Memo: Back-propagation

Tomomichi Sugihara

March 27, 2021

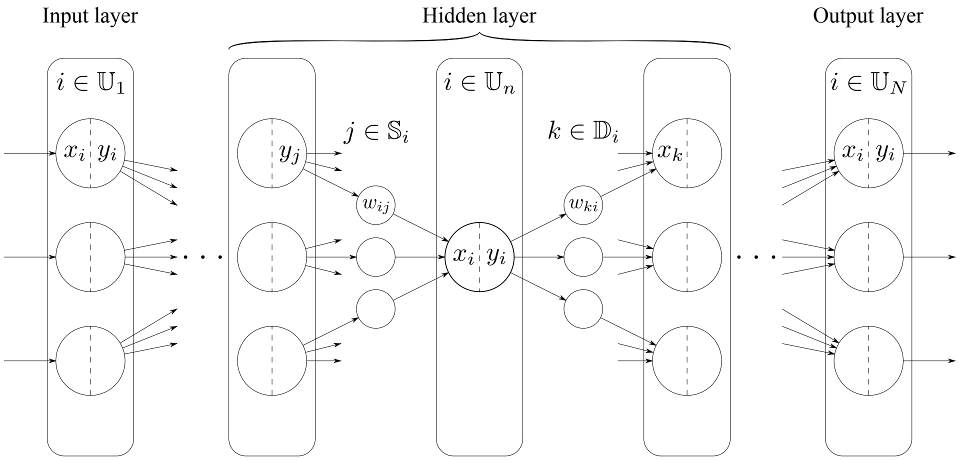

Figure 1: A non-recurrent multi-layered neural network

Let us consider a non-recurrent N-layered neural network illustrated in Fig. 1. The transmission of information in it is represented by the following equations:

$$x_i = \begin{cases} \text{given} & (\text{if } i \in \mathbb{U}_1) \\ \displaystyle\sum_{j \in \mathbb{S}_i} w_{ij} y_j + b_i & (\text{otherwise}) \end{cases} \tag{1}$$

$$y_i = \begin{cases} x_i & (\text{if } i \in \mathbb{U}_1) \\ f_i(x_i), & (\text{otherwise}) \end{cases} \tag{2}$$

where x_i is the input to the ith neural unit, $\mathbb{U}_n$ for $n = 1, \cdots, N$ is the set of indices of units in the nth layer, $\mathbb{S}_i$ is the set of indices of upstream neural units connected to the ith unit, y_i is the output of the ith unit, w_{ij} is the weight on y_j that is input to the ith unit, b_i is the bias of the ith unit, and $f_i(\cdot)$ is the activation function associated with the ith unit. By propagating the output of each unit from the input layer to the output layer based on the above equations, the network transforms the input $\boldsymbol{x} = \{x_i | \forall i \in \mathbb{U}_1\}$ to the output $\boldsymbol{y} = \{y_i | \forall i \in \mathbb{U}_N\}$. Namely, it represents a multi-input-multi-output algebraic mapping $\mathcal{N}_{\{w_{ij}\},\{b_i\}} : \boldsymbol{x} \mapsto \boldsymbol{y}$ or $\boldsymbol{y} = \boldsymbol{F}(\boldsymbol{x})$.

Provided an input $\boldsymbol{x}^*$ paired with the desired output $\boldsymbol{y}^*$, the loss function of the network $E(\boldsymbol{x}^*, \boldsymbol{y}^*; \mathcal{N}_{\{w_{ij}\},\{b_i\}})$ is defined. A typical choice of the function is the sum of squared errors as

$$E(\boldsymbol{x}^*, \boldsymbol{y}^*; \mathcal{N}_{\{w_{ij}\},\{b_i\}}) \overset{\text{def}}{=} \frac{1}{2} \left\| \boldsymbol{F}(\boldsymbol{x}^*) - \boldsymbol{y}^* \right\|^2 . \tag{3}$$

The back-propagation is one of the representative methods to train the network, *i.e.*, all the weights $\{w_{ij}\}$ and the biases $\{b_i\}$ so as to minimize the above loss function based on the following update rule:

$$w_{ij} \leftarrow w_{ij} - \eta \Delta w_{ij} \tag{4}$$

$$b_i \leftarrow b_i - \eta \Delta b_i, \tag{5}$$

where η is the learning rate. Δw_{ij} and Δb_i are decided to align the steepest descent direction as

$$\Delta w_{ij} = \frac{\partial E_{\mathbb{L}}}{\partial w_{ij}} \tag{6}$$

$$\Delta b_i = \frac{\partial E_{\mathbb{L}}}{\partial b_i}, \tag{7}$$

where $E_{\mathbb{L}}$ is the following loss function of a mini-batch

$$E_{\mathbb{L}} = \sum_{l \in \mathbb{L}} E(\boldsymbol{x}^{*(l)}, \boldsymbol{y}^{*(l)}; \mathcal{N}_{\{w_{ij}\},\{b_i\}}), \tag{8}$$

$\mathbb{L}$ is the set of identifiers of runs included in the mini-batch, $\boldsymbol{x}^{*(l)}$ is the input in the lth run to the network, and $\boldsymbol{y}^{*(l)}$ is the corresponding desired output of the network. Obviously,

$$\frac{\partial E_{\mathbb{L}}}{\partial w_{ij}} = \sum_{l \in \mathbb{L}} \frac{\partial E}{\partial w_{ij}}(\boldsymbol{x}^{*(l)}, \boldsymbol{y}^{*(l)}; \mathcal{N}_{\{w_{ij}\},\{b_i\}}) \tag{9}$$

$$\frac{\partial E_{\mathbb{L}}}{\partial b_i} = \sum_{l \in \mathbb{L}} \frac{\partial E}{\partial b_i}(\boldsymbol{x}^{*(l)}, \boldsymbol{y}^{*(l)}; \mathcal{N}_{\{w_{ij}\},\{b_i\}}), \tag{10}$$

and hence, the goal here is to find $\dfrac{\partial E}{\partial w_{ij}}$ and $\dfrac{\partial E}{\partial b_i}$.

w_{ij} and b_i only affect x_i directly, and thus,

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial x_i}\frac{\partial x_i}{\partial w_{ij}} = p_i y_j \tag{11}$$

$$\frac{\partial E}{\partial b_i} = \frac{\partial E}{\partial x_i}\frac{\partial x_i}{\partial b_i} = p_i, \tag{12}$$

where

$$p_i \overset{\text{def}}{=} \frac{\partial E}{\partial x_i}. \tag{13}$$

x_i affects y_i, and y_i affects $\{x_k | \forall k \in \mathbb{D}_i\}$, where $\mathbb{D}_i$ is the set of indices of downstream units connected to the ith unit, if it belongs to a hidden layer. Hence,

$$p_i = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial x_i} = \begin{cases} \dfrac{\partial E}{\partial y_i} f_i'(x_i) & (\text{if } i \in \mathbb{U}_N) \\ \left(\displaystyle\sum_{k \in \mathbb{D}_i} \frac{\partial E}{\partial x_k}\frac{\partial x_k}{\partial y_i}\right)\dfrac{\partial y_i}{\partial x_i} = \left(\displaystyle\sum_{k \in \mathbb{D}_i} p_k w_{ki}\right) f_i'(x_i). & (\text{otherwise}) \end{cases} \tag{14}$$

In the typical case of the sum of squared errors,

$$\frac{\partial E}{\partial y_i} = y_i - y_i^* \qquad \text{for} \quad \forall i \in \mathbb{U}_N. \tag{15}$$

$f_i'(x_i)$ depends on the definition of $f_i(x_i)$. If it is the sigmoid function,

$$f_i(x_i) = \frac{1}{1 + e^{-x_i}} \qquad \Rightarrow \qquad f_i'(x_i) = \frac{e^{-x_i}}{(1 + e^{-x_i})^2} = y_i\,(1 - y_i)\,. \tag{16}$$

Or, if it is ReLU,

$$f_i(x_i) = \max\{x_i, 0\} \qquad \Rightarrow \qquad f_i'(x_i) = \begin{cases} 1 & (\text{if } x_i \geq 0) \\ 0 & (\text{otherwise}) \end{cases}. \tag{17}$$

Note that ReLU is not differentiable at $x_i = 0$ in the strict sense, although it is usually ignored.

A pseudocode of an algorithm for a neural network $\mathcal{N}_{\{w_{ij}\},\{b_i\}}$ to train from data set $\left\{ (\boldsymbol{x}^{*(l)}, \boldsymbol{y}^{*(l)}) \middle| \forall l \in \mathbb{L} \right\}$ and the loss function $E(\boldsymbol{x}^*, \boldsymbol{y}^*; \mathcal{N}_{\{w_{ij}\},\{b_i\}})$ based on the above back-propagation is as follows.

Algorithm 1 $\textsc{TrainNN}(\mathcal{N}_{\{w_{ij}\},\{b_i\}}, \left\{ (\boldsymbol{x}^{*(l)}, \boldsymbol{y}^{*(l)}) \middle| \forall l \in \mathbb{L} \right\}, \eta)$

Require: $\mathcal{N}_{\{w_{ij}\},\{b_i\}} \cdots$ initialized
Ensure: $\mathcal{N}_{\{w_{ij}\},\{b_i\}} \cdots$ updated
$\textsc{InitGrad}(\mathcal{N}_{\{w_{ij}\},\{b_i\}})$ ▷ Initialize gradients of weights and biases
for $l \in \mathbb{L}$ **do** ▷ Evaluate each run in a mini-batch
$\textsc{BackPropagate}(\mathcal{N}_{\{w_{ij}\},\{b_i\}}, (\boldsymbol{x}^{*(l)}, \boldsymbol{y}^{*(l)}))$
end for
$\textsc{UpdateNN}(\mathcal{N}_{\{w_{ij}\},\{b_i\}}, \eta)$ ▷ Update weights and biases of neural network

Algorithm 2 $\textsc{InitGrad}(\mathcal{N}_{\{w_{ij}\},\{b_i\}})$

for $i \in \mathbb{U}_2 \cup \cdots \cup \mathbb{U}_N$ **do**
for $j \in \mathbb{S}_i$ **do**
$\mathit{\Delta} w_{ij} \leftarrow 0$
end for
$\mathit{\Delta} b_i \leftarrow 0$
end for

Algorithm 3 $\textsc{BackPropagate}(\mathcal{N}_{\{w_{ij}\},\{b_i\}}, (\boldsymbol{x}^{*(l)}, \boldsymbol{y}^{*(l)}))$

Ensure: $\{\mathit{\Delta} w_{ij}\}, \{\mathit{\Delta} b_i\}$
$\textsc{Propagate}(\mathcal{N}_{\{w_{ij}\},\{b_i\}}, \boldsymbol{x}^{*(l)})$
$\textsc{InitParam}(\mathcal{N}_{\{w_{ij}\},\{b_i\}})$
for $i \in \mathbb{U}_N$ **do** ▷ for output layer
$p_i \leftarrow \partial E / \partial y_i(\boldsymbol{x}^{*(l)}, \boldsymbol{y}^{*(l)}; \mathcal{N}_{\{w_{ij}\},\{b_i\}})$ (dependent on the definition of E)
end for
for $n = N$ downto 2 **do** ▷ Backpropagation
for $i \in \mathbb{U}_n$ **do**
$p_i \leftarrow p_i v_i$
for $j \in \mathbb{S}_i$ **do**
$p_j \leftarrow p_j + p_i w_{ij}$
$\mathit{\Delta} w_{ij} \leftarrow \mathit{\Delta} w_{ij} + p_i y_j$
end for
$\mathit{\Delta} d_i \leftarrow \mathit{\Delta} d_i + p_i$
end for
end for

Algorithm 4 $\textsc{InitParam}(\mathcal{N}_{\{w_{ij}\},\{b_i\}})$

for $i \in \mathbb{U}_1 \cup \cdots \cup \mathbb{U}_N$ **do**
$p_i \leftarrow 0$
$v_i \leftarrow f'(x_i)$
end for

Algorithm 5 $\textsc{UpdateNN}(\mathcal{N}_{\{w_{ij}\},\{b_i\}}, \eta)$

Require: $\{\mathit{\Delta} w_{ij}\}, \{\mathit{\Delta} b_i\} \cdots$ computed
Ensure: $\{w_{ij}\}, \{b_i\} \cdots$ updated
for $i \in \mathbb{U}_2 \cup \cdots \cup \mathbb{U}_N$ **do**
$d_i \leftarrow d_i - \eta \mathit{\Delta} d_i$
for $j \in \mathbb{S}_i$ **do**
$w_{ij} \leftarrow w_{ij} - \eta \mathit{\Delta} w_{ij}$
end for
end for