# Neural networks

Course of Machine Learning
Master Degree in Computer Science

University of Rome “Tor Vergata”

a.a. 2020-2021

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.

# 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 $\overline{\mathbf{x}} = (1, x_1, \ldots, x_d)^T$.

# 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 $\mathbf{z}^{(1)} = (z_1^{(1)}, \ldots, z_{m_1}^{(1)})^T$ 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 $z_j^{(1)}$.

# 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

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

and then applying function $h_2$, as follows

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

# 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

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

and non linear transformation

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

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

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

and then applying function $h_t$

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

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

# 3 layer networks

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

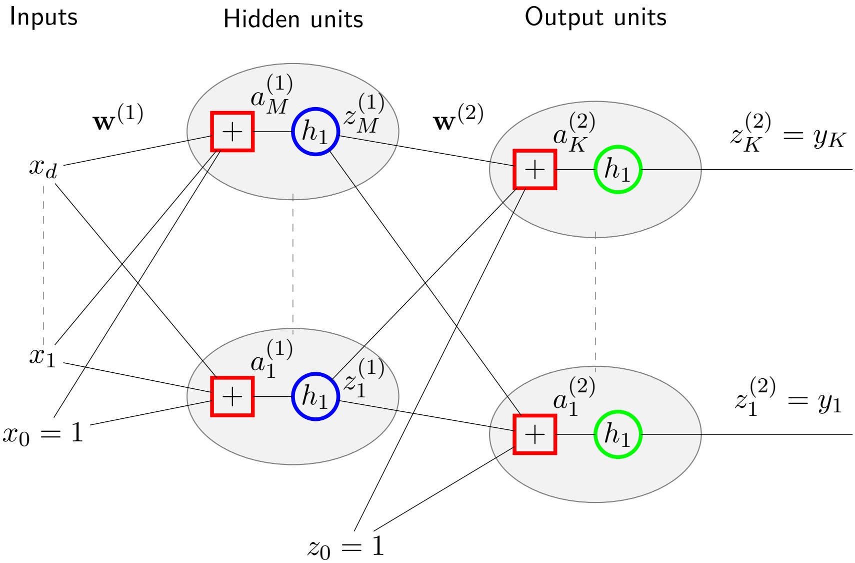
For example, applying this model for $K$-class classification corresponds to the following overall network function for each $y_k$, $k = 1, \ldots, K$

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

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



# 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.

# Maximum likelihood and neural networks

The training phase of a neural network implies learning the values of all parameters from a training set $(\mathbf{X}, \mathbf{t}) = \{(\mathbf{x}_1, \mathbf{t}_1), (\mathbf{x}_2, \mathbf{t}_2), \dots, (\mathbf{x}_n, \mathbf{t}_n)\}$. 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

The likelihood of the training set is

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

where $y_i = y(\mathbf{x}_i, \mathbf{w})$ and the log-likelihood

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

# ML and regression

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

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

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

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

this results in

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

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

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 general, has several local minima.

# ML and binary classification

The likelihood of the traning set is

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

with log-likelihood

$$l(\mathbf{t}|\mathbf{X},\mathbf{w}) = \sum_{i=1}^{n} \left(t_i \ln y_i + (1-t_i)\ln(1-y_i)\right)$$

where, again, we denote $y(\mathbf{x}_i,\mathbf{w})$ as $y_i$

# ML and binary classification

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

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

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

# ML and binary classification

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 $t$.

# ML and multiclass classification

The log-likelihood is then defined as

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

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

By construction, the function $y_k$ is defined as

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

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

$$\frac{\partial E(\mathbf{W})}{\partial a_j^{(2)}}=-\sum_{i=1}^{n}\sum_{k=1}^{K}t_{ik}\frac{1}{y_{ik}}\frac{\partial}{\partial a_j^{(2)}}y_{ik}$$

# ML and multiclass classification

As a consequence, it is easy to verify that

$$\frac{\partial E(\mathbf{W})}{\partial a_j^{(2)}} = -\sum_{i=1}^{n} \left( t_{ij} - y_{ij} \sum_{k=1}^{K} t_{ik} \right) = \sum_{i=1}^{n} (y_{ik} - t_{ij})$$

Again, as before, each item $\mathbf{x}, t$ considered contributes to the derivative with the difference between the value $y_j(\mathbf{x}, \mathbf{W})$ computed by the network at the $j$-th output node and the corresponding target $t_j$.

# 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

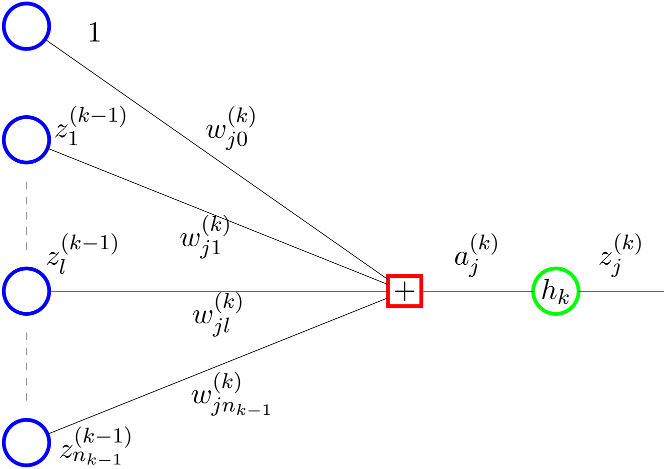
Algorithm applied to evaluate derivatives of the error wrt all weights

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.



# Backpropagation

- 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 $E_i$.

# Backpropagation

- 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

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

# Backpropagation

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

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

It is possible to show that

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

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

# Backpropagation

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

As observed before, in this case we have

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

hence,

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

The derivatives of the error $E_i$ wrt weights from the next-to-last to the output layer can be immediately derived by observing the values computed in the forward step

# Backpropagation

Hidden unit.

Here, it is possible to prove that

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

for example, if $h_k(x) = \sigma(x)$,

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

and

$$\frac{\partial E_i}{\partial w_{jl}^{(k)}} = z_l^{(k-1)} a_j^{(k)}(1 - a_j^{(k)}) \sum_{r=1}^{n_{k+1}} \delta_r^{(k+1)} w_{rj}^{(k+1)}$$

# 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

- $w_{rj}^{(k+1)}$, $r = 1, \ldots, k+1$: 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
- $\delta_r^{(k+1)}$, $r = 1, \ldots, k+1$: 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 $\delta$ values for each output variables is derived, as $\delta_j^{(2)} = y_j - t_j$

# Backpropagation

3. For each hidden unit, the corresponding $\delta$ 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

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

# Backpropagation

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

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

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

# Backpropagation

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 $\mathbf{w}$.

# Backpropagation

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

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

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