Aprenentatge Automàtic 1

GCED

Lluís A. Belanche

belanche@cs.upc.edu

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

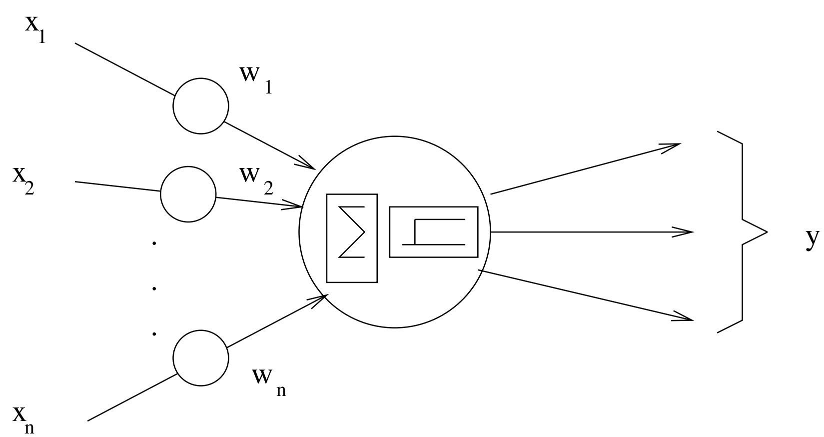
Universitat Politècnica de Catalunya

2019-2020

LECTURE 8a: Artificial neural networks (I)

Artificial neural networks (I): the MLP

An (artificial) **neuron** is an abstract computing unit that gets an **input** vector, combines this vector with a vector of local parameters (called **weights**) and –sometimes– with other local information (e.g., the neuron's previous state) and then **outputs** a scalar quantity:



The **output** can be delivered as part of the input of another neuron or to the neuron itself (self connection)

Artificial neural networks (I): the MLP

- A **directed graph** (DG) is a structure composed by a set of nodes and a set of labelled directed segments (vertexes) that connect the nodes.

- An **artificial neural network** (ANN) is a parallel and distributed information-processing structure that takes the form of a DG, where the nodes are neurons and the labels correspond to the weights

Graph	ANN
node	neuron/unit
vertex	connection
label	weight/parameter
layout	architecture/topology
with cycles	recurrent
w/o cycles	feed-forward

Artificial neural networks (I): the MLP

- A **layer** is a collection of neurons:

  1. sharing a common input vector (usually computing the same function) and

  2. not connected with one another

- The **output layer** is the last in the direction of the arrows. All other layers are called **hidden**. A **hidden neuron** is a neuron in a hidden layer

- An ANN is **recurrent** if its graph contains cycles; otherwise it is a **feed-forward** network. A recurrent network represents a dynamical system; a feed-forward network represents a function

Artificial neural networks (I): the MLP

The simplest choice of an ANN is a linear combination of the inputs:

$$y(\boldsymbol{x}) = \sum_{i=1}^{d} w_i x_i + w_0$$

What kind of network gives rise to this function?

Which can be extended to multiple outputs ...

$$y_k(\boldsymbol{x}) = \sum_{i=1}^{d} w_{ki} x_i + w_{k0}, \; k = 1, \ldots, m$$

And now?

Finally, let us add a non-linearity to the output:

$$y_k(\boldsymbol{x}) = g\left(\sum_{i=1}^{d} w_{ki} x_i + w_{k0}\right), \; k = 1, \ldots, m$$

And now?

Artificial neural networks (I): the MLP

1. Define $\boldsymbol{x} := (1, x_1, \ldots, x_d)^\top$ and $\boldsymbol{w}_k := (w_{k0}, w_{k1}, \ldots, w_{kd})^\top$

   We have

$$y_k(\boldsymbol{x}) = g\left(\sum_{i=0}^{d} w_{ki} x_i\right) = g(\boldsymbol{w}^\top \boldsymbol{x}),\ 1 \leq k \leq m$$

2. Let the weight matrix $W_{(d+1)\times m}$ gather all the weight vectors by columns

   Introduce the notation $g[\cdot]$ to mean that g is applied component-wise

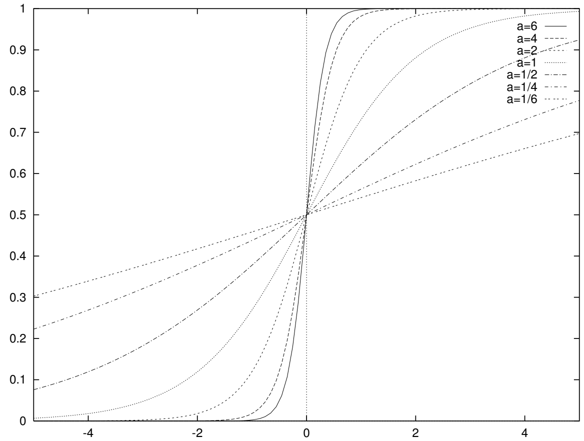
   The network then computes:

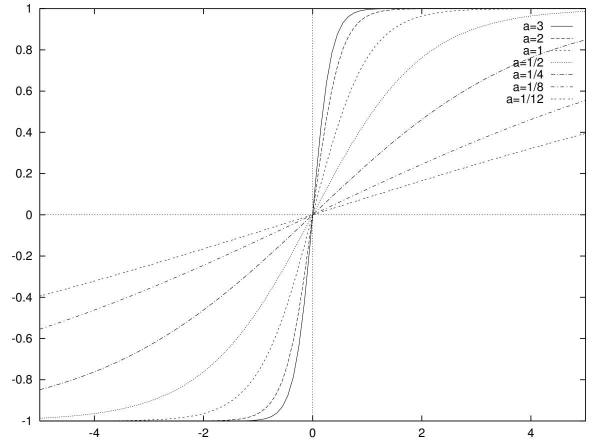
$$y(\boldsymbol{x}) = g[W^\top \boldsymbol{x}]$$

Artificial neural networks (I): the MLP

The **activation function** g is often a sigmoidal one:

- differentiable
- non-negative (or non-positive) bell-shaped first derivative
- horizontal asymptotes in $\pm\infty$





(left) : $g_\beta^{\text{log}}(z) = \dfrac{1}{1+e^{-\beta z}} \in (0,1),\ \beta > 0$ logistic

(right) : $g_\beta^{\text{tanh}}(z) = \dfrac{e^{\beta z} - e^{-\beta z}}{e^{\beta z} + e^{-\beta z}} \in (-1,1),\ \beta > 0$ tanh

(Note 'a' is β in the plots)

Artificial neural networks (I): the MLP

How could we obtain a model that is non-linear in the parameters (a **non-linear model**)? We depart from the basic linear model:

$$y_k(\boldsymbol{x}) = g\left(\sum_{i=1}^{d} w_{ki}x_i + w_{k0}\right), k = 1, \ldots, m$$

where g is a sigmoidal function. Suppose we apply non-linear functions to the input data:

$$y_k(\boldsymbol{x}) = g\left(\sum_{i=0}^{h} w_{ki}\phi_i(\boldsymbol{x})\right), k = 1, \ldots, m$$

We recover the previous "linear" situation making $h = d$ and $\phi_i(\boldsymbol{x}) = x_i$, with $\phi_0(\boldsymbol{x}) = 1$.

Artificial neural networks (I): the MLP

Approach 1. Make $\Phi = (\phi_0, \ldots, \phi_h)$ a set of predefined functions

Example: $d = 1$ and polynomial fitting. Consider the problem of fitting the function

$$p(x) = w_0 + w_1 x + \ldots + w_h x^h = \sum_{i=0}^{h} w_i x^i$$

to $x_1, \ldots, x_N \in \mathbb{R}$, which is a special case of linear regression, where the set of **regressors** is $1, x, x^2, \ldots, x^h$. Therefore $\phi_i(\boldsymbol{x}) = x^i$

The weights $w_0, w_1, \ldots, w_h$ can be estimated by standard techniques (ordinary least squares)

Artificial neural networks (I): the MLP

What if we have a multivariate input $\boldsymbol{x} = (x_1, \ldots, x_d)^\top$? The corresponding polynomial is:

$$p(x) = w_0 + \sum_{i_1=1}^{d} w_{i_1} x_{i_1} + \sum_{i_1=1}^{d} \sum_{i_2=i_1+1}^{d} w_{i_1 i_2} x_{i_1} x_{i_2} + \sum_{i_1=1}^{d} \sum_{i_2=i_1+1}^{d} \sum_{i_3=i_2+1}^{d} w_{i_1 i_2 i_3} x_{i_1} x_{i_2} x_{i_3} \ldots$$

The number of possible regressors grows as $\binom{d+h}{h}$!

So many regressors (while holding N fixed) causes increasing troubles for estimating their parameters:

- It is mandatory to have more observations N than regressors h
- Statistical significance of the weights decreases with h and increases with N

Artificial neural networks (I): the MLP

Approach 2. Why not trying to engineer **adaptive regressors**? By adapting the regressors to the problem, it is reasonable to expect that we shall need a much smaller number of them for a correct fit.

The basic neural network idea is to **duplicate the model**:

$$y_k(\boldsymbol{x}) = g\left(\sum_{i=0}^{h} w_{ki}\phi_i(\boldsymbol{x})\right), k = 1, \ldots, m$$

where

$$\phi_i(\boldsymbol{x}) = g\left(\sum_{j=0}^{d} v_{ij}x_j\right), \text{ with } \phi_0(\boldsymbol{x}) = 1, x_0 = 1$$

Artificial neural networks (I): the MLP

- We have a new set of regressors $\Phi(\boldsymbol{x}) = (\phi_0(\boldsymbol{x}), \ldots, \phi_h(\boldsymbol{x}))^\top$, which are adaptive via the $\boldsymbol{v}_i$ parameters (called the non-linear parameters).

- Once the new regressors are fully specified (i.e, the $\boldsymbol{v}_i$ parameters are estimated), the remaining task is linear (via the $\boldsymbol{w}_k$ parameters).

- What kind of network gives rise to this function if we keep duplicating? The **Multilayer Perceptron** or **MLP**.

- Under other choices for the regressors, other networks are obtained:

$$\phi_i(\boldsymbol{x}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2}{2\sigma_i^2}\right),\ \text{is the standard } \textbf{RBF network}$$

Artificial neural networks (I): the MLP

Error functions for classification

In **classification** we model the posteriors $P(\omega_k|\boldsymbol{x})$. In two-class problems, we model by creating an ANN with one output neuron ($m = 1$) to represent $y(\boldsymbol{x}) = P(\omega_1|\boldsymbol{x})$ and thus $1 - y(\boldsymbol{x}) = P(\omega_2|\boldsymbol{x})$.

Suppose we have a set of learning examples $S = \{(\boldsymbol{x}_n, t_n)\}_{n=1,\dots,N}$, where $\boldsymbol{x}_n \in \mathbb{R}^d, t_n \in \{0, 1\}$ (assume S is i.i.d.).

We take the convention that $t_n = 1$ means $\boldsymbol{x}_n \in \omega_1$ and $t_n = 0$ means $\boldsymbol{x}_n \in \omega_2$, to **model**:

$$P(t|\boldsymbol{x}) = \begin{cases} y(\boldsymbol{x}) & \text{if } \boldsymbol{x}_n \in \omega_1 \\ 1 - y(\boldsymbol{x}) & \text{if } \boldsymbol{x}_n \in \omega_2 \end{cases}$$

which is conveniently expressed as $P(t|\boldsymbol{x}) = y(\boldsymbol{x})^t(1 - y(\boldsymbol{x}))^{1-t}$, $t = 0, 1$.

Artificial neural networks (I): the MLP

Error functions for classification

This is a Bernoulli distribution. Assuming an i.i.d sample, the **likelihood function** is:

$$\mathcal{L} = \prod_{n=1}^{N} y(\boldsymbol{x}_n)^{t_n} (1 - y(\boldsymbol{x}_n))^{1-t_n}$$

So which error should we use? Let us define and minimize (again) the negative log-likelihood as the **error**:

$$E := -\ln \mathcal{L} = -\sum_{n=1}^{N} \{t_n \ln y(\boldsymbol{x}_n) + (1 - t_n) \ln(1 - y(\boldsymbol{x}_n))\}$$

popularly known as the "cross-entropy".

Error functions for classification

The case for more than two classes ($K > 2$) is obtained analogously, though with a bit more work.

The error function for the multiclass classification problem turns out to be:

$$E := -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{n,k} \ln y_k(\boldsymbol{x}_n)$$

known as the generalized cross-entropy.