Multilayer perceptrons

Course of Machine Learning
Master Degree in Computer Science
University of Rome "Tor Vergata"
a.a. 2023-2024

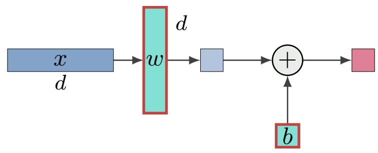
Giorgio Gambosi

Multilayer networks

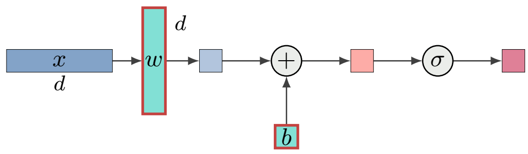
- Up to now, only models with a single level of parameters to be learned were considered.
- The model has a generalized linear model structure such as $y = f(\mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}))$: model parameters are directly applied to input values.
- More general classes of models can be defined by means of sequences of transformations applied on input data, corresponding to multilayered networks of functions.

Extended linear models

Linear regression

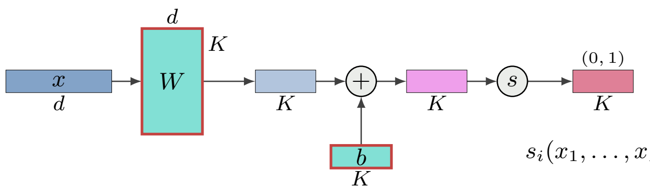


Logistic regression



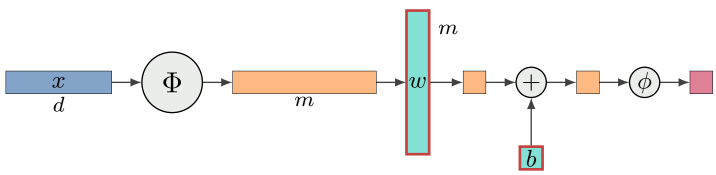
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Softmax regression

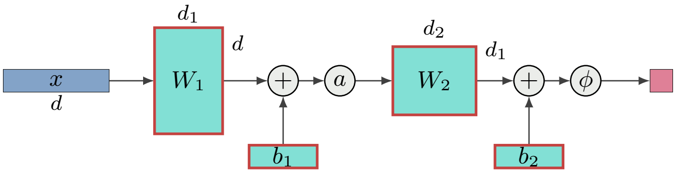


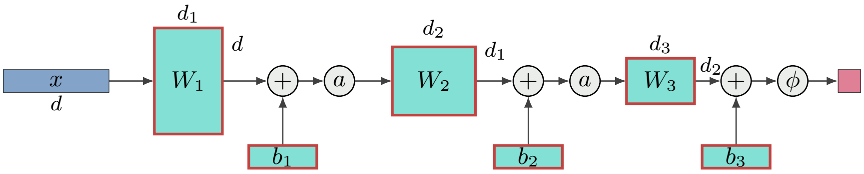
$$s_i(x_1, \ldots, x_K) = \frac{e^{x_i}}{\sum_{j=1}^K e^{x_j}}$$

Adding base functions



Adding a layer





Multilayer network structure: first layer

For any d-dimensional input vector $\mathbf{x} = (x_1, \ldots, x_d)$, the first layer of a neural network derives $m_1 > 0$ activations $a_1^{(1)}, \ldots, a_{m_1}^{(1)}$ through suitable linear combinations of $x_1, \ldots, x_d$

$$a_j^{(1)} = \sum_{i=1}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)} = \mathbf{w}_j^{(1)} \cdot \overline{\mathbf{x}}$$

where M is a given, predefined, parameter and $\overline{\mathbf{x}} = (1, x_1, \ldots, x_d)^T$.

Each activation $a_j^{(1)}$ is tranformed by means of a non-linear activation function h_1 to provide a vector $\mathbf{z}^{(1)} = (z_1^{(1)}, \ldots, z_{m_1}^{(1)})^T$ as output from the layer, as follows

$$z_j^{(1)} = h_1(a_j^{(1)}) = h_1(\mathbf{w}_j^{(1)} \cdot \overline{\mathbf{x}})$$

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

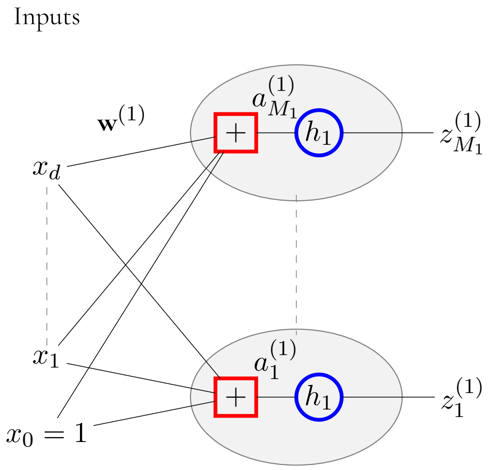
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

or a hyperbolic tangent

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

Observe that this corresponds to defining m_1 units, where unit j implements a GLM on $\mathbf{x}$ to derive $z_j^{(1)}$.

First layer



Multilayer network structure: inner layers

Vector $\mathbf{z}^{(1)}$ provides an input to the next layer, where m_2 **hidden units** compute a vector $\mathbf{z}^{(2)} = (z_1^{(2)}, \ldots, z_{m_2}^{(1)})^T$ by first performing linear combinations of the input values

$$a_k^{(2)} = \sum_{i=1}^{m_1} w_{ki}^{(2)} z_i^{(1)} + w_{k0}^{(2)} = \mathbf{w}_k^{(2)} \cdot \overline{\mathbf{z}}^{(1)}$$

and then applying function h_2, as follows

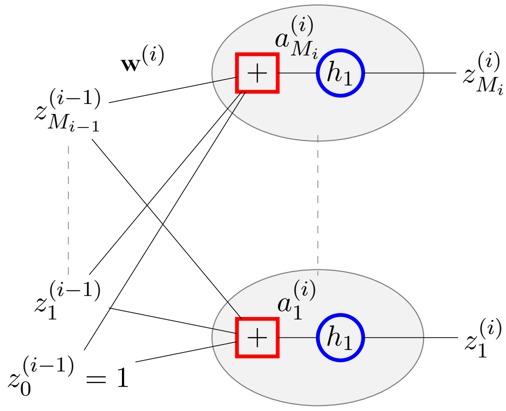
$$z_k^{(2)} = h_2(\mathbf{w}_k^{(2)} \cdot \overline{\mathbf{z}}^{(1)})$$

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

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

and non linear transformation

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



Multilayer network structure: output layer

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

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

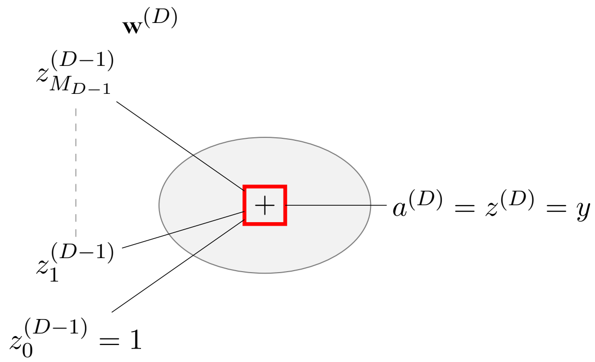
and then applying function h_t

$$y_k = z_k^{(t)} = h_t(\mathbf{w}_k^{(t)} \cdot \overline{\mathbf{z}}^{(t-1)})$$

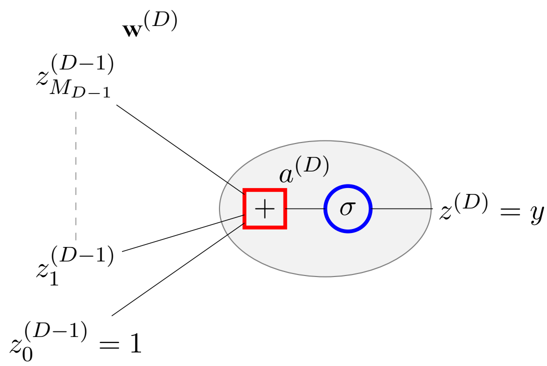
where:

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

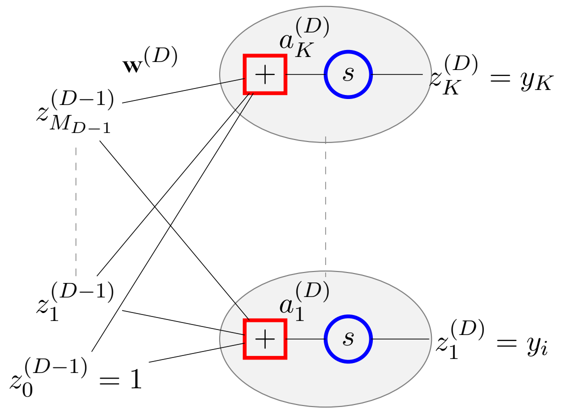
Output layer: regression



Output layer: binary classification



Output layer: K-class classification



3 layer networks

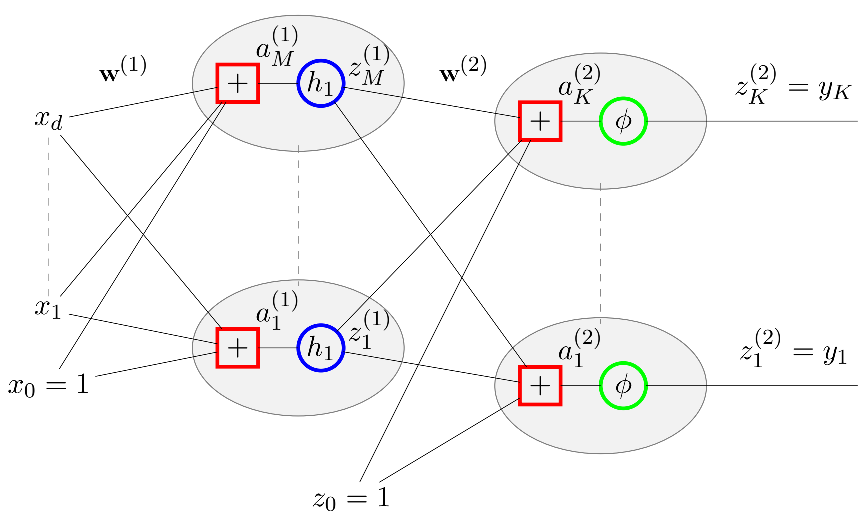
A sufficiently powerful model is provided in the case of 3 layers (input, hidden, output).

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

$$y_k = s\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{d} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

where the number M 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 $\mathbf{w}^{(1)}$.



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 $E(\mathbf{w})$

The error function $E(\mathbf{w})$ 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).

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

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

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

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

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \frac{\partial E_i(\mathbf{w})}{\partial \mathbf{w}}\Big|_{\mathbf{w}^{(k)}}$$

- 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

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \sum_{\mathbf{x}_i \in B} \frac{\partial E_i(\mathbf{w})}{\partial \mathbf{w}}\Big|_{\mathbf{w}^{(k)}}$$

Computing gradients

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

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

must be derived for all i, j, k in order to be iteratively evaluated for different values of $\mathbf{w}$ during gradient descent.

As we shall see, in order to evaluate

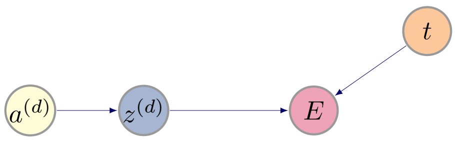
$$\frac{\partial E(\mathbf{w})}{\partial w_{ij}^{(k)}}$$

we may start by evaluating

$$\frac{\partial E(\mathbf{w})}{\partial a_i^{(d)}}$$

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

Regression



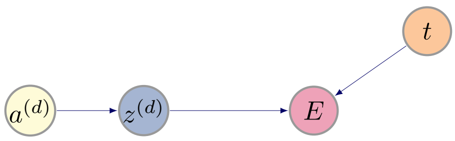
Here, we have $y = z^{(d)} = \sigma(a^{(d)})$ and

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

as a consequence,

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

Binary classification



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

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

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

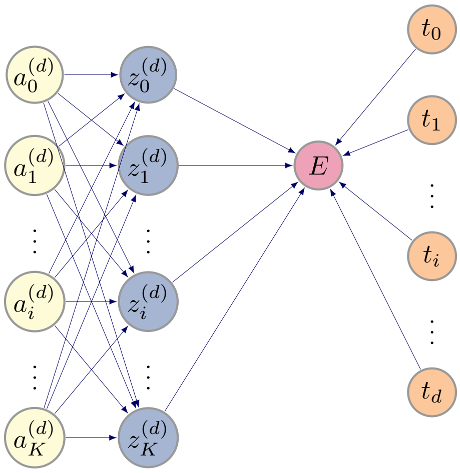
since, by the properties of the logistic function,

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

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

$$\begin{aligned}\frac{\partial E}{\partial a_i^{(d)}} &= -\sum_{j=1}^{K} \frac{\partial l}{\partial z_j^{(d)}} \frac{\partial z_j^{(d)}}{\partial a_i^{(d)}} = -\sum_{j=1}^{K} \frac{t_j}{z_j^{(d)}} \frac{\partial z_j^{(d)}}{\partial a_i^{(d)}} \\ &= -\frac{t_i}{z_i^{(d)}} z_i^{(d)}(1 - z_i^{(d)}) + \sum_{\substack{1 \le j \le K \\ j \neq i}} \frac{t_j}{z_j^{(d)}} z_i^{(d)} z_j^{(d)} = -t_i(1 - z_i^{(d)}) + \sum_{\substack{1 \le j \le K \\ j \neq i}} t_j z_i^{(d)} \\ &= z_i^{(d)} \sum_{j=1}^{K} t_j - t_i = z_i^{(d)} - t_i\end{aligned}$$

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 $w_{ij}^{(s)}$ and the values $a_i^{(s)}, z_i^{(s)}$ resulting by submitting the current item to the network, the knowledge of the derivatives

$$\frac{\partial E}{\partial a_j^{(r)}} \qquad 1 \leq j \leq n_r$$

makes it possible to compute the derivatives

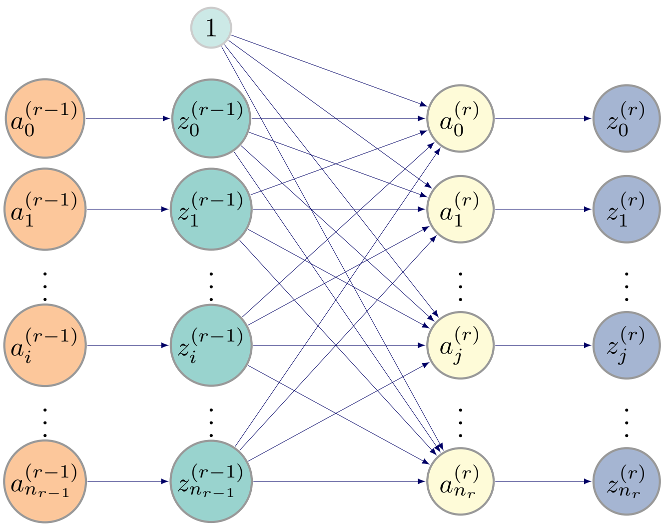
$$\frac{\partial E}{\partial w_{ij}^{(r)}} \qquad 0 \leq i \leq n_{r-1}, 1 \leq j \leq n_r$$

to be applied for gradient descent, and

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

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

Backpropagation at layer r



Here,

$$a_j^{(r)} = \sum_{i=1}^{n_{r-1}} w_{ij}^{(r)} z_i^{(r-1)} + w_{i0}^{(r)}$$

$$\frac{\partial a_j^{(r)}}{\partial w_{ij}^{(r)}} = z_i^{(r-1)} \qquad \frac{\partial a_j^{(r)}}{\partial w_{0j}^{(r)}} = 1$$

and, as a consequence,

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

$$\frac{\partial E}{\partial w_{0j}^{(r)}} = \frac{\partial E}{\partial a_j^{(r)}} \frac{\partial a_j^{(r)}}{\partial w_{0j}^{(r)}} = \frac{\partial E}{\partial a_j^{(r)}}$$

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

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

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

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

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 $r = d, \ldots, 2$

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

$$\frac{\partial E}{\partial w_{0j}^{(r)}} = \frac{\partial E}{\partial a_j^{(r)}}$$

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

For the first layer,

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

$$\frac{\partial E}{\partial w_{0j}^{(1)}} = \frac{\partial E}{\partial a_j^{(1)}}$$

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 $r = d, \dots, 2$

$$\frac{\partial E}{\partial w_{ij}^{(r)}} = \frac{\partial E}{\partial a_j^{(r)}} z_i^{(r-1)}$$
$$\frac{\partial E}{\partial w_{0j}^{(r)}} = \frac{\partial E}{\partial a_j^{(r)}}$$
$$\frac{\partial E}{\partial a_i^{(r-1)}} = h'(a_i^{(r-1)}) \sum_{j=1}^{n_r} \frac{\partial E}{\partial a_j^{(r)}} w_{ij}^{(r-1)}$$

For the first layer,

$$\frac{\partial E}{\partial w_{ij}^{(1)}} = \frac{\partial E}{\partial a_j^{(1)}} x_i$$
$$\frac{\partial E}{\partial w_{0j}^{(1)}} = \frac{\partial E}{\partial a_j^{(1)}}$$

Backpropagation and activation functions

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

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

while if a RELU activation function is applied, we get

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

Backpropagation

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

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

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 $\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$ at the current point $\mathbf{w}$.

Once $\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$ is known, a single step of gradient descent can be performed

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\Big|_{\mathbf{w}^{(i)}}$$

Computational efficiency of backpropagation

A single evaluation of error function derivatives requires $O(|\mathbf{w}|)$ 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 $O(|\mathbf{w}|)$ steps for each weight, hence $O(|\mathbf{w}|^2)$ steps overall.