Deep Learning

Ulf Brefeld & Soham Majumder

build: April 15, 2024

Machine Learning Group
Leuphana University of Lüneburg

Multi-layer perceptrons

We will separate the linear operators and the non-linearities in different blocks in the figures.

We can combine several 'layers':

with $x^{(0)} = x$,

$$\forall \ell = 1, \ldots, L; x^{(\ell)} = \sigma\left(w^{(\ell)} x^{(\ell-1)} + b^{(\ell)}\right)$$

and $f(x; w, b) = x^{(L)}$.

We call this model a **Multi-layer Perceptron (MLP)**.

Note that if σ is a linear transformation

$x = w^T x + b$

$$\forall x \in \mathbb{R}^N, \sigma(x) = \alpha x + \beta \mathbf{1}$$

E.g. using $\alpha, \beta \in \mathbb{R}$

$$\forall \ell = 1, \ldots, L; x^{(\ell)} = \alpha w^{(\ell)} x^{(\ell-1)} + \alpha b^{(\ell)} + \beta,$$

the entire network reduces to an affine transformation

$$x^{\ell} = \alpha w^{(\ell)} x^{(\ell-1)} + \alpha b^{(\ell)} + \beta$$

$$f(x; w, b) = A^{(L)} x + B^{(L)}$$

where $A^{(0)} = \mathbf{1}, B^{(0)} = 0$, and

$$x^{\ell+1} = \alpha^{\ell+1} \left(\alpha w^{(\ell)} x^{(\ell-1)} + \alpha b^{(\ell)} \right) + \beta \mathbf{1}$$

$$\forall \ell < L, A^{(\ell)} = \alpha w^{(\ell)} A^{(\ell-1)}$$

$$B^{(\ell)} = \alpha w^{(\ell)} B^{(\ell-1)} + \alpha b^{(\ell)} + \beta$$

$$\boxed{A} * + b$$

Consequently, activation functions need to be non-linear, or the resulting MLP is an overparameterized linear mapping

Note that if σ is a linear transformation

$$\forall x \in \mathbb{R}^N, \sigma(x) = \alpha x + \beta \mathbf{1}$$

E.g. using $\alpha, \beta \in \mathbb{R}$

$$\forall \ell = 1, \ldots, L; x^{(\ell)} = \alpha w^{(\ell)} x^{(\ell-1)} + \alpha b^{(\ell)} + \beta,$$

the entire network reduces to an affine transformation

$$f(x; w, b) = A^{(L)} x + B^{(L)}$$

where $A^{(0)} = \mathbf{1}$, $B^{(0)} = 0$, and

$$\forall \ell < L: \quad A^{(\ell)} = \alpha w^{(\ell)} A^{(\ell-1)}$$
$$B^{(\ell)} = \alpha w^{(\ell)} B^{(\ell-1)} + \alpha b^{(\ell)} + \beta$$

!

Consequently, activation functions need to be non-linear, or the resulting MLP is an overparameterized linear mapping

Note that if σ is a linear transformation

$$\forall x \in \mathbb{R}^N, \sigma(x) = \alpha x + \beta \mathbf{1}$$

E.g. using $\alpha, \beta \in \mathbb{R}$

$$\forall \ell = 1, \ldots, L; x^{(\ell)} = \alpha w^{(\ell)} x^{(\ell-1)} + \alpha b^{(\ell)} + \beta,$$

the entire network reduces to an affine transformation

$$f(x; w, b) = A^{(L)} x + B^{(L)}$$

where $A^{(0)} = \mathbf{1}$, $B^{(0)} = 0$, and

$$\forall \ell < L: \quad \begin{aligned} A^{(\ell)} &= \alpha w^{(\ell)} A^{(\ell-1)} \\ B^{(\ell)} &= \alpha w^{(\ell)} B^{(\ell-1)} + \alpha b^{(\ell)} + \beta \end{aligned}$$

Consequently, activation functions need to be non-linear, or the resulting MLP is an overparameterized linear mapping

Rectified Linear Unit

$$\sigma(x) = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise} \end{cases}$$

Hyperbolic Tangent

$$\sigma(x) = \frac{\exp(2x) - 1}{\exp(2x) + 1}$$

Sigmoid

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

Softplus

$$\sigma(x) = \frac{1}{\beta} \log(1 + \exp(\beta x))$$

β is usually set to 1

ELU $(a = 1.5)$

Exponential Linear Unit

$$\sigma(x) = \begin{cases} x & \text{if } x > 0, \\ a(\exp(x) - 1) & \text{otherwise,} \end{cases}$$

where $a \geq 0$

Clevert, D. A., Unterthiner, T., and Hochreiter, S. (2016). Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). International Conference on Learning Representations.

Overview of activation functions

Universal approximation

In supervised learning, we are interested in approximating an unknown function by only knowing samples from that function.

Before we dive into how to train complicated neural networks: can neural networks learn any function?

In supervised learning, we are interested in approximating an unknown function by only knowing samples from that function.

Before we dive into how to train complicated neural networks: can neural networks learn any function?

We are going to show that a neural network with one hidden layer is able to approximate any given continuous function defined on some finite interval.

Main idea: use a linear combination of translated/scaled ReLU functions

$$f(x) = \alpha_1 \sigma(w_1 x + b_1)$$

$$f(x) = \alpha_1 \sigma(w_1 x + b_1) + \alpha_2 \sigma(w_2 x + b_2)$$

$$f(x) = \alpha_1 \sigma(w_1 x + b_1) + \alpha_2 \sigma(w_2 x + b_2) + \alpha_3 \sigma(w_3 x + b_3)$$

$$f(x) = \alpha_1 \sigma(w_1 x + b_1) + \alpha_2 \sigma(w_2 x + b_2) + \alpha_3 \sigma(w_3 x + b_3) + \ldots$$

$$f(x) = \alpha_1 \sigma(w_1 x + b_1) + \alpha_2 \sigma(w_2 x + b_2) + \alpha_3 \sigma(w_3 x + b_3) + \ldots$$

$$f(x) = \alpha_1 \sigma(w_1 x + b_1) + \alpha_2 \sigma(w_2 x + b_2) + \alpha_3 \sigma(w_3 x + b_3) + \ldots$$

$$f(x) = \alpha_1 \sigma(w_1 x + b_1) + \alpha_2 \sigma(w_2 x + b_2) + \alpha_3 \sigma(w_3 x + b_3) + \ldots$$

$$f(x) = \alpha_1 \sigma(w_1 x + b_1) + \alpha_2 \sigma(w_2 x + b_2) + \alpha_3 \sigma(w_3 x + b_3) + \ldots$$

$$f(x) = \alpha_1 \sigma(w_1 x + b_1) + \alpha_2 \sigma(w_2 x + b_2) + \alpha_3 \sigma(w_3 x + b_3) + \ldots$$

Similar results exist for other activation functions.

Cybenko, George. ”Approximation by superpositions of a sigmoidal function.” Mathematics of control, signals and systems 2.4 (1989): 303-314.

Extension to any real-valued continuous function ψ

First, we can use the previous result for the $\sin$ function:

$$\forall A > 0, \epsilon > 0, \exists N, (\alpha_n, a_n) \in \mathbb{R} \times \mathbb{R},$$
$$s.t. \max_{x \in [-A,A]} \left| \sin(x) - \sum_{n=1}^{N} \alpha_n \sigma(x - a_n) \right| \leq \epsilon$$

The Fourier series provides

$$\forall \psi \in \mathcal{P}([0,1]^D, \mathbb{R}), \delta > 0, \exists M, (v_m, \gamma_m, c_m) \in \mathbb{R}^D \times \mathbb{R} \times \mathbb{R},$$
$$s.t. \max_{x \in [0,1]^D} \left| \psi(x) - \sum_{m=1}^{M} \gamma_m \sin(v_m \cdot x + c_m) \right| \leq \delta$$

Extension to any real-valued continuous function ψ

First, we can use the previous result for the $\sin$ function:

$$\forall A > 0, \epsilon > 0, \exists N, (\alpha_n, a_n) \in \mathbb{R} \times \mathbb{R},$$
$$s.t. \max_{x \in [-A,A]} \left| \sin(x) - \sum_{n=1}^{N} \alpha_n \sigma(x - a_n) \right| \leq \epsilon$$

The Fourier series provides

$$\forall \psi \in \mathcal{P}([0,1]^D, \mathbb{R}), \delta > 0, \exists M, (v_m, \gamma_m, c_m) \in \mathbb{R}^D \times \mathbb{R} \times \mathbb{R},$$
$$s.t. \max_{x \in [0,1]^D} \left| \psi(x) - \sum_{m=1}^{M} \gamma_m \sin(v_m \cdot x + c_m) \right| \leq \delta$$

So, $\forall \xi > 0$ with

$$\delta = \frac{\xi}{2}, A = \max_{1 \le m \le M} \max_{x \in [0,1]^D} |v_m \cdot x + c_m| \quad \text{and} \quad \epsilon = \frac{\xi}{2 \sum_m |\gamma_m|}$$

we get for every $x \in [0,1]^D$,

$$\left| \psi(x) - \sum_{m=1}^{M} \gamma_m \left(\sum_{i=1}^{N} \alpha_n \sigma(v_m \cdot x + c_m - a_n) \right) \right|$$

$$\le \underbrace{\left| \psi(x) - \sum_{m=1}^{M} \gamma_m \sin(v_m \cdot x + c_m) \right|}_{\le \frac{\xi}{2}}$$

$$+ \underbrace{\sum_{m=1}^{M} |\gamma_m| \underbrace{\left| \sin(v_m \cdot x + c_m) - \sum_{n=1}^{N} \alpha_n \sigma(v_m \cdot x + c_m - a_n) \right|}_{\le \frac{\xi}{2 \sum_m |\gamma_m|}}}_{\le \frac{\xi}{2}}$$

So, $\forall \xi > 0$ with

$$\delta = \frac{\xi}{2}, A = \max_{1 \le m \le M} \max_{x \in [0,1]^D} |v_m \cdot x + c_m| \quad \text{and} \quad \epsilon = \frac{\xi}{2 \sum_m |\gamma_m|}$$

we get for every $x \in [0,1]^D$,

$$\left| \psi(x) - \sum_{m=1}^{M} \gamma_m \left(\sum_{i=1}^{N} \alpha_n \sigma(v_m \cdot x + c_m - a_n) \right) \right|$$

$$\le \underbrace{\left| \psi(x) - \sum_{m=1}^{M} \gamma_m \sin(v_m \cdot x + c_m) \right|}_{\le \frac{\xi}{2}}$$

$$+ \underbrace{\sum_{m=1}^{M} |\gamma_m| \underbrace{\left| \sin(v_m \cdot x + c_m) - \sum_{n=1}^{N} \alpha_n \sigma(v_m \cdot x + c_m - a_n) \right|}_{\le \frac{\xi}{2 \sum_m |\gamma_m|}}}_{\le \frac{\xi}{2}}$$

So, we can approximate any continuous function in a fixed interval, e.g., $\psi : [0,1]^D \to \mathbb{R}$, with a mapping of the form

$$x \mapsto \omega \cdot \sigma(wx + b),$$

where $b \in \mathbb{R}^K, w \in \mathbb{R}^{K \times D}$, and $\omega \in \mathbb{R}^K$, ie., with a one hidden layer perceptron.

Note: bettter approximations require a larger hidden layer (larger K)

This is what standard algorithms from the literature can learn in the same problem.

Training and gradient descent

We saw that training consists of finding the model parameters minimizing an empirical risk or loss, for instance the mean-squared error (MSE)

$w = \ldots$

$$E(w, b) = \frac{1}{N} \sum_n \left(f(x_n; w, b) - y_n \right)^2$$

There are other loss functions for classification, certain regression problems, or density estimation.

In our previous examples, we minimized the loss with ad-hoc recipes for the empirical error rate (perceptron).

There is generally no ad-hoc method. The logistic regression for instance

$$P_w(Y = 1|X = x) = \sigma(w \cdot x + b), \quad \text{with} \quad \sigma(x) = \frac{1}{1 + \exp(-x)}$$

leads to the loss

$$E(w, b) = -\sum_n \log \sigma\left(y_n(w \cdot x_n + b)\right)$$

which cannot be minimized analytically.

The general minimization method used in such a case is **gradient descent**.

Given a functional

$$f : \mathbb{R}^D \to \mathbb{R} \quad \text{with} \quad x \mapsto f(x_1, \ldots, x_D)$$

its gradient is the mapping

$$\nabla f : \mathbb{R}^D \to \mathbb{R}^D \quad \text{with} \quad x \mapsto \left(\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_D}(x) \right)$$

To minimize a functional

$$E : \mathbb{R}^{\mathbb{D}} \to \mathbb{R}$$

the gradient descent uses local information to iteratively move toward a (local) minimum.

For $w_0 \in \mathbb{R}^D$, consider an approximation of E around w_0,

$$\tilde{E}_{w_0} = E(w_0) + \nabla E(w_0)^\top (w - w_0) + \frac{1}{2\eta} \|w - w_0\|^2$$

Note that the quadratic term does not depend on E.

$$\tilde{E}_{w_0} = E(w_0) + \nabla E(w_0)^\top (w - w_0) + \frac{1}{2\eta}\|w - w_0\|^2$$

We have

$$\begin{aligned}
\underset{w}{\operatorname{argmin}}\, \tilde{E}_{w_0}(w) &= \cancel{E(w_0)} + \nabla E(w_0)^\top (w - \cancel{w_0}) + \frac{1}{2\eta}\|w - w_0\|^2 \\
&= \nabla E(w_0)^\top w + \frac{1}{2\eta}\|w - w_0\|^2 \\
&= 2\eta \nabla E(w_0)^\top w + \|w - w_0\|^2 \\
&= 2\eta \nabla E(w_0)^\top w + \|w\|^2 - 2w^\top w_0 + \cancel{\|w_0\|^2} \\
&= \|w - (w_0 - \eta \underbrace{\nabla E(w_0)}_{\text{gradient}})\|^2
\end{aligned}$$

which leads to

$$\underset{w}{\operatorname{argmin}}\, \tilde{E}_{w_0}(w) = \underbrace{w_0}_{\text{output point}} - \underbrace{\eta}_{\text{step size}} \nabla E(w_0) \rightarrow \text{perceptron! update}$$

The resulting iterative rule, which goes to the minimum of the approximation at the current location, takes the form:

$$w_{t+1} = w_t - \eta \nabla E(w_t)$$

Which corresponds intuitively to 'following the steepest descent'.

This finds a **local** minimum, and the choices of w_0 and η are important.

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.125$

$\eta = 0.5$

$\eta = 0.5$

$\eta = 0.5$

$\eta = 0.5$

$\eta = 0.5$

$\eta = 0.5$

$\eta = 0.5$

$\eta = 0.5$

$\eta = 0.5$

$\eta = 0.5$

$\eta = 0.5$

$\eta = 0.5$

Backpropagation

Train an MLP by minimizing a loss over the training set

$$E(w, b) = \sum_n V(f(x_n; w, b), y_n)$$

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

$$\frac{\partial E}{\partial w_{i,j}^{(\ell)}} \quad \text{and} \quad \frac{\partial E}{\partial b_i^{(\ell)}}$$

batch

mini batch

So, with $E_n = V(f(x_n; w, b), y_n)$, what we need is

$$\frac{\partial E_n}{\partial w_{i,j}^{(\ell)}} \quad \text{and} \quad \frac{\partial E_n}{\partial b_i^{(\ell)}}$$

online (perceptron)

Thus, we consider a single training instance x, and introduce $s^{(1)}, \ldots, s^{(L)}$ as the summations before activation functions:

$$x^{(0)} = x \xrightarrow{w^{(1)}, b^{(1)}} s^{(1)} \xrightarrow{\sigma} x^{(1)} \xrightarrow{w^{(2)}, b^{(2)}} s^{(2)} \xrightarrow{\sigma} \ldots \xrightarrow{w^{(L)}, b^{(L)}} s^{(L)} \xrightarrow{\sigma} x^{(L)}$$

Formally, we set $x^{(0)} = x$ and $\forall \ell = 1, \ldots, L$

$$s^{(\ell)} = w^{(\ell)} x^{(\ell-1)} + b^{(\ell)} \quad \text{and} \quad x^{(\ell)} = \sigma(s^{(\ell)})$$

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

This is the **forward pass**.

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

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

which generalizes to longer compositions and higher dimensions

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

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

What follows is exactly this principle applied to an MLP.

$$\ldots \xrightarrow{\sigma} x^{(\ell-1)} \xrightarrow{w^{(\ell)},b^{(\ell)}} s^{(\ell)} \xrightarrow{\sigma} x^{(\ell)} \xrightarrow{w^{(\ell+1)},b^{(\ell+1)}} s^{(\ell+1)} \xrightarrow{\sigma} \ldots$$

We have

$$s_i^{(\ell)} = \sum_j w_{i,j}^{(\ell)} x_j^{(\ell-1)} + b_i^{(\ell)},$$

so $w_{i,j}^{(\ell)}$ influences ℓ only through $s_i^{(\ell)}$ and we get

$$\frac{\partial E}{\partial w_{i,j}^{(\ell)}} = \frac{\partial E}{\partial s_i^{(\ell)}} \frac{\partial s_i^{(\ell)}}{\partial w_{i,j}^{(\ell)}} = \frac{\partial E}{\partial s_i^{(\ell)}} x_j^{(\ell-1)}$$

ℓ: either last layer L or we computed error back to layer $\ell+1$ already

and similarly

$$\frac{\partial E}{\partial b_i^{(\ell)}} = \frac{\partial E}{\partial s_i^{(\ell)}}.$$

Since we know $x_j^{(\ell-1)}$ from the forward pass, we only need $\frac{\partial E}{\partial s_i^{(\ell)}}$

$$\ldots \xrightarrow{\sigma} x^{(\ell-1)} \xrightarrow{w^{(\ell)},b^{(\ell)}} s^{(\ell)} \xrightarrow{\sigma} x^{(\ell)} \xrightarrow{w^{(\ell+1)},b^{(\ell+1)}} s^{(\ell+1)} \xrightarrow{\sigma} \ldots$$

We have

$$x_i^{(\ell)} = \sigma(s_i^{(\ell)}),$$

and since $s_i^{(\ell)}$ influences E only through x_i^{ℓ}, the chain rule gives

$$\frac{\partial E}{\partial s_i^{(\ell)}} = \frac{\partial E}{\partial x_i^{(\ell)}} \frac{\partial x_i^{(\ell)}}{\partial s_i^{(\ell)}} = \frac{\partial E}{\partial x_i^{(\ell)}} \sigma'\left(s_i^{(\ell)}\right)$$

Since we know $s_i^{(\ell)}$ from the forward pass, we only need $\frac{\partial E}{\partial x_i^{(\ell)}}$

$$\ldots \xrightarrow{\sigma} x^{(\ell-1)} \xrightarrow{w^{(\ell)},b^{(\ell)}} s^{(\ell)} \xrightarrow{\sigma} x^{(\ell)} \xrightarrow{w^{(\ell+1)},b^{(\ell+1)}} s^{(\ell+1)} \xrightarrow{\sigma} \ldots$$

Finally, we have

$$\frac{\partial E}{\partial x_i^{(L)}} = (\nabla_1 E)_i$$

where $\nabla_1 E$ is the gradient of E wrt its first parameter, that is the predicted value.

Also, $\forall \ell = 1, \ldots, L-1$, since

$$s_h^{(\ell+1)} = \sum_i w_{h,i}^{(\ell+1)} x_i^{(\ell)} + b_h^{(\ell+1)},$$

and $x_i^{(\ell)}$ influences E only through the $s_h^{(\ell+1)}$, we have

$$\frac{\partial E}{\partial x_i^{(\ell)}} = \sum_h \frac{\partial E}{\partial s_h^{(\ell+1)}} \frac{\partial s_h^{(\ell+1)}}{\partial x_i^{(\ell)}} = \sum_h \frac{\partial E}{\partial s_h^{(\ell+1)}} w_{h,i}^{(\ell+1)}$$

To write all this in tensorial form, if $\psi : \mathbb{R}^N \to \mathbb{R}^M$, we will use the standard Jacobian notation

$$\left[\frac{\partial \psi}{\partial x}\right] = \begin{pmatrix} \frac{\partial \psi_1}{\partial x_1} & \cdots & \frac{\partial \psi_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial \psi_M}{\partial x_1} & \cdots & \frac{\partial \psi_M}{\partial x_N} \end{pmatrix}$$

and if $\psi : \mathbb{R}^{N \times M} \to \mathbb{R}$, we will use the compact notation, also tensorial

$$\left[\!\left[\frac{\partial \psi}{\partial w}\right]\!\right] = \begin{pmatrix} \frac{\partial \psi}{\partial w_{1,1}} & \cdots & \frac{\partial \psi}{\partial w_{1,M}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \psi}{\partial w_{N,1}} & \cdots & \frac{\partial \psi}{\partial w_{N,M}} \end{pmatrix}$$

A standard notation (that we do not use here) is

$$\left[\frac{\partial E}{\partial x^{(\ell)}}\right] = \nabla_{x^{(\ell)}} E, \left[\frac{\partial E}{\partial s^{(\ell)}}\right] = \nabla_{s^{(\ell)}} E, \left[\!\left[\frac{\partial E}{\partial w^{(\ell)}}\right]\!\right] = \nabla_{w^{(\ell)}} E$$

$w^{(l)}$
$b^{(l)}$
$x^{(l-1)}$
$\times$
$+$
$s^{(l)}$
σ
$x^{(l)}$

$w^{(l)}$
$\left[\!\left[\frac{\partial \cancel{\ell} E}{\partial w^{(l)}} \right]\!\right]$
$b^{(l)}$
$\left[\frac{\partial \cancel{\ell} E}{\partial b^{(l)}} \right]$
$x^{(l-1)}$
$\times$
$+$
$s^{(l)}$
σ
$x^{(l)}$

$$\left[\frac{\partial \ell}{\partial x^{(l)}}\right]$$

$$\frac{\partial E}{\partial s_i^{(\ell)}} = \frac{\partial E}{\partial x_i^{(\ell)}} \sigma'\left(s_i^{(\ell)}\right)$$

← from forward pass

$$\frac{\partial E}{\partial x_j^{(\ell-1)}} = \sum_i w_{i,j}^{(\ell)} \frac{\partial E}{\partial s_i^{(\ell)}}$$

$$\frac{\partial E}{\partial b_i^{(\ell)}} = \frac{\partial E}{\partial s_i^{(\ell)}} \cdot 1$$

$$\frac{\partial E}{\partial w_{i,j}^{(\ell)}} = \frac{\partial E}{\partial s_i^{(\ell)}} x_j^{(\ell-1)}$$

$w^{(l)}$
$\left[\!\left[\frac{\partial \ell}{\partial w^{(l)}}\right]\!\right]$
$b^{(l)}$
$\left[\frac{\partial \ell}{\partial b^{(l)}}\right]$
$\times\ .^T$
$x^{(l-1)}$
$\times$
$+$
$s^{(l)}$
σ
$x^{(l)}$
$\left[\frac{\partial \ell}{\partial x^{(l-1)}}\right]$
$.^T\ \times$
$\left[\frac{\partial \ell}{\partial s^{(l)}}\right]$
$\odot\ \sigma'$
$\left[\frac{\partial \ell}{\partial x^{(l)}}\right]$

Forward pass

Compute the activations

$$x^{(0)} = x, \;\; s^{(\ell)} = w^{(\ell)} x^{(\ell-1)} + b^{(\ell)} \text{ and } x^{(\ell)} = \sigma(s^{(\ell)})$$

Backward pass

Compute the derivations of the loss wrt the activations

$$\left[\frac{\partial E}{\partial x^{(L)}}\right] = \nabla_1 E\left(x^{(L)}\right), \qquad \left[\frac{\partial E}{\partial s^{(\ell)}}\right] = \left[\frac{\partial E}{\partial x^{(\ell)}}\right] \odot \sigma'\left(s^{(\ell)}\right)$$

$$\left[\frac{\partial E}{\partial x^{(\ell)}}\right] = \left(w^{(\ell+1)}\right)^\top \left[\frac{\partial E}{\partial s^{(\ell+1)}}\right]$$

Compute the derivatives of the loss wrt the parameters

$$\left[\!\left[\frac{\partial E}{\partial w^{(\ell)}}\right]\!\right] = \left[\frac{\partial E}{\partial s^{(\ell)}}\right]\left(x^{(\ell-1)}\right)^\top, \qquad \left[\frac{\partial E}{\partial b^{(\ell)}}\right] = \left[\frac{\partial E}{\partial s^{(\ell)}}\right]$$

Gradient step

Update the parameters

$$w^{(\ell)} \leftarrow w^{(\ell)} - \eta\left[\!\left[\frac{\partial E}{\partial w^{(\ell)}}\right]\!\right], \qquad b^{(\ell)} \leftarrow b^{(\ell)} - \eta\left[\frac{\partial E}{\partial b^{(\ell)}}\right]$$

In spite of its hairy formalization, the backward pass is a simple algorithm: apply the chain rule again and again.

As for the forward pass, it can be expressed in tensorial form. Heavy computation is concentrated in linear operations, and all the non-linearities go into component-wise operations.

Regarding computation, since the costly operation for the forward pass is

$$s^{(\ell)} = w^{(\ell)} x^{(\ell-1)} + b^{(\ell)}$$

and for the backward pass

$$\left[\frac{\partial E}{\partial x^{(\ell)}}\right] = \left(w^{(\ell+1)}\right)^{\top} \left[\frac{\partial E}{\partial s^{(\ell+1)}}\right]$$

and

$$\left[\!\!\left[\frac{\partial E}{\partial w^{(\ell)}}\right]\!\!\right] = \left[\frac{\partial E}{\partial s^{(\ell)}}\right] \left(x^{(\ell-1)}\right)^{\top}$$

the rule of thumb is that the backward pass is twice more expensive than the forward one.

References

D. Perekrestenko, P. Grohs, D. Elbrächter, H. Bölcskei. The Universal Approximation Power of Finite-Width Deep ReLU Networks, arXiv preprint arXiv:1703.02518