BACKPROP IS NOT ONLY FOR THE ERROR MINIMIZATION

NAOTO YOSHIDA

Abstract. The backpropagation algorithm (backprop) is an elegant algorithm for the multilayer perceptrons. Unfortunately, in the most of the explanations of the backprop, this algorithm is treated as if this algorithm is used only for the gradient calculation of the error function. We introduce the backprop as the general algorithm for the calculation of the gradient of the target function, and later we show the application of the backprop to the several target functions. This is the personal note for my future study.

Contents

1. The Target function and notations

We assume that some target function E is the element-wise sum of the outputs

$$E = \sum_j f_j(y_j),$$

where f_j is the element function with respect to the j-th output unit, and y_j is the j-th output of the multilayer perceptron. We denotes j is the index of the outer (nearer to the output) units, i is the index of the inner (nearer to the input) units. w_{ij} is the weight between the j-th unit and the i-th unit. We define e_j as

$$e_j = \frac{\partial E}{\partial y_j} = \frac{\partial f_j}{\partial y_j}.$$

Date: March 2, 2015.

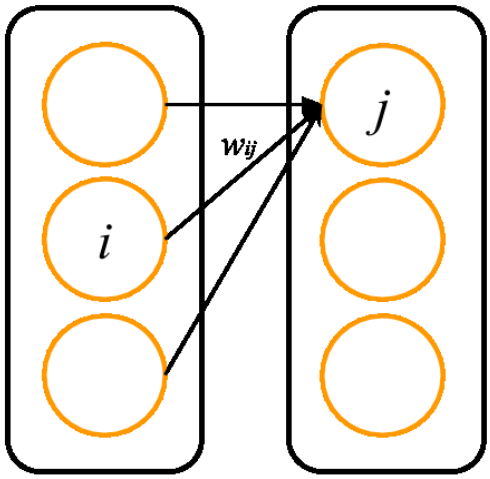

FIGURE 1. The output layer j and its previous layer i. The weights for the j-th unit are shown in this figure.

And the j unit is activated by the previous layer following the equation

$$\begin{aligned} y_j &= \phi(v_j) \\ &= \phi\Big(\sum_i w_{ij} y_i\Big), \end{aligned}$$

we call $\phi(v)$ the activation function.

2. When j is the output unit

Here we assume that the j unit is the output unit of the network (Figure.1). Then the derivative of the E with respect to w_{ij} is given by

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j}\frac{\partial v_j}{\partial w_{ij}} \\ &= e_j \phi'(v_j) y_i, \end{aligned}$$

where $\phi'(v)$ is the derivative of the activation function $\phi'(v) = \frac{\partial \phi(v)}{\partial v}$. We define

$$\delta_j^{out} = \frac{\partial E}{\partial v_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j} = e_j \phi'(v_j). \tag{1}$$

Therefore the derivative of the cost function with respect to w_{ij} is represented by

$$\frac{\partial E}{\partial w_{ij}} = \delta_j^{out} y_i. \tag{2}$$

Then the gradient of the target function is given by the product of the error signal δ_j^{out} and the outputs of the previous layer y_i.

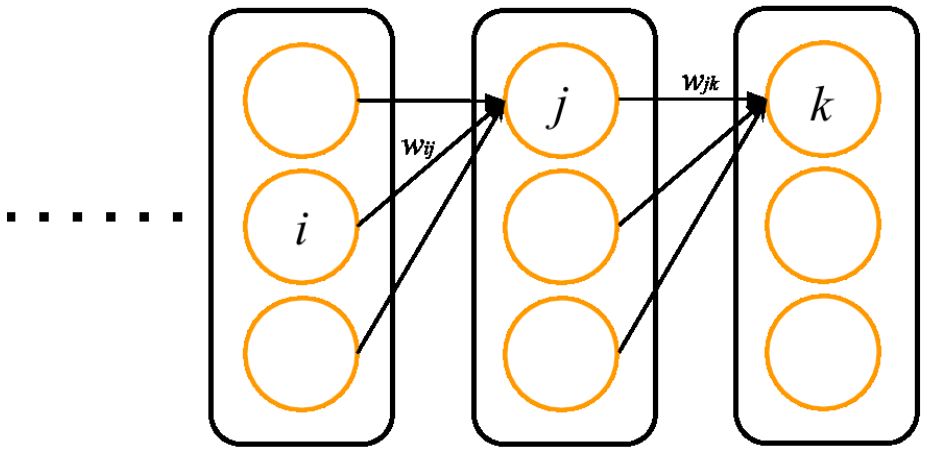

FIGURE 2. The output layer k and its previous layers i, j. The weights for the j-th unit and k-th unit are shown in this figure.

3. When j is the hidden unit

Here we assume that the j is the hidden unit of the network. And k is the index of the output units. Then

$$\begin{aligned}
\frac{\partial E}{\partial w_{ij}} &= \sum_k \frac{\partial f_k}{\partial y_k}\frac{\partial y_k}{\partial v_k}\frac{\partial v_k}{\partial y_j}\frac{\partial y_j}{\partial v_j}\frac{\partial v_j}{\partial w_{ij}} \\
&= \sum_k \Big(e_k \phi'(v_k) w_{jk}\Big)\phi'(v_j) y_i \\
&= \Big(\phi'(v_j)\sum_k \delta_k^{out} w_{jk}\Big) y_i \\
&= \delta_j^{hidden} y_i. \qquad (3)
\end{aligned}$$

In this case, we defined δ_j^{hidden} as

$$\delta_j^{hidden} = \phi'(v_j)\sum_k \delta_k^{out} w_{jk}. \qquad (4)$$

Again, the gradient of the target function is given by the product of the error signal δ_j^{hidden} and the outputs of the previous layer y_i.

The propagation of δ^{hidden} toward the input units is same, that is

$$\delta_i^{hidden} = \phi'(v_i)\sum_j \delta_j^{hidden} w_{ij}. \qquad (5)$$

4. Summary of the Backprop

In the most of the literature, the back propagation algorithm is introduced in the context of the minimization of the squared error or the cross entropy error. However, these explanations may mislead the students, they may misunderstand that the backprop algorithm is the only algorithm for the minimization of the above error functions. As explained above, backprop is the algorithm for the efficient calculation of the gradient of the function which is expressed by the element-wise sum of the functions of output units.

In a summary, the process of the backprop algorithm is as follows.

1: Evaluate the output y by the forward propagation.
2: Calculate the error signal δ^{out} by the equation (1).
3: Propagate the error signal by the equation (4).
4: Continue the propagation by the equation (5).
5: Update the weights by equations (2) and (3).

Practically, the element-wise calculation is much slower than the matrix operation in the most of the high-level computer languages. Because such computer languages or the scientific computation libraries are very optimized for the matrix operations. Then the direct implementation of the algorithm described above can take much longer than the matrix-based implementation or using the neural network libraries. So, to use the large multilayer perceptrons for the large data set, we strongly recommend to write the matrix-based implementation or use the neural network libraries.

Again, the algorithm is exactly same for any element-wise sum target function. The only difference is the definition of the δ^{out} given by

$$\delta_j^{out} = \frac{\partial E}{\partial v_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial v_j} = \frac{\partial f_j}{\partial y_j}\frac{\partial y_j}{\partial v_j} = e_j\phi'(v_j).$$

5. Applications

5.1. **Gradient of the Squared Cost and the Cross-Entropy.** The calculation of the gradient of the squared cost and the cross-entropy is the most popular application of the backprop. For the squared cost, we use the target function

$$E_{sq} = \frac{1}{2}\sum_j (t_j - y_j)^2$$

where t_j denotes the continuous teacher signal. We can understand that this is the special case of the element-wise sum form of the element cost $f_j(y_j) = \frac{1}{2}(t_j - y_j)^2$. The square loss is usually used for the multilayer perceptrons with continuous outputs. Then the natural choice is the linear activation function $\phi(x) = x \Rightarrow$

$\phi'(x) = 1$ at the output units. In this case,

$$\delta_j^{out} = \frac{\partial f_j}{\partial y_j}\frac{\partial y_j}{\partial v_j} = e_j\phi'(v_j) = y_j - t_j$$

is the output error signal.

For cross-entropy error, we use the target function

$$E_{cross} = -\sum_j t_j \log y_j + (1 - t_j)\log(1 - y_j)$$

where t_j is the binary teacher signal. In this case, the sigmoid activation function $\phi(x) = 1/(1 + e^{-x}) \Rightarrow \phi'(x) = \phi(x)(1 - \phi(x))$ is usually used for the outputs. As a result,

$$\delta_j^{out} = \frac{\partial f_j}{\partial y_j}\frac{\partial y_j}{\partial v_j} = e_j\phi'(v_j) = \Big\{\frac{t_j - y_j}{y_j(1 - y_j)}\Big\}y_j(1 - y_j) = y_j - t_j$$

is the output error signal.

In a summary, the both cost function produce the same output error signal

$$\delta_j^{out} = y_j - t_j.$$

5.2. **Gradient of the Entropy.** We can use backprop for other target functions, for example, the entropy

$$E = H = -\sum_j y_j \log y_j + (1 - y_j)\log(1 - y_j).$$

We assume that the output y_j is activated by the sigmoid function $\phi(x) = 1/(1 + e^{-x}) \Rightarrow \phi'(x) = \phi(x)(1 - \phi(x))$. Then

$$\delta_j^{out} = \frac{\partial H}{\partial v_j} = \frac{\partial H}{\partial y_j}\frac{\partial y_j}{\partial v_j} = \Big(\log(1 - y_j) - \log y_j\Big)y_j(1 - y_j).$$

We can minimize/maximize the entropy by using the gradient generated by this δ^{out} and propagated δ^{hidden}, and the stochastic gradient descent or other gradient-based methods.

5.3. **Gradient of the output.** Another important example is the case when the network has only one output unit $E = y$ (or, equivalently, we focus on the gradient of the one of the output units) and

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial y}{\partial w_{ij}}$$

as the target function. In this case, δ^{out} is given by

$$\delta^{out} = \frac{\partial E}{\partial v} = \frac{\partial y}{\partial v} = \phi'(v).$$

This equation suggests that $\delta^{out} = 1$ for $\phi(x) = x$, $\delta^{out} = y(1 - y)$ for $\phi(x) = 1/(1 + e^{-x})$.

5.4. **Calculation of the Jacobian.** We may sometimes want to calculate the jacobian of the multilayer perceptron, that is

$$\Delta_{in} = \frac{\partial y_{out}}{\partial y_{in}}$$

for all output units y_{out} and the input units y_{in}. We can efficiently calculate by the similar propagation method as the backprop. We use the same index i, j, k used in the section 3. At the output layer, we can obtain

$$\Delta_k = \frac{\partial y_k}{\partial y_k} = 1.$$

Then the jacobian at the next layer is given by

$$\begin{aligned} \Delta_j &= \frac{\partial y_k}{\partial y_j} \\ &= \sum_k \frac{\partial y_k}{\partial y_k}\frac{\partial y_k}{\partial v_k}\frac{\partial v_k}{\partial y_j} \\ &= \sum_k \Delta_k \phi'(v_k) w_{jk}. \end{aligned}$$

Notice that the jacobian with respect to y_j is given by the propagation of Δ_k. We can obtain the jacobian of the next layer by the same manner

$$\begin{aligned} \Delta_i &= \frac{\partial y_k}{\partial y_i} \\ &= \sum_j \sum_k \Big(\frac{\partial y_k}{\partial y_k}\frac{\partial y_k}{\partial v_k}\frac{\partial v_k}{\partial y_j} \Big) \frac{\partial y_j}{\partial v_j}\frac{\partial v_i}{\partial y_i} \\ &= \sum_j \Delta_j \phi'(v_j) w_{ij}. \end{aligned}$$

To obtain Δ_{in}, we simply repeat the above propagation until the algorithm reaches the input units as

$$\begin{aligned} \Delta_i &= \frac{\partial y_k}{\partial y_i} \\ &= \sum_{j \in \{outer\ \ layer\ \ units\}} \Delta_j \phi'(v_j) w_{ij}. \end{aligned}$$

References

[1] Haykin, Simon S., et al. Neural networks and learning machines. Vol. 3. Upper Saddle River: Pearson Education, 2009.