij}_k(t+1) = -\omega_{\min}$, if $\delta^{ij}(t) > 0$.

Algorithm of a redundant link deletion. On each iteration of a neuron we make the following steps:

Step 1. Check if a neuron $N^{ij}_{\dots c}$ has $|\omega^{ij}_k|$ lower than $\omega_{\min}$ during t_o iterations:

$$C^{ij}_{del\,k}(t) = 1, \quad \text{if } \left| \sum_{t-t_o \leq \tau \leq t} \left(|\omega^{ij}_k(\tau)| - \omega_{\min} \right) \cdot \sigma(a^{ij}(\tau)) \right| < 0.$$

Step 2. Delete all connections with $C^{ij}_{del\,k}(t) = 1$, assuming $c^{ij}_k(t+1) = (0,0,0)$. An exception to this rule will be a connection to the external data source $c^{ij}_k(t) = (0, 0, r)$, and also the previously deleted one $c^{ij}_k(t) = (0,0,0)$, for which our algorithm of a new link creation has found l, p, r on the current iteration $c^{ij}_k(t+1) = (l, p, r)$.

Compared with previous models, we add the following parameters:

- $x_{\max}$ — maximum absolute value for input data of the neuron;

[1] We use the factor $2^{-|l-i|}$ in order to ensure the priority creation of links with a close layers.

- t_o — control time for an old link deletion;
- P_{deep1} — probability of creating a deep link bypassing a previous layer;
- P_{rec} — probability of a new deep link to be a recurrent one.

We can suggest the following general guidelines for selecting these parameters. The choice of a probability P_{rec} very much depends on the desired topology of the neural network. Values of $P_{rec} > 0.5$ are selected if the network has a hybrid architecture and prefers creating recurrent connections, while $P_{rec} < 0.5$ is chosen if, on the contrary, it is preferable to create deep direct links. For a P_{deep1} probability, large values should not be chosen, preferably $P_{deep1} \leq 0.2$. The control time t_o should be chosen small enough $t_o \approx 3 \ldots 5$, because otherwise a neuron may fail to reorganize connections in time, which can lead to a paralysis of weight coefficients. The choice of $x_{\max}$ depends heavily on how input data of neurons was normalized. Most often, we assume $x_{\max} = 1$.
For the optimal link creation, it is advisable to alternate a supply of training examples in all possible variants of their sequential submission to a neural network. In theory, an algorithm with adaptive connection readjustment can solve the problem with overfitting in deep neural networks. The idea is that it should start its operation almost completely devoid of any connections and with a maximally generalizing output function. New links are added during the course of training, which leads to a gradual decrease in a degree of generalization of training examples.

§ 4. Examples of building a neural network system

All the neural networks from our examples will have:

- external data inputs $\overline{X}(t) = (X_1(t), \ldots, X_n(t))$;
- external data outputs $\overline{Y}(t) = (Y_1(t), \ldots, Y_m(t))$;
- reference inputs $\overline{E}(t) = (E_1(t), \ldots, E_m(t))$;
- a general training control signal $a(t)$;
- a general detection of local minimum $\xi(t)$;
- a general detection of paralysis $p(t)$.

The training control signal $a(t)$ will be applied to all the neurons N^{ij} from our network as $a^{ij}(t) = a(t)$. The general detection signals $\xi(t)$ and $p(t)$ will be constructed with a logical disjunction $\xi(t) = \vee\xi^{ij}(t)$ and $p(t) = \vee p^{ij}(t)$.

Example 1. Long sort-term memory network with integrated training. Our LSTMIT (LSTM + Integrated Training) network will consist of 9 layers ($j = \overline{1,m}$):

1. $N^{1j}_{\text{th}\,r}$ — recurrent input layer. According to our notation this neurons will use equations from model 2 partially replaced by (2.1)–(2.5).
2. $N^{2j}_{\sigma\,r}$ — recurrent input gates. They will use equations from model 1 changed by analogy to (2.1)–(2.5).
3. B^{3j}_{*} — multiplier blocks (see model 9).
4. $N^{4j}_{\sigma\,r}$ — recurrent forget gates. They are completely analogous to input gates.
5. $B^{5j}_{*\,r}$ — recurrent multiplier blocks. Will use equations of model 9 changed by analogy with (2.1)–(2.3)
6. B^{6j}_{+} — summation blocks (see model 10).
7. B^{7j}_{th} — tangent activation blocks (see model 11).
8. $N^{8j}_{\sigma\,r}$ — recurrent output gates. They are fully analogous to gates of layer 2 and 4.
9. $B^{9j}_{*\,re}$ — recurrent output multiplier blocks with reference inputs $e^{9\,j}(t) = E_j(t)$. They will use equations of model 10, changed by analogy with (2.6)–(2.8).

Since we have already described the operation of all the models considered, it would be sufficient for a full description to define only static connections c^{ij} for all the layers.

- Gate and input layers: $c_k^{1\,j} = c_k^{2\,j} = c_k^{4\,j} = c_k^{8\,j} = \begin{cases} (0,0,k), & \text{if } k = \overline{1,n}, \\ (9,k,1), & \text{if } k = \overline{n+1,n+m}. \end{cases}$
- Multiplier blocks $B_*^{3\,j}$ of layer 3: $c_1^{3\,j} = (1,j,1)$ and $c_2^{3\,j} = (2,j,1)$.
- Multiplier blocks $B_{*r}^{5\,j}$ of layer 5: $c_1^{5\,j} = (4,j,1)$ and $c_2^{5\,j} = (6,j,1)$.
- Summation blocks $B_+^{6\,j}$ of layer 6: $c_1^{6\,j} = (3,j,1)$ and $c_2^{6\,j} = (5,j,1)$.
- Tangent blocks $B_{\text{th}}^{7\,j}$ of layer 7: $c_1^{7\,j} = (6,j,1)$.
- Multipliers $B_{*re}^{9\,j}$ of the last layer: $c_1^{9\,j} = (7,j,1)$ and $c_2^{9\,j} = (8,j,1)$.

To prepare LSTMIT network for training on $m \le MaxM$ etalon values, we will have to supply this data values $\overline{X}(t), \ldots, \overline{X}(t+m)$ with corresponding reference values $\overline{E}(t), \ldots, \overline{E}(t+m)$, while holding the control signal low: $a(t) = \ldots = a(t+m) = 0$. To complete one full cycle of training we will need to turn the training signal high and hold it for an additional m time steps $a(t+m+1) = \ldots = a(t+2m) = 1$.

Neurons with adjustable connections could be used for models, composed of many LSTMIT networks. However, this should be done only to LSTMIT networks without external data connections and only to their first layers, changing $N_{\text{th}\,r}^{1\,j}$ to $N_{\text{th}\,r\,c}^{1\,j}$ and assuming $P_{deep1} = P_{rec} = 0$. At the initial time at least half of the direct links of this layer $N_{\text{th}\,r\,c}^{1\,j}$ should be disabled to suppress the overfitting. In addition to this, we will also have to forbid a deletion of recurrent links, in order to prevent a disruption of a base LSTM logic.

Remark 2. The main difference of our LSTMIT from classical LSTM is in the introduction of the stack memory $S_{x_k}^{i\,j}$ for n external data inputs of $N_{\text{th}\,r}^{1\,j}$, $N_{\sigma r}^{2\,j}$, $N_{\sigma r}^{4\,j}$ and $N_{\sigma r}^{8\,j}$, as well as the stack memory $S_e^{9\,j}$ for external reference inputs of $B_{*re}^{9\,j}$. It is important to note, that for a model composed of many consequential LSTMIT networks a stack memory is not required for inner LSTMIT without external connections.

Example 2. Radial basis functions with integrated training. This RBFIT network will consist of three layers ($j = \overline{1,m}$):

1) $N_{\text{Ed}}^{1\,j}$ — Euclidean distance neurons;
2) $B_{\text{norm}}^{2\,j}$ — Gaussian activators;
3) $N_{\text{id}\,e}^{3\,1}$ — single linear neuron with reference input $e(t)$.

All static connections are very simple and straightforward:

- for all the input neurons $N_{\text{Ed}}^{1\,j}$ we have: $c_k^{1\,j} = (0,0,k)$, where $k = \overline{1,n}$;
- Gaussian activators $B_{\text{norm}}^{2\,j}$ are linked directly: $c_1^{2\,j} = (1,j,1)$, where $j = \overline{1,m}$;
- $N_{\text{id}\,e}^{3\,1}$ is connected to all the second layer: $c_k^{3\,1} = (2,k,1)$, where $k = \overline{1,m}$.

It is possible to use $N_{\text{id}\,c\,e}^{3\,1}$ with adjustable connections instead of $N_{\text{id}\,e}^{3\,1}$. In this variant we will connect it at the initial time to at least half of the neurons from layer 2, and assume $P_{deep1} = P_{rec} = 0$. As a result, our network will start with a high degree of input generalization and will gradually decrease it during a training process.

Example 3. Recurrent radial basis network for chaotic series ($j = \overline{1,m}$):

1) $B_{+r}^{1\,j}$ — recurrent summation blocks;

2) N_{Ed}^{2j} — Euclidean distance neurons;

3) B_{norm}^{3j} — Gaussian activators;

4) $N_{\mathrm{th}\,re}^{41}$ — hyperbolic tangent neuron with reference input $e(t)$.

This network has only one data input $X_1(t)$, and all static connections are very simple:

- for the first recurrent layer $c_1^{1j} = (0,0,1)$ and $c_2^{1j} = (4,1,1)$ for all $j = \overline{1,m}$;
- second layer neurons N_{Ed}^{2j} are linked directly: $c_1^{2j} = (1,j,1)$, where $j = \overline{1,m}$;
- Gaussian blocks B_{norm}^{3j} are also linked directly: $c_1^{3j} = (2,j,1)$, where $j = \overline{1,m}$;
- $N_{\mathrm{th}\,re}^{41}$ is connected to all the third layer: $c_k^{41} = (3,k,1)$, where $k = \overline{1,m}$.

The stack memory $S_{x_1}^{1j}(t)$ for data inputs will be used only for blocks B_{+r}^{1j} and the stack memory for reference inputs $S_e^{41}(t)$ would be used only for a single neuron $N_{\mathrm{th}\,re}^{41}$. A training algorithm for the approximation of chaotic sequence $Y(0), Y(1), \ldots, Y(n)$ will be composed of two steps. At first we will send $X_1(t_0) = Y(0)$, $X_1(t_0+\tau) = 0$ to data inputs and $E_1(t_0) = Y(1)$, $E_1(t_0+\tau) = Y(\tau+1)$ to reference inputs, while holding the training signal low: $a(t_0+\tau) = 0$ for $\tau = \overline{1,n-1}$. After that we will set the training signal to one and wait additional n cycles: $a(t_0+\tau) = 1$ for $\tau = \overline{n,2n}$.

It is possible to use $N_{\mathrm{th}\,rce}^{41}$ with adjustable links instead of $N_{\mathrm{th}\,re}^{41}$. In this variant we will connect it at the initial time to only one neuron from layer 2, and assume $P_{deep1} = P_{rec} = 0$. As a result parallel approximation branches will be added only when they are necessary, which will gradually increase the probability of successful training.

Example 4. Convolutional neural networks CONVIT with integrated training. In base variant this network consists of:

1) N_{Conv}^{1j} — convolutional layer, where $j = \overline{1,n_1}$ and all neurons have m_1 outputs;

2) B_{ReLu}^{2j} — linear rectification layer, where $j = \overline{1,n_1 m_1}$;

3) B_{Pool}^{3j} — pooling layer, where $j = \overline{1,n_1 m_2}$ and $m_2 \ll m_1$;

4) $N_{\sigma e}^{4j}$ — output sigmoid layer, where $j = \overline{1,n_1 m_3}$ and $m_3 \ll m_2$.

In general case, we assume that the input data is I_{lwh} in the form of a 3D matrix ($h = 1$ for monochrome and $h = 3$ for colour). We transform this three-dimensional matrix to a vector form according to a standard algorithm:

$$X_{\alpha+l(\beta-1)+lw(\gamma-1)} = i_{\alpha\beta\gamma}, \quad \alpha = \overline{1,l} \quad \beta = \overline{1,w} \quad \gamma = \overline{1,h}.$$

In addition to this, we will need to introduce the following three auxiliary functions for working with indices ($x \operatorname{rest} y$ is the remainder of x divided by y; $[x]$ is the integer part of x):

$$\theta(x,y) = \begin{cases} y, & \text{if } y|x, \\ x \operatorname{rest} y, & \text{if otherwise;} \end{cases}$$

$$\lambda^+(x,y) = \begin{cases} [x/y], & \text{if } y|x, \\ [x/y]+1, & \text{if otherwise;} \end{cases}$$

$$\lambda^-(x,y) = \begin{cases} [x/y]-1, & \text{if } y|x, \\ [x/y], & \text{if otherwise.} \end{cases}$$

We assume that convolutional neurons $N^{1\,j}_{\text{Conv}}$ will read the input data with a sliding window of size $f \cdot f$ from each slice of a three-dimensional matrix $I_{l\,w\,h}$. Also, we restrict that this window will move along the image with a unit step $st = 1$. After a convolution operation, we will get $m_1 = l_1 \cdot w_1 = (l - f + 1) \cdot (w - f + 1)$ data at the outputs of each convolutional neurons. Then these values will pass through the linear rectification blocks and enter the pooling layer. We choose a pooling window to be of size $g \cdot g$, which finally yields $m_1 = g^2\, m_2 = g^2\, l_2\, w_2$ and $l_2 = l_1/g$, $w_2 = w_1/g$. In this case, the organization of network connections can be carried out as follow:

- Every $N^{1\,j}_{\text{Conv}}$ will have $c^{1\,j}_{(\alpha-1)m_1+\beta}(t) = (0, 0, k)$, where $\alpha = \overline{1, f^2 h}$, $\beta = \overline{1, m_1}$ and

  $$k = \theta(\alpha, f) + l\left\{\lambda^+(\alpha, f) - 1\right\} + lw\left\{\lambda^-(\alpha, f^2)\right\} + l\left\{\theta(\beta, w - f + 1) - 1\right\} + \lambda^-(\beta, w - f + 1).$$

  Such large number of terms is responsible for five types of transitions when reading data with a sliding window $f \cdot f$. The term $\theta(\alpha, f)$ is responsible for a movement along a separate column of the sliding window, $l\left\{\lambda^+(\alpha, f) - 1\right\}$ is included for a transition between this columns, and the term $lw\left\{\lambda^-(\alpha, f^2)\right\}$ for a transition between separate h colour layers. By analogy, $l\left\{\theta(\beta, w - f + 1) - 1\right\}$ is responsible for a horizontal shift of the sliding window and the final term $\lambda^-(\beta, w - f + 1)$ for its vertical shift.

- Second layer $B^{2\,j}_{\text{ReLu}}$ is linked directly $c^{2\,j}_1 = (1, \alpha, \beta)$, where:

  $$j = (\alpha - 1)m_1 + \beta \quad \text{and} \quad \alpha = \overline{1, n_1},\ \beta = \overline{1, m_1}.$$

- For $B^{3\,j}_{\text{Pool}}$ we have $c^{3j}_k = (2, \zeta, 1)$, $j = (\alpha - 1)m_2 + \beta$ and $\alpha = \overline{1, n_1}$, $\beta = \overline{1, m_2}$,

  $$\zeta = (\alpha - 1)m_1 + \theta(k, g) + l_1\left\{\lambda^+(k, g) - 1\right\} + g\, l_1\left\{\theta(\beta, w_2) - 1\right\} + g\,\lambda^-(\beta, w_2), \quad k = \overline{1, g^2}.$$

  The first term in this expression $(\alpha-1)m_1$ is responsible for a transition between n_1 information channels (from n_1 neurons of the first layer). The next term $\theta(k, g)$ denotes the movement along the columns of the sliding window $g \cdot g$. By analogy, $g\, l_1\left\{\theta(\beta, w_2) - 1\right\}$ is responsible for a horizontal shift of the sliding window and $g\,\lambda^-(\beta, w_2)$ is used for its vertical shift.

- The layer $N^{4\,j}_{\sigma e}$ is connected as: $c^{4j}_k = (3, k, 1)$ and $k = \overline{1, n_1 m_2}$, $j = \overline{1, n_1 m_3}$.

For a considered basic architecture of a convolutional network, each layer is, in fact, performing some manipulation over three-dimensional data. In particular, the first layer has $h \cdot l \cdot w$ input values, and outputs $n_1 \cdot l_1 \cdot w_1$ values to the next layer, which are reduced by pooling layer to $n_1 \cdot l_2 \cdot w_2$. The main option of scaling such network is to connect successive layers of pooling and convolution. In this case, the convolutional layers will increase the depth of the three-dimensional data $h < n_1 < n_2 < \ldots$, and, at the same time, will reduce the length and width of data $l > l_1 > \ldots$ and $w > w_1 > \ldots$, while the pooling layers will only reduce the length and width without altering the depth of data.

It is possible to use $N^{4\,j}_{\sigma\, c\, e}$ with adjustable connections instead of $N^{4\,j}_{\sigma\, e}$. In this variant we will connect it at the initial time to at least half of the neurons from layer 3, and assume $P_{deep1} = P_{rec} = 0$. As a result, a network will start with a high degree of input generalization and will gradually decrease it during the training process. The comparison with dropout algorithm from [17] on MNIST data set is presented in a table below. The overall time for a training of convit networks was constraint to not exceed the corresponding training time for dropout networks for more then 20 %.

Considering the high effectiveness of dropout algorithm we can also add its support in our neuron models. For this we will have to add some variables to store a dropout state:

- $\overline{r}^{\,i\,j}(t) = (r^{i\,j}_1(t), \ldots, r^{i\,j}_k(t))$ — dropout state values for outputs of $N^{i\,j}_{\ldots}(t)$.

Table 1. Comparison with dropout networks on MNIST data set

Method	Unit type	Architecture	Error %	Time
Dropout NN	Logistic	3 layers, 1024 units	1.35	t_1
Convit NN	Logistic	3 layers, 1024 units	1.38	$\leq 1.2\,t_1$
Dropout NN + max-norm	RELU	3 layers, 1024 units	1.06	t_2
Convit NN + max-norm	RELU	3 layers, 1024 units	1.1	$\leq 1.2\,t_2$

Updating this variables will be according to a standard rule (p^{ij} is included in neuron parameters):

$$\forall\, i,j,l \quad r_l^{ij}(t) \sim \text{Bernoulli}(p^{ij}).$$

For example, if we would like to incorporate dropout in model 1, then we will have to change only the formulas for the output $y^{ij}(t)$ and general correction factors $\delta^{ij}(t)$:

$$y^{ij}(t) = \begin{cases} \varphi^{ij}\left(\sum \omega_k^{ij}(t)\cdot x_k^{ij}(t) + b^{ij}(t)\right), & \text{if } a^{ij}(t)=0; \\ r^{ij}(t)\cdot\varphi^{ij}\left(\sum \omega_k^{ij}(t)\cdot x_k^{ij}(t) + b^{ij}(t)\right), & \text{if } a^{ij}(t)=1. \end{cases}$$

$$\delta^{ij}(t) = \begin{cases} (y^{ij}(t) - e^{ij}(t)), & \text{for neurons with } e^{ij};\ \text{denote them by } N_{\sigma e}^{ij}; \\ \sum\limits_{\substack{l,p,k:\\ c_k^{lp}(t)=(i,j,1)}} \Delta_k^{lp}(t)\, r^{ij}(t), & \text{for neurons without } e^{ij};\ \text{denote them by } N_{\sigma}^{ij}. \end{cases}$$

The integration of dropout in all the other neuron models will follow a similar scheme with a sole exception of convolutional neurons. For them, we will have to use a dot product for vector output $\overline{y}^{ij}(t)$ and also incorporate dropout coefficients $r_p^{ij}(t)$ in formula (1.8) as:

$$\omega_k^{ij}(t+1) = \omega_k^{ij}(t) - \mu \sum_m x_{km}^{ij} \begin{cases} (y_m^{ij}(t) - e_m^{ij}(t)), & \text{for } N_{\text{Conv}\,e}^{ij}; \\ \sum\limits_{\substack{l_1,l_2,p:\\ c_p^{l_1\,l_2}(t)=(i,j,r)}} \Delta_p^{l_1\,l_2}(t)\, r_m^{ij}(t), & \text{for } N_{\text{Conv}}^{ij}. \end{cases}$$

Example 5. Multilayer perceptron PERCIT with integrated training and link adjustment. In a basic configuration, this neural network consists of k layers:

1. $N_{\sigma}^{1\,j}$ — input layer, where $j = \overline{1,n_1}$ and all neurons have n data inputs.
2. $N_{\sigma\,c}^{2\,j}$ — layer with adjustable links, where $j = \overline{1,n_2}$ and neurons have n_1 inputs.

$k-1$. $N_{\sigma\,c}^{k-1\,j}$ — adjustable layer, where $j = \overline{1,n_{k-1}}$ and all neurons have n_{k-2} inputs.

k. $N_{\sigma\,c\,e}^{k\,j}$ — output layer, where $j = \overline{1,n_k}$ and all neurons have n_{k-1} inputs.

The initial connections will be organized as follow:

— all neurons $N_{\sigma}^{1\,j}$ are connected to all the external inputs $c_k^{1\,j} = (0,0,k)$, $k = \overline{1,n}$;

— for all other layers, we set 75% of all connections to be blank $c_k^{ij} = (0,0,0)$, and other 25 % to be linked to random neurons from the previous layer.

For the adjustable neurons, we set the following parameters: the probability of a recurrent connection $P_{rec} = 0$, the probability of creating a deep link bypassing the previous layer $P_{deep_1} = 0.1$, the control time for deleting the old links $t_o = 4$ and the maximum absolute value for input data $x_{\max} = 1$. For deep neural networks, one of the main problems of training them with gradient methods is the vanishing gradient problem. However, if we allow the creation of deep links with a 10 % probability, then we will significantly reduce this effect by passing through the error via several layers.

COMPUTER SCIENCE 2018. Vol. 28. Issue 2

Remark 3. For the optimal creation of new connections, it is advisable to alternate the supply of training examples in all possible variants of their sequential submission. In this case the condition (3.1) will be used to its full potential.

Results and Discussion

An important methodological advantage of our approach is standardization with the development of a universal general formalism for a broad range of neuron models. First of all, this greatly simplifies an integration of any new models with other activation functions or aggregation of the input data. Secondly, our approach enables us to construct a hierarchical networks $Nnet_1, \ldots, Nnet_k$, where each $Nnet_j$ is controlling the training process of the next network $Nnet_{j+1}$. For example, the first network $Nnet_1$ could be trained to spot some basic visual stimuli in video data, which will be used to issue training command for a much bigger second network $Nnet_2$. On its part this network $Nnet_2$ could be trained with the assistance of the $Nnet_1$ to spot a more complex training stimuli, maybe not only in the video data, but in the additional audio data supplied (like verbal command: "train please") and learn to associate the corresponding data and issue the training signal for the next one $Nnet_3$, and so on.

All of the basic neuron models considered can be easily generalized by switching from standard stochastic gradient descent to a more advanced algorithm. For example, one can integrate stochastic descent with momentum in that models, or stochastic descent with adaptive momentum estimation.

REFERENCES

1. Dreyfus S.E. Artificial neural networks, back propagation, and the Kelley–Bryson gradient procedure, *Journal of Guidance, Control and Dynamics*, 1990, vol. 13, no. 5, pp. 926–928. DOI: 10.2514/3.25422
2. Broomhead D.S., Lowe D. Multivariable functional interpolation and adaptive networks, *Complex Systems*, 1988, vol. 2, pp. 321–355. http://sci2s.ugr.es/keel/pdf/algorithm/articulo/1988-Broomhead-CS.pdf
3. Lecun Y., Bottou L., Bengio Y., Haffner P. Gradient-based learning applied to document recognition, *Proceedings of the IEEE*, 1998, vol. 86, issue 11, pp. 2278–2324. DOI: 10.1109/5.726791
4. Greff K., Srivastava R.K., Koutnik J., Steunebrink B.R., Schmidhuber J. LSTM: A search space odyssey, *IEEE Transactions on Neural Networks and Learning Systems*, 2017, vol. 28, issue 10, pp. 2222–2232. DOI: 10.1109/TNNLS.2016.2582924
5. Chen G. A gentle tutorial of recurrent neural network with error backpropagation, 2016, arXiv: 1610.02583v3 [cs]. https://arxiv.org/pdf/1610.02583.pdf
6. Krizhevsky A., Sutskever I., Hinton G.E. ImageNet classification with deep convolutional neural networks, *Communications of the ACM*, 2017, vol. 60, issue 6, pp. 84–90. DOI: 10.1145/3065386
7. Girshick R., Donahue J., Darrell T., Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014, arXiv: 1311.2524v5 [cs]. https://arxiv.org/pdf/1311.2524.pdf
8. Park J., Sandberg I.W. Universal approximation using radial-basis-function networks, *Neural Computation*, 1991, vol. 3, issue 2, pp. 246–257. DOI: 10.1162/neco.1991.3.2.246
9. Pham V., Bluche T., Kermorvant C., Louradour J. Dropout improves recurrent neural networks for handwriting recognition, 2013, arXiv: 1312.4569v2 [cs]. https://arxiv.org/pdf/1312.4569.pdf
10. Graves A. Generating sequences with recurrent neural networks, 2014, arXiv: 1308.0850v5 [cs]. https://arxiv.org/pdf/1308.0850.pdf
11. Sutskever I., Vinyals O., Le Q.V. Sequence to sequence learning with neural networks, 2014, arXiv: 1409.3215v3 [cs]. https://arxiv.org/pdf/1409.3215.pdf
12. Sak H., Senior A., Beaufays F. Long short-term memory recurrent neural network architectures for large scale acoustic modeling, *Proceedings of the Annual Conference of the International Speech Communication Association*, Singapore, 2014, pp. 338–342. https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43905.pdf
13. Fan Y., Qian Y., Xie F., Soong F.K. TTS synthesis with bidirectional LSTM based recurrent neural networks, *Proceedings of the Annual Conference of the International Speech Communication Association*, Singapore, 2014, pp. 1964–1968. https://pdfs.semanticscholar.org/564f/ed868f652f361bb3e345f6f94073d8f6f261.pdf

14. Donahue J., Hendricks L.A., Guadarrama S., Rohrbach M., Venugopalan S., Saenko K., Darrell T. Long-term recurrent convolutional networks for visual recognition and description, 2016, arXiv: 1411.4389v4 [cs]. https://arxiv.org/pdf/1411.4389.pdf
15. Nazarov M.N. Artificial neural network with modulation of synaptic coefficients, *Vestn. Samar. Gos. Tekhn. Univ., Ser. Fiz.-Mat. Nauki*, 2013, vol. 2, no. 31, pp. 58–71. DOI: 10.14498/vsgtu1052
16. Maslennikov O.V., Nekorkin V.I. Adaptive dynamical networks, *Physics-Uspekhi*, 2017, vol. 60, no. 7, pp. 694–704. DOI: 10.3367/UFNe.2016.10.037902
17. Srivastava N., Hinton G., Krizhevsky A., Sutskever I., Salakhutdinov R. Dropout: A simple way to prevent neural networks from overfitting, *Journal of Machine Learning Research*, 2014, vol. 15, pp. 1929–1958. http://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

Received 22.05.2018

Nazarov Maxim Nikolaevich, Senior Lecturer, Department of Higher Mathematics 1, National Research University of Electronic Technology, pl. Shokina, 1, Zelenograd, Moscow, 124498, Russia
E-mail: nazarov-maximilian@yandex.ru

М. Н. Назаров

Нейронные сети с динамическими коэффициентами и перестраиваемыми связями на основе интегрированного обратного распространения

Цитата: Вестник Удмуртского университета. Математика. Механика. Компьютерные науки. 2018. Т. 28. Вып. 2. С. 260–274.

Ключевые слова: искусственные нейроны, обратное распространение ошибки, адаптивная перестройка связей, рекуррентные нейронные сети.

УДК 519.68, 007.5

DOI: 10.20537/vm180212

Рассматриваются искусственные нейроны, чьи весовые коэффициенты будут изменяться по специальному закону, основанному на интегрированном в их модели обратном распространении. Для этого коэффициенты погрешности обратного распространения вводятся в явном виде во все модели нейронов и осуществляется передача их значений вдоль межнейронных связей. В дополнение к этому вводится специальный тип нейронов с эталонными входами, которые будут выступать в качестве основного источника первичной оценки погрешности для всей нейронной сети. В последнюю очередь вводится контрольный сигнал для запуска обучения, который будет управлять процессом передачи коэффициентов погрешности и корректировкой весов нейронов. Для рекуррентных нейронных сетей демонстрируется как провести интеграцию обратного распространения во времени в их формализм с помощью стековой памяти для внешних входов нейронов. Дополнительно к этому рассматриваются примеры как формализовать в рамках данного подхода такие популярные нейронные сети, как сети долгой кратковременной памяти, сети радиально-базисных функций, многослойные перцептроны и сверточные нейронные сети. Основным практическим следствием данного подхода является возможность описания нейронных сетей с перестраиваемыми связями на основе интегрированного алгоритма обратного распространения.

Поступила в редакцию 22.05.2018

Назаров Максим Николаевич, старший преподаватель, кафедра высшей математики 1, Национальный исследовательский университет «МИЭТ», 124498, г. Москва, г. Зеленоград, площадь Шокина, 1.
E-mail: nazarov-maximilian@yandex.ru