Tutorial of Back Propagation

Xingyu Jin

2016 年 2 月 21 日

目录

1 Back propagation of MLP

In this section, we will introduce the theoretical formulas and practical implementation of MLP.

1.1 Symbols

First of all, we define some variable symbols. Bold name denotes vectors. $\mathbf{y}^i$ represents the output vector of layer i and y^i_k represents the kth output node of layer i. n^i means ith layer has n^i nodes in total except bias node. b^i is the bias node in layer i. $\mathbf{W}^i$ represents weight parameter matrix between layer i and layer $i+1$ and w^i_{kj} means the weight parameter between the kth output node in layer i and the jth input node in layer $i+1$. $\mathbf{u}^i$

represents the input vector of layer i and u_k^i represents the kth input node of layer i. $\phi(\cdot)$ is element-wise nonlinear function to convert input node data to output node data in one layer. The notations can be seen in the schematic diagram as shown in 1

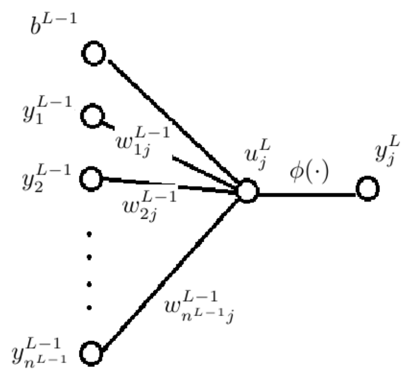

图 1: Schematic diagram for symbols

1.2 Forward Calculation

Suppose the network has L layers and $\mathbf{y}^L$ is exactly the binary coding output layer. $\mathbf{y}^{k-1}$ is a hidden layer. We can easily get:

$$\mathbf{u}^k = \mathbf{W}^{k-1}\mathbf{y}^{k-1} + \mathbf{b}^{k-1}$$

$$\mathbf{y}^k = \phi(\mathbf{u}^k)$$

1. If $\phi()$ is sigmoid, i.e. $\phi(x) = \frac{1}{1+e^{-x}}$, we have,

   $$\phi'(\mathbf{u}^k) = \mathbf{y}^k. * (1 - \mathbf{y}^k)$$

   where, $.*$ represents element wise multiplication.

2. if $\phi()$ is relu, i.e. $\phi(x) = max(0, x)$, we have,

   $$\phi'(\mathbf{u}^k) = \mathbf{y}^k > 0$$

   where $>$ is element wise in MATLAB codes.

1.3 Loss function

We utilize the mean squared error as the loss function of the neural network.

$$E = \frac{1}{n}\sum_{i=1}^{n}(t_i - y_i^L)^2 \tag{1}$$

E is the loss function of total training data. n is the number of training data. y_i^L represent the ith prediction when the input is the ith training data. t_i is the corresponding label.

1.4 Backward calculation

The error slope in the jth input node in layer k is δ_j^k

$$\delta_j^k = \frac{\partial E}{\partial u_j^k}$$

From loss function Equation 1, we can deduce that:

$$\delta_j^L = \frac{\partial E}{\partial y_j^L}\frac{\partial y_j^L}{\partial u_j^L} \tag{2}$$

$$= -\frac{2}{n}(t_j - y_j^L)\phi'(u_j^L) \tag{3}$$

Then we have:

$$\frac{\partial E}{\partial w_{ij}^{L-1}} = \frac{\partial E}{\partial u_i^L}\frac{\partial u_i^L}{\partial w_{ij}^{L-1}} \tag{4}$$

$$= \delta_i^L y_j^{L-1} \tag{5}$$

We can use back propagation algorithm between layer k and layer $k+1$ $(k < L-1)$:

$$\delta_i^k = \frac{\partial E}{\partial u_i^k} \tag{6}$$

$$= \sum_{j=1}^{n^{k+1}} \frac{\partial E}{\partial u_j^{k+1}}\frac{\partial u_j^{k+1}}{\partial y_i^k}\frac{\partial y_i^k}{\partial u_i^k} \tag{7}$$

$$= \sum_{j=1}^{n^{k+1}} \delta_j^{k+1} w_{ji}^k \phi'(u_i^k) \tag{8}$$

i.e.

$$\delta^k = ({W^k}^T \delta^{k+1}). * \phi'(u^k)$$

where, $.*$ represents element wise multiplication.

and :

$$\frac{\partial E}{\partial w_{ij}^k} = \frac{\partial E}{\partial u_i^{k+1}} \frac{\partial u_i^{k+1}}{\partial w_{ij}^k} \tag{9}$$

$$= \delta_i^{k+1} y_j^k \tag{10}$$

$$\frac{\partial E}{\partial b_i^k} = \sum_{j=1}^{n^{k+1}} \frac{\partial E}{\partial u_j^{k+1}} \frac{\partial u_j^{k+1}}{\partial b_i^k} \tag{11}$$

$$= \delta_i^{k+1} \tag{12}$$

When we get $\frac{\partial E}{\partial w_{ij}^k}$, we can update weight parameters by iterations like:

$$w_{ij}^k <= w_{ij}^k - \frac{\partial E}{\partial w_{ij}^k} \Delta w$$

Here $<=$ is used as value assignment in order to avoid confusion with mathematical equal.

i.e.

$$\mathbf{W}^k <= \mathbf{W}^k - \delta^{k+1} {\mathbf{y}^k}^T \Delta w$$

$$\mathbf{b}^k <= \mathbf{b}^k - \delta^{k+1} \Delta b$$

1.5 Multiple Input

If the input is $X = [x^{(1)} x^{(2)} ... x^{(M)}]$, a $N \times M$ matrix, then we can compute pararelly, however, the parameters should be shared, and a slight change should be made. $x^{(m)}$ is the m th input vector.

1. Forward Part:

$$\mathbf{u}^{(m)k} = \mathbf{W}^{k-1} \mathbf{y}^{(m)k-1} + \mathbf{b}^{(m)k-1}$$

$$\mathbf{y}^{(m)k} = \phi(\mathbf{u}^{(m)k})$$

2. Loss Function:
$$E = \sum_{m=1}^{M} \sum_{i=1}^{n} (t_i^{(m)} - y_i^{(m)L})^2$$
Let $T = [t^{(1)} t^{(2)} ... t^{(M)}]$, $Y^k = [y^{(1)k} y^{(2)k} ... y^{(M)k}]$, then we have,
$$E = [1 \quad 1...1](T - Y^L)^T (T - Y^L)$$

3. Backward: Let $\Delta^k = [\delta^{(1)k} \delta^{(2)k} ... \delta^{(M)k}]$
$$\delta_j^{(i)L} = \frac{\partial E}{\partial y_j^{(i)L}} \frac{y_j^{(i)L}}{u_j^{(i)L}}$$
Thus,
$$\delta^{(i)L} = -2(t^{(i)} - y^{(i)L}). * \phi'(u^{(i)L})$$
Finally,
$$\Delta^L = -2(T - Y). * \phi'(U^L)$$

$$\frac{\partial E}{\partial w_{ij}^k} = \sum_{m=1}^{M} \frac{\partial E}{\partial u_i^{(m)k+1}} \frac{\partial u_i^{(m)k+1}}{\partial w_{ij}^k} \tag{13}$$
$$= \sum_{m=1}^{M} \delta_i^{(m)k+1} y_j^{(m)k} \tag{14}$$
Thus,
$$\frac{\partial E}{\partial \mathbf{W}} = \Delta^k {Y^k}^T$$

1.6 Softmax Classification

A softmax layer output $\mathbf{y}$ is a $K \times 1$ vector and $\mathbf{x}$ is input vector.
$$y_k = P(t_k = 1) = \frac{exp(x_k)}{\sum_{i=1}^{K} exp(x_i)}$$
Cross entropy error function is defined as follows.
$$E = -ln(p) = -\sum_{i=1}^{K} t_i ln(y_i)$$

If the input of the neural network is multiple, i.e. $\mathbf{X} = [x^{(1)}, x^{(2)}, ...x^{(n)}]$, then the loss function E satisfies,

$$E = \sum_{i=1}^{n} E^{(i)}$$

$$E^{(i)} = -\sum_{i=1}^{K} t_i^{(i)} ln(y_i^{(i)})$$

Backward propagation:

$$\begin{aligned}
\frac{\partial E}{\partial x_i^{(m)}} &= \frac{\partial E^{(m)}}{\partial x_i^{(m)}} && (15)\\
&= \sum_{j=1}^{K} \frac{\partial E^{(m)}}{\partial y_j^{(m)}} \frac{\partial y_j^{(m)}}{\partial x_i^{(m)}} && (16)\\
&= -\frac{t_i^{(m)}}{y_i^{(m)}}(y_i^{(m)} - {y_i^{(m)}}^2) + \sum_{j!=i} \frac{t_j^{(m)}}{y_j^{(m)}} y_j^{(m)} y_i^{(m)} && (17)\\
&= -t_i^{(m)}(1 - y_i^{(m)}) + \sum_{j!=i} t_j^{(m)} y_i^{(m)} && (18)\\
&= -t_i^{(m)} + \sum_{j=1}^{K} t_j^{(m)} y_i^{(m)} && (19)\\
&= -t_i^{(m)} + y_i^{(m)} && (20)
\end{aligned}$$

2 Back propagation of CNN

Convolutional Neural Network is different from MLP mentioned before. The most important difference is that CNN layer contain 3 dimentions, 2 of spatial dimension, like height and width, and 1 dimension describing depth or the number of filters. The parameters in a filter is 3 dimensional, local connected on height and width, but fully connected in previous depth. The input size is $W_1 \times H_1 \times n_{in}$, W_1 is width,H_1 is height, n_{in} is depth, usually three channel of a colorful image as input or the number of filters of previous Conv layer, and the hyperparameters of current Conv layer:

1. Number of filters n_{out}

2. their spatial extent K
3. the stride S
4. the amount of zero padding P

Each filter is $K \times K \times n_{in}$. And the Conv layer produces a volume of size $W_2 \times H_2 \times D_2$, where

1. $W_2 = \frac{W_1 - K + 2P}{S} + 1$
2. $H_2 = \frac{H_1 - K + 2P}{S} + 1$
3. $D_2 = n_{out}$

With parameter sharing, the Conv layer has a total of $k \times k \times n_{in} \times n_{out}$ weights and n_{out} biases. Let w^k_{ijd} be the ith height, jth width, dth depth, kth layer weight entry.

$$u^k = (W^{k-1} * y^{k-1})_{valid} + b^{k-1}$$

where $*$ represents convolutional operation. i.e.

$$u^k_{ijm} = \sum_{p=1}^{K} \sum_{q=1}^{K} \sum_{d=1}^{D^{k-1}} w^{k-1}_{pqdm} y^{k-1}_{(p+i-1)(q+j-1)d} + b^{k-1}$$

$1 \leq i \leq H^k, 1 \leq j \leq W^k, 1 \leq m \leq D^k$

$$\begin{aligned}
\delta^k_{pqw} &= \frac{\partial E}{\partial u^k_{pqw}} && (21)\\
&= \sum_{i=1}^{H^{k+1}} \sum_{j=1}^{W^{k+1}} \sum_{d=1}^{D^{k+1}} \frac{\partial E}{\partial u^{k+1}_{ijd}} \frac{\partial u^{k+1}_{ijd}}{\partial y^k_{pqw}} \frac{\partial y^k_{pqw}}{\partial u^k_{pqw}} && (22)\\
&= \sum_{i=1}^{H^{k+1}} \sum_{j=1}^{W^{k+1}} \sum_{d=1}^{D^{k+1}} \delta^{k+1}_{ijd} w^k_{(p-i+1)(q-j+1)wd} \phi'(u^k_{pqw}) && (23)\\
&= \sum_{i=1}^{K} \sum_{j=1}^{K} \sum_{d=1}^{D^{k+1}} w^k_{ijwd} \delta^{k+1}_{(p-i+1)(q-j+1)d} \phi'(u^k_{pqw}) && (24)\\
& && (25)
\end{aligned}$$

i.e.

$$\delta^k = (\delta^{k+1} * W^k)_{full} . * \phi'(u^k)$$

$$\frac{\partial E}{\partial w^k_{pqwn}} = \sum_{i=1}^{H^{k+1}} \sum_{j=1}^{W^{k+1}} \sum_{m=1}^{D^{k+1}} \frac{\partial E}{\partial u^{k+1}_{ijm}} \frac{\partial u^{k+1}_{ijm}}{\partial w^k_{pqwn}} \tag{26}$$

$$= \sum_{i=1}^{H^{k+1}} \sum_{j=1}^{W^{k+1}} \delta^{k+1}_{ijn} y^k_{(p+i-1)(q+j-1)w} \tag{27}$$

$$\frac{\partial E}{\partial b^k_n} = \sum_{i=1}^{H^{k+1}} \sum_{j=1}^{W^{k+1}} \sum_{m=1}^{D^{k+1}} \frac{\partial E}{\partial u^{k+1}_{ijm}} \frac{\partial u^{k+1}_{ijm}}{\partial b^k_n} \tag{28}$$

$$= \sum_{i=1}^{H^{k+1}} \sum_{j=1}^{W^{k+1}} \delta^{k+1}_{ijn} \tag{29}$$

im2col function can be applied like this.

im2col(Y(:,:,m,n),[K,K],'sliding') will produce a $(K \cdot K) \times [(H - K + 1) \cdot (W - K + 1)]$ matrix, use a for loop across all depth m and batch n could build a $K \cdot K \cdot n_{in} \times (H - K + 1) \cdot (W - K + 1) \cdot batch\_size$ matrix A.

$$U^k = W^{k-1} * A^{k-1} + b^{k-1}$$

where U^k is $n_{out} \times ((H - K + 1) \cdot (W - K + 1) \cdot batch\_size)$ matrix which can be reformed to $(H - K + 1) \times (W - K + 1) \times n_{out} \times batch\_size$ output tensor