Practical introduction to artificial neural networks

Laurent Bougrain

PRACTICAL INTRODUCTION TO ARTIFICIAL NEURAL NETWORKS

Laurent Bougrain *

* *Loria, Campus Scientifique - BP 239 - 54506 Vandœuvre-lès-nancy Cedex - France*

Abstract: What are they ? What for are they ? How to use them ? This article wants to answer these three fundamental questions about artificial neural networks that every engineer interested by this machine learning technique asks to oneself. We present the most useful architectures. We explain how to train them using a supervised or an unsupervised learning depending on the task we want to do : regression, discrimination or clustering. What kind of data can one use and how to prepare them ? Finally, we will be interested to which confidence can we give to the observed results ?

Keywords: Artificial neural networks, methodology, introduction.

1. INTRODUCTION

Artificial neural networks (ANN), also called connectionism, start to be a standard method to model a phenomenon from examples with a good performance. A lot of famous software such as Matlab™, Mathematica™or Weka have included them in their toolbox. Engineers or researchers can easily access to this technique. Nevertheless, several choices should be done before to get interesting results. The following sections present the most useful neural networks, their goal and the methodology to use them.

2. FROM A BIOLOGICAL NEURON TO ARTIFICAL NEURAL NETWORKS

In this section, we present the two principal components of a neural network : the architectures and the learning processes. The next section will explain how to choose the most well-matched couples according to the task.

2.1 Architecture

2.1.1. The neuron The basic computing unit of an artificial neural network is a formal neuron (see Figure 1). Directly inspired by the biological neuron, and reduced to the simplest expression by Mc Culloch and Pitts (McCulloch and Pitts 1943), it is a little system combining the information it receives into a value called potential and returns a single output value applying its activation function to its potential. Then, this output value can be sent to other neurons through connections.

A connection is an unidirected edge, also called synapsis in reference to the neurobiologic vocabulary. It connects a pre-synaptic neuron to a post-synaptic neuron. It is characterized by a value: its weight.

The potential $a(.)$ of a neuron j is a function of the output x_i of each neuron i at the origin of a connection linked to it and the weights w_{ji} of these connections. Usually, it is the weighted sum plus a bias w_{j0} (neuron with the output always equal to 1 used to learn the best threshold of the activation function).

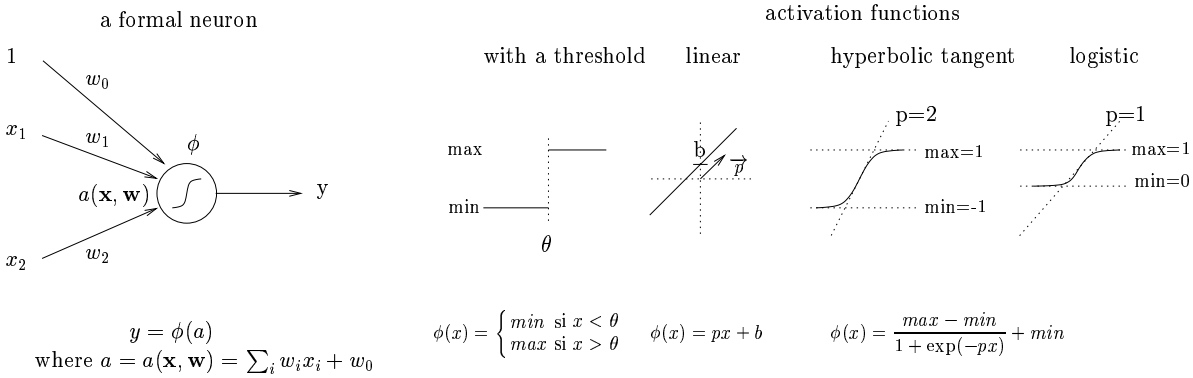

Fig. 1. A mathematical formalization of a neuron and some standard activation functions to assign to it.

$$a_j = a(\mathbf{x}, \mathbf{w}) = \sum_i w_{ji} x_i + w_{j0}$$

The activation function $\phi(.)$ takes as input the potential and returns the output value, or activation, of the neuron. The choice of the activation function depends on the flexibility we want to give it and of the bounds of the output space. Usually, the functions used are a threshold function, a linear function or some sigmoid functions (see figure 1).

2.1.2. Neural networks Within a neural network, three types of units can be distinguished : the input neurons which receive the input values, the output neurons which transmit the output values of the system and the hidden neurons which do not have a bond with the outside world. The neurons are gathered by type in a layer. Usually, the neurons of the same layer have the same activation function. The input layer should not be defined has a layer of the network because no computation is done. Nevertheless, it is very common to use this term and in very rare cases, it may carry out a transformation of the input values.

Feed-forward networks propagate the flow of information in a single direction since the input layer up to the output layer. There is no cycle. This kind of networks is the most usually observed in the literature. This category includes for example the perceptron (Rosenblatt 1961) and the multi-layer perceptron (Rumelhart *et al.* 1986) (Figure 2).

Recurrent networks belong to another category of ANNs. They contain some cycles to introduce a time relation or a space relation between the examples and the answers, or between the answers using a contextual information (the architecture of neural networks are comparable to an oriented graph). Thus, at least one connection links one neuron to another neuron of the same layer or of a previous layer (Figure 3). This category includes, for example, the Elman's network (Elman 1990), the Jordan's network (Jordan 1986) and the self-organising map by Kohonen (Kohonen 1989).

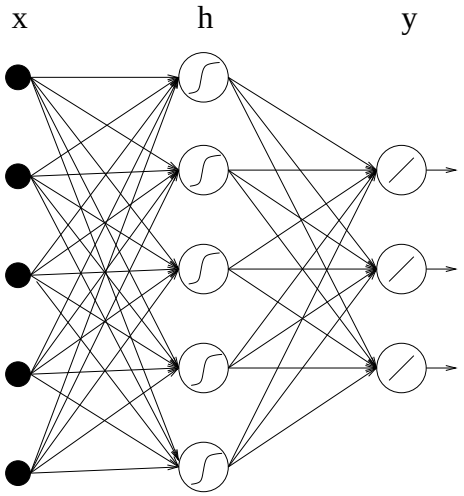

Fig. 2. A multi-layer perceptron with one hidden layer.

2.2 Learnings

A neural network must process the data presented as inputs in such a way that the produced outputs answer what we want. When we do not know *a priori* the characteristics to be given to the network (what is often the case), the management of the process is done using an iterative training, by modification of the weights and more rarely by modification of the architecture. The transformations brought depend on the learning rule.

2.2.1. Supervised learning If the values explicitly required as outputs are known, the learning is called supervised. A cost function measures the

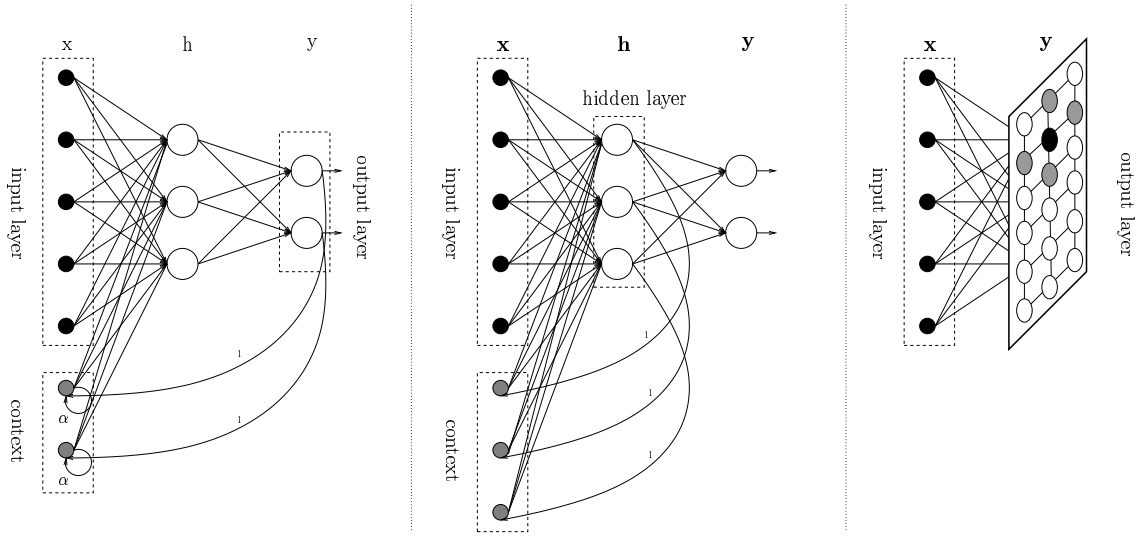

Fig. 3. Recurrents networks : a Jordan's model, a Elman's model and a self-organising map by Kohonen.

difference between the desired outputs and the produced outputs. Then, a learning rule updates the network to reduce this difference.

Whatever the learning paradigm (supervised or unsupervised learning presented below), it changes the value of the weights. The learning rule determines this modification with regard to the contribution of the weight on the quality of the final result.

The supervised learning th most used is the back-propagation algorithm gives a rule to update the weights of a multi-layer perceptron or a recurrent network for a supervised task. This learning rule follows the gradient descent optimisation $\Delta w = -\epsilon \frac{\partial E}{\partial w}$ where ϵ is the learning rate (usually $\epsilon << 1$). A learning with a small learning rate needs a long training stage to obtain a good model while a big learning rate does not allow to tune correctly the weight. A compromise can be to decrease the learning rate during the learning stage. The update of the weight of a connection linking a hidden neuron j to an output neuron k depends on :

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial w_{jk}}$$

where a_k, $\phi_k()$ and y_k are, respectively, the potential, the activation function and the output of neuron k. We have $y_k = \phi_k(a_k)$ and $a_k = \sum_j x_j w_{jk}$.

On one hand, if we choose, as error measure, the sum-of-square error function defined by $E = \frac{1}{2}\sum_k (d_k - y_k)^2$ where d_k is the desired value, or target value, for the output neuron k, and y_k is the output value of the output neuron k then :

$$\frac{\partial E}{\partial y_k} = \left[\frac{1}{2}\sum_k (d_k - y_k)^2\right]' = -(d_k - y_k)$$

On the other hand,

$$\frac{\partial y_k}{\partial a_k} = \frac{\partial \phi_k(a_k)}{\partial a_k} = \phi_k'(a_k)$$

So, if the activation function of the output neuron ϕ_k is the logistic sigmoid one $\frac{1}{1+e^{-x}}$ then $\phi_k'(x) = \phi_k(x)\,(1 - \phi_k(x))$.
From which

$$\frac{\partial y_k}{\partial a_k} = y_k(1 - y_k)$$

Finally,

$$\frac{\partial a_k}{\partial w_{jk}} = \frac{\partial \sum_j x_j w_{jk}}{\partial w_{jk}} = x_j$$

Then

$$\Delta w_{jk} = \epsilon (d_k - y_k) y_k (1 - y_k) x_j$$

The update of the weight of a connection linking an input neuron i and a hidden neuron j depends on :

$$\frac{\partial E}{\partial w_{ij}} = \sum_k \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial y_j} \frac{\partial y_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}}$$

where $y_k = \psi_j(a_k)$ et $a_k = \sum_j x_j w_{jk}$.
In the same way, the update value depends on the error function and the activation functions chosen. We have a recursive calculation of the modification of the weights.

This constitutes the back-propagation algorithm of the error applicable when the supervised network is made up of derivable functions of activation.

2.2.2. Unsupervised learning Here, the information concerning the desired output value is not available. No knowledge is given to the network *a priori*. It must discover regularities in the patterns

which are presented to it. The number of ouput neurons specifies the number of categories which one wants to see emerging. The system must develop its own representation of the shapes of the inputs by discovering the statistically redundant features (see section 3.3).

2.2.3. Incremental and batch learnings The frequency of the modifications can be moderate in two ways :

(1) on-line training (or incremental learning) : the update of the weights takes place after each example.
(2) off-line training (or batch learning) : the update of the weights takes place after all examples were presented i.e. after a cycle. The modifications to bring to the weights are cumulated and take effect at the end of the cycle. In this case, the learning rate should be divided by the number of training patterns.

3. WHAT FOR

3.1 Regression

A prediction task, within the framework of ANNs, wants to forecast one or several numerical variables from a certain number of numeric variables. When the variables to forecast are continuous, we speak about regression. Classically, the first model to use is a multi-layer perceptron. Indeed, this model is a nonlinear multivariate multiple regression model. The input layer will contain as many input neurons as input variables and as many output neurons as target variables. The input and output neurons have a linear activation function. One or eventually two hidden layers should be sufficient to obtain a model as good as by using more hidden layers. The number of hidden neurons is depending upon the number of input variables and the complexity of the task to model. It should not exceed 20 units unless to have more than 200 input variables, and a number between 5 and 15 is reasonable. To have many hidden neurons lets model a complex function but they need many examples to be learned correctly. If not, the model will not generalize well (we speak about overtraining, see section 4.5 and 4.7). A good choice is to assign an hyperbolic tangent function as activation function to the hidden neurons, rather than a linear or a logistic one, to prevent a too large flexibility requiring a long training, and to allows negative output values on the hidden layers. Finally, the sum-of-square error function which is equal to

$$E(\mathbf{t},\mathbf{y}) = \frac{1}{2}\sum_{k=1}^{m}(t_k - y_k)^2$$

is well adapted to a regression task. It "was obtained from the maximum likehood principle assuming the target data was generated from a smooth deterministic function with added Gaussian noise" (Bishop 1995).

3.2 Discrimination

A discrimination task is a specific prediction task where the variable to forecast is a nominal variable. In this case, we want to classify (i.e. to assign to pre-existent classes) patterns characterized by a certain number of numeric variables. If the classes are linearly separable one from another then the use of a perceptron, i.e. a feed-forward network without hidden layer, is enough. Let us recall that for a forecasting task, we apply a supervised learning such as the backpropagation algorithm. Then, the architecture of the network contains as many output neurons as values taken by the nominal variable, i.e. as classes. The class will be recode as a 1-of-c target scheme. For example, if you want to predict if a pattern belongs to class A, class B or class C, the output layer will have three neurons and the class of the examples will be recoded respectively as (1, 0, 0), (0, 1, 0) or (0, 0, 1). Using the sum-of-square error function and a linear activation function for the output neurons allows to approximate the *posterior* probabilities of class membership, conditioned on the input vector (see (Bishop 1995), chapter 6 for a complete explanation). This property can also be obtained using the cross-entropy error function and the softmax activation function which are, from a theorical point of view, more indicated for a discrimination problem (Bridle 1990).

The cross-entropy error function is as follows :

$$E(\mathbf{t},\mathbf{y}) = -\sum_{k=1}^{c} t_k \ln \frac{y_k}{t_k}$$

The softmax activation function is a generalization of the logistic sigmoid activation function :

$$y_k = \frac{\exp(a_k)}{\sum_{k'=1}^{c}\exp(a_{k'})}$$

The discrimination between two classes can also be treated using only one output neuron. The cross-entropy error function becomes

$$E(t,y) = -t\ln\frac{y}{t} - (1-t)\ln\frac{(1-y)}{(1-t)}$$

and the associated activation function is the logistic one. Compared to a 2-target code, the performance is the same but the 1-target representation minimizes the number of weights to learn.

3.3 Clustering

A clustering task is a technique of data analysis. The data are gathered by similarity in non pre-defined homogeneous classes. To do that, the architecture of the network contains as many output neurons as categories and the unsupervised learning uses a competitive algorithm to produce a regrouping inherent in the data. A competition system can, without supervised learning, lead to worked out treatments, such as a data partition, a vector quantization of the data space, a diminution of the number of patterns and an extraction of characteristics. Within the framework of artificial neural networks, the useful information for clustering the data is the weight vector associated to each class, termed prototype. The closest prototype using an Euclidian distance, or better a Mahalanobis' distance, identifies the class to which belongs the current data. The prototype converges towards the gravity center of the cluster. As a particular and useful illustration, the goal of self-organising maps is to obtain a clustering of the examples with a neighborhood constraint between the classes (which imposes that two elements of close classes are close in the data space but the reciprocal one is not always true. The topology is defined according to the problem. Usually, it projects into a 2 or 3-dimentional space and will have as an elementary form a grid with square, hexagonal or cubic cells. Various topologies define a specific number of neighbors and specific relations of distance. The distance between the classes on the grid is used to define the force with which the representatives of the classes are modified compared to the closest. In this manner, the topology of the classes reflects the topology of the inputs. The distance can be, for example, the Euclidean distance or the Manhattan distance. The use of the topographic representations is so widespread that it constitutes a relevant data processing obviously.

4. PRE-PROCESSING AND POST-PROCESSING

4.1 binary variables

Each binary variable should be recode into -1 and +1 in place of 0 and 1 because, when you apply the backpropagation algorithm, the update to assign to each weight of a connection located between the input layer and the first hidden layer is a product of the input value and another term. So, using zero to describe the value of a binary variable, you will never change the weight of the connections taking as input the value of a binary variable when this value equals zero. It is not necessary to apply this transformation to an output binary variable. Moreover, this can make a problem in some cases. Thus, if you use the entropy function, for example, with a supervised network to discriminate classes (see previous section), the cross-entropy is computed for 0 and 1. In the same idea, using a sigmoid function as the activation function of an output unit will always give as output value a value between 0 and 1 (if you want to obtain a value between -1 and 1, you can use for example the hyperbolic tangent function).

4.2 Nominal variables

A nominal variable should be transformed into as binary inputs as values. Firstly, because neural networks only take numerical value as input. Secondly, because to use a single continus variable where each modality has been assigned to a particular value does not respect the equidistance between the modalities. So, it is better to use a 1-to-n code where one binary variable will be equal to 1 and the (n - 1) others binary variables will be equal to 0 (or -1, see section 4.1). Thus, each modality is represented by a binary variable.

4.3 Continus variables

Each continus variable should be normalized. The potential of a neuron is the weighted sum of its inputs and its weights. If you use a single learning rate to update every weight, the variation of any variable will influence in the same way the potential. But each variable has a specific domain of definition. So a small variation for a variable with a smal domain of definition can reflect an important change where as the same variation for a variable with a large domain of definition can be unsignificant. To normalize the values, it is possible to apply a linear interpolation which project linearly the input value between -1 and 1, or a statistical normalisation removing their mean and dividing by their standard deviation.

4.4 Missing values

Patterns with missing values cannot be used by ANNs just as they are. The missing value should be forecasted starting from the other values of this pattern. Using a supervised learning, the missing value could be replaced by the average value of the variable while with an unsupervised learning, the missing value can be replaced sometimes by the weight associated to this variable to obtain no influence of this variable upon the competitive process.

4.5 Training set, validation set and test set

The training is stopped according to a stopping criterion. For the supervised methods, it can be given by the performance obtained on a validation dataset after training on the data of a training set :

(1) The training dataset contains most of the data. The model updates its parameters, i.e. its weights, according to the errors which it observes on the examples of the training dataset. So, the function to model is learned on the training dataset.
(2) The validation dataset constains data unused for training. The performance observed on this dataset indicates when to stop the training, i.e. the epoch when the performance on this dataset goes down meaning that the model has finished to learn the general characteristics of the task and starts to learn by heart the examples from the training dataset.
(3) The test dataset contains new data to estimate the performance of the model under real conditions of use. These data are never used to tune the model unlike the training and the validation sets.

4.6 Cross-validation

The initialization of the weights is random and the complete dataset is split, more or less arbitrarily, into two or three subsets to train, to validate and test the model. Unfortunatly, the results we observed depend on these choices. With an aim of having a better approximation of the real performance of the system, it is better to make several tries changing the initialization and more important, changing the distribution of the examples into the subsets. Several procedures exist such as the cross-validation, the bootstrap and the jackknife.

4.7 Overtraining

A key point with machine learning systems is their capability to obtain a general model from a set of examples. The problem is that if the system is very flexible, which means for a neural network to have a large architecture, you need a lot of examples to learn it. The use of a validation dataset solves this problem but the best choice is to use a pruning algorithm to obtain the minimum architecture (Reed 1993).

4.8 Unbalanced classes and confusion matrix

The fact of presenting during the training stage more examples of a class than of another one will influence the model obtained. In the same way, the fact of presenting during the test stage more examples of a class than of another one will influence the evaluation of its performance. Thus, it is necessary, in the case of a discrimination task, to avoid a too great unbalance between the number of examples of each class and to check the matrix of confusion to see the performance and the weakness of the model in a more detailed way than the rate of good discrimination.

5. CONCLUSIONS

Artificial neural networks can be very useful for engineers or researchers who want to obtain a model from a dataset of examples. This article tried to present a short introduction to this machine learning technique with a significant part reserved to the data pre-processing, the choice and use of the neural networks as well as the result analysis for a practical use.

REFERENCES

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Clarendon Press. Oxford.

Bridle, J. S. (1990). Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In: *Neuro-computing: Algorithms, Architectures and Applications* (F. Fogelman Soulié and J. Hérault, Eds.). Springer. Berlin. pp. 227–236.

Elman, J.L. (1990). Finding structure in time. *Cognitive Science* **14**, 179–211.

Jordan, M. I. (1986). Serial order: A parallel distributed processing approach. Technical Report ICS Report 8604. Institute for Cognitive Science, University of California at San Diego. La Jolla, CA.

Kohonen, T. (1989). *Self-Organization and Associative Memory*. 3 ed.. Springer-Verlag. Berlin.

McCulloch, W. S. and W. Pitts (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* **5**, 115–133.

Reed, Russel (1993). Pruning algorithms — A survey. *IEEE Transactions on Neural Networks* **4**(5), 740–746.

Rosenblatt, F. (1961). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books. Washington DC.

Rumelhart, D. E., G. E. Hinton and R. J. Williams (1986). Learning representations by back-propagating errors. *Nature* **323**, 533–536.