Gradient Descent and Backpropagation in Machine Learning

Jinn-Liang Liu
Institute of Computational and Modeling Science,
National Tsing Hua University, Taiwan
2017/7/11, 2018/12/8

The change of the function (height) $f(\mathbf{x})$ at $\mathbf{x} \in R^2$ in the direction $\mathbf{p} \in R^2$ is a **directional derivative** defined as

$$
\begin{aligned}
D_{\mathbf{p}} f(\mathbf{x}) &= \lim_{t\to 0} \frac{f(\mathbf{x}+t\mathbf{p}) - f(\mathbf{x})}{t} && (1)\\
(\mathbf{x} &= (x,y) \text{ any point, } \mathbf{p} = (p_1,p_2) \text{ any unit direction})\\
&= \lim_{t\to 0} \frac{f(x+tp_1, y+tp_2) - f(x,y)}{t}\\
&= \lim_{t\to 0} \left\{ \frac{f(x+tp_1, y+tp_2) - f(x, y+tp_2)}{t} \right.\\
&\quad \left. + \frac{f(x, y+tp_2) - f(x,y)}{t} \right\}\\
&= \lim_{t\to 0} \left\{ \frac{[f(x+tp_1, y+tp_2) - f(x, y+tp_2)]p_1}{tp_1} \right.\\
&\quad \left. + \frac{[f(x, y+tp_2) - f(x,y)]p_2}{tp_2} \right\}\\
&= \lim_{t\to 0} \left\{ \frac{[f(x+\Delta x, y+tp_2) - f(x, y+tp_2)]p_1}{\Delta x} \right.\\
&\quad \left. + \frac{[f(x, y+\Delta y) - f(x,y)]p_2}{\Delta y} \right\}\\
&= \frac{\partial f(x,y)}{\partial x} p_1 + \frac{\partial f(x,y)}{\partial y} p_2\\
(\nabla &= \left\langle \frac{\partial}{\partial x}, \frac{\partial}{\partial y} \right\rangle = \text{grad} = \text{del, the } \textbf{gradient operator}) && (2)\\
&= \nabla f(x,y) \cdot \mathbf{p} = |\nabla f|\,|\mathbf{p}| \cos\theta && (3)
\end{aligned}
$$

$\Rightarrow$ Max value of $D_{\mathbf{p}} f$ is $|\nabla f|$ in $\mathbf{p} = \nabla f / |\nabla f|$ with $\theta = 0$

$\Rightarrow$ Min value of $D_{\mathbf{p}} f$ is $-|\nabla f|$ in $\mathbf{p} = -\nabla f / |\nabla f|$ with $\theta = 180^\circ$

Therefore, we should choose the direction $\mathbf{p} = -\nabla f(\mathbf{x})/\left|\nabla f(\mathbf{x})\right|$ (negative gradient) if we want to minimize $f(\mathbf{x})$ (to go down to the valley) in the **fastest** way (**steepest descent** or **gradient descent**) when we are at $\mathbf{x} = (x, y)$ on a mountain surface described by the function $z = f(\mathbf{x})$ at the elevation z. The **method of gradient (steepest) descent** is thus an iterative process

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_{k-1}\mathbf{p}_{k-1} \tag{4}$$

of changing (updating) our current location $\mathbf{x}_{k-1} = (x_{k-1}, y_{k-1})$ by deciding the next **stepping length** α_k (called **learning rate** in Machine Learning (ML)) in our **predicted** (**gradient**) direction $\mathbf{p}_{k-1} = -\bigtriangledown f(\mathbf{x}_{k-1})/\left|\nabla f(\mathbf{x}_{k-1})\right|$.

<u>**Exercise**:</u> Show that the **gradient vector** $\nabla f(\mathbf{x}_0)$ is *perpendicular* to the **tangent vector** $\mathbf{r}'(t_0)$ to the level curve $f(\mathbf{x}) = c$ at $\mathbf{x} = \mathbf{x}_0 = (x_0, y_0) = \langle x(t_0), y(t_0)\rangle = \mathbf{r}(t_0)$, where $\mathbf{r}'(t) = \frac{d}{dt}\langle x(t), y(t)\rangle$. Draw a graph with a mountain, surface, level curve, tangent vector, gradient vector, valley, and all math notations.

<u>**1D Example:**</u> Consider the minimization (optimization) problem

$$\textbf{Minimize} \quad y = f(x) = x^2, \forall x \in R^1. \tag{5}$$

$$\textbf{Method:} \quad x_k = x_{k-1} + \alpha_{k-1}p_{k-1} \text{ with } x_0 = 2,\ \alpha_{k-1} = \frac{1}{2},\ \forall k. \tag{6}$$

$$\begin{aligned}
\nabla f(x) &= \frac{df(x)}{dx} = 2x \qquad (7)\\
p_0 &= \frac{-\bigtriangledown f(x_0)}{|\nabla f(x_0)|} = \frac{-4}{|4|} = -1 \text{ (go west if } x_0 > 0)\\
x_1 &= x_0 + \alpha_0 p_0 = 2 - \frac{1}{2} = \frac{3}{2} \Rightarrow\\
x_2 &= x_1 + \alpha_1 p_1 = \frac{3}{2} - \frac{1}{2} = 1 \Rightarrow\\
x_3 &= x_2 + \alpha_2 p_2 = 1 - \frac{1}{2} = \frac{1}{2} \Rightarrow\\
x_4 &= x_3 + \alpha_3 p_3 = \frac{1}{2} - \frac{1}{2} = 0 = x^* \text{ (optimizer)} \Rightarrow\\
y^* &= f(x^*) = 0 \text{ (optimal value).}
\end{aligned}$$

2D Example: Consider the minimization problem

$$\textbf{Minimize} \quad z = f(\mathbf{x}) = \frac{x^2}{4^2} + y^2, \, \forall \mathbf{x} = \langle x, y \rangle \in R^2. \tag{8}$$

$$\textbf{Method}: \quad \mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_{k-1}\mathbf{p}_{k-1} \text{ with } \mathbf{x}_0 = \left\langle 2, \frac{1}{4} \right\rangle, \, \alpha_{k-1} = \frac{\sqrt{5}}{4}, \, \forall k. \tag{9}$$

$$\begin{aligned}
\nabla f(\mathbf{x}) &= \left\langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right\rangle = \left\langle \frac{x}{8}, 2y \right\rangle \qquad (10) \\
\mathbf{p}_0 &= \frac{- \bigtriangledown f(\mathbf{x}_0)}{|\nabla f(\mathbf{x}_0)|} = \frac{-\left\langle \frac{1}{4}, \frac{1}{2} \right\rangle}{\left|\left\langle \frac{1}{4}, \frac{1}{2} \right\rangle\right|} = \frac{-\left\langle \frac{1}{4}, \frac{1}{2} \right\rangle}{\frac{\sqrt{5}}{4}} = -\left\langle \frac{1}{\sqrt{5}}, \frac{2}{\sqrt{5}} \right\rangle \\
\mathbf{x}_1 &= \mathbf{x}_0 + \alpha_0 \mathbf{p}_0 = \left\langle 2, \frac{1}{4} \right\rangle - \frac{\sqrt{5}}{4}\left\langle \frac{1}{\sqrt{5}}, \frac{2}{\sqrt{5}} \right\rangle = \left\langle \frac{7}{4}, \frac{-1}{4} \right\rangle \\
\mathbf{p}_1 &= \frac{- \bigtriangledown f(\mathbf{x}_1)}{|\nabla f(\mathbf{x}_1)|} = \frac{-\left\langle \frac{7}{32}, \frac{-1}{2} \right\rangle}{\left|\left\langle \frac{7}{32}, \frac{-1}{2} \right\rangle\right|} \\
\mathbf{x}_2 &= \mathbf{x}_1 + \alpha_1 \mathbf{p}_1 = \cdots \Rightarrow \cdots
\end{aligned}$$

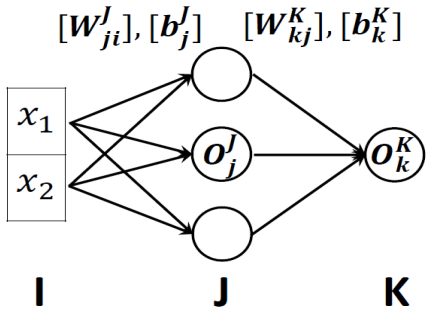

Figure 1: A Neural Network

ML Example: Consider a simple **Neural Network** (NN) as shown in Fig. 1 by Ryan Harris, *Neural Network Tutorial: The Back-Propagation Algorithm (2012)* on YouTube, where

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in R^2, \text{ input data point (vector)} \tag{11}$$

$$\mathbf{W}^J = \left[W^J_{ji}\right] = \begin{bmatrix} W^J_{11} & W^J_{12} \\ W^J_{21} & W^J_{22} \\ W^J_{31} & W^J_{32} \end{bmatrix} \in R^{3\times 2},\ \mathbf{b}^J = \begin{bmatrix} b^J_1 \\ b^J_2 \\ b^J_3 \end{bmatrix} \in R^{3\times 1} \tag{12}$$

$$\mathbf{W}^K = \left[W^K_{kj}\right] = \begin{bmatrix} W^K_{11} & W^K_{12} & W^K_{13} \end{bmatrix} \in R^{1\times 3},\ \mathbf{b}^K = b^K_k \in R^1 \tag{13}$$

$$\mathbf{O}^J = \mathbf{W}^J\mathbf{x} = \begin{bmatrix} O^J_1 \\ O^J_2 \\ O^J_3 \end{bmatrix} : \text{hidden layer } J,\ O^J_j \text{: hidden node} \tag{14}$$

$$y_k(\mathbf{W}) = O^K_k = \sigma(t) : \text{output value, } y_k \approx y^*_k \text{: } \mathbf{true\ value} \tag{15}$$

$$\sigma(t) = \sigma\left(\mathbf{W}^K\mathbf{W}^J\mathbf{x}\right) = \frac{1}{1+e^{-t}} \text{: sigmoid function, } t = \mathbf{W}^K\mathbf{O}^J \tag{16}$$

$$\sigma'(t) = \frac{d\sigma(t)}{dt} = \frac{e^{-t}}{(1+e^{-t})^2} = \frac{1+e^{-t}-1}{(1+e^{-t})^2} = \sigma(1-\sigma). \tag{17}$$

$$\textbf{Minimize Error}\quad E(\mathbf{W}) = \frac{1}{2}\sum_{k\in K}(y_k(\mathbf{W}) - y^*_k)^2,\ \forall\mathbf{W}. \tag{18}$$

$$\nabla E(\mathbf{W}) = \left\langle \cdots, \frac{\partial E(\mathbf{W})}{\partial W_{ji}^J}, \cdots, \frac{\partial E(\mathbf{W})}{\partial W_{kj}^K}, \cdots \right\rangle \tag{19}$$

$$\frac{\partial E(\mathbf{W})}{\partial W_{ji}^J} = (y_k(\mathbf{W}) - y_k^*)\frac{\partial y_k(\mathbf{W})}{\partial W_{ji}^J},\ y_k(\mathbf{W}) = \sigma(\mathbf{W}) = \sigma(t) \tag{20}$$

$$\begin{aligned}\frac{\partial y_k(\mathbf{W})}{\partial W_{ji}^J} &= \frac{\partial \sigma\left(\mathbf{W}^K\mathbf{W}^J\mathbf{x}\right)}{\partial W_{ji}^J} = \frac{d\sigma(t)}{dt}\frac{\partial\left(\mathbf{W}^K\mathbf{O}^J\right)}{\partial W_{ji}^J} \\ &= \sigma'(t)\frac{\partial}{\partial W_{ji}^J}\sum_{l\in J} W_{kl}^K O_l^J = \sigma'\sum_{l\in J} W_{kl}^K\frac{\partial O_l^J}{\partial W_{ji}^J} \\ &= \sigma'\sum_{l\in J}\left[W_{kl}^K\frac{\partial O_l^J}{\partial O_j^J}\frac{\partial}{\partial W_{ji}^J}\sum_{m\in I} W_{lm}^J x_m\right] = \sigma' W_{kj}^K x_i \end{aligned} \tag{21}$$

$$p_i = \frac{\partial E(\mathbf{W})}{\partial W_{ji}^J} = (y_k - y_k^*)O_k^K\left(1 - O_k^K\right)W_{kj}^K x_i,\ \frac{\partial E(\mathbf{W})}{\partial b_j^J} = 1 \tag{22}$$

$$\frac{\partial y_k(\mathbf{W})}{\partial W_{kj}^K} = \sigma'\frac{\partial\left(\mathbf{W}^K\mathbf{O}^J\right)}{\partial W_{kj}^K} = \sigma'\frac{\partial}{\partial W_{kj}^K}\sum_{l\in J} W_{kl}^K O_l^J = \sigma' O_j^J \tag{22}$$

$$p_j = \frac{\partial E(\mathbf{W})}{\partial W_{kj}^K} = (y_k - y_k^*)O_k^K\left(1 - O_k^K\right)O_j^J,\ \frac{\partial E(\mathbf{W})}{\partial b_k^K} = 1 \tag{23}$$

Back Propagate $(y_k - y_k^*)O_k^K\left(1 - O_k^K\right)$ from k Nodes to j Nodes!! (24)

Method: $W_m^{(n)} = W_m^{(n-1)} - \alpha^{(n-1)}p_m^{(n-1)},\ \ m = i \text{ or } j,\ \forall m.$ (25)

$\mathbf{p} = \nabla E(\mathbf{W}) = \langle \cdots,\ p_m, \cdots \rangle$, $\mathbf{W}^{(0)}$ and $\alpha^{(n-1)}$ given,

$\mathbf{W} = \langle \cdots, W_m, \cdots \rangle = \left\langle \cdots, W_{ji}^J, \cdots \right\rangle$, $\Delta W_m = W_m^{(n)} - W_m^{(n-1)}$.

Forward: $\nabla E(\mathbf{W}) = \left\langle \frac{\partial E(\mathbf{W})}{\partial W_1}, \cdots, \frac{\partial E(\mathbf{W})}{\partial W_{1,000,000}} \right\rangle = \langle \cdots, p_m, \cdots \rangle$

$\Rightarrow$ One Million Paths from j Nodes to k Nodes

$\Rightarrow$ One Million Differentiations!!!

$\Rightarrow$ Need Output Error $(y_k - y_k^*)$ One Million Times (Individually).

$\Rightarrow$ Need 116 Days!!

Backward: One Output Error $(y_k - y_k^*)$ from k Nodes to j Nodes

$\Rightarrow$ Get One Million $\frac{\partial E(\mathbf{W})}{\partial W_m}$ Layer-by-Layer Simultaneously!!

$\Rightarrow$ Need Only 1 Second, 10^7 Times Faster!!