Chapter 9

Applying Artificial Neural Network Hadron - Hadron Collisions at LHC

Amr Radi and Samy K. Hindawi

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/51273

1. Introduction

High Energy Physics (HEP) targeting on particle physics, searches for the fundamental particles and forces which construct the world surrounding us and understands how our universe works at its most fundamental level. Elementary particles of the Standard Model are gauge Bosons (force carriers) and Fermions which are classified into two groups: Leptons (i.e. Muons, Electrons, etc) and Quarks (Protons, Neutrons, etc).

The study of the interactions between those elementary particles requests enormously high energy collisions as in LHC [1-8], up to the highest energy hadrons collider in the world $\sqrt{s}$ =14 Tev. Experimental results provide excellent opportunities to discover the missing particles of the Standard Model. As well as, LHC possibly will yield the way in the direction of our awareness of particle physics beyond the Standard Model.

The proton-proton (p-p) interaction is one of the fundamental interactions in high-energy physics. In order to fully exploit the enormous physics potential, it is important to have a complete understanding of the reaction mechanism. The particle multiplicity distributions, as one of the first measurements made at LHC, used to test various particle production models. It is based on different physics mechanisms and also provide constrains on model features. Some of these models are based on string fragmentation mechanism [9-11] and some are based on Pomeron exchange [12].

Recently, different modeling methods, based on soft computing systems, include the application of Artificial Intelligence (AI) Techniques. Those Evolution Algorithms have a physical powerful existence in that field [13-17]. The behavior of the p-p interactions is complicated due to the nonlinear relationship between the interaction parameters and the output. To understand the interactions of fundamental particles, multipart data analysis are needed and AI techniques are vital. Those techniques are becoming useful as alternate approaches to conventional ones [18]. In this sense, AI techniques, such as Artificial Neural Network (ANN) [19], Genetic Algorithm (GA) [20], Genetic Programming (GP) [21 and Gene Expression Programming (GEP) [22], can be used as alternative tools for the simulation of these interactions [13-17, 21-23].

The motivation of using a NN approach is its learning algorithm that learns the relationships between variables in sets of data and then builds models to explain these relationships (mathematically dependant).

In this chapter, we have discovered the functions that describe the multiplicity distribution of the charged shower particles of p-p interactions at different values of high energies using the GA-ANN technique. This chapter is organized on five sections. Section 2, gives a review to the basics of the NN & GA technique. Section 3 explains how NN & GA is used to model the p-p interaction. Finally, the results and conclusions are provided in sections 4 and 5 respectively.

2. An overview of Artificial Neural Networks (ANN)

An ANN is a network of artificial neurons which can store, gain and utilize knowledge. Some researchers in ANNs decided that the name ``neuron'' was inappropriate and used other terms, such as ``node''. However, the use of the term neuron is now so deeply established that its continued general use seems assured. A way to encompass the NNs studied in the literature is to regard them as dynamical systems controlled by synaptic matrixes (i.e. Parallel Distributed Processes (PDPs)) [24].

In the following sub-sections we introduce some of the concepts and the basic components of NNs:

2.1. Neuron-like Processing Units

A processing neuron based on neural functionality which equals to the summation of the products of the input patterns element $\{x_1, x_2,..., x_p\}$ and its corresponding weights $\{w_1, w_2,..., w_p\}$ plus the bias θ. Some important concepts associated with this simplified neuron are defined below.

A single-layer network is an area of neurons while a multilayer network consists of more than one area of neurons.

Let u_i^{ℓ} be the ith neuron in ℓth layer. The input layer is called the xth layer and the output layer is called the Oth layer. Let $n\ell$ be the number of neurons in the ℓth layer. The weight of the link between neuron u_j^{ℓ} in layer ℓ and neuron $u_i^{\ell+1}$ in layer $\ell+1$ is denoted by w_{ij}^{ℓ}. Let $\{x_1, x_2,..., x_p\}$ be the set of input patterns that the network is supposed to learn its classification and let $\{d_1, d_2,..., d_p\}$be the corresponding desired output patterns. It should be noted that x_p is an n dimension vector $\{x_{1p}, x_{2p},..., x_{np}\}$ and d_p is an n dimension vector $\{d_{1p},d_{2p},...,d_{np}\}$. The pair (x_p, d_p) is called a training pattern.

The output of a neuron u_i^{0} is the input x_{ip} (for input pattern p). For the other layers, the network input $net_{pi}^{\ell+1}$ to a neuron $u_i^{\ell+1}$ for the input $x_{pi}^{\ell+1}$ is usually computed as follows:

$$net_{pi}^{\iota+1} = \sum_{j=1}^{n_\iota} w_{ij}^{\iota} o_{pj}^{\iota} - \theta_i^{\iota+1} \tag{1}$$

where $O_{pj}^{\ell} = x_{pi}^{\ell+1}$ is the output of the neuron u_j^{ℓ} of layer ℓ and $\theta_i^{\ell+1}$ is the neuron's bias value of neuron $u_i^{\ell+1}$ of layer ℓ+1. For the sake of a homogeneous representation, θ_i is often substituted by a ``bias neuron'' with a constant output 1. This means that biases can be treated like weights, which is done throughout the remainder of the text.

2.2. Activation Functions

The activation function converts the neuron input to its activation (i.e. a new state of activation) by f (net_p). This allows the variation of input conditions to affect the output, usually included as O_p.

The sigmoid function, as a non-linear function, is also often used as an activation function. The logistic function is an example of a sigmoid function of the following form:

$$o_{pj}^{\iota} = f(net_{pi}^{\iota}) = \frac{1}{1+e^{-\beta net_{pi}^{\iota}}} \tag{2}$$

where β determines the steepness of the activation function. In the rest of this chapter we assume that β=1.

2.3. Network Architectures

Network architectures have different types (single-layer feedforward, multi-layer feedforward, and recurrent networks) [25]. In this chapter the Multi-layer Feedforward Networks are considered, these contain one or more hidden layers. Hidden layers are placed between input and output layers. Those hidden layers enable extraction of higher-order features.

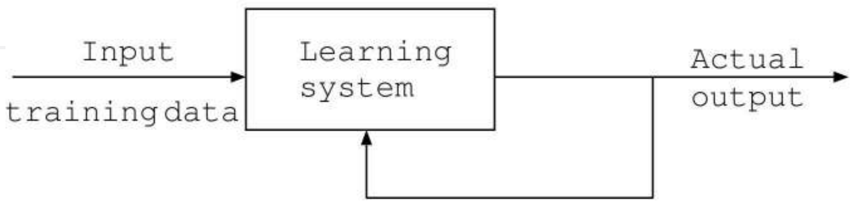

Figure 1. the three layers (input, hidden and output) of neurons are fully interconnected.

The input layer receives an external activation vector, and passes it via weighted connections to the neurons in the first hidden layer [25]. An example of this arrangement, a three layer NN, is shown in Fig 1. This is a common form of NN.

2.4. Neural Networks Learning

To use a NN, it is essential to have some form of training, through which the values of the weights in the network are adjusted to reflect the characteristics of the input data. When the network is trained sufficiently, it will obtain the most nearest correct output for a presented set of input data.

A set of well-defined rules for the solution of a learning problem is called a learning algorithm. No unique learning algorithm exists for the design of NN. Learning algorithms differ from each other in the way in which the adjustment of Δw_{ij} to the synaptic weight w_{ij} is formulated. In other words, the objective of the learning process is to tune the weights in the network so that the network performs the desired mapping of input to output activation.

NNs are claimed to have the feature of generalization, through which a trained NN is able to provide correct output data to a set of previously (unseen) input data. Training determines the generalization capability in the network structure.

Supervised learning is a class of learning rules for NNs. In which a teaching is provided by telling the network output required for a given input. Weights are adjusted in the learning system so as to minimize the difference between the desired and actual outputs for each input training data. An example of a supervised learning rule is the delta rule which aims to minimize the error function. This means that the actual response of each output neuron, in the network, approaches the desired response for that neuron. This is illustrated in Fig 2.

The error ε_{pi} for the ith neuron $u_i^{\,o}$ of the output layer o for the training pair (x_p, t_p) is computed as:

$$\varepsilon_{pi} = t_{pi} - o_{pi}^{\,o} \tag{3}$$

This error is used to adjust the weights in such a way that the error is gradually reduced. The training process stops when the error for every training pair is reduced to an acceptable level, or when no further improvement is obtained.

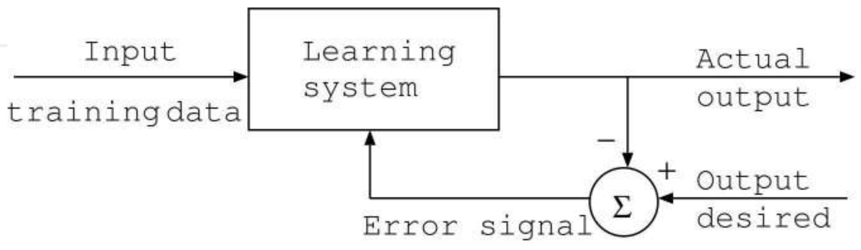

Figure 2. Example of Supervised Learning.

A method, known as "learning by epoch", first sums gradient information for the whole pattern set and then updates the weights. This method is also known as "batch learning" and most researchers use it for its good performance [25]. Each weight-update tries to minimize the summed error of the pattern set. The error function can be defined for one training pattern pair (x_p, d_p) as:

$$E_p = 1/2\sum_{i=1}^{n_o} \varepsilon_{pi} \tag{4}$$

Then, the error function can be defined for all the patterns (Known as the Total Sum of Squared, (TSS) errors as:

$$E = \frac{1}{2}\sum_{p=1}^{m}\sum_{i=1}^{n} \varepsilon_{pi} \tag{5}$$

The most desirable condition that we could achieve in any learning algorithm training is $\varepsilon_{pi} \geq 0$. Obviously, if this condition holds for all patterns in the training set, we can say that the algorithm found a global minimum.

The weights in the network are changed along a search direction, to drive the weights in the direction of the estimated minimum. The weight updating rule for the batch mode is given by:

$$w_{ij}^{s+1} = \Delta w_{ij}^{\ell}(s) + w_{ij}^{\ell}(s) \tag{6}$$

where w_{ij}^{s+1} is the update weight of w_{ij}^{ℓ} of layer ℓ in the sth learning step, and s is the step number in the learning process.

In training a network, the available input data set consists of many facts and is normally divided into two groups. One group of facts is used as the training data set and the second group is retained for checking and testing the accuracy of the performance of the network after training. The proposed ANN model was trained using Levenberg- Marquardt optimization technique [26].

Data collected from experiments are divided into two sets, namely, training set and testing set. The training set is used to train the ANN model by adjusting the link weights of network model, which should include the data covering the entire experimental space. This means that the training data set has to be fairly large to contain all the required information and must include a wide variety of data from different experimental conditions, including different formulation composition and process parameters.

Linearly, the training error keeps dropping. If the error stops decreasing, or alternatively starts to rise, the ANN model starts to over-fit the data, and at this point, the training must be stopped. In case over-fitting or over-learning occurs during the training process, it is usually advisable to decrease the number of hidden units and/or hidden layers. In contrast, if the network is not sufficiently powerful to model the underlying function, over-learning is not likely to occur, and the training errors will drop to a satisfactory level.

3. An overview of Genetic Algorithm

3.1. Introduction

Evolutionary Computation (EC) uses computational models of evolutionary processes based on concepts in biological theory. Varieties of these evolutionary computational models have been proposed and used in many applications, including optimization of NN parameters and searching for new NN learning rules. We will refer to them as Evolutionary Algorithms (EAs) [27-29]

EAs are based on the evolution of a population which evolves according to rules of selection and other operators such as crossover and mutation. Each individual in the population is given a measure of its fitness in the environment. Selection favors individual with high fitness. These individuals are perturbed using the operators. This provides general heuristics for exploration in the environment. This cycle of evaluation, selection, crossover, mutation and survival continues until some termination criterion is met. Although, it is very simple from a biological point of view, these algorithms are sufficiently complex to provide strong and powerful adaptive search mechanisms.

Genetic Algorithms (GAs) were developed in the 70s by John Holland [30], who strongly stressed recombination as the energetic potential of evolution [32]. The notion of using abstract syntax trees to represent programs in GAs, Genetic Programming (GP), was suggested in [33], first implemented in [34] and popularised in [35-37]. The term Genetic Programming is used to refer to both tree-based GAs and the evolutionary generation of programs [38,39]. Although similar at the highest level, each of the two varieties implements genetic operators in a different manner. This thesis concentrates on the tree-based variety. We will discuss GP further in Section 3.4. In the following two sections, whose descriptions are mainly based on [30,32,33,35,36,37], we give more background information about natural and artificial evolution in general, and on GAs in particular.

3.2. Natural and Artificial Evolution

As described by Darwin [40], evolution is the process by which a population of organisms gradually adapt over time to enhance their chances of surviving. This is achieved by ensuring that the stronger individuals in the population have a higher chance of reproducing and creating children (offspring).

In artificial evolution, the members of the population represent possible solutions to a particular optimization problem. The problem itself represents the environment. We must apply each potential solution to the problem and assign it a fitness value, indicating its performance on the problem. The two essential features of natural evolution which we need to maintain are propagation of more adaptive features to future generations (by applying a selective pressure which gives better solutions a greater opportunity to reproduce) and the heritability of features from parent to children (we need to ensure that the process of reproduction keeps most of the features of the parent solution and yet allows for variety so that new features can be explored) [30].

3.3. The Genetic Algorithm

GAs is powerful search and optimization techniques, based on the mechanics of natural selection [31]. Some basic terms used are:

- A *phenotype* is a possible solution to the problem;
- A *chromosome* is an encoding representation of a phenotype in a form that can be used;
- A *population* is the variety of chromosomes that evolves from generation to generation;
- A *generation* (a population set) represents a single step toward the solution;
- *Fitness* is the measure of the performance of an individual on the problem;
- *Evaluation* is the interpretation of the genotype into the phenotype and the computation of its fitness;
- *Genes* are the parts of data which make up a chromosome.

The advantage of GAs is that they have a consistent structure for different problems. Accordingly, one GA can be used for a variety of optimization problems. GAs are used for a number of different application areas [30]. GA is capable of finding good solutions quickly [32]. Also, the GA is inherently parallel, since a population of potential solutions is maintained.

To solve an optimization problem, a GA requires four components and a termination criterion for the search. The components are: a representation (encoding) of the problem, a fitness evaluation function, a population initialization procedure and a set of genetic operators.

In addition, there are a set of GA control parameters, predefined to guide the GA, such as the size of the population, the method by which genetic operators are chosen, the probabilities of each genetic operator being chosen, the choice of methods for implementing probability in selection, the probability of mutation of a gene in a selected individual, the method used to select a crossover point for the recombination operator and the seed value used for the random number generator.

The structure of a typical GA can be described as follows [41]

In the algorithm, an initial population is generated in line 2. Then, the algorithm computes the fitness for each member of the initial population in line 3. Subsequently, a loop is entered based on whether or not the algorithm's termination criteria are met in line 4. Line 6 contains the control code for the inner loop in which a new generation is created. Lines 7 through 10 contain the part of the algorithm in which new individuals are generated. First, a genetic operator is selected. The particular numbers of parents for that operator are then selected. The operator is then applied to generate one or more new children. Finally, the new children are added to the new generation.

```
(1)     $0 \rightarrow t$;
(2)     $population(s) \rightarrow P(t)$;
(3)     $evaluate(P(t))$;
(4)     REPEAT until solution is found
(5)         {
(6)             $t+1 \rightarrow t$;
(7)             $selection(P(t)) \rightarrow B(t)$;
(8)             $breeding(B(t)) \rightarrow R(t)$;
(9)             $mutation(R(t)) \rightarrow M(t)$;
(10)            $evaluate(M(t))$;
(11)            $survival(M(t), P(t-1)) \rightarrow P(t)$;
(12)        }
(13)    END REPEAT;
```

where:

s is a random generator seed;
t represents the generation;
$P(t)$ is the population in generation t;
$B(t)$ is the buffer of parents in generation t;
$R(t)$ are the children generated by recombining or cloning $B(t)$;
$M(t)$ are the children created by mutating $R(t)$.

Lines 11 and 12 serve to close the outer loop of the algorithm. Fitness values are computed for each individual in the new generation. These values are used to guide simulated natural selection in the new generation. The termination criterion is tested and the algorithm is either repeated or terminated.

The most significant differences in GAs are:

- GAs search a population of points in parallel, not a single point
- GAs do not require derivative information (unlike gradient descending methods, e.g. SBP) or other additional knowledge - only the objective function and corresponding fitness levels affect the directions of search
- GAs use probabilistic transition rules, not deterministic ones
- GA can provide a number of potential solutions to a given problem
- GAs operate on fixed length representations.

4. The Proposed Hybrid GA - ANN Modeling

Genetic connectionism combines genetic search and connectionist computation. GAs have been applied successfully to the problem of designing NNs with supervised learning processes, for evolving the architecture suitable for the problem [42-47]. However, these applications do not address the problem of training neural networks, since they still depend on other training methods to adjust the weights.

4.1. GAs for Training NNs

GAs have been used for training NNs either with fixed architectures or in combination with constructive/destructive methods. This can be made by replacing traditional learning algorithms such as gradient-based methods [48]. Not only have GAs been used to perform weight training for supervised learning and for reinforcement learning applications, but they have also been used to select training data and to translate the output behavior of NNs [49-51]. GAs have been applied to the problem of finding NN architectures [52-57], where an architecture specification indicates how many hidden units a network should have and how these units should be connected.

The process key in the evolutionary design of neural architectures is shown in Fig. The topologies of the network have to be distinct before any training process. The definition of the architecture has great weight on the network performance, the effectiveness and efficiency of the learning process. As discussed in [58], the alternative provided by destructive and constructive techniques is not satisfactory.

The network architecture designing can be explained as a search in the architecture space that each point represents a different topology. The search space is huge, even with a limited number of neurons, and a controlled connectivity. Additionally, the search space makes things even more difficult in some cases. For instance when networks with different topologies may show similar learning and generalization abilities, alternatively, networks with similar structures may have different performances. In addition, the performance evaluation depends on the training method and on the initial conditions (weight initialization) [59]. Building the architectures by means of GAs is strongly reliant on how the features of the network are encoded in the genotype. Using a bitstring is not essentially the best approach to evolve the architecture. Therefore, a determination has to be made concerning how the information about the architecture should be encoded in the genotype.

To find good NN architectures using GAs, we should know how to encode architectures (neurons, layers, and connections) in the chromosomes that can be manipulated by the GA. Encoding of NNs onto a chromosome can take many different forms.

4.2. Modeling by Using ANN and GA

This study proposed a hybrid model combined of ANN and GA (We called it "GA–ANN hybrid model") for optimization of the weights of feed-forward neural networks to improve the effectiveness of the ANN model. Assuming that the structure of these networks has been decided. Genetic algorithm is run to have the optimal parameters of the architectures, weights and biases of all the neurons which are joined to create vectors.We construct a genetic algorithm, which can search for the global optimum of the number of hidden units and the connection structure between the inputs and the output layers. During the weight training and adjusting process, the fitness functions of a neural network can be defined by considering two important factors: the error is the different between target and actual outputs. In this work, we defined the fitness function as the mean square error (SSE).The approach is to use the GA-ANN model that is enough intelligent to discover functions for p-p interactions (mean multiplicity distribution of charged particles with respects of the total center of mass energy). The model is trained/predicated by using experimental data to simulate the p-p interaction. GA-ANN has the potential to discover a new model, to show that the data sets are subdivided into two sets (training and predication). GA-ANN discovers a new model by using the training set while the predicated set is used to examine their generalization capabilities.To measure the error between the experimental data and the simulated data we used the statistic measures. The total deviation of the response values from the fit to the response values. It is also called the summed square of residuals and is usually labeled as *SSE*. The statistical measures of sum squared error (SSE),

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{7}$$

where $\hat{y}_i = b_0 + b_1 x_i$ is the *predicted value* for x_i and y_i is the observed data value occurring at x_i .

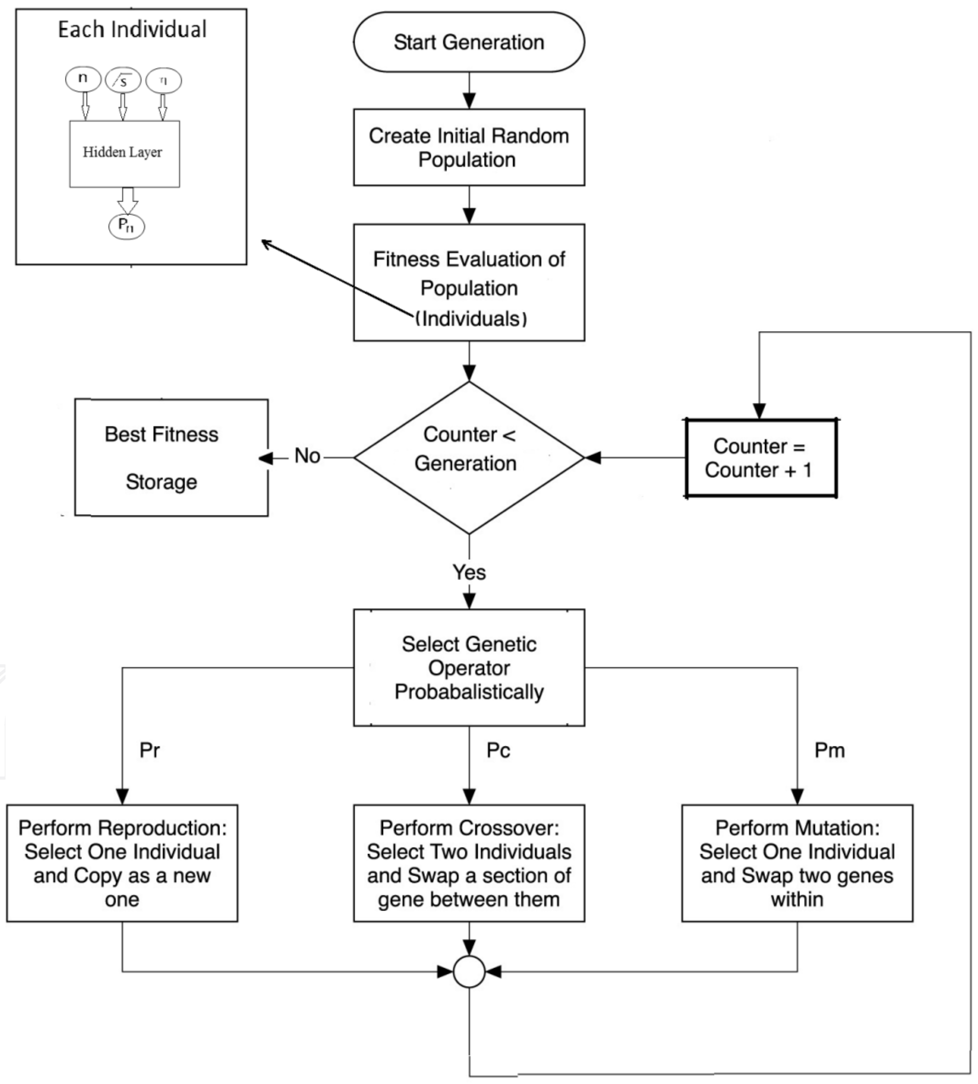
Each Individual
n
√s
η
Hidden Layer
Pη
Start Generation
Create Initial Random Population
Fitness Evaluation of Population (Individuals)
Best Fitness Storage
No
Counter < Generation
Counter = Counter + 1
Yes
Select Genetic Operator Probabalistically
Pr
Pc
Pm
Perform Reproduction: Select One Individual and Copy as a new one
Perform Crossover: Select Two Individuals and Swap a section of gene between them
Perform Mutation: Select One Individual and Swap two genes within

Figure 3. Overview of GA-ANN hybrid model.

The proposed GA-ANN hybrid model has been used to model the multiplicity distribution of the charged shower particles. The proposed model was trained using Levenberg-Marquardt optimization technique [26]. The architecture of GA-ANN has three inputs and one output. The inputs are the charged particles multiplicity (n), the total center of mass energy ($\sqrt{s}$), and the pseudo rapidity (η).The output is the charged particles multiplicity distribution (P_n). Figure 1 shows the schematic of GA-ANN model.

Data collected from experiments are divided into two sets, namely, training set and testing set. The training set is used to train the GA- ANN hybrid model. The testing data set is used to confirm the accuracy of the proposed model. It ensures that the relationship between inputs and outputs, based on the training and test sets are real. The data set is divided into two groups 80% for training and 20% for testing. For work completeness, the final weights and biases after training are given in Appendix A.

5. Results and discussion

The input patterns of the designed GA-ANN hybrid have been trained to produce target patterns that modeling the pseudo-rapidity distribution. The fast Levenberg-Marquardt algorithm (LMA) has been employed to train the ANN. In order to obtain the optimal structure of ANN, we have used GA as hybrid model.

Simulation results based on both ANN and GA-ANN hybrid model, to model the distribution of shower charged particle produced for P-P at different the total center of mass energy, $\sqrt{s}$ 0.9 TeV, 2.36 Tev and 7 TeV, are given in Figure 2-a, b, and c respectively. We notice that the curves obtained by the trained GA-ANN hybrid model show an exact fitting to the experimental data in the three cases.

Then, the GA-ANN Hybrid model is able to exactly model for the charge particle multiplicity distribution. The total sum of squared error SSE, the weights and biases which used for the designed network are provided in the Appendix A.

Structure	Number of connections	Error values	Learning rule
ANN: 3 x15x15x1	285	0.01	LMA
GA optimization structure	229	0.0001	GA

Table 1. Comparison between the different training algorithms (ANN and GA-ANN) for the for charge particle Multiplicity distribution.

In this model we have obtained the minimum error (=0.0001) by using GA. Table 1 shows a comparison between the ANN model and the GA-ANN model for the prediction of the pseudo-rapidity distribution. In the 3x15x15x1 ANN structure, we have used 285 connections and obtained an error equal to 0.0001, while the connection in GA-ANN model is 225. Therefore, we noticed in the ANN model that by increasing the number of connections to 285 the error decreases to 0.01, but this needs more calculations. By using GA optimization search, we have obtained the structure which minimizes the number of connections equals to 229 only and the error (= 0.0001). This indicates that the GA-ANN hybrid model is more efficient than the ANN model.

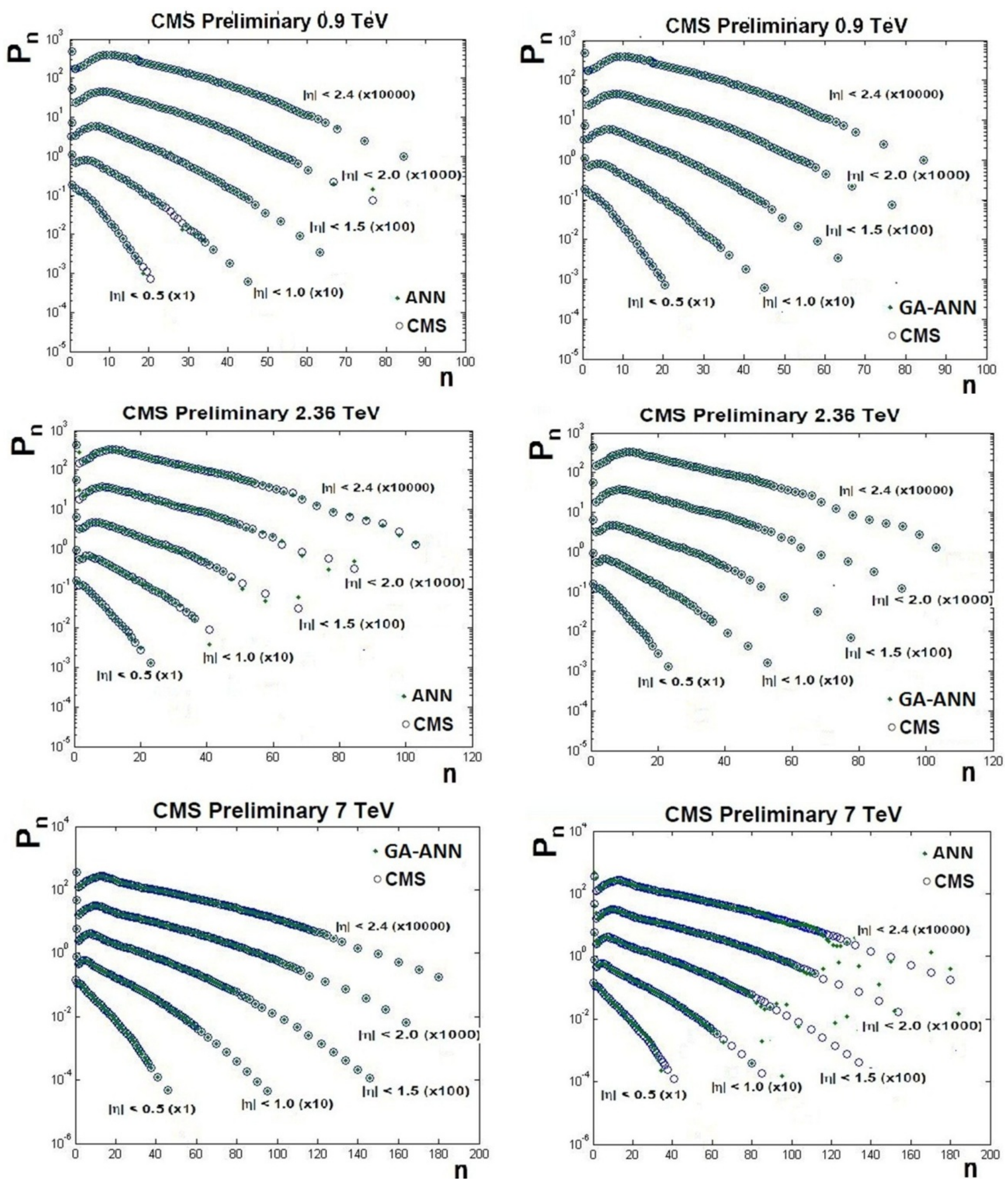

Figure 4. ANN and GA-ANN simulation results for charge particle Multiplicity distribution of shower p-p.

6. Conclusions

The chapter presents the GA-ANN as a new technique for constructing the functions of the multiplicity distribution of charged particles, P_n (n, η, $\sqrt{s}$) of p-p interaction. The discovered models show good match to the experimental data. Moreover, they are capable of testing experimental data for P_n (n, η, $\sqrt{s}$) that are not used in the training session.

Consequence, the testing values of P_n (n, η, $\sqrt{s}$) in terms of the same parameters are in good agreement with the experimental data from Particle Data Group. Finally, we conclude that GA-ANN has become one of important research areas in the field of high Energy physics.

Appendices

The efficient ANN structure is given as follows: [3x15x15x1] or [ixjxkxm].

Weights coefficient after training are:

Wji =	[3.5001	-1.0299	1.6118
	0.7565	-2.2408	3.2605
	-1.4374	1.1033	-3.1349
	2.0116	2.8137	-1.7322
	-3.6012	-1.5717	-0.2805
	-1.6741	-2.5844	2.7109
	-2.0600	-3.1519	1.2488
	-0.1986	1.0028	-4.0855
	2.6272	0.8254	3.6292
	-2.3420	3.0259	-1.9551
	-3.2561	0.4683	3.0896
	1.2442	-0.8996	-3.4896
	-3.2589	-1.1887	2.0875
	-1.0889	-1.2080	4.3688
	-2.7820	-1.4291	2.3577
	3.1861	-0.6309	2.0691
	3.4979	0.2456	-2.6633
	-0.4889	2.4145	-2.8041
	2.1091	-0.1359	-3.4762
	-0.1010	4.1758	-0.2120
	3.5538	-1.5615	-1.4795
	-3.4153	1.2517	2.1415
	2.6232	-3.0757	0.0831
	1.7632	1.9749	-2.5519
	7.6987	0.0526	0.4267].

Wkj =	[0.3294	0.5006	0.0421	0.3603	0.5147
	0.5506	0.2498	0.2678	0.2670	0.3568
	0.3951	0.2529	0.2169	0.4323	0.0683
	0.1875	0.2948	0.2705	0.2209	0.1928
	0.2207	0.6121	0.0693	0.0125	0.4214
	0.4698	0.0697	0.4795	0.0425	0.2387
	0.1975	0.1441	0.2947	0.1347	0.0403
	0.0745	0.2345	0.1572	0.2792	0.3784
	0.1043	0.4784	0.2899	0.2012	0.4270
	0.5578	0.7176	0.3619	0.2601	0.2738
	0.1081	0.2412	0.0074	0.3967	0.2235
	0.0466	0.0407	0.0592	0.3128	0.1570
	0.4321	0.4505	0.0313	0.5976	0.0851
	0.4295	0.4887	0.0694	0.3939	0.0354
	0.1972	0.1416	0.1706	0.1719	0.0761

Columns 6 through 10				
0.2102	0.0185	-0.1658	-0.1943	-0.4253
0.2685	0.4724	0.4946	-0.3538	0.1559
0.3198	0.1207	0.5657	-0.3894	0.1497
-0.5528	0.4031	0.5570	0.4562	-0.5802
0.3498	-0.3870	0.2453	0.4581	0.2430
0.2047	-0.0802	0.1584	0.2806	-0.2790
0.0981	-0.5055	0.2559	-0.0297	-0.2058
-0.3498	-0.5513	0.0022	-0.3034	0.2156
-0.6226	-0.4085	0.4338	-0.0441	-0.4801
-0.0093	0.0875	0.0815	0.3935	0.1840
0.0063	0.2790	0.7558	0.3383	0.5882
-0.5506	-0.0518	0.5625	0.2459	-0.0612
0.0036	0.4404	-0.3268	-0.5626	-0.2253
0.5591	-0.2797	-0.0408	0.1302	-0.4361

Columns 11 through 15				
-0.6123	0.4833	-0.0457	0.3927	-0.3694
-0.0746	-0.0978	0.0710	-0.7610	0.1412
-0.3373	0.4167	0.3421	-0.0577	0.2109
0.2422	0.2013	-0.1384	-0.3700	-0.4464
0.0868	-0.5964	-0.0837	-0.7971	-0.4299
-0.6500	-1.1315	-0.4557	1.6169	-0.3205
0.2205	1.0185	0.4752	-0.4155	0.1614
1.2311	0.0061	-0.0539	0.6813	0.9395
-0.4295	-0.3083	0.2768	-0.1151	0.0802
-0.6988	0.2346	-0.3455	0.0432	0.1663
-0.0601	0.0527	0.3519	0.3520	-0.7821
-0.6241	-0.1201	-0.4317	0.7441	0.7305
0.5433	-0.6909	0.4848	-0.3888	0.3710
-0.6920	-0.0190	-0.4892	0.1678	0.0808
-0.3752	-0.1745	-0.7304	0.0462	-0.3883].

Wmk = [0.9283 1.6321 0.0356 -0.4147 -0.8312 -3.0722 -1.9368 1.7113 0.0100 -0.4066 0.0721 0.1362 0.4692 -0.9749 1.7950].

bi = [-4.7175 -2.2157 3.6932].

bj = [-4.1756 -3.8559 3.9766 -3.3430 2.7598 2.5040 2.1326 1.9297

-0.6547 0.7272 0.5859 -1.1575 0.3029 0.3486 -0.4088].

bk = [1.7214 -1.7100 1.5000 -1.2915 1.1448 1.0033 -0.6584 -0.4397

-0.4963 -0.3211 0.2594 -0.1649 0.0603 -0.1078].

bm = [-0.2071].

The optimized GA-ANN:

The standard GA has been used. The parameters are given as follows: Generation = 1000, Population = 4000, probability of crossover = 0.9, probability of mutation = 0.001, Fitness function is SSE. A neural network had been optimized as 229 of neurons.

Acknowledgements

The authors highly acknowledge and deeply appreciate the supports of the Egyptian Academy of Scientific Research and Technology (ASRT) and the Egyptian Network for High Energy Physics (ENHEP).

Author details

Amr Radi[1*] and Samy K. Hindawi[2]

*Address all correspondence to: Amr.radi@cern.ch

1 Department of Physics, Faculty of Sciences, Ain Shams University, Abbassia, Cairo, Egypt / Center of Theoretical Physics at the British University in Egypt (BUE), Egypt

2 Department of Physics, Faculty of Sciences, Ain Shams University, Abbassia, Cairo, Egypt

References

[1] CMS Collaboration. (2011). *J. High Energy Phys.*, 01, 079.

[2] CMS Collaboration. (2011). *J. High Energy Phys.*, 08, 141.

[3] CMS Collaboration. (2010). *Phys. Rev. Lett.*, 105, 022002.

[4] ATLAS Collaboration. (2012). *Phys. Lett. B*, 707, 330-348.

[5] ATLAS Collaboration,. (2011). *Phys. Lett. B*.

[6] ALICE Collaboration,. (2010). *Phys. Lett. B*, 693, 53-68.

[7] ALICE Collaboration,. (2010). *Eur. Phys. J. C*, 68345-354.

[8] TOTEM Collaboration,. (2011). *EPL*, 96, 21002.

[9] Jacob, M., & Slansky, R. (1972). *Phys. Rev.*, D5, 1847.

[10] Hwa, R. (1970). *Phys. Rev.*, D1, 1790.

[11] Hwa, R. (1971). *Phys. Rev. Lett.*, 26, 1143.

[12] Engel, R., Phys, Z., & , C. (1995). *R. Engel, J. Ranft and S. Roesler, Phys. Rev.*, D52.

[13] Teodorescu, L., & Sherwood, D. (2008). *Comput. Phys. Commun.*, 178, 409.

[14] Teodorescu, L. (2006). *IEEE T. Nucl. Sci.*, 53, 2221.

[15] Link, J. M. (2005). *Nucl. Instrum. Meth. A*, 551, 504.

[16] El-Bakry, S. Yaseen, & Radi, Amr. (2007). *Int. J. Mod. Phys. C*, 18, 351.

[17] El-dahshan, E., Radi, A., & El-Bakry, M. Y. (2009). *Int. J. Mod. Phys. C*, 20, 1817.

[18] Whiteson, S., & Whiteson, D. (2009). *Eng. Appl. Artif. Intel.*, 22, 1203.

[19] Haykin, S. (1999). *Neural networks a comprehensive foundation* (2nd ed.), Prentice Hall.

[20] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.

[21] Koza, J. R. (1992). Genetic Programming: On the Programming of Computers by means of Natural Selection. The MIT Press, Cambridge, MA.

[22] Ferreira, C. (2006). Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence. 2nd Edition, Springer-Verlag, Germany.

[23] Eiben, A. E., & Smith, J. E. (2003). Introduction to Evolutionary Algorithms. Springer, Berlin.

[24] Radi, Amr. (2000). Discovery of Neural network learning rules using genetic programming. *PHD, the School of computers Sciences*, Birmingham University.

[25] Teodorescu, L. (2005). High energy physics data analysis with gene expression programming. *In 2005 IEEE Nuclear Science Symposium Conference Record*, 1, 143-147.

[26] Hagan, M. T., & Menhaj, M. B. (1994). Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks*, 6, 861-867.

[27] Back, T. (1996). *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York.

[28] Fogel, D. B. (1994). An Introduction to Simulated Evolutionary Optimization. *IEEE Trans. Neural Networks*, 5(1), 3-14.

[29] Back, T., Hammel, U., & Schwefel, H. P. (1997). Evolutionary Computation: Comments on the History and Current State. *IEEE Trans. Evolutionary Computation*, 1(1), 3-17.

[30] Holland, . H. ((1975).) Adaptation in Natural and Artificial Systems The University of Michigan Press Ann Arbor, Michigan

[31] Fogel, D. B. (1995). Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. *IEEE Press*, Piscataway, NJ.

[32] Goldberg, D. E. (1989). *Genetic Algorithm in Search Optimization and Machine Learning*, Addison-Wesley, New York.

[33] Richard. Forsyth (1981). BEAGLE A Darwinian Approach to Pattern Recognition. Kybernetes , 10(3), 159-166.

[34] Cramer, Nichael Lynn. (1985). A representation for the Adaptive Generation of Simple Sequential Programs. *in Proceedings of an International Conference on Genetic Algorithms and the Applications*, Grefenstette, John J. (ed.), CMU.

[35] Koza, J. R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press.

[36] Koza, J. R. (1994). Genetic Programming II: Automatic Discovery of Reusable Programs. MIT Press.

[37] Koza, J. R., Bennett, F. H., Andre, D., & Keane, M. A. (1999). *Genetic Programming III: Darwinian Invention and Problem Solving*, Morgan Kaufmann.

[38] Banzhaf, W., Nordin, P., Keller, R. E., & Francone, F. D. (1998). *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and its Applications*, Morgan Kaufmann.

[39] Mitchell, M. (1996). *An Introduction to Genetic Algorithms*, MIT Press.

[40] Darwin, C. (1959). *The Autobiography of Charles Darwin: With original omissions restored, edited with appendix and notes by his grand-daughter*, Nora Barlow, Norton.

[41] Whitley, Darrel. (1994). A genetic algorithm tutorial. *Statistics and Computing*, 4, 65-85.

[42] A new algorithm for developing dynamic radial basis function neural network models based on genetic algorithms

[43] Sarimveis, H., Alexandridis, A., Mazarkakis, S., & Bafas, G. (2004). *in Computers & Chemical Engineering*.

[44] An optimizing BP neural network algorithm based on genetic algorithm

[45] Ding, Shifei., & Su, Chunyang. (2010). *in Artificial Intelligence Review*.

[46] Hierarchical genetic algorithm based neural network design

[47] Yen, G. G., & Lu, H. (2000). *in IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*.

[48] Genetic Algorithm based Selective Neural Network Ensemble

[49] Zhou, Z. H., Wu, J. X., Jiang, Y., & Chen, S. F. (2001). *in Proceedings of the 17th International Joint Conference on Artificial Intelligence*.

[50] Modified backpropagation algorithms for training the multilayer feedforward neural networks with hard-limiting neurons

[51] Yu, Xiangui, Loh, N. K., Jullien, G. A., & Miller, W. C. (1993). *in Proceedings of Canadian Conference on Electrical and Computer Engineering*.

[52] Training Feedforward Neural Networks Using Genetic Algorithms

[53] Montana, David J., & Davis, Lawrence. (1989). *in Machine Learning*.

[54] van Rooij, A. J. F., Jain, L. C., & Johnson, R. P. (1996). *Neural network training using genetic algorithms*, Singapore, World Scientific.

[55] Maniezzo, Vittorio. (1994). Genetic Evolution of the Topology and Weight Distribution of Neural Networks. *in: IEEE Transactions of Neural Networks*, 5(1), 39-53.

[56] Bornholdt, Stefan., & Graudenz, Dirk. (1992). General Asymmetric Neural Networks and Structure Design by Genetic Algorithms. *in: Neural Networks*, 5, 327-334, Pergamon Press.

[57] Kitano, Hiroaki. (1990a). Designing Neural Networks Using Genetic Algorithms with Graph Generation Systems. *in: Complex Systems* [4], 461-476.

[58] Nolfi, S., & Parisi, D. (1994). Desired answers do not correspond to good teaching inputs in ecological neural networks. *Neural processing letters*, 1, 1-4.

[59] Nolfi, S., & Parisi, D. (1996). Learning to adapt to changing environments in evolving neural networks. *Adaptive Behavior*, 5, 75-97.

[60] Nolfi, S., Parisi, D., & Elman, J. L. (1994). Learning and evolution in neural networks. *Adaptive Behavior*, 3(1), 5-28.

[61] Pujol, J. C. F., & Poli, R. (1998). Efficient evolution of asymmetric recurrent neural networks using a two-dimensional representation. *Proceedings of the first European workshop on genetic programming (EUROGP)*, 130-141.

[62] Miller, G. F., Todd, P. M., & Hedge, S. U. (1989). Designing neural networks using genetic algorithms. *Proceedings of the third international conference on genetic algorithms and their applications*, 379-384.

[63] Mandischer, M. (1993). Representation and evolution of neural networks. Paper presented at Artificial neural nets and genetic algorithms proceedings of the international conference at Innsbruck, Austria. 643-649, Wien and New York, Springer.

[64] Figueira Pujol, Joao Carlos. (1999). Evolution of Artificial Neural Networks Using a Two-dimensional Representation. *PhD thesis*, School of Computer Science, University of Birmingham, UK.

[65] Yao, X. (1995f). Evolutionary artificial neural networks. *In Encyclopedia of Computer Science and Technology, ed. A. Kent and J. G. Williams, Marcel Dekker Inc., New York*, 33, 137-170, Also appearing in Encyclopedia of Library and Information Science.