ISSN 2229-5518

Power System Economic Dispatch Using Traditional and Neural Networks Programs

Draidi Abdellah, Labed Djamel.

Abstract— The introduction of techniques of artificial intelligence in software of control and decision is an essential element in research and development of tomorrow's power systems. Neural networks are among the techniques most used in the field of artificial intelligence. The economic dispatch is a key sector in the electricity networks, where it must generate less energy for the same demand with good economic operation reducing repartition grid losses to have the least cost of kWh possible. In this paper, we will opt for a quicker economic dispatch; we will program a mesh network of 8 buses including 3 generation units using traditional program then backpropagation learning neural network program, finally, we will compare the two programs in terms of speed and reliability.

Index Terms— Power systems, economic dispatch, artificial intelligence, neural networks, grid losses, traditional program, backpropagation learning.

1 Economic Dispatch

1.1 Introduction

THE Economic dispatch is a static optimization problem which consists in distributing active power production requested by different grid buses from generation unites in the most economical way. This distribution must of course respect the limits of production of generation units. The variable to be optimized is the production cost.

1.2 The cost function

The cost of production of a plant is generally modeled by a polynomial function of second degree in P_{Gi} (active power generated by the plant i) whose coefficients are constants specific to each plant: [1]

$$C_i(P_{Gi}) = a_i + b_i P_{Gi} + c_i P_{Gi}^2 \quad (1)$$

1.3 Economic dispatch solution

To minimise the total production cost of an interconnected power system we must minimize the sum of cost functions of production units

Minimise $$C = \sum_{i=1}^{ng} C_i(P_{Gi}) \quad (2)$$

taking into consideration the following constraints:

Equality constraints: $$\sum_{i=1}^{ng} P_{Gi} = \sum_{j=1}^{nd} P_{Dj} \quad (3)$$

Inequality constraints: $$P_{Gi}^{\min} \le P_{Gi} \le P_{Gi}^{\max} \quad (4)$$

where C is total cost function, ng is a total number of producer nodes and nd is a total number of consumer nodes. P_{Gi} represents the active power generated by the ith generator, P_{Dj} is the active power consumed by the jth load, P_{Gi}^{max} is the maximum active power of the ith generator and P_{Gi}^{min} is the minimum active power of the ith generator. [2]

The solution of this problem is obtained by using the Lagrange function which is obtained by multiplying the function of equality constraints by the Lagrange multiplier λ, adding to the total cost function (5).

$$L(P_{Gi}, \lambda) = C + \lambda\left(\sum_{i=1}^{ng} P_{Gi} - \sum_{j=1}^{nd} P_{Dj}\right) \quad (5)$$

The derivatives of the Lagrange equation with respect to each independent variable (P_{Gi}, λ) give us: [3]

$$\frac{\delta L}{\delta P_{Gi}} = \frac{dC_i(P_{Gi})}{dP_{Gi}} - \lambda = 0 \Rightarrow \lambda = \frac{dC_i(P_{Gi})}{dP_{Gi}} \quad (6)$$

$$\frac{\delta L}{\delta \lambda} = \sum_{j=1}^{nd} P_{Dj} - \sum_{i=1}^{ng} P_{Gi} = 0 \quad (7)$$

So, from (6) λ represents the incremental cost of the ith generator, then, for each energy packet the generator having the least λ is responsible of production (least cost) respecting the constraints of (3) and (4).

1.4 Insertion of losses formula in the economic dispatch

1.4.1 Calculation of Losses (P_L):

The general formula of losses following the equations of power flows is:

$$P_L = \psi^T G \psi \quad (8)$$

with $\psi = M\,\delta$. *M* and δ are matrices of *line's incidence* and *phases of nodes* respectively. *G is the diagonal matrix of line conductances.*

$$G = diag[G_{12} G_{13} G_{14} \ldots G_{(n-1)n}] \quad (9)$$

δ can be approximated by a *DC Load Flow* so,

$$P_G - P_D = A\delta \Rightarrow \delta = A^{-1}(P_G - P_D) \quad (10)$$

A represents the *DC Load flow Matrix*, therefore,

$$P_L = P_D^T B P_D - 2P_D^T B P_G + P_G^T B P_G \quad (11)$$

where $B = A^{-1} M^T G M A^{-1}$

- *Draidi Abdellah is currently pursuing Doctorat degree in electric power engineering at the electrical engineering department, and member of Electrical Engineering Laboratory of Constantine, University of Constantine, Algeria, PH-00213551826942. E-mail: abdellah.draidi@gmail.com*
- *Dr.Labed Djamel is currently a teacher at the electrical engineering department and member of Electrical Engineering Laboratory of Constantine University of Constantine, Algeria, PH-00213773312865. E-mail: djamel_labed@yahoo.fr*

ISSN 2229-5518

1.4.2 Penalty Factor: supposing

P_{Gi} Power generated by the ith plant.

P_{Ci} Part of the power generated that is really consumed by loads.

P_{Li} Part of the power generated that is lost in the lines, we know that: $$P_{Gi} = P_{Ci} + P_{Li} \quad (12)$$

and from(1) $$\frac{dC}{dP_{Gi}} = b_i + 2c_i P_{Gi} \quad (13)$$

by substitutions $$\frac{dC}{dP_{Ci}} = b_i' + 2c_i' P_{Gi} \quad (14)$$

where $b_i' = b_i f_i \ \& c_i' = c_i f_i$ are the new coefficients of (1), with $f_i = (1 - \frac{dP_L}{dP_{Gi}})^{-1}$ is the penalty factor of the incremental cost.

1.4.3 Criterion of convergence:

If $\left| \sum_{i=1}^{ng} P_{Gi} - \sum_{j=1}^{nd} P_{Dj} - P_L \right| \le \varepsilon$ the system has converged. [1]

2 Neural Networks

2.1 Introduction

The origin of artificial neural networks comes from the biological neuron modeling test by Warren McCulluch and Walter Pitts. They assumed that the nerve impulse is the result of a simple calculation made by each neuron and the thought is born with the collective effect of a network of interconnected neurons. [4]

2.2 Neuron Model

A neuron consists essentially of an integrator that performs the weighted sum of its inputs. The result n of this sum is then transformed by a transfer function f which produces the neuron output a. Fig. 1

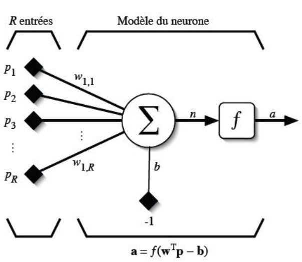

Fig. 1. Artificial neuron model

$$n = \sum_{j=1}^{R} w_{1,j} p_j - b \quad (15)$$
$$= w_{1,1}p_1 + w_{1,2}p_2 + \ldots + w_{1,R}p_R - b$$

This output corresponds to a weighted sum of the weights $w_{i,j}$ and inputs p_i minus the bias b. [5]

2.3 Neural network learning

There are essentially two types of learning, unsupervised and supervised. In our paper we will use a supervised learning where we impose to the network specific operations by forcing it from inputs submitted, the outputs to take by changing the synaptic weights. [6]

2.4 Error backpropagation learning

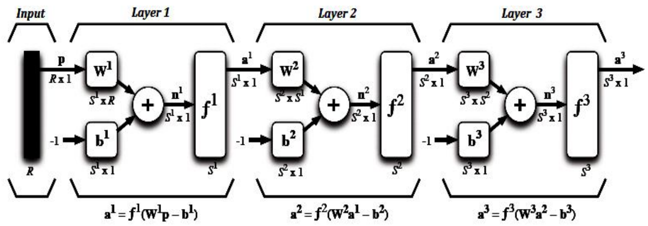

Fig. 2 Representation of three layers network

The simple perceptron consists of a single layer of S neurons which are fully connected to vector p of R entries. In a multilayer perceptron the equation that describes the output of layer k (Fig. 2) is given by:

$$a^k = f^k (W^k a^{k-1} - b^k) \text{ for } k = 1,\ldots,M \quad (16)$$

where M is the total number of layers. The network outputs correspond to a^M. The backpropagation algorithm uses the mean squared error as performance index, and allows a supervised learning with a set of associations (stimulus, target) $\{(p_q,d_q)\}$, q=1,...,Q where p_q represents the stimulus vector (inputs) and d_q the target vector (desired outputs). At each time t, we can forwardpropagate a different stimulus p(t) through the network of Fig. 2 to obtain an output vector a(t). This allows us to calculate the error e(t) between what the network produces as output for the stimulus and the target d(t) associated with it:

$$e(t) = d(t) - a(t) \quad (17)$$

The performance index *F* minimizes the mean square error. This index is approximated by the instantaneous error $\hat{F}(x)$, the vector *x* includes all the weights and biases of the network,

$$\hat{F}(x) = e^T(t)e(t) \quad (18)$$

we use the *gradient descent method* to optimize *x*:

$$\Delta w_{i,j}^k(t) = -\eta \frac{\partial \hat{F}}{\partial w_{i,j}^k} \ ; \ \Delta b_i^k(t) = -\eta \frac{\partial \hat{F}}{\partial b_i^k} \quad (19)$$

where η represents the learning rate, so:

$$\frac{\partial \hat{F}}{\partial w_{i,j}^k} = \frac{\partial \hat{F}}{\partial n_i^k} \times \frac{\partial n_i^k}{\partial w_{i,j}^k} \quad (20)$$

$$\frac{\partial \hat{F}}{\partial b_i^k} = \frac{\partial \hat{F}}{\partial n_i^k} \times \frac{\partial n_i^k}{\partial b_i^k} \quad (21)$$

n_i^k represent the *activation levels* of a layer *k* which depend directly on weights and bias on this layer;

$$n_i^k = \sum_{j=1}^{s^{k-1}} w_{i,j}^k a_j^{k-1} - b_i^k \quad (22)$$

so, the second term of (20) and (21) becomes:

$$\frac{\partial n_i^k}{\partial w_{i,j}^k} = a_j^{k-1} \ ; \ \frac{\partial n_i^k}{\partial b_i^k} = -1 \quad (23)$$

for the first term of (20) and (21), we define the ty s_i^k where $s_i^k = \frac{\partial \hat{F}}{\partial n_i^k}$, then (19) becomes:

ISSN 2229-5518

$$\Delta w_{i,j}^{k}(t) = -\eta s_i^{k}(t) a_j^{k-1}(t)\,;\ \Delta b_i^{k}(t) = \eta s_i^{k}(t) \qquad (24)$$

with

$$s^{k} = \left(\frac{\partial n^{k+1}}{\partial n^{k}}\right)^{T} \frac{\partial \hat{F}}{\partial n^{k+1}} = \dot{F}^{k}(n^{k})(W^{k+1})(s^{k+1}) \qquad (25)$$

In this case, we get a recursive formula where sensitivity layers upstream (input s^k) depend on the sensitivity of the layers downstream (output s^{k+1}). That's where the term «*back-propagation*» comes, because the direction of information propagation is reversed compared to that of (16). [5]

3 Economic Dispatch Using Neural Networks

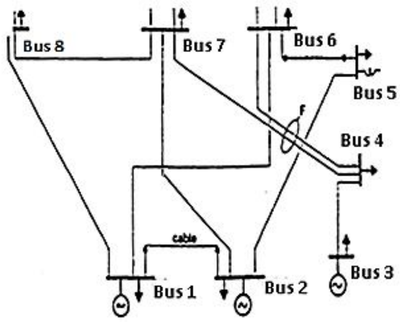

Fig. 3 The 138KV area from IEEE 24-Bus Test Network[7]

In this example we will study and program the economic dispatch of a mesh network using the traditional and the neural network programs. The network is the 138KV area in the IEEE 24-Bus Test Network (Fig. 3) and constituted of 8 busbars; three Generation buses (Bus 1, Bus 2 and Bus 3) and five Load buses (Bus 4, Bus 5, Bus 6, Bus 7 and Bus 8) [7]

First, we show the characteristics of generators;

Table 1
Characteristics of Generators [7]

Gen Bus	Pmax [MW]	Pmin [MW]	a [$/h]	b [$/MWh]	c [$/MW²h]
Bus 1	308	0	646.99	19.18	0.0322
Bus 2	350	0	646.99	19.18	0.0322
Bus 3	250	0	1829.71	27.22	0.0628

Second, we give busbars distances and impedances;

Table 2
Distances and Impedances between Busbars [7]

Line	from	to	Distance [Miles]	R[Ω]	X [Ω]
1	1	8	55	0.055	0.21
2	1	6	45	0.02	0.08
3	1	2	3	0.003	0.014
4	2	7	60	0.015	0.115
5	2	5	50	0.05	0.192
6	3	4	16	0.016	0.06
7	4	7	43	0.043	0.165
8	4	6	43	0.043	0.165
9	5	6	16	0.014	0.061
10	7	8	31	0.031	0.119

Afterwards, we present energy demands of load buses;

Table 3
Energy Demands of Load Buses

DEMAND [MW]					TOTAL DEMAND [MW]
B4	B5	B6	B7	B8	
0	46	1	15	90	152
0	56	10	25	100	191
0	66	20	35	110	231
0	76	30	45	120	271
0	86	40	55	130	311
0	96	50	65	140	351
0	106	60	75	150	391
10	116	70	85	160	441
20	126	80	95	170	491
30	136	90	105	180	541
40	146	100	115	190	591
50	156	110	125	200	641
60	166	120	135	210	691
70	176	130	145	220	741
80	186	140	155	230	791
90	196	150	165	240	841
100	206	160	175	250	891

3.1 Traditional Program

The traditional program (dispatch with losses) gives us the results shown in Table 4.

3.2 Associated Neural Network Program

Based on the results of the traditional dispatch program, we will create a neural network with P: input matrix [5x17]taken from Table 3, T: target matrix [3x17]taken from Table 4 representing the power generated by the three stations using traditional program. We will use two layers neural network with error backpropagation learning (Fig. 4):

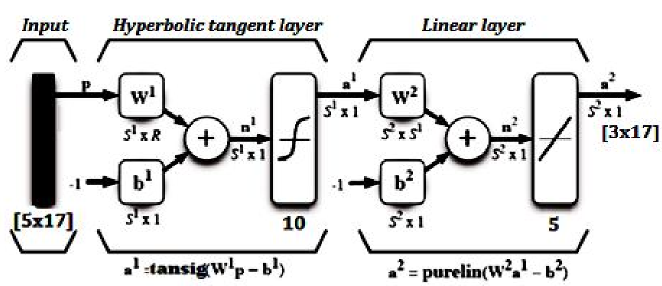

Fig. 4 Representation of the neural network used

The network contains: an input (P matrix), a hidden layer of 10 neurons with tansig (hyperbolic tangent sigmoid) activation

ISSN 2229-5518

function and an output layer of 5 neurons with Pureline (linear) activation function. The learning results are shown in Table 4.

TABLE 4
RESULTS OF PLANTS GENERATIONS FOLLOWING TOTAL DEMANDS OF TABLE 3 USING TRADITIONAL AND NEURAL NETWORK PROGRAMS

TRADITIONAL PROGRAM RESULTS [MW] (target)			NEURAL NETWORK PROGRAM RESULTS [MW] (output)		
B1	B2	B3	B1	B2	B3
153.053	0	0	152.451	1.859	1.622
192.352	0	0	192.758	1.464	0.589
232.705	0	0	235.009	0.292	0.631
273.112	0	0	272.580	0.599	1.231
307.207	6.291	0	300.584	8.783	1.924
307.633	46.175	0	310.004	38.134	1.909
288.103	106.935	0	288.254	107.243	0.371
307.412	117.122	21.097	298.709	124.038	21.392
307.91	127.326	61.005	307.791	126.953	60.894
307.444	137.548	102.136	309.066	131.349	105.869
307.271	150.482	140.448	310.001	150.728	137.788
307.745	180.552	160.551	309.189	180.089	160.627
307.244	211.715	180.666	307.778	210.695	181.743
307.769	241.992	200.792	307.894	239.753	203.061
307.322	273.376	220.930	309.280	269.228	222.915
307.900	303.865	241.08	310.083	303.132	238.014
307.906	349.977	244.867	308.067	346.016	244.817

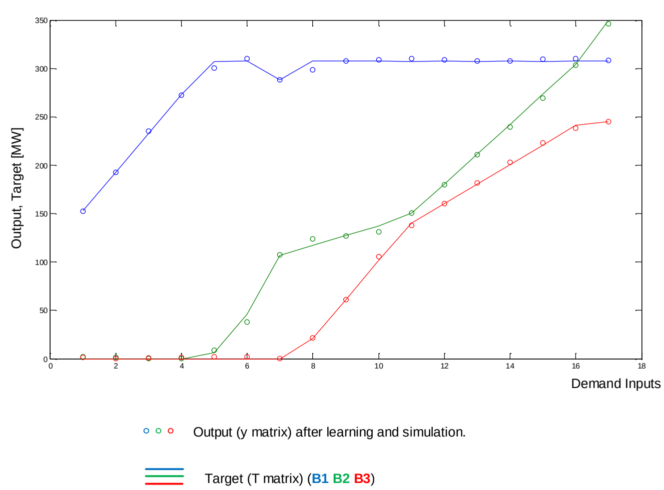

Fig. 5 Comparison between output (Y) and Target (T)

We can see from Table 4 and Fig. 5 that the values of the output are close to the target. We can deduce then that the training of our neural network is considered good. We must therefore test the network to judge the reliability of the learning.

3.3 Network Test

We introduce the matrix P_{test} which contains values in the borders of training matrix P, but did not participate in it.

TABLE 5
BUSBAR DEMANDS FOR NEURAL NETWORK TESTING

Ptest [MW]				
B4	B5	B6	B7	B8
0	75	29	44	119
0	77	31	46	121
11	117	71	86	161
25	131	85	100	175
53	159	113	128	203
65	171	125	140	215
72	178	132	147	222
86	192	146	161	236

then, P_{test} is passed through the traditional and Neural network programs. This will give us Ttest and Ytest respectively (Table 6).

TABLE 6
TEST RESULTS MATRICES TTEST AND YTEST

Ttest [MW]			Ytest [MW]		
B1	B2	B3	B1	B2	B3
269.069	0	0	269.784	0.001	0.004
277.156	0	0	276.264	0.971	0.003
307.460	118.142	25.079	309.767	116.283	24.360
307.172	132.435	82.044	307.092	128.125	86.656
307.892	189.592	166.584	307.629	189.400	167.042
307.503	226.841	190.727	307.038	228.499	189.488
307.878	248.060	204.819	306.984	250.808	202.875
307.666	291.657	233.018	307.694	290.522	234.198

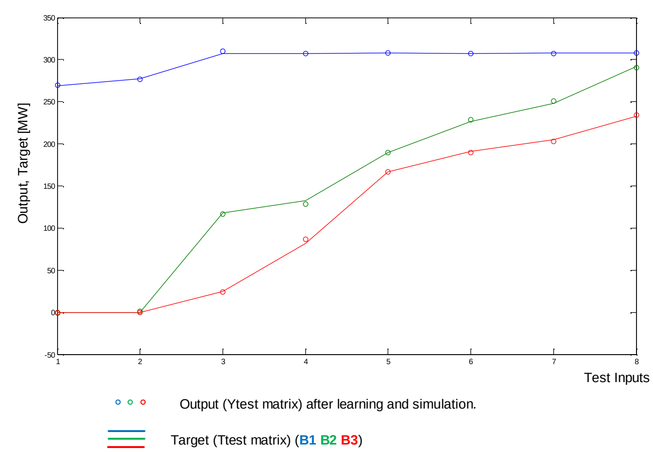

Fig. 6 Comparison between Ytest and Ttest

Based on data in Table 6 and Fig. 6, we can confirm that the learning of our neural network is considered good. To have a very good to excellent learning, output must be perfectly identical to the target (Y ≡ T), this requires:

- Using a good economic dispatch traditional program, based on precise data: power plants parameters, lines impedances and grid topography;

- Simulating the largest number of cases based on actual statistics of busbars energy demands;
- Simulating random and unpredictable cases (fault cases) which are not included in the statistics;
- Reducing learning step of the neural network to make a significant interpolation for unknown data (unpredictable);
- Adding more data, i.e. increasing the size of P and T matrices, thus making a more detailed learning which covers the majority of cases from the traditional program (caution: very lengthy P and T matrices ask for more powerful processors otherwise we risk having slow program).

A good economic dispatch software based on neural networks, must have a complete and excellent learning and test, then, it could execute directly the dispatching, i.e. demand data could be presented in real time to the neural network program where the decisions take two cases: if the neural network finds data (real time) in its matrix P, it gives immediate release to predefined outputs from matrix T; if the neural network cannot find data in matrix P, it makes a direct interpolation to give results.

4 Conclusion

The most important difference between traditional and neural network programs is the execution time. The traditional program is slow because:

- It uses iterative loops : *if, while* and *for* ; which affects directly the execution time;
- It stocks a largest number of parameters: parameters of plants, lines and each busbar;
- The more a network is meshed the more the program is slow.

For the neural network program, the execution time is very rapid (milliseconds), because:

- It contains loops only on training; then after training the problem of dispatching becomes a classification problem.
- It is an executor; it performs data already stored and interpolates intermediate ones.

In practice, we opt for an economic dispatch with the fastest possible frequency (5 or 15 minutes instead of every hour [8]), even in real time [9], which is practically impossible with the traditional program.

To control power grids, time is very important and vital especially in economic dispatch and many other disciplines: protection, stability, power flow, etc. Using techniques of artificial intelligence and more specifically neural networks reduces the execution time; this will bring a huge economic gain by reducing losses, thus restraining the consumption of fuels (coal, oil, gas, uranium etc.). The decrease of production implies a contribution to the preservation of the environment by reducing pollution and global warming.

5 Acknowledgment

The authors gratefully acknowledge the contributions of Peter Kelen, research officer and owner of Power Optimization Software.

6 References

[1] ELEC234 : "Conduite des réseaux électriques", *Projet : dispatching économique*. Université libre de Bruxelles.

[2] M.Younes, M.Rahli and M.Kandouci, "Répartition économique des puissances par un algorithme génétique en code réel "*6ème Conférence francophone de modélisation et simulation* - MOSIM'06 - du 3 au 5 avril 2006 – Rabat- Maroc.

[3] A.Merev, "Comparison of the economic dispatch solutions with and without transmission losses" *Istanbul university engineering faculty.* JEE, Vol.2, No. 2, 2002.

[4] J-F. Jodouin, *Les réseaux de neurones : principe et définition*. Lavoisier, Septembre 1994.

[5] M. Parizeau, *Réseau de neurones GIF-21140 et GIF -64326*. Université LAVAL, Automne 2004.

[6] P. Borne, M.Benrejeb and J.Haggège, *Les réseaux de neurones : présentation et application*. TECHNIP, 2007.

[7] C-W.Chua, *A stochastic pool-based electricity market simulator. McGill University*, Montreal, November 2000.

[8] "Economic Dispatch: Concepts, Practices and Issues" *FERC Staff,* California*,* November 13, 2005.

[9] Subject: "Security Constrained Unit Commitment and Real-Time Commitment Rules" TECHNICAL BULLETIN 51, 11/10/2004; revised 9/9/09.