A NOTE ON BACKPROP

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 more general target functions, and later we show the application of the backprop to the several target functions. Also we describe the vanishing gradient problem in the backprop with conventional neural networks, and intuitively explain why rectifier linear units works well in the deep neural networks. This is the personal note for my future study.

Contents

1. The Target function and notations

We assume that some target function E is given as

$$E = E(y_1, \ldots, y_j, \ldots, y_J).$$

So E is the function of the output units of the multilayer perceptron y_j ($j \in 1, 2, .., J$). We denotes j is the index of the outer (nearer to the output) units, i is

Date: August 29, 2015.

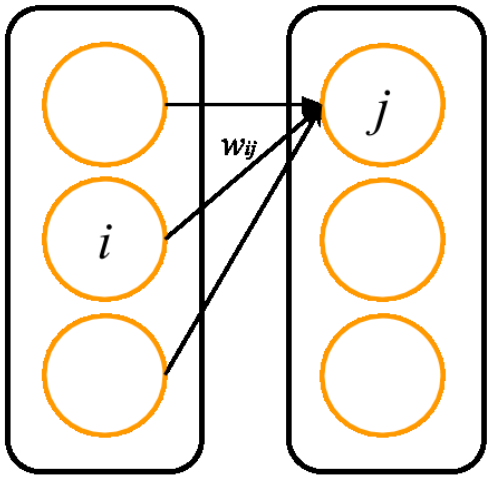

FIGURE 1. The output layer j and its previous layer i. The weights for the j-th unit are shown in this figure.

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

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

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

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

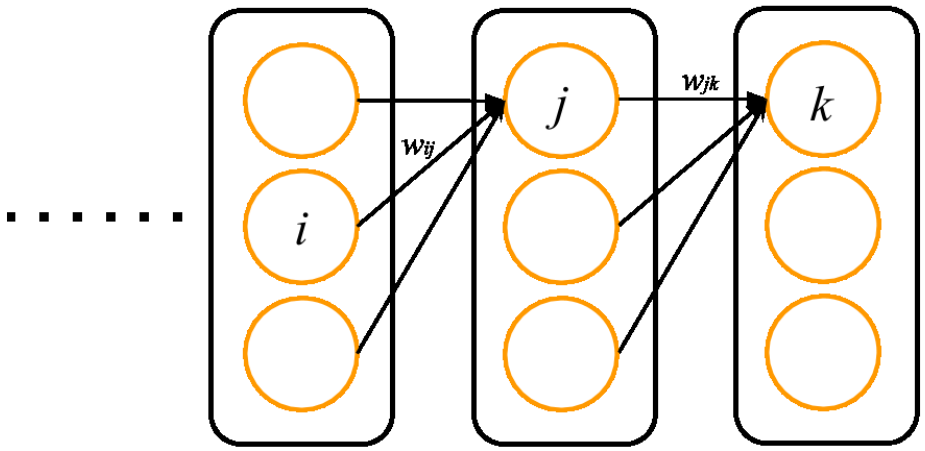

FIGURE 2. The output layer k and its previous layers i, j. The weights for the j-th unit and k-th unit are shown in this figure.

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.

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

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

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

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

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 previous layer is same, that is

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

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 target functions. The only difference is the definition of the δ^{out} given by

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

5. Applications

5.1. **Gradient of the Squared Cost and the Cross-Entropy.** The calculation of the gradient of the squared error cost and the cross-entropy cost 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. The squared error cost 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,

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

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,

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

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

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

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

$$\Delta_{kk} = \frac{\partial y_k}{\partial y_k} = 1.$$

Then the jacobian at the next layer is given by

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

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

$$\begin{aligned} \Delta_{ik} &= \frac{\partial y_k}{\partial y_i} \\ &= \sum_j \Big(\frac{\partial y_k}{\partial y_k}\frac{\partial y_k}{\partial v_k}\frac{\partial v_k}{\partial y_j}\Big)\frac{\partial y_j}{\partial v_j}\frac{\partial v_j}{\partial y_i} \\ &= \sum_j \Delta_{jk}\phi'(v_j)w_{ij}. \end{aligned}$$

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

$$\begin{aligned} \Delta_{tk} &= \frac{\partial y_k}{\partial y_t} \\ &= \sum_i \Delta_{ik}\phi'(v_i)w_{ti}. \end{aligned}$$

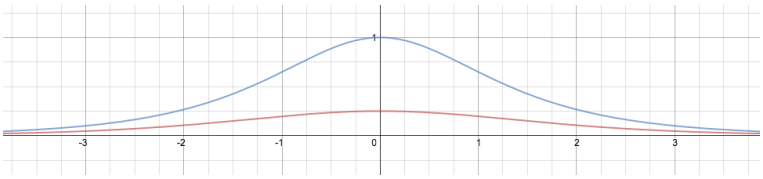

FIGURE 3. The derivatives of the sigmoid activation functions $\phi'(s)$. The red line represents the derivative of the logistic activation. The blue line represents that of the hyperbolic tangent.

6. How to Understand the Vanishing Gradient Intuitively

Vanishing gradient is a well known phenomenon which caused the second winter of the neural network community. In this section, we intuitively understand why the vanishing gradient occurs, and why the recently developed rectifier linear units (ReLU) break the vanishing gradient problem.

Let us we now consider a four-layers multilayer perceptron (four cell layers). Calculating the partial derivatives following the section 1-3, we can obtain the derivative of the error function with respect to the weights w_{ti} between the input layer t and its previous (outer) layer i.

$$\begin{aligned}
\frac{\partial E}{\partial w_{ti}} &= \sum_j \sum_k \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial v_k}\frac{\partial v_k}{\partial y_j}\frac{\partial y_j}{\partial v_j}\frac{\partial v_j}{\partial y_i}\frac{\partial y_i}{\partial v_i}\frac{\partial v_i}{\partial w_{ti}} \\
&= \sum_j \sum_k \frac{\partial E}{\partial y_k}\Big(\frac{\partial y_k}{\partial v_k}\frac{\partial v_k}{\partial y_j}\Big)\Big(\frac{\partial y_j}{\partial v_j}\frac{\partial v_j}{\partial y_i}\Big)\Big(\frac{\partial y_i}{\partial v_i}\Big)\frac{\partial v_i}{\partial w_{ti}} \\
&= \sum_j \sum_k e_k\Big(\phi'(v_k)w_{jk}\Big)\Big(\phi'(v_j)w_{ij}\Big)\Big(\phi'(v_i)\Big)x_t \\
&= \sum_j \sum_k e_k\Big(\phi'(v_k)\phi'(v_j)\phi'(v_i)\Big)\Big(w_{jk}w_{ij}\Big)x_t.
\end{aligned}$$

The inputs (the activations of the t layer units) are denoted by x_t. Recalling that e_k is the derivative of the target function with respect to the activation of the output units, and $\phi'(x)$ is the derivative of the activation function $\frac{\partial\phi(x)}{\partial x}$, now we can easily understand how the activation function affects this derivative.

Conventional neural networks usually use the sigmoidal activation functions such as the logistic function $\phi(x) = \frac{1}{1+e^{-x}}$ or the hyperbolic tangent function $\phi(x) = \tanh(x)$. The derivatives of both functions ($\phi'(x) = \frac{1}{1+e^{-x}}(1 - \frac{1}{1+e^{-x}})$ and $\phi'(x) = 1 - \tanh^2(x)$) are $\phi'(x) \leq 1$ for any $x \in \mathcal{R}$. Figure 3 is the actual plot of the derivatives of the sigmoidal activation functions. This figure clearly shows that the derivatives of the logistic function (red) always output the values less than

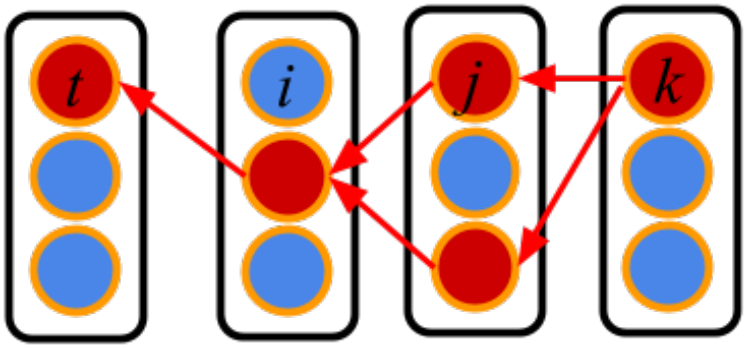

FIGURE 4. The propagation of the derivatives in the neural network with rectifier liner units. The red cells are firing ($v > 0$) units and the blue cells are non-firing ($v < 0$) units.

one. Hence the product of derivatives $\phi'(x)$ quickly decrease the error gradients. Equivalently, the increase of the layers result in the "vanishing of the gradients".

LeCun *et al.* introduced that the networks with hyperbolic tangent units often converge faster than the logistic activation units. We can (at least, partially) understand that the faster convergence of hyperbolic tangent units results from its derivative (blue, in Figure 3), which is close to one when the input x is close to zero and always takes larger values than that of the logistic activation units. However, usually the derivative of the hyperbolic tangent function results $\phi'(x) < 1$ in the training of neural networks. Then the vanishing gradient (exponential decay of the derivatives) still suffers when we use the VERY deep networks.

In the record-breaking paper of Krizhevsky *et al.*, the rectifier linear unit (ReLU) $\phi(x) = \max(0, x)$ was introduced. Even though ReLU is non-differential at $x = 0$, ReLU is one of the simplest non-linear activation function and enable the fast computation. Obviously, the derivative of the ReLU is given by $\phi'(x) = 1$ when $0 < x$ and $\phi'(x) = 0$ when $x < 0$. In this case, all "paths" in the network including $\phi'(x) = 0$ disappear in the summation at the last row of the previous equation, then it becomes

$$\frac{\partial E}{\partial w_{ti}} = \begin{cases} \sum_{j\in\{j_{fire}\}} \sum_{k\in\{k_{fire}\}} e_k \Big(w_{jk} w_{ij} w_{ti} \Big) x_t & (v_i > 0) \\ 0 & (v_i < 0) \end{cases}$$

$\{j_{fire}\}$ is the set of firing units (that is, $v_j > 0$) in that layer j. Remarkable point of this propagation is that the vanishing component (the product of $\phi'(s)$ s) in this summation are always one and the derivatives always reach without vanishing, except that no cells are firing in the any of the single layers (this would be avoided by using sufficiently "wide" layers).

Figure 4 describes how the derivatives are propagated in the network. The red cells are firing ($v > 0$) units and the blue cells are that of non-firing ($v < 0$) units.

The red arrows are the weights updated in the backprop. The derivative of the activation function $\phi'(v)$ works as a sort of "gates" of the flows of the derivatives.

References

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

[2] LeCun, Yann A., et al. "Efficient backprop." Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012. 9-48.

[3] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.