Artificial neural networks in high-energy physics

Liliana Teodorescu
Brunel University, United Kingdom

Abstract
Artificial neural networks are the machine learning technique best known in the high-energy physics community. Introduced in the field in 1988, followed by a decade of tests and applications received with reticence by the community, they became a common tool in high-energy physics data analysis. Important physics results have been extracted using this method in the last decade. This lecture makes an introduction of the topic discussing various types of artificial neural networks, some of them commonly used in high-energy physics, other not explored yet. Examples of applications in high-energy physics are also briefly discuss with the intention of illustrating types of problems which can be addressed by this technique rather than providing a review of such applications.

1 Introduction

Artificial Neural Networks are modest attempts of algorithmic modelling of biological neural systems. The human brain abilities to perform simultaneously complex tasks, to learn, memorise and generalise, inspired computer scientists to develop computer algorithms based on the same principles as those of the human brain functioning. Artificial Neural Networks are the results of such efforts. So far they are able to solve single objective problems which are of moderate complexity relative to the human brain capabilities. Reproducing the complex human brain abilities is still a desideratum.

A biological neural system is a network of neural cells called neurons. Signals propagate from one neuron to another when the cell "fires". Hence, a neuron can either excite or inhibit a signal.

Artificial Neural Networks (ANN) are layered networks of artificial neurons (AN) which are models of the biological neurons. Each AN receives signals from another AN or from the environment, collects them and forms an output signal which is transmitted to another AN or to the environment. An ANN consists of one input layer, one or more hidden layers and one output layer of ANs. Each AN in a layer is connected, fully or partially, to the ANs in the next layer. In some ANN configurations feedback connections to the previous layers are introduced.

2 Artificial neural network architecture

2.1 Artificial neuron

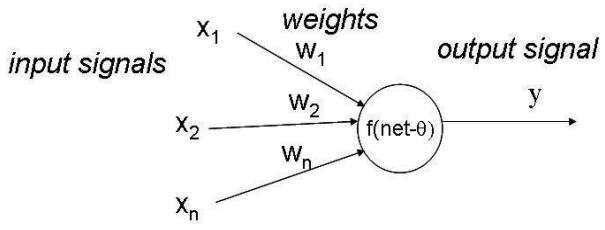

Fig. 1: An artificial neuron

A representation of an AN is depicted in Fig. 1. AN receives a set of input signals $(x_1, x_2, ..., x_n)$ from the environment or from another AN. A weight w_i $(i = 1, .., n)$ is associated to each input signal.

If the weight is positive then the signal is excited. Otherwise the signal is inhibited. AN collects all the input signals, calculates a net signal and transmits an output signal.

The net signal can be calculated as the the sum of the input signals, in which case the AN represents a summation unit:

$$net = \sum_{i=1}^{n} w_i x_i, \tag{1}$$

or as the product of the input signals in which case the AN represents a product unit:

$$net = \prod_{i=1}^{n} w_i x_i. \tag{2}$$

The product unit has a higher information capacity than the summation unit as it allows higher order combinations of inputs.

The output signal is calculated using a function called an activation function which depends on the value of the net signal net and a threshold value θ called bias.

For a more compact treatment, the bias can be considered as an additional input unit (signal) called a bias unit with the value $x_{i+1} = -1$ and the weight $w_{i+1} = \theta$. In this case the net signal is calculated as:

$$net = \sum_{i=1}^{n+1} w_i x_i, \tag{3}$$

or

$$net = \prod_{i=1}^{n+1} w_i x_i, \tag{4}$$

and the activation function becomes dependent only on this net signal.

Different types of activation functions are possible. Some common choices are:

– linear functions

$$f(net) = \beta \cdot net, \tag{5}$$

where β is a constant;

– step functions

$$f(net) = 1, net \geq 0 \tag{6}$$

$$f(net) = 0, net < 0; \tag{7}$$

– Sigmoid functions

$$f(net) = \frac{1}{1 + e^{-\lambda \cdot net}}, \tag{8}$$

where, usuallym $\lambda = 1$;

– Gaussian functions

$$f(net) = e^{-\frac{\bar{net}^2}{\sigma^2}}, \tag{9}$$

where $\bar{net}$ is the mean value of the net signal and σ^2 is the variance of the Gaussian distribution of the net signal.

2.2 Types of artificial neural networks

A single AN is quite limited in its functionality as it can produce linearly separable functions only. For more complex functions a network of interconnected ANs are necessary. One simple way to organise the ANs is in layers resulting a class of ANN called multi-layer ANNs.

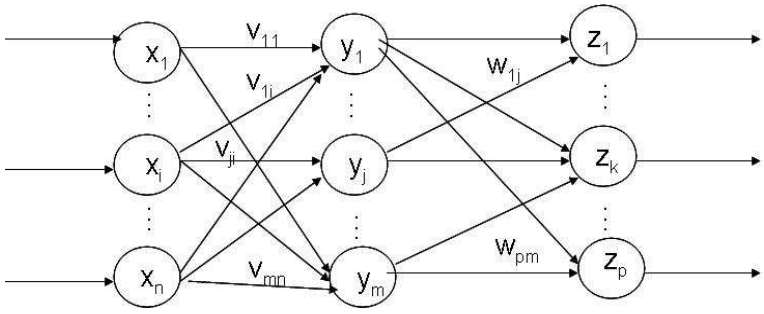

Fig. 2: A multi-layer ANN

An example of a muti-layer ANN is shown in Fig. 2. It contains a layer of n input neurons x_i ($i = 1, ..., n$), a hidden-layer of m neurons y_j with the activation function f_{y_j} ($j = 1, ..., m$) and an output layer of p neurons z_k with the activation function f_{z_k} ($k = 1, ..., p$). The activation functions could be different for different layers. Also the input layer could have an activation function. In the example shown in Fig. 2 no activation function is considered for the input layer.

The connections between the neurons are weighted with different values. v_{ji} are the weights between the input layer and the hidden layer, and w_{kj} are the weights between the hidden layer and the output layer. Using these weights, the network propagates the external signal through the layers producing the output signal which is of the form (for a network with summation units):

$$z_k = f_{z_k}(net_{z_k}) = f_{z_k}(\sum_{j=1}^{m+1} w_{kj} f_{y_j}(net_{y_j})) = f_{z_k}(\sum_{j=1}^{m+1} w_{kj} f_{y_j}(\sum_{i=1}^{n+1} v_{ji} x_i)). \tag{10}$$

This type of ANN, which simply propagates the input through all the layers, is called a feed-forward multi-layer ANN.

The example discussed here contains only one hidden layer. The network architecture can be extended to contain as many hidden layers as necessary. It was demonstrated, however, that a feed-forward multi-layer ANN with monolithically increasing differentiable functions can approximate any continuous function with only one hidden layer, provided this layer has enough nodes (neurons) [1]. In practice it was observed that some variations in ANN performance can be obtained by increasing the number of hidden layers. Hence, it is recommended to explore various network configurations for each application, starting with a one hidden layer architecture and gradually increasing the number of hidden layers in order to identify the optimal configuration.

The input layer of a feed-forward ANN can be extended into a layer of functional units, as a means of implementing an activation function for the input layer. Such a version of an ANN is called functional link ANN [2]. An example is shown in Fig. 3. This network is similar with the previously discussed ANN, except it has an additional layer, called the functional layer, which contains q functions $h_l(x_1, ..., x_n)$ ($l = 1, ..., q$). The weights between the input layer and the functional layer are $u_{li} = 1$ if h_l depends on x_i, and $u_{li} = 0$ otherwise. The output of this ANN is:

$$z_k = f_{z_k}(\sum_{j=1}^{m+1} w_{kj} f_{y_j}(\sum_{l=1}^{q+1} v_{jl} h_l(x_1, ..., x_n))). \tag{11}$$

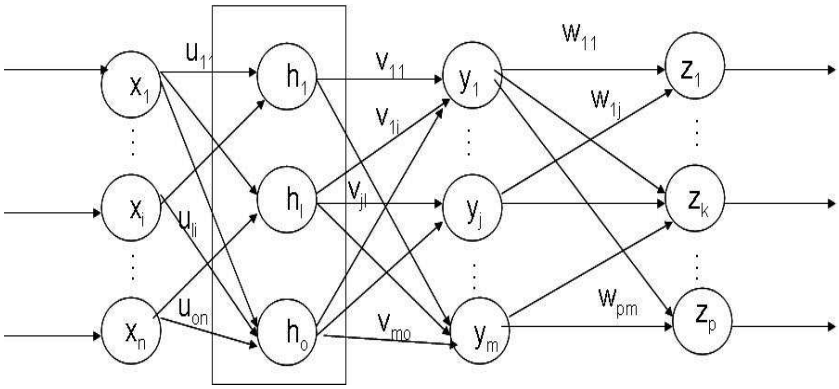

Fig. 3: A functional link ANN

Functional link ANNs were found to give better performance in terms of computational time and accuracy of the results than a simple feed-forward multi-layer ANN [2].

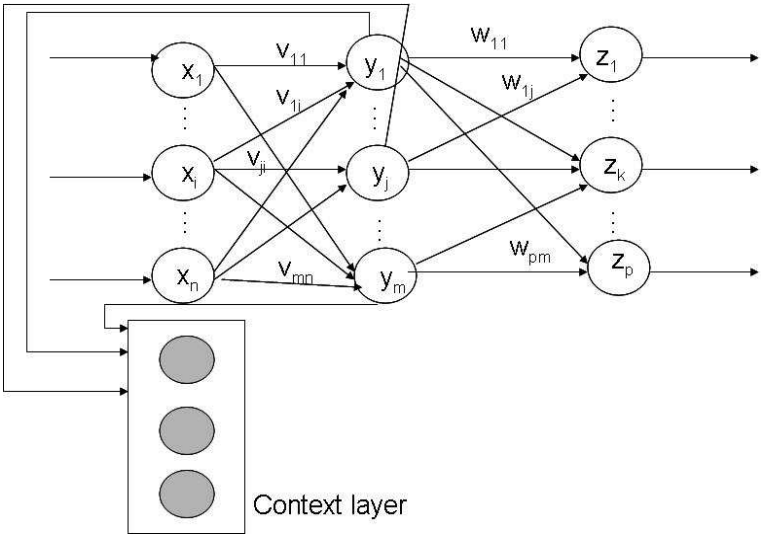

Fig. 4: A recurrent ANN

A different class of ANNs can be obtained by introducing a feedback connection in the propagation of the signal through the layers. Such networks are called recurrent ANNs [3]. An example of such a network is shown in Fig. 4. In this case the network contains an additional layer, called the context layer, which is, in fact, the copy of the hidden layer from the previous state of the network (previous iteration in the learning process). The input layer is now extended with the units of the context layer. This means the input signals will be made of the actual inputs $(x_1, ..., x_n)$ and the context units $(x_{n+2}, ..., x_{n+1+m}) = (y_1(t-1), ..., y_m(t-1)))$, where t represents the current iteration of the network. The weight of the hidden unit connected to the corresponding context unit is, in general, equal to one, but different values can be used. The output signal of this type of network is:

$$z_k = f_{z_k}(\sum_{j=1}^{m+1} w_{kj} f_{y_j}(\sum_{i=1}^{n+1+m} v_{ji} x_i)). \tag{12}$$

The recurrent ANNs have the advantage of being able to learn temporal characteristics of the data sets to which they are exposed.

The type of ANNs discussed here are only a few from the many proposed in the literature. Specialized books cover largely the topic, in different levels of details [4]. A specialized IEEE journal [5] publishes the last developments in the field.

3 ANN learning

The learning process for an ANN is the process through which the weights of the network are determined. This is achieved by adjusting the weights until certain criteria are satisfied.

There are there main types of learning:

- supervised learning - the ANN is presented with a training data set which contains the input vectors and a target associated with each input vector. The target is the desired output. The weights of the ANN are adjusted iteratively such that the difference between the actual output of the ANN and the target is minimized.
- unsupervised learning - the ANN is presented with a data set which contains only the input vectors. The weights of the ANN are adjusted such that the output provides a clustering of the input vectors based on certain criteria. This type of learning allows the discovery of patterns (clusters or regularities) in the data.
- reinforcement learning - the ANN is presented with a data set which contains only the input vectors. The weights of the ANN are adjusted such that those parts of the ANN which give a good performance (based on certain criteria) are rewarded while the other parts are penalised.

In this lecture only the supervised learning will be discussed as being the one used in the vast majority of the high-energy physics applications.

The most common supervised learning method is based on the gradient descent learning rule. The method optimises the network weights such that a certain objective function E is minimised by calculating the gradient of E in the weight space and moving the weight vector along the negative gradient.

For a single AN the objective function is usually an error function which measures the AN‘s error in approximating the target vector:

$$E = \sum_{p}^{P} (t_p - y_p)^2, \tag{13}$$

where t_p is the target value for the pattern p, y_p is the actual output for the pattern p and P is the total number of patterns in the data set. In this context a pattern represents a data instance or, in high energy physics terminology, an event, together with its associated target value.

The weights of the AN are adjusted by exposing it more times to the training data set. For a single training pattern the weights of an AN are adjusted with the formula:

$$w_i(t) = w_i(t-1) + \Delta w_i(t), \tag{14}$$

$$\Delta w_i(t) = \eta(-\frac{\delta E}{\delta w_i}), \tag{15}$$

$$\frac{\delta E}{\delta w_i} = -2(t_p - y_p)\frac{\delta f}{\delta net_p} x_{i,p}, \tag{16}$$

where t represents one iteration in the training process, f the activation function, $i = 1, ..., n$ with n being the number of elements of the input vector, and η is the learning rate which represents the size of the steps taken in the negative direction of the gradient.

In the case of an ANN the gradient descent weight optimisation contains, for each iteration (usually called epoch), two phases:

- feed-forward pass in which the output of the network is calculated with the current value of the weights,
- backward propagation in which the errors of the output signal are propagated back from the output layer towards the input layer and the weights are adjusted as a function of the back-propagated errors.

For the ANN with a hidden layer discussed previously the objective function of the learning process is:

$$E_p = \frac{1}{2}\frac{\sum_{k=1}^{K}(t_{k,p} - z_{k,p})^2}{K}, \tag{17}$$

where K is the number of neurons in the output layer and the p index refers to a training pattern.

The weights are updated with the formula:

$$w_{kj}(t) = w_{kj}(t) + \Delta w_{kj}(t) + \alpha \Delta w_{kj}(t-1), \tag{18}$$

$$v_{ji}(t) = v_{ji}(t) + \Delta v_{ji}(t) + \alpha \Delta v_{ji}(t-1), \tag{19}$$

$$\Delta w_{kj} = \eta(-\frac{\delta E}{\delta w_{kj}}) = -\eta \delta_{z_k} y_j, \tag{20}$$

$$\Delta v_{ji} = \eta(-\frac{\delta E}{\delta w_{ji}}) = -\eta \delta_{y_j} z_i, \tag{21}$$

where $\delta_{z_k} = \frac{\delta E}{\delta net_{z_k}}$ is the output error, $\delta_{y_j} = \frac{\delta E}{\delta net_{y_j}}$ is the hidden layer error, η the learning rate, and α is a parameter called momentum (it will be discussed further in this section).

To summarise, the supervised learning process implies the following steps:

1. initialisation of the weights, the η and α parameters, and the number of epochs $\xi = 0$;
2. initialisation of the error function $E_T = 0$;
3. for each training pattern
   a) calculate $y_{j,p}$ and $z_{k,p}$ (feed-forward phase);
   b) calculate the output error $\delta_{z_k,p}$ and the hidden layer error $\delta_{y_j,p}$;
   c) adjust the weights w_{kj} and v_{ji} (back-propagation phase);
   d) update of the error function $E_T = E_T + E_p$;
4. update the number of epochs $\xi = \xi + 1$;
5. test the stopping criteria; if this is not met then the process continues from the step 2.

As stopping criteria, common choices are a maximum number of epochs, a minimum value of the error function evaluated for the training data set, and the over-fitting point (which will be discussed in the next section).

The initialisation of the weights, the learning rate and the momentum are very important for the convergence and the efficiency of the learning process.

For weights, it is recommended to start with small random values around zero or in the interval $[-1/\sqrt{fanin}, 1/\sqrt{fanin}]$ where $fanin$ is the number of connections leading to a unit of the network [6]. This methods will avoid any bias toward a particular solution and the trap of the optimisation in a local minimum.

The learning rate η is a parameter which controls the size of each step toward the minimum of the objective function. If the value is too small the adjustment of the weights will also be small and a

high number of epochs is needed to reach a minimum. If, instead, the value is too high the convergence might be initially fast but the optimisation might oscillate around a minimum without reaching it, or might jump a good minimum and converge to a bad one. For these reasons it is recommended to start with small values around 0, such as 0.1, and to gradually increase it if the convergence is too low, or to decrease it if the error does not decrease fast enough.

Momentum η is a parameter which ensure the search path is in the average downhill direction of the error. When the weights of the network are updated after each data pattern, the weight changes fluctuate in the sign of the error derivative making the network to go back and forth and to unlearn what it learned in the previous steps. To avoid this problem, the weight changes are averaged by adding a term in the learning rule which is the previous change weighted with the momentum parameter. The usual value for this parameter is 0.9.

4 ANN performance measures

ANN performance can be analysed from different points of view. Common performance measures are the accuracy of predictions, the time complexity and the convergence of the learning process.

The accuracy is usually measured in terms of mean squared error:

$$E = \frac{\sum_{p=1}^{P} \sum_{k=1}^{K} (t_{k,p} - z_{k,p})^2}{PK}, \tag{22}$$

where P is the number of patterns in the data sample and K is the number of outputs of the network.

When this error is calculated for the training data it is called the training error. When it is calculated for another independent data set, not used in the training, the error is called the generalisation error.

The generalisation error characterises how well the network is able to model new data, not used in the learning process. Hence the objective of the learning process is to obtain a network with a low generalisation error.

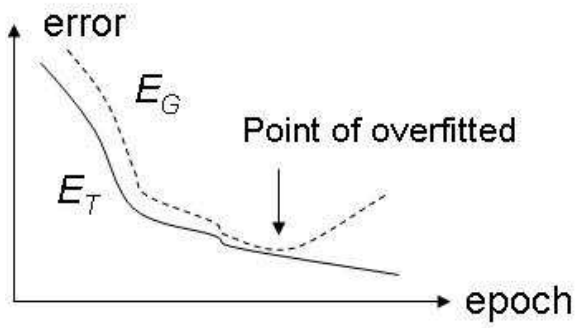

Fig. 5: Representation of the over-fitting effect. E_T represents the training error and E_G the generalisation error.

The generalisation error does not necessarily follow the behavior of the training error. With the increase of the number or epochs, the training error tends to decrease . The generalisation error also decrease up to a certain point, called the over-training or over-fitting point, after which the generalisation error tends to increase while the training error continues to decrease (see Fig. 5).

The over-fitting happens when the ANN tends to memorise the training patterns. In this case the network has a low capability to generalise to new data sets. The effect occurs particularly when ANN is very large, meaning that it has too many hidden nodes and hence, too many weights to be optimised. To avoid the over-fitting, the network architecture needs to be optimised for the problem at hand and to be exposed to a sufficiently large training data set.

The over-fitting point is determined by estimating the generalisation error on an independent data set called the validation data set. Quantitatively, the over-fitting point is the point for which

$$E_V > \bar{E_V} + \sigma_{E_V}, \tag{23}$$

where E_V is the validation error (the generalisation error calculated for the validation data set), $\bar{E_V}$ is the validation error averaged over the previous epochs and σ_{E_V} is the standard deviation of the validation error values.

Alternative or complementary accuracy measures to the generalisation errors can be used for evaluating the ANN performance.

For specialised problems such as classification a common choice is the classification accuracy defined as the percentage of the data patterns correctly classified by the network.

The correlation between the network outputs and the target values is also of interest. It is measured by the correlation coefficient defined as:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})\sum_{i=1}^{n}(y_i - \bar{y})}{\sigma_x \sigma_y}, \tag{24}$$

where $x = z_{k,p}$ and $y = t_{k,p}$.

The ability of an ANN to converge to a certain error level is another performance measure. This ability, called convergence, is quantitatively (and empirically) expressed as the number of times, out of a fixed number of simulations, the ANN succeeds to reach the specific error level. Extensive theoretical studies exists on this topic which will not be discussed here.

For practical problems the computational complexity, meaning the computational time needed to train the network, is of significant importance. Various measures can be used to evaluate it: the number of epochs to reach a certain error, the number of patterns presentations, the number of weight updates or the total number of calculations made during training. Among the factors which influence this complexity, the network architecture, the size of the training data set and the complexity of the learning rules used were found to have considerable impact.

5 Applications in high-energy physics

Artificial Neural Networks had a late and timid start in high-energy physics. They were introduced in this field in 1988 [7], [8], almost 40 years after their invention. For other ten years their exploitation remained sporadic and received with scepticism by a significant part of the high-energy physics community. Only since late nineties ANNs started to be used more broadly, probably due to an increased complexity of the data and physics process investigated which imposed more demands on the analysis techniques. A good review of these applications until 1999 was published in [9]. During the last couple of years ANNs have become quite routinely used in experiments such as BaBar [10], CDF [11] and D0 [12].

In terms of types of ANNs, the vast majority of applications in high-energy physics are based on feed-forward multi-layer ANN with back-propagation. The first application, however, in 1988, was of a recurrent ANN for tracking reconstruction [9], [8]. A recurrent ANN was also used for tracking reconstruction in DELPHI experiment [14]. Other types of ANNs remained almost unexplored.

In terms of types of applications, ANN were used for both online triggers and offline data analysis.

The application of ANNs for online triggering of particle physics events can be easily understood if the high similarity between a basic coincidence circuit (which is the basic unit of a triggering system) and an Artificial Neuron is noticed. Fig. 6 shows a coincidence circuit which is to be compared with Fig. 1 of an AN. The potential V_i can be considered analogous to the input signal x_i, the resistors R_i analogous to the weights, the threshold of the circuit analogous to the AN bias and the transfer function σ_l analogous to the activation function $f(net)$. Both the circuit unit and the AN can be "fired" or not.

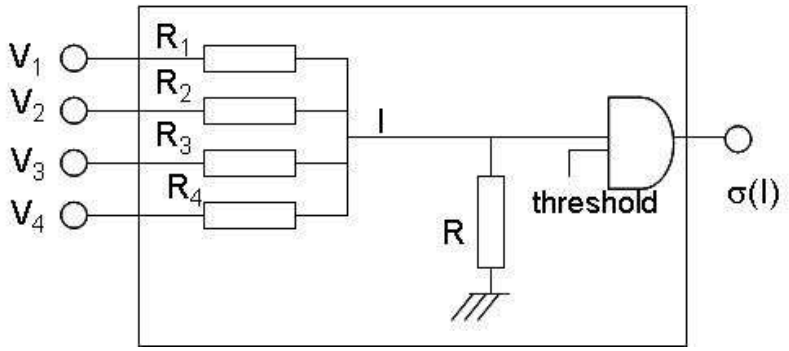

Fig. 6: Basic coincidence electric circuit

Two running experiments which used neural network triggers are DIRAC [13] and H1 [15]. Previous applications were reviewed in [9]. The H1 neural network trigger, for example, was implemented in the level 2 trigger of the experiment. It is based on feed-forward ANNs with 3 layers and used to separate physics from background events. The trigger contains an ANN for each physics process (twelve in this case). The input variables were energy sums in subsets of the calorimeter, information on vertex position, and charge tracks multiplicities. The output was one or zero, for physics or background events, respectively. A full description of the system can be found on the project webpage [16] and in the corresponding publications listed there.

For offline data analysis ANNs were used or tested for a variety of tasks such as track and vertex reconstruction, particle identification and discrimination, calorimeter energy estimation and jet tagging.

In terms of physics processes studied, the first application from which a physics results was extracted with an ANN was for the decay of the Z boson [17]. A feed-forward network was used to discriminate the decay of the Z boson into c, b or s quarks and the results used further to determine the decay probability of Z into the corresponding states.

Other important physics results were obtained with feed-forward neural networks at Tevatron experiments such as the direct measurement of the top quark mass [18] or laptoquark searches [19]. ANNs continue to be important techniques in the search for the Higgs boson an the Tevatron experiments.

Also the BaBar experiment extracted numerous physics results concerning the decay of B mesons using feed-forward neural networks.

The LHC experiments tested ANN for various applications, many of the studies being published in the proceedings of the ACAT conference [20] over the last couple of years. It remains to be seen if the method will be, indeed, used in the running experiments.

In these applications ANNs gave better results than the standard methods mainly due to the highly non-linearity character of the method. Its drawback is the lack of transparency of the results which is the main reason a significant number of particle physicists were reticent about the method. Today, however, this drawback seems to be overcome.

Another important issue with the method is that it is based on supervised learning which requires a data set, in many situations of considerable size, for which the target output is known. In the vast majority of cases such data is obtained by Monte Carlo simulations. In this way the result is very much dependent of the quality of the simulation. While for low level pattern recognition applications a highly reliable simulation can be easier achieved, for high level physics analysis the simulation incorporates theoretical model which might bias the results. When the physics process is poorly known, the method can become unusable.

6 Conclusions

Artificial neural networks were proven as valuable methods in high-energy physics for online and offline data analysis tasks. They have a number of characteristics which make them appreciated. They provide highly non-linear outputs, taking into account the correlation between the inputs, and hence their potential in event selection goes beyond the conventional cut-based approach. They are also tolerant to noise, are fast due to the intrinsic parallelism and hence easily implementable in electronic circuits for online event selection, and have a good programmability which makes them quickly adaptable to new running conditions.

There are, however, a number of disadvantages associated with this technique. The interpretation of the network output is not obvious and this lack of transparency makes physicists to be reticent. Extensive tests are needed in order to prove the correctness of the results. Another big disadvantage is the need to rely on Monte Carlo simulations of the physics process studied and of the response of the experimental apparatus for the network learning, as the supervised learning is used in the vast majority of cases. This restricts the applicability of the technique to situations when both the physics process and the experimental apparatus are very well understood.

While continuing to use the Artificial Neural Networks in the same manner as so far will very probably provide important results for high-energy physics, new directions should be explored in order to adapt them to new demands such as those impose by the LHC experiments, for example. Two such directions could be the investigation of more complex types of neural networks than the simple feed-forward ones, and the investigation of the unsupervised neural networks which might be useful in searching for new physics processes and particles.

References

[1] K. Hornik, *Neural Networks* **2** (1989) 359.

[2] J. Ghosh, Y Shin, *International Journal of Neural Systems* **3** (1992) 323.

[3] D. Mandic, J. Chambers, Recurrent Neural Networks, Wiley, 2001.

[4] P. Picton, Neural Networks, Palgrave, 2000; C. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995; X.-S. Zhang, Neural Networks in Optimisation, Kluwer Academic Publisher, 2000.

[5] IEEE Transactions on Neural Networks.

[6] L. Wessels, E. Barnard, *IEEE Transactions on Neural Networks* **3** (1992) 899.

[7] B. Denby, *Computer Physics Communications* **49** (1988) 429.

[8] C. Peterson, *Nuclear Instruments and Methods* **A279** (1988) 537.

[9] B. Denby, *Computer Physics Communications* **119** (1999) 219.

[10] BaBar experiment,`http://www.slac.stanford.edu/BFROOT/` .

[11] CDF experiment, `http://www-cdf.fnal.gov` .

[12] DZero experiment,`http://www-d0.fnal.gov` .

[13] DIRAC experiment, `http://www.cern.ch/DIRAC` .

[14] R. Fruhwirth, *Computer Physics Communications* **78** (1993) 23.

[15] H1 experiment,`http://www-h1.desy.de` .

[16] `http://wwwh1.mppmu.mpg.de/projects/neuro/neuro.html` .

[17] P. Abreu *et. al., Phys. Lett. B* **295** (1992) 382.

[18] S. Abachi *et. al., Phys. Rev, Lett.* **79** (1997) 1197.

[19] B. Abbott *et. al., Phys. Rev. Lett.* **79** (1997) 4321.

[20] International Workshop oh Advanced Computing and Analysis Techniques in Physics Research (ACAT).