2006 Autumn semester Pattern Information Processing
Topic 4. Pattern recognition

Session 11. (2) Neural networks and learning algorithm

Neuron and neural network

The research on artificial neural networks started from the modeling of biological neuron. From the physiological point of view, a typical model in the following was proposed:

1. A neuron accepts electrical and/or chemical stimuli from other neurons.
2. If the total amount of accepted stimuli exceeds a threshold, the neuron is excited and distributes stimuli to connected neurons.

This neuron model is schematically illustrated in Fig. 1. Let us assume that x_i, the status of the i-th neuron, is either 1 (excited) or 0 (not excited), and that the amount of stimulus distributed to each of connected neurons is unity. A weight coefficient is assigned to the interconnection between neurons, and the stimulus from a connected neuron is magnified by the corresponding weight coefficient. Formally, Let x_j be the status of the j-th neuron, and w_{ji} be the weight coefficient on the interconnection between the j-th neuron and the i-th neuron. The amount of the stimulus accepted by the i-th neuron from the j-th neuron is $w_{ji}x_j$, and the i-th neuron is excited if the total amount of accepted stimuli, $\sum_j w_{ji}x_j$, exceeds a certain threshold. Consequently, Let T be the threshold and f be the threshold function defined as follows:

$$f(x) = \begin{cases} 1 & x \geq T, \\ 0 & x < T. \end{cases} \tag{1}$$

The i–th neuron is excited, i. e. $x_i = 1$, if $f(\sum_j w_{ji}x_j) = 1$; $x_i = 0$ otherwise.

The information processing by the neural network is performed by the interconnection of many neurons. The input information is given by the initial status of the neurons, the status of each neuron is modified by interchanging the stimuli, and the output information is presented by the final status of the neurons.

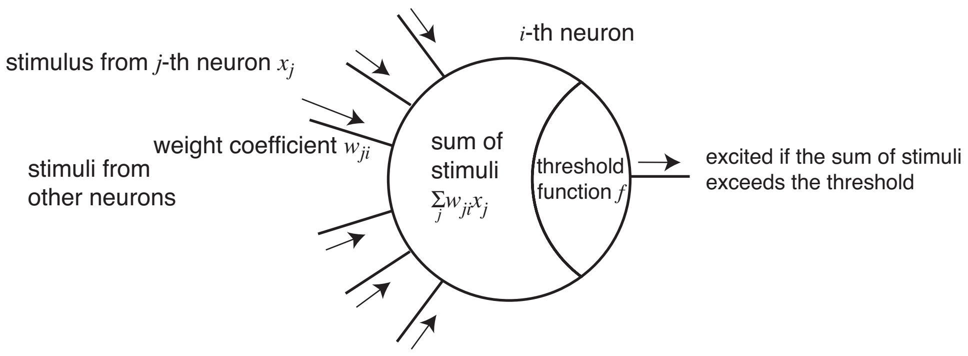
The interconnection topology of neurons is briefly categorized into *layered* or *fully-interconnected* ones. The layered network, as illustrated in Fig. 2(a), was proposed as a model of information processing by human optic nerves. A set of information, for example a visual pattern, is recorded by the whole neurons on the first layer, and it is fed from one layer to the next layer as stimuli flowing along one direction. The final status of the final layer after the arrival of the stimuli will be the output of the network. The layered network is mainly used for image processing and pattern recognition.

On the other hand, the fully-connected network, as illustrated in Fig. 2(b), has no layers and each neuron is connected to all the other neurons. The interconnections of this kind of network are “programed” for each problem, and the status is modified from the initial one by interchanging stimuli. The output of the network, or the solution of the problem, is given by the status of neurons after the interchanging operations converge, i. e. no more interchangings occur. The fully-connected network is mainly used for the optimum location problem and the associative memory.

The layered network, which is closely related to the image processing, is solely explained in this session.

Perceptron learning and its limitation

Perceptron is one of the simplest layered neural network proposed at the early stage of neural network research. The perceptron has three layers, which are input, intermediate, and output ones, as shown in Fig. 3. The perceptron was proposed for pattern recognition. If the pattern “A” is put into the input layer, the neuron corresponding to the character “A” is excited in the output layer. If “B” is input, the output “B” neuron is excited, etc.



stimulus from j-th neuron x_j
i-th neuron
weight coefficient w_{ji}
stimuli from other neurons
sum of stimuli $\sum_j w_{ji} x_j$
threshold function f
excited if the sum of stimuli exceeds the threshold

Fig. 1: neuron model.

The main characteristic of layered networks is learning ability. The learning tunes so as to output the desired values without any information except examples of input-output pair. The learning of the perceptron is carried out only between the intermediate layer and the output layer, explained in the followings. It is assumed that the output layer contains only one neuron in the output layer, and its status is denoted as x^O, as shown in Fig. 4.

1. Some examples of ideal input-output pair are prepared. An ideal input-output pair is, for example, as the input pattern “A” causes the status “1” of the output neuron. In each input-output pair, the status of the $j-$th neuron in the intermediate layer is denoted as x^H_j.

2. Initial weight coefficients between the intermediate and output layers, denoted w^O_j, are set, for example randomly.

3. An example input is given to the input layer, and the output x^O is examined.

4. Let the ideal output at the output layer be y^O. If the current output $x^O = 0$ while the ideal output $y^O = 1$, i. e. $x^O - y^O = -1$, it is indicated that $\sum_j w^O_j x^H_j$, the sum of stimuli to the neuron in the output layer, is too small. Thus the weight coefficients of the connections to the neurons whose status x^H_j is 1 in the intermediate layer are increased. If the current output $x^O = 1$ while the ideal output $y^O = 0$, i. e. $x^O - y^O = +1$, it is indicated that $\sum_j w^O_j x^H_j$ is too large. Thus the weight coefficients of the connections to the neurons whose status x^H_j is 1 in the intermediate layer are decreased. The above procedure is formulated as updating the weight coefficient of the connection between the j-th neuron in the intermediate layer and

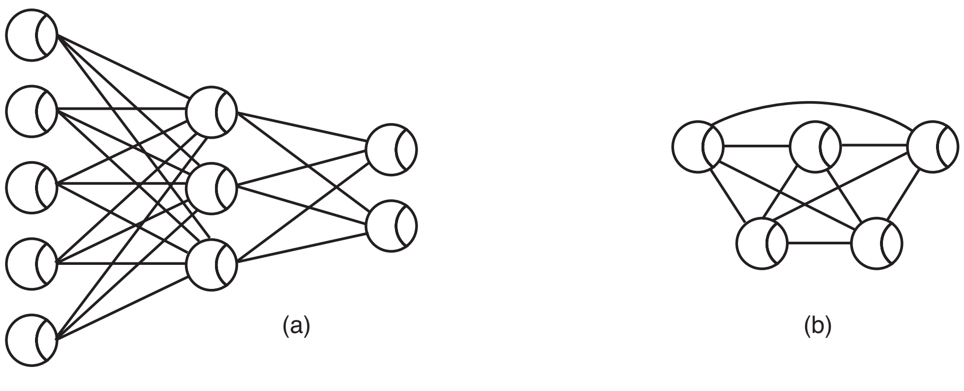


Fig. 2: Topologies of neural networks. (a) layered network. (b) fully-interconnected network.

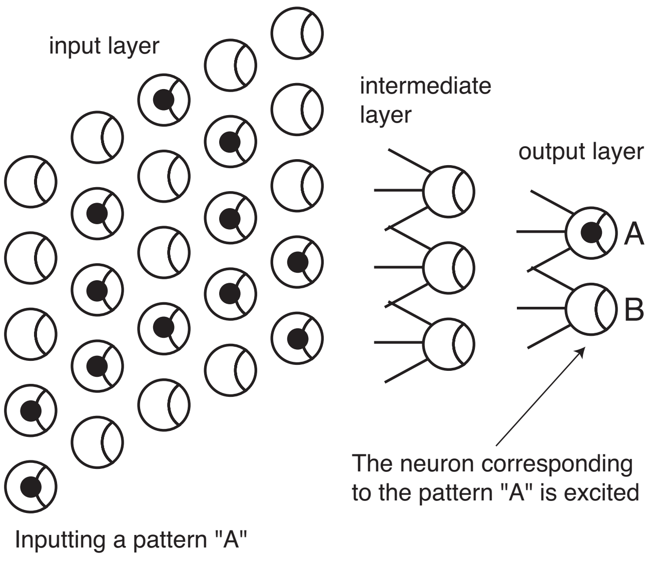


Fig. 3: Layered neural network such as perceptron.

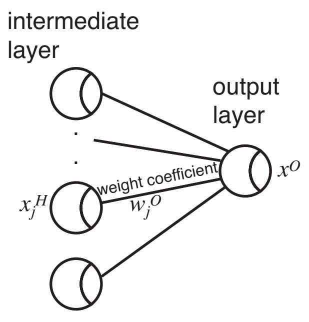


Fig. 4: Learning of the weight coefficients between two layers.

the output neuron to $w_j'^O$, as follows:

$$w_j'^O = w_j^O - (x^O - y^O)x_j^H. \tag{2}$$

5. Steps 3 and 4 are repeated until the ideal output is obtained.

The method is called the *perceptron learning*. This method is simple but has a problem that the algorithm does not always converge and w_j^O walks on a cycle of possible values forever. This is because the amount of modification from w_j^O to $w_j'^O$ is too large. To solve this problem, the threshold function f is replaced with a continuous function, for example the sigmoid function as follows:

$$f(x) = \frac{1}{1 + \exp(-\lambda(x - T))}, \tag{3}$$

as shown in Fig. 5. The sigmoid function reduces to the threshold function if the parameter $\lambda to \infty$. The statuses of neurons, x_j^H and x^O, are considered continuous, and Eq. (2) is modified by introducing a small positive number ϵ, called the *learning coefficient,* as follows:

$$w_j'^O = w_j^O + \epsilon(y^O - x^O)x_j^H. \tag{4}$$

This gradual learning method is called δ-rule.

The status of the output neuron of the perceptron is determined as follows:

$$x^O = f(\sum_j w_j^O x_j^H). \tag{5}$$

This operation indicates that the status of the output neuron is determined whether the weighted sum of the status of neurons in the intermediate layer is larger than the threshold value T or not. Let us consider the simplest case that the intermediate layer contains only two neurons x_1^H and x_2^H, and the status of these neurons are indicated on the coordinate plane, as shown in Fig. 6. The calculation in Eq. (5) is expressed in the coordinate plane in Fig. 7(a) as dividing the coordinate plane into two domains by a straight line

$$w_1^O x_1^H + w_2^O x_2^H = T, \tag{6}$$

and determines the output status x^O by the domain where the combination of values of x_1^H and x_2^H are located.

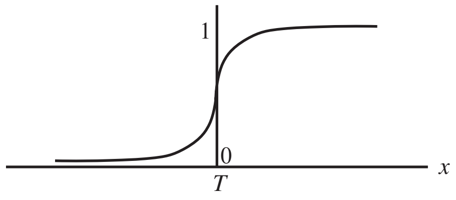


Fig. 5: Sigmoid function.

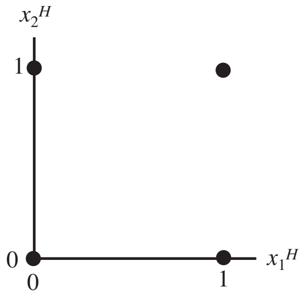


Fig. 6: Status of neuron. The pair (x_1^H, x_2^H) is on one of four dots.

In case of the operation "x^O =Exclusive OR (XOR) of x_1^H and x_2^H," however, the domain where $x^O = 1$ and the domain where $x^O = 0$ cannot be separated by a straight line. Such operations is called *linearly nonseparable.* Thus the learning of the weight coefficients of the connections between the intermediate and output layers, explained above, cannot obtain XOR of x_1^H and x_2^H [1]. Since the interconnections between the input and intermediate layers are not modified, the desired pattern recognition is not always achieved by the learning.

This is a limitation of the perceptron. This was pointed out by Minsky and Papert, and they also stated that there were many problem that could not be solved by the perceptron under the practical restrictions on the construction of the input and intermediate layers, and that the learning of the interconnections between the intermediate and output layers could not achieve practically meaningful issues. Their results had very big impact, and researchers' interests in neural networks were suddenly lost. To overcome this limitation, a learning method not only for the interconnections between the intermediate and output layers but also for the interconnections of overall network must be developed. It is the error back propagation algorithm, which is explained in the next section.

Error back propagation algorithm

In the cases of the perceptron learning and the δ-rule in the previous section, the ideal input-output pairs are given as the status of the input and output layers only. Thus the difference (or error) between the current and ideal outputs, minimized by the learning, is obtained only for the output layer. The perceptron learning and the δ-rule modify the weight coefficients between the output layer and the adjacent layer (the $(n-1)$th layer if the network has n layers) using this error.

To modify the weight coefficients between the $(n-1)$th and $(n-2)$th layers by the similar method, it is necessary to determine the error at each neuron in the $(n-1)$th layer. To achieve this, the error at a neuron on the output layer must be assigned to each neuron on the $(n-1)$th layer. The error at a neuron of the output layer is assigned based on the following idea:

[1] The larger the weight coefficient between a neuron on the $(n-1)$th layer and the output neuron is, the larger the amount of error should be assigned to this neuron, since the modification at this neuron becomes more effective at the output layer.

[2] The status of each neuron on the $(n-1)$th layer is determined by the weighted sum of the stimuli from the $(n-2)$th layer transferred by a nonlinear function like the sigmoid function. The larger the modification ratio of the status of a neuron subject to the modifi-

[1]The perceptron learning does not converge if the ideal input-output pairs are not linearly separable. The δ-rule converges to the weight coefficients where the square error from the ideal output is the minimum.

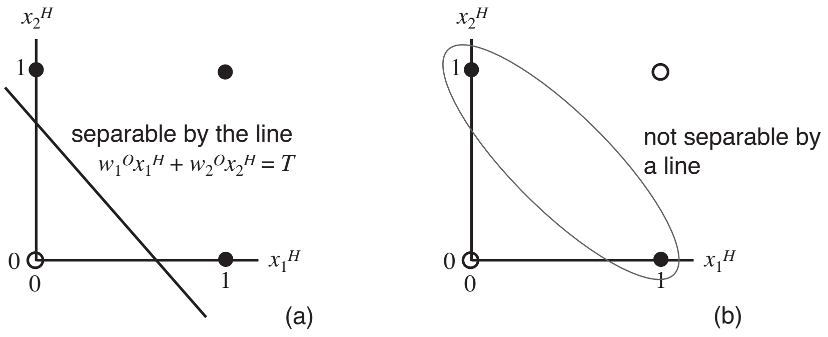
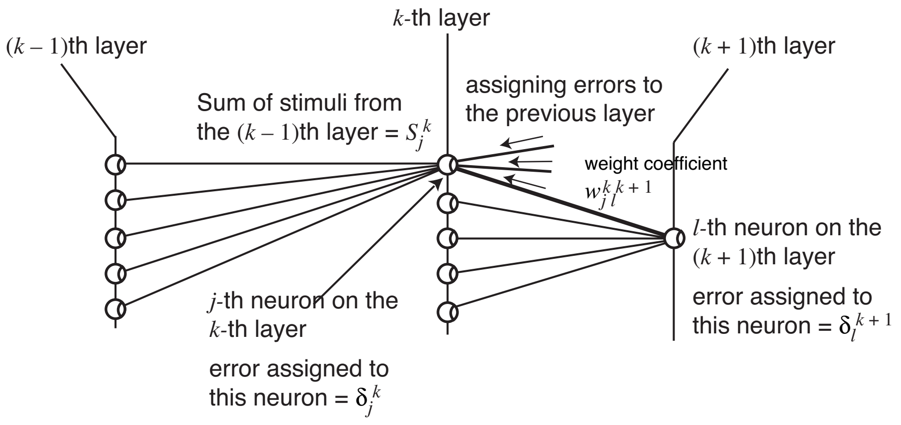


Fig. 7: Linear separability. (a) the white dot and the black dots are linearly separable. (b) Exclusive OR. The white dots and the black dots are not linearly separable.



k-th layer
$(k-1)$th layer
$(k+1)$th layer
Sum of stimuli from the $(k-1)$th layer = S_j^k
assigning errors to the previous layer
weight coefficient $w_{j\,l}^{k\,k+1}$
l-th neuron on the $(k+1)$th layer
error assigned to this neuron = δ_l^{k+1}
j-th neuron on the k-th layer
error assigned to this neuron = δ_j^k

Fig. 8: Error back propagation method.

cation of the sum of stimuli is, the larger the amount of error should be assigned to this neuron, since the modification at this neuron becomes more effective at the output layer. In other words, the more "sensitive" a neuron is, the more error should be assigned.

These ideas are formulated generally for the k-th and $(k+1)$th layers as follows, as illustrated in Fig. 8. Let the error assigned to the j-th neuron on the k-th layer be δ_j^k, and the error assigned to the l-th neuron on the $(k+1)$th layer be δ_l^{k+1}. Let the weight coefficient between the j-th neuron on the k-th layer and the l-th neuron on the $(k+1)$th layer be $w_{j\,l}^{k\,(k+1)}$. Let the status of the j-th neuron on the k-th layer be x_j^k.

The idea [1] mentioned above indicates that the error assignment of δ_j^k from the l-th neuron on the $(k+1)$th layer is proportional to $w_{j\,l}^{k\,(k+1)}$. Since the sum of stimuli from the $(k-1)$th layer to the j-th neuron on the k-th layer, denoted S_j^k, is as follows:

$$S_j^k = \sum_i w_{j\,l}^{(k-1)\,k} x_i^{k-1}, \tag{7}$$

the idea [2] indicates that the error assignment of δ_j^k from the l-th neuron on the $(k+1)$th layer is also proportional to dx_j^k/dS_j^k. Since these errors are assigned from the neurons on the $(k+1)$th layer and these sum is δ_j^k, we get

$$\delta_j^k = \sum_l \left(w_{j\,l}^{k\,k+1} \frac{dx_j^k}{dS_j^k} \right) \delta_l^{k+1}. \tag{8}$$

Let $f()$ be a nonlinear function, for example the sigmoid function, used here instead of the threshold function. Since

$$x_j^k = f(S_j^k), \tag{9}$$

we get

$$\frac{dx_j^k}{dS_j^k} = f'(S_j^k), \tag{10}$$

and Eq. (8) is rewritten to

$$\begin{aligned}\delta_j^k &= \sum_l \left(w_{j\,l}^{k\,k+1} f'(S_j^k) \right) \delta_l^{k+1} \\ &= \left[\sum_l (w_{j\,l}^{k\,k+1} \delta_l^{k+1}) \right] f'(S_j^k). \end{aligned} \tag{11}$$

Equation (11) determines the error amount assigned to the j-th neuron on the k-th layer from the error at each neuron on the k-th layer. In case $k = n$, i. e. at the output layer, this equation is not applicable since the $(k + 1)$th layer does not exist. However,

$$\delta_l^n = (x_l^n - y_l) f'(S_l^n), \tag{12}$$

where y_l is the ideal output at the l-th neuron on the output layer, since the error at the output layer is the difference between the ideal and current outputs.

Equations (11) and (12) determine the error assigned to every neuron on every layer. Applying the δ-rule in Eq. (4) to these errors, the updated weight coefficient $w'{}_{j\,l}^{k\,k+1}$ is obtained as follows:

$$w'{}_{j\;\;l}^{k-1\,k} = w_{j\;\;l}^{k-1\,k} - \epsilon \delta_l^k x_j^{k-1} \tag{13}$$

The above method of learning for multilayer neural networks is called the *error back propagation (EBP) method*, since the errors are propagated from the output layer back to the previous layers.

The EBP method in Eq. (13) derived above from the ideas [1] and [2] has been proved to modify the weight coefficients along the direction of the largest error reduction at each modification. It is called that the modification is the *steepest descent*. The proof is on the appendix, and the meaning of the steepest descent is explained here.

The sum of square errors is uniquely determined for a set of the weight coefficients. If we consider the coordinate space each of whose bases is each weight coefficient, we can consider a "field" whose value at each point corresponds to the sum of square errors. The modification of weight coefficients is moving along a pass in this space. The steepest descent means moving from the current point along the direction where the reduction of the sum is the largest.

We now consider what this direction is. Let $\boldsymbol{x}$ be a point in the space, and $f(\boldsymbol{x})$ be a function giving the sum of square errors in our problem. Let $\boldsymbol{x}(s)$ be a point on a pass in this space with a parameter s.

The gradient vector at $\boldsymbol{x}(s)$ on the pass is expressed as

$$\boldsymbol{b} = \frac{d\boldsymbol{x}}{ds}. \tag{14}$$

Differentiating f along the direction of the pass, we get

$$\frac{\partial f}{\partial s} = \frac{\partial f}{\partial \boldsymbol{x}} \cdot \frac{d\boldsymbol{x}}{ds} = \boldsymbol{b} \cdot \mathrm{grad} f, \tag{15}$$

where grad$f$ is the vector derived by the differentiation of $f(\boldsymbol{x})$ along each coordinate. For example, if $\boldsymbol{x}$ is three-dimensional and denoted $\boldsymbol{x} = (x, y, z)$, grad$f$ is the vector

$$(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}). \tag{16}$$

Equation (15) indicates that the largest reduction of f is achieved if $\boldsymbol{b}$, the gradient on the pass, is parallel to grad$f$ but in the opposite direction. Consequently, the direction of the steepest descent of f is $-$grad$f$.

Note that the direction of the steepest descent is the direction where the reduction of f is *currently* the largest, but *not the direction to the minimum of f*.

Appendix. Proof that the EBP method achives the steepest descent

We define the sum of square errors between the current output, or the status of neurons on the n-th layer, and the ideal output for an example input-output pair as follows:

$$E = \frac{1}{2} \sum_l (x_l^n - y_l)^2. \tag{A1}$$

It can be considered that E is a function of the weight coefficients. By differentiating E with respect to the weight coefficient $w_{j\;\;l}^{k-1\,k}$, we find

$$\frac{\partial E}{\partial w_{j\;\;l}^{k-1\,k}} = \frac{\partial E}{\partial S_l^k} \cdot \frac{\partial S_l^k}{\partial w_{j\;\;l}^{k-1\,k}}, \tag{A2}$$

and it follows from Eq. (7) in the main text that

$$\frac{\partial S_l^k}{\partial w_j^{k-1}{}_l^k} = \frac{\partial}{\partial w_j^{k-1}{}_l^k} \sum_i w_i^{k-1}{}_l^k x_i^{k-1} = x_j^{k-1}. \quad (A3)$$

By setting

$$\delta_l^k = \frac{\partial E}{\partial S_l^k}, \quad (A4)$$

the substitution of Eqs. (A3) and (A4) into Eq. (A2) yields

$$\frac{\partial E}{\partial w_j^{k-1}{}_l^k} = \delta_l^k x_j^{k-1}. \quad (A5)$$

Comparing this equation with Eq. (13), i. e.

$$w'{}_j^{k-1}{}_l^k = w_j^{k-1}{}_l^k - \epsilon \delta_l^k x_j^{k-1},$$

which is the equation expressing how to modify the weight coefficients, we find that Eq. (13) modifies the weight coefficients along the direction of $\partial E / \partial w_j^{k-1}{}_l^k$, i. e. $-\mathrm{grad} E$, and that the modification by Eq. (13) achieves the steepest descent of E.

It is shown in the following that δ in Eq. (A4) is equivalent to δ derived from the ideas [1] and [2] in the main text. In the case of $k \neq n$, we get by rewriting Eq. (A4)

$$\frac{\partial E}{\partial \partial S_j^k} = \sum_l \frac{\partial E}{\partial S_l^{k+1}} \cdot \frac{\partial S_l^{k+1}}{\partial x_j^k} \cdot \frac{\partial x_j^k}{\partial S_j^k}. \quad (A6)$$

It follows from Eq. (10) in the main text,

$$\frac{dx_j^k}{dS_j^k} = f'(S_j^k), \quad (A7)$$

and by replacing the suffices of Eq. (7), we get

$$S_l^{k+1} = \sum_i w_i^k{}_l^{k+1} x_i^k, \quad (A8)$$

then we get

$$\frac{\partial S_l^{k+1}}{\partial x_j^k} = w_j^k{}_l^{k+1}. \quad (A9)$$

The substitution of Eqs. (A7) and (A9) into (A6) yields

$$\frac{\partial E}{\partial \partial S_j^k} = \sum_l \frac{\partial E}{\partial S_l^{k+1}} \cdot w_j^k{}_l^{k+1} \cdot f'(S_j^k). \quad (A10)$$

It follows from the definition of δ in Eq. (A4) that

$$\delta_j^k = \frac{\partial E}{\partial S_j^k}, \; \delta_j^{k+1} = \frac{\partial E}{\partial S_l^{k+1}}. \quad (A11)$$

The substitution of Eq. (A11) into Eq. (A10) yields

$$\begin{aligned} \delta_j^k &= \sum_l \delta_j^{k+1} \cdot w_j^k{}_l^{k+1} \cdot f'(S_j^k) \\ &= \left[\sum_l (w_j^k{}_l^{k+1} \delta_j^{k+1}) \right] f'(S_j^k), \quad (A12) \end{aligned}$$

and it is identical to the definition of δ in the main text.

In the case of $k = n$, we get from Eq. (A4)

$$\frac{\partial E}{\partial S_l^n} = \frac{\partial E}{\partial x_l^n} \cdot \frac{dx_l^n}{dS_l^n}. \quad (A13)$$

From Eq. (A1), the definition of the sum of square errors, we get

$$\begin{aligned} \frac{\partial E}{\partial x_l^n} &= \frac{\partial}{\partial x_l^n} \left\{ \frac{1}{2} \sum_l (x_l^n - y_l)^2 \right\} \\ &= x_l^n - y_l \quad (A14) \end{aligned}$$

The substitution of Eq. (A14) into Eq. (A13) yields

$$\frac{\partial E}{\partial S_l^n} = (x_l^n - y_l) \cdot \frac{dx_l^n}{dS_l^n}. \quad (A15)$$

From the definition of δ in Eq. (A4) we get

$$\delta_l^n = \frac{\partial E}{\partial S_l^n}, \quad (A16)$$

and it follows from Eq. (S11app07) that

$$\frac{dx_l^n}{dS_l^n} = f'(S_l^n), \quad (A17)$$

and the substitution of Eqs. (A16) and (A17) into Eq. (A15) yields

$$\delta_l^n = (x_l^n - y_l) f'(S_l^n), \quad (A18)$$

which is identical to the definition of δ by Eq. (12) in the main text.