Aprentatge Automàtic 1

GCED

Lluís A. Belanche

belanche@cs.upc.edu

Soft Computing Research Group
Dept. de Ciències de la Computació (Computer Science)

Universitat Politècnica de Catalunya

2019-2020

LECTURE 8b: Artificial neural networks (I)

Artificial neural networks (I): the MLP

A gentle exposition of backpropagation

A regression MLP of c hidden layers is a function $F : \mathbb{R}^d \to \mathbb{R}^m$ made up of pieces $F_1, \ldots, F_m$ of the form:

$$F_k(\boldsymbol{x}) = g\left(\sum_{j=0}^{H_c} w_{kj}^{(c+1)} \phi_j^{(c)}(\boldsymbol{x})\right), k = 1, \ldots, m$$

where, for every $l = 1, \ldots, c$, $W^{(l)} = [w_{ji}^{(l)}]$ is the matrix of weights connecting layers $l-1$ and l, H_l is the size of hidden layer l and

$$\phi_j^{(l)}(\boldsymbol{x}) = g\left(\sum_{i=0}^{H_{l-1}} w_{ji}^{(l)} \phi_i^{(l-1)}(\boldsymbol{x})\right), \text{ for } l = 1, \ldots, c$$

with $\phi_i^{(0)}(\boldsymbol{x}) = x_i, \phi_0^{(l)}(\boldsymbol{x}) = 1$ (in particular, $x_0 = 1$) and $H_0 = d$.

Artificial neural networks (I): the MLP

A gentle exposition of backpropagation

The goal in **regression** is to minimize the empirical error of the network on the training data sample $S = \{(\boldsymbol{x}_n, \boldsymbol{t}_n)\}_{n=1,\dots,N}$, where $\boldsymbol{x}_n \in \mathbb{R}^d, \boldsymbol{t}_n \in \mathbb{R}^m$.

$$E_{emp}(\boldsymbol{\omega}) := \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{m} (t_{nk} - F_k(\boldsymbol{x}_n))^2$$

where $\boldsymbol{\omega}$ is the vector of all network weights

Artificial neural networks (I): the MLP

A gentle exposition of backpropagation

- Note that, if g admits a derivative everywhere, $E_{emp}(\boldsymbol{\omega})$ is a differentiable function of every weight $w_{ji}^{(l)}$

- If we want to apply **gradient descent**, we need to compute the partial derivative of the error w.r.t. every weight, the gradient vector:

$$\nabla E_{emp}(\boldsymbol{\omega}) = \left(\frac{\partial E_{emp}(\boldsymbol{\omega})}{\partial w_{ji}^{(l)}} \right)_{l,j,i}$$

- There exists a reasonably efficient algorithm for computing this gradient vector: the **backpropagation algorithm**

Artificial neural networks (I): the MLP

A gentle exposition of backpropagation

Consider a MLP where, for notational simplicity, we define:

$$z_j^{(l)} := g(a_j^{(l)}) := g\left(\sum_i w_{ji}^{(l)} z_i^{(l-1)}\right),\; z_j^{(0)} = x_j$$

- Note that E_{emp} is the sum of the (independent) errors for every input/output example $(\boldsymbol{x}_n, \boldsymbol{t}_n)$:

$$E_{emp}(\boldsymbol{\omega}) = \sum_{n=1}^{N} \frac{1}{2} \sum_{k=1}^{m} (t_{nk} - F_k(\boldsymbol{x}_n))^2 := \sum_{n=1}^{N} E_{emp}^{(n)}(\boldsymbol{\omega})$$

Artificial neural networks (I): the MLP

A gentle exposition of backpropagation

Therefore

$$\frac{\partial E_{emp}(\boldsymbol{\omega})}{\partial w_{ji}^{(l)}} = \sum_{n=1}^{N} \frac{\partial E_{emp}^{(n)}(\boldsymbol{\omega})}{\partial w_{ji}^{(l)}}$$

The updating formula for the weights is:

$$w_{ji}^{(l)}(t+1) := w_{ji}^{(l)}(t) - \alpha \left. \frac{\partial E_{emp}(\boldsymbol{\omega})}{\partial w_{ji}^{(l)}} \right|_{\boldsymbol{\omega}=\boldsymbol{\omega}(t)}$$

Artificial neural networks (I): the MLP

A gentle exposition of backpropagation

Suppose we present $\boldsymbol{x}_n$ to the network and compute all the neuron's outputs $z_j^{(l)}$ (this is known as the **forward propagation**). Now,

$$\Delta^n w_{ji}^{(l)} := \frac{\partial E_{emp}^{(n)}(\boldsymbol{\omega})}{\partial w_{ji}^{(l)}} = \frac{\partial E_{emp}^{(n)}(\boldsymbol{\omega})}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial w_{ji}^{(l)}} = \delta_j^{(l)} \cdot z_i^{(l-1)}$$

where we have defined $\delta_j^{(l)} := \frac{\partial E_{emp}^{(n)}(\boldsymbol{\omega})}{\partial a_j^{(l)}}$.

Artificial neural networks (I): the MLP

Backpropagation algorithm (BPA)

Set initial values for the weights $w_{ji}^{(l)}$

repeat

forall n **in** $1 \leq n \leq N$

1. **Forward pass** Present $\boldsymbol{x}_n$ and compute the outputs $z_i^{(l)}$ of all the units
2. **Backward pass** Compute the deltas $\delta_j^{(l)}$ of all the units, from $l = c+1$ down to $l = 1$:
   a. **if** $l = c+1$ **then** $\delta_j^{(l)} := g'(a_j^{(c+1)}) \cdot (z_j^{(c+1)} - t_{nj})$
   b. **if** $l < c+1$ **then** $\delta_j^{(l)} := g'(a_j^{(l)}) \sum_q \delta_q^{(l+1)} w_{qj}^{(l+1)}$
3. Set $\Delta^n w_{ji}^{(l)} := \delta_j^{(l)} \cdot z_i^{(l-1)}$

end

Update the weights as $w_{ji}^{(l)}(t+1) := w_{ji}^{(l)}(t) + \alpha \sum_{n=1}^{N} \Delta^n w_{ji}^{(l)}$

until convergence **or** max. epochs

Artificial neural networks (I): the MLP

A gentle exposition of backpropagation

Since $\left(g_\beta^{\text{log}}(z)\right)' = \beta g_\beta^{\text{log}}(z)\left[1 - g_\beta^{\text{log}}(z)\right]$ we obtain:

$$g'(a_j^{(c+1)}) = \beta g(a_j^{(c+1)})\Big(1 - g(a_j^{(c+1)})\Big) = \beta z_j^{(c+1)}(1 - z_j^{(c+1)})$$

$$g'(a_j^{(l)}) = \beta g(a_j^{(l)})\Big(1 - g(a_j^{(l)})\Big) = \beta z_j^{(l)}(1 - z_j^{(l)})$$

Analogously for $\left(g_\beta^{\text{tanh}}(z)\right)' = \beta^2 \left(1 - \left(g_\beta^{\text{tanh}}(z)\right)^2\right)$