Multilayer perceptrons

Course of Machine Learning
Master Degree in Computer Science
University of Rome "Tor Vergata"
a.a. 2024-2025

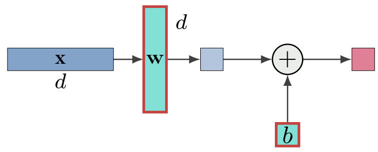
Giorgio Gambosi

Multilayer networks

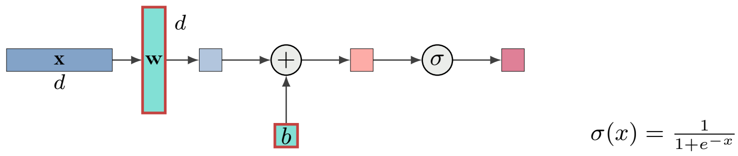
Up to now, only models with a single level of parameters to be learned were considered. Such models are based on generalized linear model structures of the type $y(\mathbf{x}) = f(\mathbf{w}^T\mathbf{x} + b)$, where model parameters are directly applied to input values. More general classes of models can be defined by defining sequences of transformations applied on input data, corresponding to multilayered networks of functions.

For example,

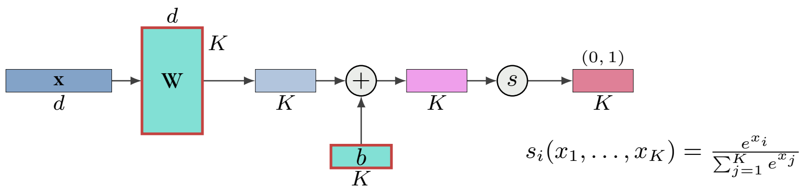
Linear regression



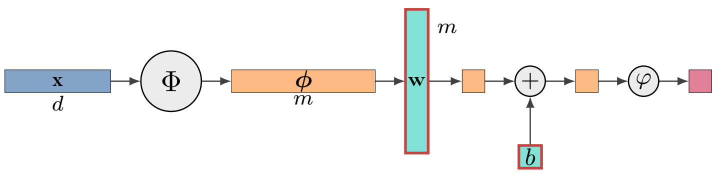
Logistic regression



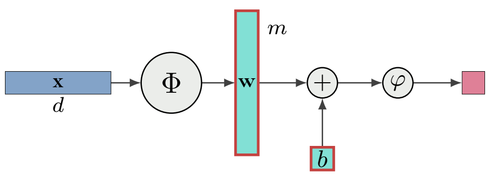
Softmax regression



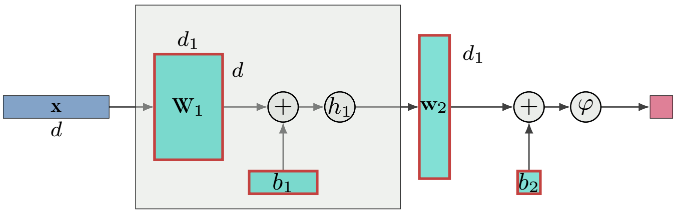
The case when base functions are defined and applied can be represented as



A simplified description,



The role of predefined base functions can be taken by a set of generalized linear models with parameter values learnable from data.



This corresponds to adding a first layer of computing units, which from the d-dimensional input vector $\mathbf{x} = (x_1, \dots, x_d)$ derives a vector $\mathbf{a} = (a_1, \dots, a_{d_1})$ of $d_1 > 0$ activations through suitable linear combinations of $x_1, \dots, x_d$

$$a_j = \sum_{i=1}^{d} w_{ji} x_i + b_j = \overline{\mathbf{w}}_j^T \overline{\mathbf{x}}$$

Each activation a_j is transformed by means of a non-linear activation function h_1 to provide an ouput value

$$z_j = h_1(a_j) = h_1(\mathbf{w}_j^T \mathbf{x} + b_j)$$

here h_1 is some approximate threshold function, such as a sigmoid

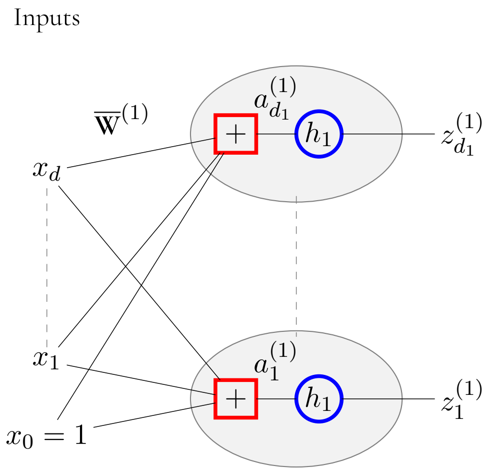
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

or a hyperbolic tangent

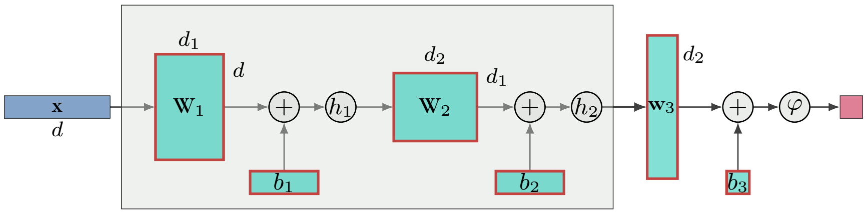
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1}{1 + e^{-2x}} - \frac{1}{1 + e^{2x}} = \sigma(2x) - \sigma(-2x)$$

An output vector $\mathbf{z} = (z_1, \dots, z_{d_1})$ is then produced.

First layer



The approach can be iterated, adding more layers with the same structure, and resulting in a multilayer network. Here, one additional layer is added.



Now, vector $\mathbf{z}^{(1)}$ provides an input to the next layer, where d_2 hidden units compute a vector $\mathbf{z}^{(2)} = (z_1^{(2)}, \ldots, z_{d_2}^{(1)})^T$ by first performing linear combinations of the input values

$$a_k^{(2)} = \sum_{i=1}^{d_1} w_{ki}^{(2)} z_i^{(1)} + b_k^{(2)} = \overline{\mathbf{w}}_k^{(2)T} \overline{\mathbf{z}}^{(1)}$$

and then applying function h_2, as follows

$$z_k^{(2)} = h_2(\overline{\mathbf{w}}_k^{(2)T} \overline{\mathbf{z}}^{(1)})$$

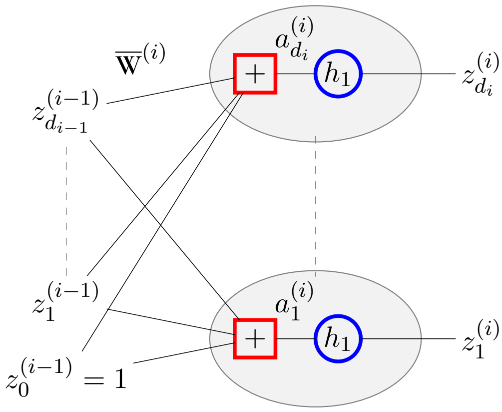
Usually, $h_2 = h_1$, that is, a same activation function is applied for all layers (except the last one).

The same structure can be repeated for each inner layer, where layer r has d_r units which, from input vector $\mathbf{z}^{(r-1)}$, derive output vector $\mathbf{z}^{(r-1)}$ through linear combinations

$$a_k^{(r)} = \overline{\mathbf{w}}_k^{(r)T} \overline{\mathbf{z}}^{(r-1)}$$

and non linear transformation

$$z_k^{(r)} = h_r(\overline{\mathbf{w}}_k^{(r)T} \overline{\mathbf{z}}^{(r-1)})$$



Multilayer network structure: output layer

For what concerns the last layer, say layer D, an output vector $\mathbf{y} = \mathbf{z}^{(D)}$ is again produced by means of d_D output units by first performing linear combinations on $\mathbf{z}^{(D-1)}$

$$a_k^{(D)} = \overline{\mathbf{w}}_k^{(D)T} \overline{\mathbf{z}}^{(D-1)}$$

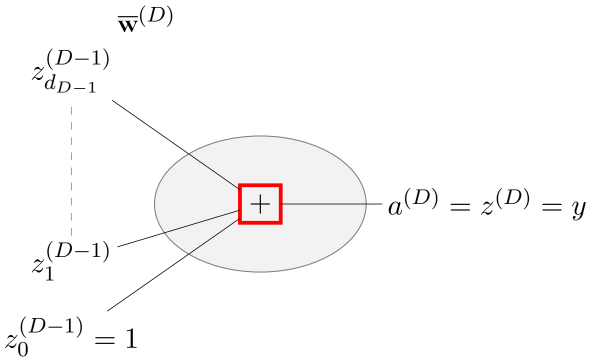
and then applying function φ

$$y_k = z_k^{(D)} = \varphi({\overline{\mathbf{w}}_k^{(D)}}^T \overline{\mathbf{z}}^{(D-1)})$$

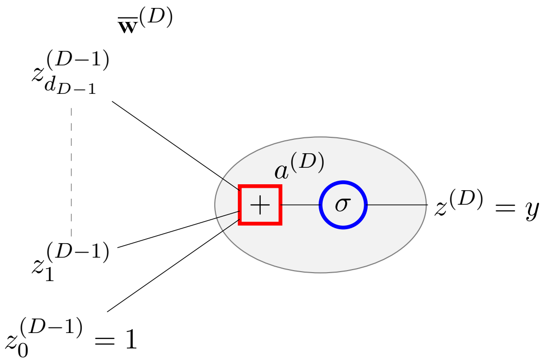
where:

- φ is the identity function in the case of regression
- φ is a sigmoid in the case of binary classification
- φ is a softmax in the case of multiclass classification

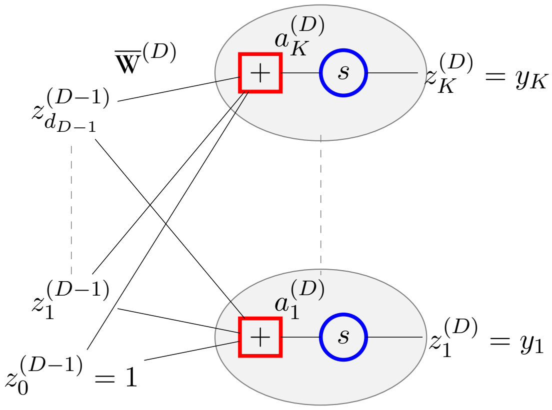
Output layer: regression



Output layer: binary classification



Output layer: K-class classification



3 layer networks

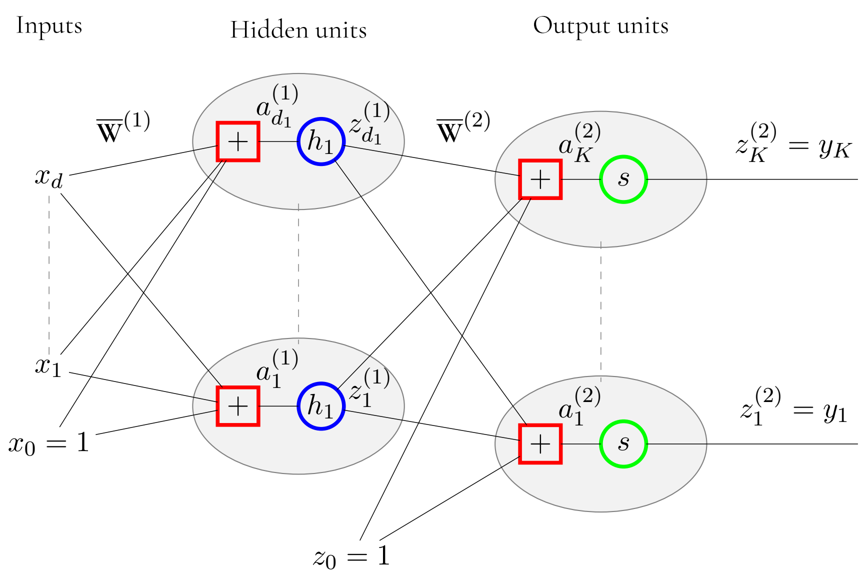
A sufficiently powerful model is provided in the case of 3 layers (input, hidden, output).

For example, applying this model for K-class classification corresponds to the following overall network function for each y_k, $k = 1, \ldots, K$

$$y_k = s\left(\sum_{j=1}^{d_1} w_{kj}^{(2)} h\left(\sum_{i=1}^{d} w_{ji}^{(1)} x_i + b_j^{(1)}\right) + b_k^{(2)}\right)$$

where the number d_1 of hidden units is a model structure parameter and s is the softmax function.

The resulting network can be seen as a GLM where base functions are not predefined wrt to data, but are instead parameterized by coefficients in $\overline{\mathbf{W}}^{(1)}$.



Approximating functions with neural networks

Neural networks, despite their simple structure, are sufficient powerful models to act as **universal approximators**.

It is possible to prove that any continuous function can be approximated, at any by means of two-layered neural networks with sigmoidal activation functions. The approximation can be indefinitely precise, as long as a suitable number of hidden units is defined.

Iterative methods to minimize $E(\overline{\mathbf{W}})$

The error function $E(\overline{\mathbf{W}})$ is usually quite hard to minimize:

- there exist many local minima
- for each local minimum there exist many equivalent minima
  - any permutation of hidden units provides the same result
  - changing signs of all input and output links of a single hidden unit provides the same result

Analytical approaches to minimization cannot be applied: resort to iterative methods (possibly comparing results from different runs).

$$\overline{\mathbf{W}}^{(k+1)} = \overline{\mathbf{W}}^{(k)} + \Delta\overline{\mathbf{W}}^{(k)}$$

Gradient descent

At each step, two stages:

1. the derivatives of the error functions wrt all weights are evaluated at the current point
2. weights are adjusted (resulting into a new point) by using the derivatives

On-line (stochastic) gradient descent

We exploit the property that the error function is the sum of a collection of terms, each characterizing the error corresponding to each observation

$$E(\overline{\mathbf{W}}) = \sum_{i=1}^{n} E_i(\overline{\mathbf{W}})$$

the update is based on one training set element at a time

$$\overline{\mathbf{W}}^{(k+1)} = \overline{\mathbf{W}}^{(k)} - \eta \nabla E_i(\overline{\mathbf{W}})\Big|_{\overline{\mathbf{W}}^{(k)}}$$

- at each step the weight vector is moved in the direction of greatest decrease wrt the error for a specific data element
- only one training set element is used at each step: less expensive at each step (more steps may be necessary)
- makes it possible to escape from local minima

Batch gradient descent

The gradient is computed by considering a subset (batch) B of the training set

$$\overline{\mathbf{W}}^{(k+1)} = \overline{\mathbf{W}}^{(k)} - \eta \sum_{\mathbf{x}_i \in B} \nabla E_i(\overline{\mathbf{W}})\Big|_{\overline{\mathbf{W}}^{(k)}}$$

Computing gradients

In order to apply a gradient based method, the set of derivatives

$$\frac{\partial E(\overline{\mathbf{W}})}{\partial w_{ij}^{(k)}}$$

and

$$\frac{\partial E(\overline{\mathbf{W}})}{\partial b_{j}^{(k)}}$$

must be derived for all i, j, k in order to be iteratively evaluated for different values of $\mathbf{w}$ during gradient descent. For the sake of brevity, assume $b_j^{(k)} = b_j^{(k)}$ in the following.

As we shall see, in order to evaluate

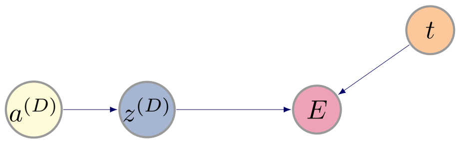
$$\frac{\partial E((\overline{\mathbf{W}}))}{\partial w_{ij}^{(k)}}$$

we may start by evaluating

$$\frac{\partial E((\overline{\mathbf{W}}))}{\partial a_{i}^{(D)}}$$

that is the derivatives of the cost function wrt each activation value $a_1^{(D)}, \ldots, a_{d_D}^{(D)}$ at the final layer (the D-th, here) of the network.

Regression



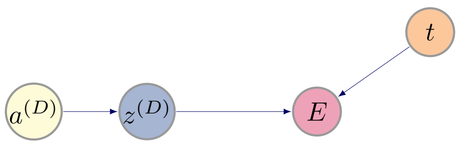
Here, we have $y = z^{(D)} = a^{(D)}$ and

$$E = \frac{1}{2}(y-t)^2 = \frac{1}{2}(z^{(D)}-t)^2 = \frac{1}{2}(a^{(D)}-t)^2$$

as a consequence,

$$\frac{\partial E}{\partial a^{(D)}} = a^{(D)} - t = z^{(D)} - t$$

Binary classification



Here, we have $y = z^{(D)} = \sigma(a^{(D)})$ and

$$E = -(t \log y + (1-t)\log(1-y)) = -(t \log z^{(D)} + (1-t)\log(1-z^{(D)}))$$

$$\frac{\partial E}{\partial z^{(D)}} = -\left(\frac{t}{z^{(D)}} - \frac{1-t}{1-z^{(D)}}\right) = \frac{z^{(D)}-t}{z^{(D)}(1-z^{(D)})}$$

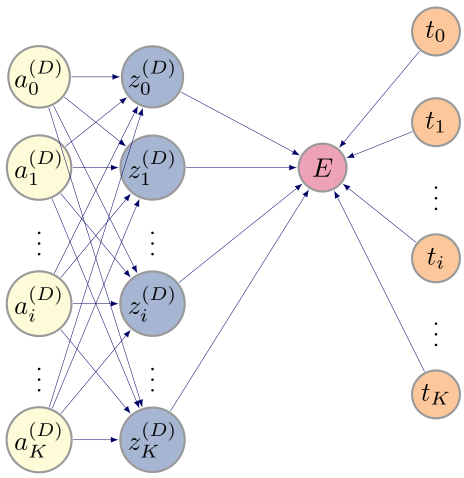
since, by the properties of the logistic function,

$$\frac{\partial z^{(D)}}{\partial a^{(D)}} = \sigma(a^{(D)})(1 - \sigma(a^{(D)})) = z^{(D)}(1 - z^{(D)})$$

it results

$$\frac{\partial E}{\partial a^{(D)}} = \frac{\partial E}{\partial z^{(D)}} \frac{\partial z^{(D)}}{\partial a^{(D)}} = z^{(D)} - t$$

K-class classification



Here, we have

$$y_i = z_i^{(D)} = \frac{e^{a_i^{(D)}}}{\sum_{j=1}^{K} e^{a_j^{(D)}}}$$

and

$$E = -\sum_{i=1}^{K} t_i \log z_i^{(D)}$$

$$\frac{\partial E}{\partial z_i^{(D)}} = -\frac{t_i}{z_i^{(D)}}$$

Since

$$\frac{\partial z_i^{(D)}}{\partial a_i^{(D)}} = \frac{e^{a_i^{(D)}} \sum_{j=1}^{K} e^{a_j^{(D)}} - e^{a_i^{(D)}} e^{a_i^{(D)}}}{\left(\sum_{j=1}^{K} e^{a_j^{(D)}}\right)^2} = z_i^{(D)} - z_i^{(D)} z_i^{(D)} = z_i^{(D)}(1 - z_i^{(D)})$$

$$\frac{\partial z_i^{(D)}}{\partial a_j^{(D)}} = \frac{-e^{a_i^{(D)}} e^{a_j^{(D)}}}{\left(\sum_{j=1}^{K} e^{a_j^{(D)}}\right)^2} = -z_i^{(D)} z_j^{(D)} \qquad i \neq j$$

it results

$$\frac{\partial E}{\partial a_i^{(D)}} = -\sum_{j=1}^{K} \frac{\partial l}{\partial z_j^{(D)}} \frac{\partial z_j^{(D)}}{\partial a_i^{(D)}} = -\sum_{j=1}^{K} \frac{t_j}{z_j^{(D)}} \frac{\partial z_j^{(D)}}{\partial a_i^{(D)}}$$

$$= -\frac{t_i}{z_i^{(D)}} z_i^{(D)}(1 - z_i^{(D)}) + \sum_{\substack{1\le j\le K\\ j\ne i}} \frac{t_j}{z_j^{(D)}} z_i^{(D)} z_j^{(D)} = -t_i(1 - z_i^{(D)}) + \sum_{\substack{1\le j\le K\\ j\ne i}} t_j z_i^{(D)}$$

$$= z_i^{(D)} \sum_{j=1}^{K} t_j - t_i = z_i^{(D)} - t_i$$

Backpropagation

Algorithm applied to evaluate derivatives of the error wrt all weights

It can be interpreted in terms of backward propagation of a computation in the network, from the output towards input units.

It provides an efficient method to evaluate derivatives wrt weights. It can be applied also to compute derivatives of output wrt to input variables, to provide evaluations of the Jacobian and the Hessian matrices at a given point.

Let us now show that, for any layer, knowing the current weights $w_{ij}^{(r)}$ and the values $a_i^{(r)}, z_i^{(r)}$ resulting by submitting the current item to the network, the knowledge of the derivatives

$$\frac{\partial E}{\partial a_j^{(r)}} \qquad 1 \le j \le d_r$$

makes it possible to compute the derivatives

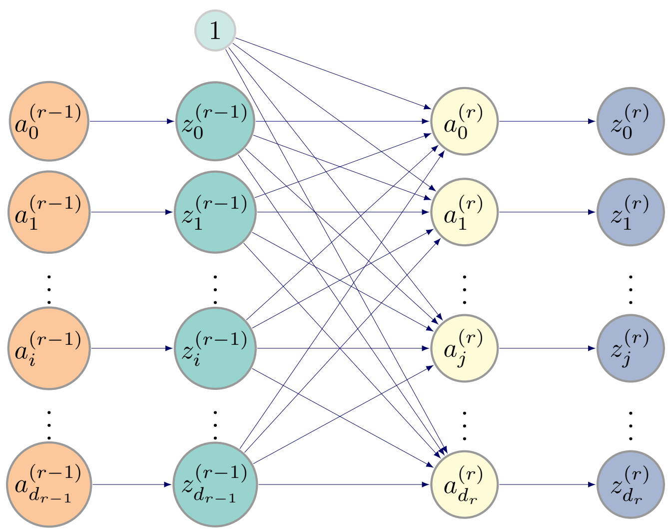
$$\frac{\partial E}{\partial w_{ij}^{(r)}} \qquad 0 \le i \le d_{r-1}, 1 \le j \le d_r$$

to be applied for gradient descent, and

$$\frac{\partial E}{\partial a_i^{(r-1)}} \qquad 1 \le i \le d_{r-1}$$

where d_s is the number of units at the s-th layer

Backpropagation at layer r



Here,

$$a_j^{(r)} = \sum_{i=1}^{d_{r-1}} w_{ij}^{(r)} z_i^{(r-1)} + b_i^{(r)}$$

$$\frac{\partial a_j^{(r)}}{\partial w_{ij}^{(r)}} = z_i^{(r-1)} \qquad \frac{\partial a_j^{(r)}}{\partial b_j^{(r)}} = 1$$

and, as a consequence,

$$\frac{\partial E}{\partial w_{ij}^{(r)}} = \frac{\partial E}{\partial a_j^{(r)}} \frac{\partial a_j^{(r)}}{\partial w_{ij}^{(r)}} = \frac{\partial E}{\partial a_j^{(r)}} z_i^{(r-1)}$$

$$\frac{\partial E}{\partial b_j^{(r)}} = \frac{\partial E}{\partial a_j^{(r)}} \frac{\partial a_j^{(r)}}{\partial b_j^{(r)}} = \frac{\partial E}{\partial a_j^{(r)}}$$

Moreover, since $\frac{\partial a_j^{(r)}}{\partial z_i^{(r-1)}} = w_{ij}^{(r)}$, it results

$$\frac{\partial E}{\partial z_i^{(r-1)}} = \sum_{j=1}^{d_r} \frac{\partial E}{\partial a_j^{(r)}} \frac{\partial a_j^{(r)}}{\partial z_i^{(r-1)}} = \sum_{j=1}^{d_r} \frac{\partial E}{\partial a_j^{(r)}} w_{ij}^{(r)}$$

and since $z_j^{(r)} = h(a_j^{(r)})$, then $\frac{\partial z_j^{(r)}}{\partial a_j^{(r)}} = h'(a_j^{(r)})$ and

$$\frac{\partial E}{\partial a_i^{(r-1)}} = \frac{\partial E}{\partial z_i^{(r-1)}} \frac{\partial z_i^{(r-1)}}{\partial a_i^{(r-1)}} = \frac{\partial E}{\partial z_i^{(r-1)}} h'(a_i^{(r-1)}) = h'(a_i^{(r-1)}) \sum_{j=1}^{d_r} \frac{\partial l}{\partial a_j^{(r)}} w_{ij}^{(r)}$$

Reassuming, it results

$$\frac{\partial E}{\partial a^{(D)}} = z^{(D)} - t \qquad \text{for regression and binary classification}$$

$$\frac{\partial E}{\partial a_i^{(D)}} = z_i^{(D)} - t_i \qquad \text{for multiclass classification}$$

and, for each layer $r = D, \dots, 2$

$$\frac{\partial E}{\partial w_{ij}^{(r)}} = \frac{\partial E}{\partial a_j^{(r)}} z_i^{(r-1)}$$

$$\frac{\partial E}{\partial b_j^{(r)}} = \frac{\partial E}{\partial a_j^{(r)}}$$

$$\frac{\partial E}{\partial a_i^{(r-1)}} = h'(a_i^{(r-1)}) \sum_{j=1}^{d_r} \frac{\partial E}{\partial a_j^{(r)}} w_{ij}^{(r-1)}$$

For the first layer,

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \frac{\partial E}{\partial a_j^{(1)}} x_i$$

$$\frac{\partial E}{\partial b_j^{(1)}} = \frac{\partial E}{\partial a_j^{(1)}}$$

Backpropagation and activation functions

In the case of a sigmoidal activation function $h(x) = \sigma(x)$, it results, in particular,

$$\frac{\partial E}{\partial a_i^{(r-1)}} = \sigma(a_i^{(r-1)})(1 - \sigma(a_i^{(r-1)})) \sum_{j=1}^{d_r} \frac{\partial E}{\partial a_j^{(r)}} w_{ij}^{(r-1)}$$

while if a RELU activation function is applied, we get

$$\frac{\partial E}{\partial a_i^{(r-1)}} = \begin{cases} \displaystyle\sum_{j=1}^{d_r} \frac{\partial E}{\partial a_j^{(r)}} w_{ij}^{(r-1)} & \text{if } a_i^{(r-1)} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Backpropagation

Iterate the preceding steps on all items in the batch set. In fact, since

$$E(\overline{\mathbf{W}}) = \sum_{i=1}^{n} E_i(\overline{\mathbf{W}})$$

it is

$$\frac{\partial E}{\partial w_{jl}^{(r)}} = \sum_{i=1}^{n} \frac{\partial E_i}{\partial w_{jl}^{(r)}}$$

This provides an evaluation of $\nabla E(\overline{\mathbf{W}})$ at the current point $\overline{\mathbf{W}}^{(k)}$.

Once $\nabla E(\overline{\mathbf{W}})\big|_{\overline{\mathbf{W}}^{(k)}}$ is known, a single step of gradient descent can be performed

$$\overline{\mathbf{W}}^{(k+1)} = \overline{\mathbf{W}}^{(k)} - \eta \nabla E(\overline{\mathbf{W}})\Big|_{\overline{\mathbf{W}}^{(k)}}$$

Computational efficiency of backpropagation

A single evaluation of error function derivatives requires $O(|\overline{\mathbf{W}}|)$ steps

Alternative approach: finite differences. Perturb each weight w_{ij} in turn and approximate the derivative as follows

$$\frac{\partial E_i}{\partial w_{ij}} = \frac{E_i(w_{ij} + \varepsilon) - E_i(w_{ij} - \varepsilon)}{2\varepsilon} + O(\varepsilon^2)$$

This requires $O(|\overline{\mathbf{W}}|)$ steps for each weight, hence $O(|\overline{\mathbf{W}}|^2)$ steps overall.