EE-559 – Deep learning

3.6. Back-propagation

François Fleuret
https://fleuret.org/ee559/

Sun Dec 9 04:43:27 UTC 2018

RESEARCH INSTITUTE

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

We want to train an MLP by minimizing a loss over the training set

$$\mathscr{L}(w, b) = \sum_n \ell(f(x_n; w, b), y_n).$$

We want to train an MLP by minimizing a loss over the training set

$$\mathscr{L}(w, b) = \sum_n \ell(f(x_n; w, b), y_n).$$

To use gradient descent, we need the expression of the gradient of the loss with respect to the parameters:

$$\frac{\partial \mathscr{L}}{\partial w_{i,j}^{(l)}} \quad \text{and} \quad \frac{\partial \mathscr{L}}{\partial b_i^{(l)}}.$$

We want to train an MLP by minimizing a loss over the training set

$$\mathscr{L}(w, b) = \sum_n \ell(f(x_n; w, b), y_n).$$

To use gradient descent, we need the expression of the gradient of the loss with respect to the parameters:

$$\frac{\partial \mathscr{L}}{\partial w_{i,j}^{(l)}} \quad \text{and} \quad \frac{\partial \mathscr{L}}{\partial b_i^{(l)}}.$$

So, if we define $\ell_n = \ell(f(x_n; w, b), y_n)$, what we need is

$$\frac{\partial \ell_n}{\partial w_{i,j}^{(l)}} \quad \text{and} \quad \frac{\partial \ell_n}{\partial b_i^{(l)}}.$$

For clarity, we consider a single training sample x, and introduce $s^{(1)}, \dots, s^{(L)}$ as the summations before activation functions.

$$x^{(0)} = x \xrightarrow{w^{(1)}, b^{(1)}} s^{(1)} \xrightarrow{\sigma} x^{(1)} \xrightarrow{w^{(2)}, b^{(2)}} s^{(2)} \xrightarrow{\sigma} \dots \xrightarrow{w^{(L)}, b^{(L)}} s^{(L)} \xrightarrow{\sigma} x^{(L)} = f(x; w, b).$$

For clarity, we consider a single training sample x, and introduce $s^{(1)}, \dots, s^{(L)}$ as the summations before activation functions.

$$x^{(0)} = x \xrightarrow{w^{(1)}, b^{(1)}} s^{(1)} \xrightarrow{\sigma} x^{(1)} \xrightarrow{w^{(2)}, b^{(2)}} s^{(2)} \xrightarrow{\sigma} \dots \xrightarrow{w^{(L)}, b^{(L)}} s^{(L)} \xrightarrow{\sigma} x^{(L)} = f(x; w, b).$$

Formally we set $x^{(0)} = x$,

$$\forall l = 1, \dots, L, \begin{cases} s^{(l)} = w^{(l)} x^{(l-1)} + b^{(l)} \\ x^{(l)} = \sigma\left(s^{(l)}\right), \end{cases}$$

and we set the output of the network as $f(x; w, b) = x^{(L)}$.

For clarity, we consider a single training sample x, and introduce $s^{(1)}, \dots, s^{(L)}$ as the summations before activation functions.

$$x^{(0)} = x \xrightarrow{w^{(1)}, b^{(1)}} s^{(1)} \xrightarrow{\sigma} x^{(1)} \xrightarrow{w^{(2)}, b^{(2)}} s^{(2)} \xrightarrow{\sigma} \dots \xrightarrow{w^{(L)}, b^{(L)}} s^{(L)} \xrightarrow{\sigma} x^{(L)} = f(x; w, b).$$

Formally we set $x^{(0)} = x$,

$$\forall l = 1, \dots, L, \begin{cases} s^{(l)} = w^{(l)} x^{(l-1)} + b^{(l)} \\ x^{(l)} = \sigma\left(s^{(l)}\right), \end{cases}$$

and we set the output of the network as $f(x; w, b) = x^{(L)}$.

This is the **forward pass**.

The core principle of the back-propagation algorithm is the "chain rule" from differential calculus:

$$(g \circ f)' = (g' \circ f)f'$$

which generalizes to longer compositions and higher dimensions

$$J_{f_N \circ f_{N-1} \circ \dots \circ f_1}(x) = \prod_{n=1}^{N} J_{f_n}(f_{n-1} \circ \dots \circ f_1(x)),$$

where $J_f(x)$ is the Jacobian of f at x, that is the matrix of the linear approximation of f in the neighborhood of x.

The core principle of the back-propagation algorithm is the “chain rule” from differential calculus:

$$(g \circ f)' = (g' \circ f)f'$$

which generalizes to longer compositions and higher dimensions

$$J_{f_N \circ f_{N-1} \circ \dots \circ f_1}(x) = \prod_{n=1}^{N} J_{f_n}(f_{n-1} \circ \dots \circ f_1(x)),$$

where $J_f(x)$ is the Jacobian of f at x, that is the matrix of the linear approximation of f in the neighborhood of x.

The linear approximation of a composition of mappings is the product of their individual linear approximations.