Machine learning

Neural networks

Corso di Laurea Magistrale in Informatica

Università di Roma Tor Vergata

Prof. Giorgio Gambosi

a.a. 2021-2022

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 $y = f(\mathbf{w}^T\phi(\mathbf{x}))$: 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.

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 $x_1, \ldots, x_d$

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

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

Multilayer network structure: first layer

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

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

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 $z_j^{(1)}$.

Multilayer network structure: inner layers

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

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

and then applying function h_2, as follows

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

Multilayer network structure: inner layers

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

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

and non linear transformation

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

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 $\mathbf{z}^{(t-1)}$

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

and then applying function h_t

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

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

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

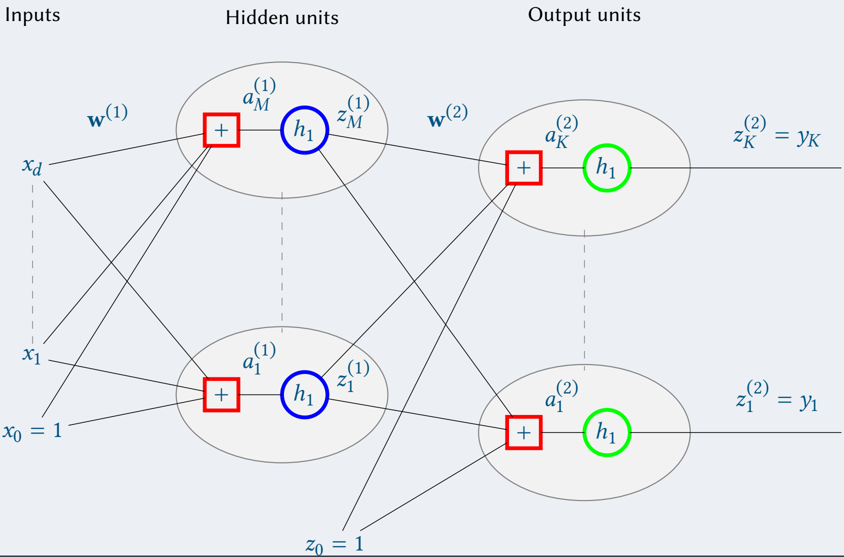
For example, applying this model for K-class classification corresponds to the following overall network function for each $y_k, k = 1, \dots, K$

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

where the number M of hidden units is a model structure parameter.

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 $\mathbf{w}^{(1)}$.

3 layer 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.

The training phase of a neural network implies learning the values of all parameters from a training set $(\mathbf{X}, \mathbf{t}) = \{(\mathbf{x}_1, \mathbf{t}_1), (\mathbf{x}_2, \mathbf{t}_2), \ldots, (\mathbf{x}_n, \mathbf{t}_n)\}$. In the case of 3-layered networks, this corresponds to learning $\mathbf{w} = \mathbf{w}^{(1)} \cup \mathbf{w}^{(2)}$.

As usual, learning can be performed by minimizing some loss function, in dependance of the problem considered and the assumed probabilistic model.

In the case of maximum likelihood, the minimization of the loss function is equivalent to the maximization of the likelihood of the training set, given the model and its parameters.

ML and regression in the case $K = 1$

Probabilistic model: for each element $(\mathbf{x}_i, t_i)$ of the training set, the value $y_i = y(\mathbf{x}_i, \mathbf{w})$ returned by the network is normally distributed around the target value t_i with variance σ^2 to be determined.

This is equivalent to assuming that, given an element $\mathbf{x}$, its unknown target value t is normally distributed around the returned value $y = y(\mathbf{x}, \mathbf{w})$ with same variance σ^2: that is,

$$p(t|\mathbf{x}, \mathbf{w}, \sigma^2) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \sigma^2)$$

The likelihood of the training set is

$$L(\mathbf{t}|\mathbf{X}, \mathbf{w}, \sigma^2) = \prod_{i=1}^{n} p(t_i|\mathbf{x}_i, \mathbf{w}, \sigma^2) = \prod_{i=1}^{n} N(t_i|y(\mathbf{x}_i, \mathbf{w}), \sigma^2)$$

and the log-likelihood

$$\begin{aligned} l(\mathbf{t}|\mathbf{X}, \mathbf{x}, \sigma^2) &= \log L(\mathbf{t}|\mathbf{x}, \mathbf{w}, \sigma^2) \\ &= -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y(\mathbf{x}_i, \mathbf{w}) - t_i)^2 \end{aligned}$$

As well known, maximizing the log-likelihood wrt $\mathbf{w}$ is equivalent to minimizing the loss function

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{n}(y(\mathbf{x}_i, \mathbf{w}) - t_i)^2$$

Let us now consider the derivative of the loss function with respect to $a^{(2)}$

$$\frac{\partial E(\mathbf{w})}{\partial a^{(2)}} = \frac{1}{2}\sum_{i=1}^{n}\frac{\partial}{\partial a^{(2)}}(y(\mathbf{x}_i, \mathbf{w}) - t_i)^2 = \sum_{i=1}^{n}(y(\mathbf{x}_i, \mathbf{w}) - t_i)\frac{\partial}{\partial a^{(2)}}(y(\mathbf{x}_i, \mathbf{w}) - t_i)$$

by construction, $y = a^{(2)}$, which implies that

$$\frac{\partial}{\partial a^{(2)}}(y - t) = \frac{\partial}{\partial a^{(2)}}(a^{(2)} - t) = 1$$

and

$$\frac{\partial E(\mathbf{w})}{\partial a^{(2)}} = \sum_{i=1}^{n}(y(\mathbf{x}_i, \mathbf{w}) - t_i)$$

that is, each item $\mathbf{x}, t$ considered contributes to the gradient with the error $y(\mathbf{x}, \mathbf{w}) - t$.

In the case of a neural network, differently than in the case of linear regression, $y(\mathbf{x}, \mathbf{w})$ is not linear and, in

As already stated, here we refer to a probabilistic model where the conditional probability of the target value, given the feature values, is distributed according to a Bernoulli

$$p(t|\mathbf{x}) = p(C_1|\mathbf{x})^t p(C_0|\mathbf{x})^{1-t}$$

assuming that value $y(\mathbf{x}, \mathbf{w})$ returned by the logistic model is an estimate of $p(C_1|\mathbf{x}; \mathbf{w})$, we get

$$p(t|\mathbf{x}, \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t (1 - y(\mathbf{x}, \mathbf{w}))^{1-t}$$

The likelihood of the traning set is then

$$L(\mathbf{t}|\mathbf{X}, \mathbf{w}) = \prod_{i=1}^{n} y(\mathbf{x}_i, \mathbf{w})^{t_i} (1 - y(\mathbf{x}_i, \mathbf{w}))^{1-t_i}$$

with log-likelihood

$$l(\mathbf{t}|\mathbf{X}, \mathbf{w}) = \sum_{i=1}^{n} (t_i \ln y(\mathbf{x}_i, \mathbf{w}) + (1 - t_i) \ln(1 - y(\mathbf{x}_i, \mathbf{w})))$$

The loss function is the cross entropy $E(\mathbf{w}) = -l(\mathbf{t}|\mathbf{X}, \mathbf{w}) = -\sum_{i=1}^{n}\left(t_i \ln y_i + (1-t_i)\ln(1-y_i)\right)$, where we denote $y(\mathbf{x}_i, \mathbf{w})$ as y_i

Its derivative wrt $a^{(2)}$ is then

$$\frac{\partial E(\mathbf{w})}{\partial a^{(2)}} = -\sum_{i=1}^{n}\left(t_i \frac{1}{y_i}\frac{\partial y_i}{\partial a^{(2)}} - (1-t_i)\frac{1}{1-y_i}\frac{\partial y_i}{\partial a^{(2)}}\right)$$

Since $y = \sigma(a^{(2)})$ by construction, we get

$$\frac{\partial y}{\partial a^{(2)}} = \frac{\partial \sigma(a^{(2)})}{\partial a^{(2)}} = \sigma(a^{(2)})(1-\sigma(a^{(2)})) = y(1-y)$$

As a consequence,

$$\begin{aligned}\frac{\partial E(\mathbf{W})}{\partial a^{(2)}} &= -\sum_{i=1}^{n}\left(t_i\frac{1}{y_i}y_i(1-y_i)-(1-t_i)\frac{1}{1-y_i}y_i(1-y_i)\right)\\ &= -\sum_{i=1}^{n}(t_i(1-y_i)-(1-t_i)y_i)\\ &= \sum_{i=1}^{n}(y_i-t_i)\end{aligned}$$

Again, as in the case of regression, each item $\mathbf{x}, t$ considered contributes to the derivative with the difference between the value $y(\mathbf{x}, \mathbf{w})$ computed by the network and the corresponding target t.

As already discussed, in the case of K-class classification, the likelihood is defined in terms of a categorical distribution, as

$$p(\mathbf{T}|\mathbf{X}) = \prod_{i=1}^{n}\prod_{k=1}^{K} p(C_k|\mathbf{x}_i)^{t_{ik}}$$

here, $\mathbf{T}$ is the $n \times K$ matrix where row i is the 1-to-K coding of t_i

Assuming that the value $y_k(\mathbf{x}, \mathbf{W})$ by the softmax model is an estimate of $p(C_k|\mathbf{x})$, we get

$$p(\mathbf{T}|\mathbf{X}, \mathbf{W}) = \prod_{i=1}^{n}\prod_{k=1}^{K} y_k(\mathbf{x}_i, \mathbf{W})^{t_{ik}}$$

where $\mathbf{W} = (\mathbf{w}_1, \dots, \mathbf{w}_K)$ is the $(d+1) \times K$ matrix of model coefficients such that $\mathbf{w}_j$ is the $d+1$-dimensional vector of coefficients for class C_j

The log-likelihood is then defined as

$$l(\mathbf{T}|\mathbf{X}, \mathbf{W}) = \sum_{i=1}^{n}\sum_{k=1}^{K} t_{ik} \log y_k(\mathbf{x}_i, \mathbf{W})$$

and the loss function to be minimized is, again, $E(\mathbf{W}) = -l(\mathbf{T}|\mathbf{X}, \mathbf{W})$.

By construction,

$$y_j = \frac{e^{a_j^{(2)}}}{\sum_{r=1}^{K} e^{a_r^{(2)}}}$$

The derivative of the loss function wrt $a_j^{(2)}$ can be derived as

$$\begin{aligned}
\frac{\partial E(\mathbf{W})}{\partial a_j^{(2)}} &= -\sum_{i=1}^{n}\sum_{k=1}^{K} t_{ik} \frac{\partial}{\partial a_j^{(2)}} \log y_k(\mathbf{x}_i, \mathbf{W}) \\
&= -\sum_{i=1}^{n}\sum_{k=1}^{K} t_{ik} \frac{1}{y_k(\mathbf{x}_i, \mathbf{W})} \frac{\partial}{\partial a_j^{(2)}} y_k(\mathbf{x}_i, \mathbf{W})
\end{aligned}$$

For what regards the derivative of $y_k(\mathbf{x}, \mathbf{W})$ wrt $a_j^{(2)}$, we have that

$$
\begin{aligned}
\frac{\partial}{\partial a_j^{(2)}} y_k(\mathbf{x}, \mathbf{W}) &= y_j(\mathbf{x}, \mathbf{W})(1 - y_j(\mathbf{x}, \mathbf{W})) && \text{if } k = j \\
\frac{\partial}{\partial a_j^{(2)}} y_k(\mathbf{x}, \mathbf{W}) &= -y_k(\mathbf{x}, \mathbf{W}) y_j(\mathbf{x}, \mathbf{W}) && \text{if } k \neq j
\end{aligned}
$$

and, as a consequence,

$$
\begin{aligned}
\frac{\partial E(\mathbf{W})}{\partial a_j^{(2)}} &= -\sum_{i=1}^{n} \left(t_{ij}(1 - y_j(\mathbf{x}_i, \mathbf{W})) - \sum_{k \neq j} t_{ik} y_j(\mathbf{x}_i, \mathbf{W}) \right) \\
&= -\sum_{i=1}^{n} \left(t_{ij} - \sum_{k=1}^{K} t_{ik} y_j(\mathbf{x}_i, \mathbf{W}) \right) \\
&= -\sum_{i=1}^{n} \left(t_{ij} - y_j(\mathbf{x}_i, \mathbf{W}) \sum_{k=1}^{K} t_{ik} \right) = \sum_{i=1}^{n} \left(y_j(\mathbf{x}_i, \mathbf{W}) - t_{ij} \right)
\end{aligned}
$$

Again, as before, each item $\mathbf{x}, t$ considered contributes to the derivative with the difference between the value

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

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

Backpropagation

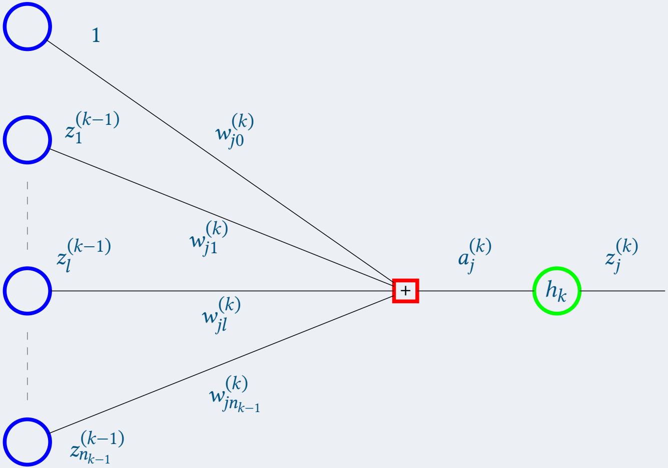
Algorithm applied to evaluate derivatives of the error wrt all weights

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.

Backpropagation

Assume a feed-forward neural network with arbitrary topology and differentiable activation functions and error function.



- All variables z_i could be either an input variable to the network or the output from a unit in the preceding layer
- The variable a_j could also be directly returned (h_k being the identity function)

Assumption on the error function: it may be expressed, given a training set, as the sum of the errors corresponding to single elements of the training set

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

If E_i is differentiable, so is E, with derivative given by the sum of the derivatives of functions E_i.

- Assume that, for each element $(\mathbf{x}_i, \mathbf{t}_i)$ of the training set, the feature values $\mathbf{x}_i$ have been given as input to the network and both the activation values for each unit and the output values are available: this step is denoted as forward propagation
- We wish to evaluate the derivative of E_i wrt to parameter $w_{jl}^{(k)}$, which associates a weight to the contribution of $z_l^{(k-1)}$ to the unit computing $a_j^{(k)}$
- E_i is a function of $w_{jl}^{(k)}$ only through the following sum

$$a_j^{(k)} = \sum_{r=1}^{m} w_{jr}^{(k)} z_r^{(k-1)}$$

Let us define $\delta_j^{(k)}$ as follows:

$$\delta_j^{(k)} = \frac{\partial E_i}{\partial a_j^{(k)}}$$

Since

$$\frac{\partial a_j^{(k)}}{\partial w_{jl}^{(k)}} = \frac{\partial}{\partial w_{jl}^{(k)}} \sum_{r=1}^{m} w_{jr}^{(k)} z_r^{(k-1)} = z_l^{(k-1)}$$

it results

$$\frac{\partial E_i}{\partial w_{jl}^{(k)}} = \delta_j^{(k)} z_l^{(k-1)}$$

To compute the derivatives of E_i wrt to all parameters, it is necessary to compute $\delta_j^{(k)}$ for all network units.

Let us first consider the output, that is $z_j^{(k)} = y_j$.

As observed before, in this case we have

$$\delta_j^{(k)} = \frac{\partial E_i}{\partial a_j^{(k)}} = y_j - t_j$$

Hidden unit.

- any change of $a_j^{(k)}$ has effect on E_i by inducing changes for all variables $a_l^{(k+1)}$
- the effect on E_i is a function of the sum of the effect of the change of $a_j^{(k)}$ on all variables $a_r^{(k+1)}$

$$\delta_j^{(k)} = \frac{\partial E_i}{\partial a_j^{(k)}} = \sum_{r=1}^{n_{k+1}} \frac{\partial E_i}{\partial a_r^{(k+1)}} \frac{\partial a_r^{(k+1)}}{\partial a_j^{(k)}} = \sum_{r=1}^{n_{k+1}} \delta_r^{(k+1)} \frac{\partial a_r^{(k+1)}}{\partial a_j^{(k)}}$$

Since by definition

$$a_r^{(k+1)} = \sum_l w_{rl}^{(k+1)} z_l^{(k)}$$

$$z_j^{(k)} = h_k(a_j^{(k)})$$

it results

$$\frac{\partial a_r^{(k+1)}}{\partial a_j^{(k)}} = \frac{\partial a_r^{(k+1)}}{\partial z_j^{(k)}} \frac{\partial z_j^{(k)}}{\partial a_j^{(k)}} = w_{rj}^{(k+1)} h_k'(a_j^{(k)})$$

and

$$\delta_j^{(k)} = h_k'(a_j^{(k)}) \sum_{r=1}^{n_{k+1}} \delta_r^{(k+1)} w_{rj}^{(k+1)}$$

Backpropagation

$$\delta_j^{(k)} = h_k'(a_j^{(k)}) \sum_{r=1}^{n_{k+1}} \delta_r^{(k+1)} w_{rj}^{(k+1)}$$

can the be evaluated if the following are known

- $w_{rj}^{(k+1)}, r = 1, \dots, k+1$: this are assumed as known for any single back propagation step
- $a_j^{(k)}$: this is computed, during forward propagation, from the current $\mathbf{w}$ and the input values
- $\delta_r^{(k+1)}, r = 1, \dots, k+1$: these can be computed from the network output and the target values by applying a backward propagation of the values from the last to the first network layers (that is, in opposite sense wrt to the output computation)

Backpropagation

Example of backpropagation on a 3-layered network:

1. The feature values $\mathbf{x}_i$ of a training set item are provided as input to the network: all values $a_j^{(1)}$, $a_j^{(2)}$, $z_j^{(1)}$, $z_j^{(2)} = y_j$ are derived and made available
2. Starting from output and target values, the δ values for each output variables is derived, as $\delta_j^{(2)} = y_j - t_j$

3. For each hidden unit, the corresponding δ value is computed, as

$$\delta_j^{(1)} = h_1'(a_j^{(1)}) \sum_{i=1}^{K} w_{ij}^{(2)} \delta_i^{(2)} = h_1'(a_j^{(1)}) \sum_{i=1}^{K} w_{ij}^{(2)} (y_j - t_j)$$

which, in the usual case $h_1(x) = \sigma(x)$, results into

$$\delta_j^{(1)} = \sigma(a_j^{(1)})(1 - \sigma(a_j^{(1)})) \sum_{i=1}^{K} w_{ij}^{(2)} (y_j - t_j) = z_j^{(1)}(1 - z_j^{(1)}) \sum_{i=1}^{K} w_{ij}^{(2)} (y_j - t_j)$$

4. For each parameter $w_{jl}^{(k)}$, where $k = 1, 2$, the value of the derivative of the function error wrt $w_{jl}^{(k)}$ at the current value $\mathbf{w}$ of all weights is computed as

$$\frac{\partial E_i}{\partial w_{jl}^{(k)}} = \delta_j^{(k)} z_l^{(k-1)}$$

which results into

$$\frac{\partial E_i}{\partial w_{jl}^{(2)}} = z_l(y_j - t_j)$$

$$\frac{\partial E_i}{\partial w_{jl}^{(1)}} = x_l z_j(1 - z_j) \sum_{i=1}^{K} w_{ij}^{(2)}(y_j - t_j)$$

Iterate the preceding steps on all items in the training set (or a subset of them). In fact, since

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

it is

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

This provides an evaluation of $\dfrac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$ at the current point $\mathbf{w}$.

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

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

The whole process can be made more efficient through on-line descent, that is by considering a single training set element at a time.

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.