Evaluation of Training Algorithms for the Laser Cutting Process

Ali Zaman and S. Mustafa Ali Zaidi
SZABIST
Karachi, Pakistan.

Abstract: - *In [1] it was shown that using orthogonal array as input to neural network for estimation of laser cutting is not a good approach while incorporating the Levenberg Marquardt training algorithm, furthermore it was argued that Levenberg Marquardt is the best training algorithm for the problem of laser cutting. But theoretically speaking Gradient descent with momentum, Quasi Newton and Levenberg Marquardt, all have a probability of finding the global minima in the error space. The probability can be higher for one and lesser for another but there is a probability none the less. In research done for this paper an exhaustive search was performed, a search in the "structure" and training algorithms so as to look at the idea objectively. It is hypothesized that may be, even if the gradient descent is given the "right" momentum and a "good" learning rate than it can find the actual global minima and thus a pattern might become recognizable in the error space which maps this language into quantification.*

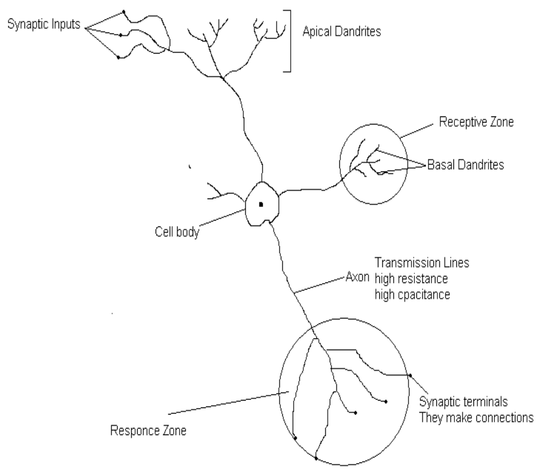
INTRODUCTION

Laser Cutting is a process by which material is cut using high amplified light. The process has four variables; the speed at which the laser moves, the power of the laser, the standoff distance and the gas pressure. The material being cut is Perspex sheet. In this paper about laser cutting an experiment will be conducted. The experiment will be controlled as that will lead to a larger domain of observational characteristics. Here the data used for the experiment is the same which has been used in [1]. A design will be used in order to come to a correct formulation. For the purpose of systematic design factorial design was selected. The idea behind factorial design is that the process being studied has discrete factors playing a role in the manipulation of its outcome so we change the factors in a way that we can understand the role of each factor individually as well as collectively. After getting data in the form of factorial design, Taguchi method was applied on the data. This generalizes the variable values, Applying Taguchi method gives us an orthogonal array which is also saturated, thus giving us the outcome mentioned above, but it is not at all as good as it sounds[2], it has been proposed that the experiment design which is sequential in nature has greater efficiency and accuracy. But the use of Factorial design and the Taguchi method was the choice in [1] and since the primary purpose of this paper is to study the different algorithms available for training neural networks, selection criteria for experiment design lies outside the boundaries of this paper. The reason for not using statistical design has been already been shown in [3] that nonlinear regression model is not sufficient for solving this problem, as a matter of fact it was shown that average error reaches 50% which is a huge error. That is why neural network approach was adopted as was the case in [1] by Mustafa. What makes this study different from that will be discussed in the discussion part of this report. Before going into the details of neural networks first a brief introduction is necessary in order to grasp the concept in this paper in its totality.

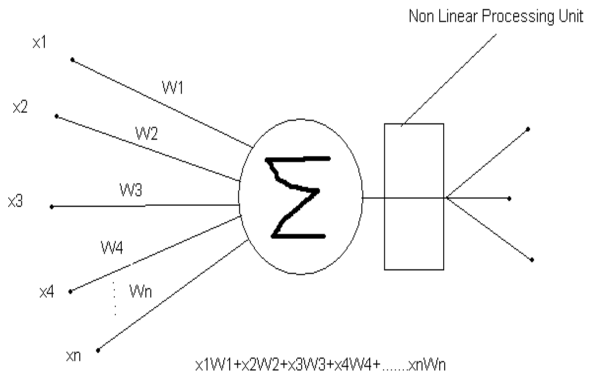
ARTIFICIAL NEURAL NETWORKS

The motivation behind artificial neural network as a model for computation lies in the work of Hebb (1949). It was observed that humans have been performing computations as long as they have existed. The system which we have in our brains is a massively parallel system through which we can perform very complicated tasks. Ever since the beginning of computers they were used to perform tasks that required computation which are algebraic in nature i.e. tasks that can be broken down to addition and subtraction. But there are certain tasks which we as humans perform exponentially better than computers e.g. recognizing a face, from thousands of faces we know we can recognize which face belongs to which person in a fraction of seconds.

Some salient features of Artificial Neural Networks are non-linearity, input output mapping, ability to adapt to environment, evidential response, and fault tolerance, analogous to neurobiological systems. Below is a diagram of a neural network in a human,



The synaptic input connections in a human are strong and weak. These connections are adjusted so that we become better able to predict desired outputs. Analogous to this system the following electronic model of computation can be abstracted.



The variables denoted by x are the inputs and the variables denoted by w are the synaptic connection strengths. These weights are the free parameters which are adjusted to get the desired output. This model of computation can be used to perform regression analysis as will be explained below.

In regression Analysis we have some data and we wish to find out the slope of the line which is closest to all the data because this line best explains the relationship between the input and the output. Now we know that the slope intercept form of a line is:

$$y=mx + b \quad (1)$$

Here m is the slope, x the input and b is a constant. In the above picture of artificial neuron the line that has no label and is going in the summer neuron is the constant input. It is usually termed as the bias. In this ANN analogy the weights are the slopes, the bias is the constant input to the neuron and the inputs are the independent variables. A neuron with one weight W1, one input x1 and one constant input b and output y could be written as y=W1x1+b. The error is calculated as:

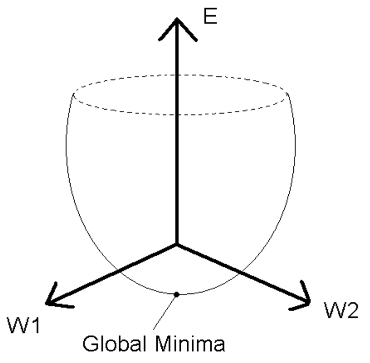
$$\text{Error} = \text{target} - \text{output} \quad (2)$$

In case of many input values and output values that are known we sum up the squared error of all known input output mappings in the following formula:

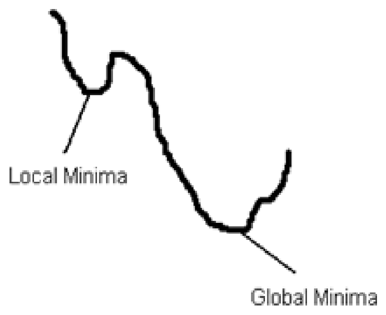
$$\sum_{p=0}^{p=n} E^{p} = \sum (t^{p} - y^{p})^{2} \quad (3)$$

This formula gives the error of one weight configuration with respect to all provided mappings between inputs and outputs, thus performing a regression. And on the basis of this we change weights and then try it again to see whether error is lesser or greater and we follow the procedure

iteratively. This gives us an error surface, e.g. consider a neural network with two weights w1 and w2 and consider the following example surface



The main purpose is to set the weights such that E becomes minimum. Usually a surface is not as simple as shown above e.g.



DISCUSSION ABOUT TRAINING ALGORITHMS USED

For the purpose of training neural networks three algorithms were selected namely, gradient descent with momentum, Quasi Newton and Levenberg Marquardt. Only Levenberg Marquardt was used in Mustafa [1] but it was seen here that all have a probability of converging towards the global minima so all were considered. 9000 neural networks were initialized, 3000 each for gradient descent with momentum, Quasi Newton and Levenberg Marquardt. In case of gradient descent with momentum one configuration had 10 instances each while in the case of Levenberg Marquardt and Quasi Newton each configuration had 600 instances of separate configurations.

The Back Propagation Algorithm

The back propagation algorithm is simply an algorithm which changes weights on the basis of previously calculated mean squared error. The algorithm is iterative in nature and can be expressed in the form of a pseudo code as follows:

- first randomly or by some other means initialize the weights
- get the output from the network with these weights
- calculate error by subtracting output from known input output mapping
- compute new weights or delta ws from output layer to hidden layers
- compute for all successive layers to input layer
- change the old weights with the new weights
- repeat from step 2 until network converges to desired error

The Gradient Descent Algorithm

In this algorithm weights are adjusted by using the following simple rule

$$\triangle W_{ji} = \propto (t_j - y_j) g'(h_j) x_i$$

(4)

new weight for i= learning rate*(desired result - actual result)*input i

This algorithm is very slow and it requires the user to give the learning rate. If the learning rate is too high the result has a high probability of being erroneous for global minima might have already passed. If the learning rate is too small then it works very slowly. It is also called steepest descent. The gradient descent algorithm often gets stuck in local minima.

The Gradient Descent With Momentum

As its name implies this algorithm adds a momentum term to the gradient descent algorithm so that it does not get stuck in a local minima. Gradient descent with momentum is a good algorithm as is proposed in [4]; still it depends on the user to tell it what the momentum term should be none the less it has a probability of finding the global minima but that probability seems lesser than Quasi Newton and Levenberg Marquardt. A mathematical representation would be:

new weight for i= learning rate*(desired result - actual result)*input i + Momentum*previous weight…(equation(5)

The Quasi Newton Algorithm

The point at which the derivative of a function is zero is called the stationary point of that particular function. This point is a value which is given as input to the function. Isaac Newton proposed a mathematical method to locate the stationary point on a function. The Quasi Newton Algorithm has its origins in that function. Isaac Newton made an assumption in this method and that assumption is that the area around the optimum can be approximated to be quadratic in nature. The advantage of the Quasi Newton method is that it does not require the computation of second derivate Hessian Matrix; instead one after another gradient vectors are analyzed and used to update the Hessian Matrix. The Quasi Newton is another option in case one has to use the conjugate gradient optimization. But usually in the domain of neural networks this algorithm is used when training is of small networks [1]. Secondly it must be noted that this algorithm requires a line which it uses to find the direction in which it starts the descent, once that is set it goes into the depth of that line. How this line is selected was not settled in this independent study, for the purpose of experiments conducted in this paper the default line function for Quasi Newton GDSM was used which is srchcha. The Quasi Newton Algorithm can be written mathematically as:

Change in weight= previous weight - (previous Hessian * previous gradient)….equation (6)

Levenberg Marquardt Algorithm

The beauty of Levenberg Marquardt is that it does not require the solution of second order derivative matrix i.e. the Hessian Matrix. Given mean squared error as the performance criteria, Hessian Matrix becomes

H= transpose (J) * J (7)

Where J is a Jacobian matrix i.e. a matrix with all first order partial derivatives. The partial derivatives are:

$$\frac{\partial \text{ Error}}{\partial \text{ Weights \& biases}} \quad (8)$$

The reason because of which partial derivative is needed is that the error is partially dependent on many variables. Here is the equation is clear, we are taking partial change in error with respect to weights and biases. The gradient g is calculated as:

g= transpose (Jacobian)*e (9)

Where e is a symbolic representation of vector consisting of errors in the network. Now that the basics are known, the Levenberg Marquardt algorithm can be expressed as:

Change in weight=previous weight - [transpose (J)*J + µI]$^{-1}$ * transpose (J)*e (10)

The genius lies in the µ symbol, for when it becomes 0 the equation becomes a Quasi Newton equation, and when this quantity becomes too large the equation represents a simple gradient descent with momentum. This can be explained very easily, all this equation is saying is that when the algorithm reaches towards a minima it starts to act like a Quasi Newton equation so that it does not get stuck in a local minima and when it is away from a minima it acts like a normal gradient descent with momentum. According to [1] the LM algorithm is very suitable for training medium sized datasets. The only drawback lies in the large memory which it requires, but since our requirement is not to find relationships between the laser cutting variables at real time, this algorithm is very suitable.

METHODOLOGY USED TO REACH SOLUTION

The Laser cutting process which is to be optimized consists of 4 controllable input variables:

- Laser Power in watts
- Cutting Speed in meters/minute
- Assist Gas Pressure in Bars
- Standoff Distance in millimeter

The outputs which are to be found out on the basis of these controllable variables are:

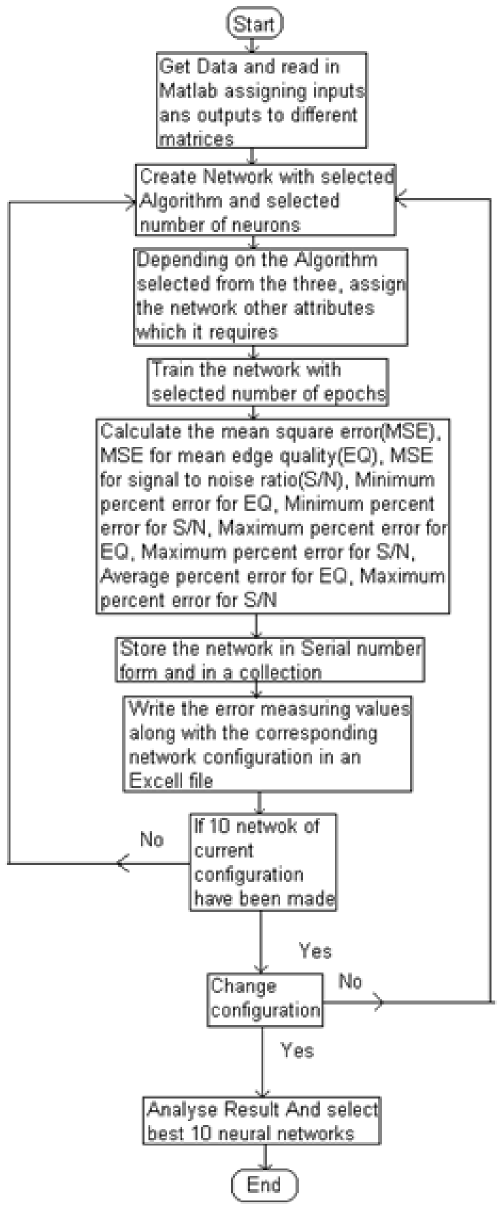
- Mean Edge Quality
- Signal to Noise Ratio

Both have to do with the edge quality of the material that has been cut. For the purpose of optimization in this paper no new experiments were conducted on laser cutting machine, simply the orthogonal array used in [1] was used as it is a generalization of the factorial design. The data provided for the experiment was in the following form:

Run	Laser Power (watt)	Cutting Speed (m/min)	Assist Gas Pressure (bars)	Stand Off Distance (mm)	Mean Edge Quality	Signal to Noise Ratio
1	100	0.2	0.5	1	0.1467	16.6666
2	100	0.2	2.5	5	0.0700	23.0980
3	100	1.2	4.5	10	0.0000	0.0000
4	300	0.2	2.5	10	0.0500	26.0206
5	300	0.7	4.5	1	0.2200	13.1515
6	300	1.2	0.5	5	0.0800	21.9382
7	500	0.2	4.5	5	0.1200	18.4164
8	500	0.7	0.5	10	0.1400	17.0115
9	500	1.2	2.5	1	0.1100	19.1721

Table 1: Inputs output mappings for training

The process of the experiment conducted in this paper can be represented easily by the following flow chart:


Start
Get Data and read in Matlab assigning inputs ans outputs to different matrices
Create Network with selected Algorithm and selected number of neurons
Depending on the Algorithm selected from the three, assign the network other attributes which it requires
Train the network with selected number of epochs
Calculate the mean square error(MSE), MSE for mean edge quality(EQ), MSE for signal to noise ratio(S/N), Minimum percent error for EQ, Minimum percent error for S/N, Maximum percent error for EQ, Maximum percent error for S/N, Average percent error for EQ, Maximum percent error for S/N
Store the network in Serial number form and in a collection
Write the error measuring values along with the corresponding network configuration in an Excell file
If 10 netwok of current configuration have been made
No
Yes
Change configuration
No
Yes
Analyse Result And select best 10 neural networks
End

ANALYSIS

Following the above procedure a total of 9000 neural networks were made, 3000 for each algorithm. The table in Appendix A shows the best 1997 neural networks whose MSE is lesser than the MSE achieved in [1]. It concluded with the remark that early stopping leads to more generalization, here it was done through built in mechanisms in Matlab and this remark was confirmed. The conclusion ended with the creation of a neural network with Max % error of 25 and an average percent error of 10 and this lead to the hypothesis that training on the basis of Orthogonal Array is not sufficient, but in the same thesis only eight of the array values were taken for training. In the experiment conducted in this independent study all nine values were used for training and the results were dramatic. The best neural networks achieved were of the following configuration and output values.

A	B	C	D	E	F	G	H	I
2.725	9E-04	5.45	0.002	0.131	0.301	11.7	0.034	1.32
3.428	9E-04	6.86	0.047	4.834	0.564	7.06	0.068	1.32
2.602	6E-04	5.2	0.057	3.601	8.341	7.69	0.933	1.25
4.652	0.003	9.3	0.032	2.555	0.418	11.8	0.05	1.6
1.57	0.002	3.14	0.005	0.961	12.52	10.3	1.391	1.26
0.753	0.002	1.5	0.032	3.573	10.94	17	1.22	2.28
2.339	7E-04	4.68	0.002	3.802	0.536	27.8	0.06	3.51
1.723	0.002	3.45	0.151	0.033	10.84	23.1	1.221	2.57
4.236	6E-04	8.47	0.012	1.384	4.887	23.3	0.544	2.74
1.375	0.001	2.75	0.032	0.899	1.77	49	0.2	5.55

Table 2: Error Values

The sequence represents the rank which a particular network was given. It was given by taking the average of all error checking parameters used in the above table.
The Best Artificial Neural Network with respect to training algorithms was:

Algo	A	B	C	D	E	F	G	H	I
Best gdm	1.58	0.02	3.1	0.093	2.27	3.362	65.1	0.38	7.48
Best QN	2.66	0.04	5.3	0.323	1.09	67.17	2.93	7.5	0.45
Best LM	2.73	0	5.4	0.002	0.13	0.301	11.7	0.03	1.32

Table 3: Best Outputs with respect to training algorithms

In the above table the columns represent the following variables:

A	MSE
B	MSE mean Edge Quality
C	MSE signal To noise ratio
D	min % error of mean edge quality
E	min % error of signal to noise ratio
F	max% error of mean edge quality
G	max% error of signal to noise ratio
H	average % error of mean edge quality
I	average % error of signal to noise ratio

Table 4: Explanation of Error Values

The values selected for network structures:

number of neurons	[2 4 6 8 10]
epochs	[500 1000]
training algorithms	[traingdm trainbfg trainlm]
performance function	[mse]
learning rate in case of gdm	[0.001 0.003 0.005 0.007 0.009]
momentum used in case of gdm	[0.1 0.3 0.5 0.6 0.7 0.8]

Table 5: Network Attributes

As it can be seen, out of the best ten neural networks only a few have performed lesser than the neural network proposed in [1]. Secondly all the best neural networks are those which used Levenberg Marquardt algorithm, the interesting fact is that the number of neurons in the best was in the following order:

best	lm with 4 neurons
2nd	lm with 8 neurons
3rd	lm with 10 neurons
4th	lm with 10 neurons
5th	lm with 10 neurons
6th	lm with 6 neurons
7th	lm with 8 neurons
8th	lm with 8 neurons
9th	lm with 4 neurons
10th	lm with 10 neurons

Table 6: Best Networks

There were 600 neurons of each structure of different number of neurons. 4/600 were of 10 neurons, 3/600 were of 8 neurons, 1/600 of 6 neurons and 2/600 of 4 neurons. Thus even though the best network was of 4 neurons, the ones with 10 had a higher probability of being optimized networks. So the experiment suggests that Levenberg Marquardt is the best algorithm, and applying exhaustive search gave a good optimized network. The entity which played the greatest role in finding the global minima it seems is the random weight initialization, which leads to more surface traversal in the error space, therefore confirming the thought that no sound mathematical principle is there to design neural networks.

Another very interesting observation which was made was that out of the best 1997 neural networks which were created the distribution with respect to training algorithms was as follows,

Training Algorithm	**Network Quantity**
Gradient Descent with momentum	334
Quasi Newton BFGS	862
Levenberg Marquardt	801

Table 7: Best Networks with respect to training algorithms

The Networks with Gradient Descent with momentum had the following properties:

neurons	2	4	6	8	10
quantity	32	75	87	72	68

epochs	500	1000
quantity	177	157

learning rate	0.001	0.003	0.005	0.007	0.009
quantity	**68**	76	82	63	45

momentum	0.1	0.3	0.5	0.6	0.7	0.8
quantity	56	77	73	49	63	16

Table 8: Gradient Descent network attributes

For Gradient descent with momentum, the following deductions were made by inferring from this data

- Neurons should be between 4 and 8
- Epochs should be a little below 1000
- Learning rate should be between 0.003 and 0.006
- Momentum should be between 0.3 and 0.5

From Quasi Newton Algorithm, following data was extracted regarding the optimum number of neurons required to solve this problem

neurons	2	4	6	8	10
quantity	145	201	208	147	161

Table 9: Neuron quantity in networks with Quasi Newton training

This suggests that number of neurons should be somewhere between 4 and 7

neurons	2	4	6	8	10
quantity	115	187	195	168	136

Table 10: Neuron quantity in networks with Levenberg Marquardt training

Table 10 also suggests that number of neurons should be somewhere between 4 and 7.

The attributes which were kept constant in all neural networks are as follows:

1. Type of neural network used:- Feed Forward Network
2. Method used to check performance:- Mean Squared error
3. Transfer Function at hidden Layer:- Tan Sigmoid
4. Transfer Function at output layer:- Pure linear
5. Number of Layers:- One

The sole purpose of keeping these entities constant is that if we do not do that the exhaustive search will become too large and analyzing data would become very hard, thus making the problem at hand far more complex than it already is.

CONCLUSION

On the basis of the experiment conducted to reach a conclusion about the value of the three training algorithms, it is clear that Levenberg Marquardt is the best choice but there are certain things which the other two algorithms which might prove to be of great value. As was mentioned in the beginning, the Quasi Newton algorithm used to train the neural network in this experiment was assigned the default line search i.e. Charalambous line search, which searched in a given direction, maybe if given a different line search technique this method would perform better for after all in the trained set the number of Quasi Newton networks was greater than LM networks. The Gradient Descent with Momentum cannot be ignored for even though its values were not that promising, the data which was received after an exhaustive search explained a lot about the error surface, it seems that to optimize a given neural network or to find out an area in the error surface from where we should start, this would prove to be a good algorithm. Even though the data suggests that the optimum number of neurons lie between 4 and 7, the best neural network for QN and GDM had 10 neurons, while the ANN which incorporated LM had 4 neurons (the top 10 networks in LM class had more 10 neuron structures). Secondly a particular configuration in GDM had only 10 weight initializations, may be if given more initializations with the structure, which its data supports, it too can perform well.

POSSIBLE DIRECTIONS FOR FUTURE

The Levenberg Marquardt algorithm can also be used using a variant proposed in [7]. An Adaptive minimization algorithm can also be implemented so that terms which increase more slowly have a better chance of optimization [8]. Swarm Intelligence [9] or some other optimization technique like genetic algorithm can also be implemented. An intelligent system can be built which modifies on the basis of different input configurations of GDM algorithm.

REFERENCES

[1]Z Mustafa, Dr G Haider "Estimation Of Errors In Predictions For Laser Cutting Process" MS Computer Science Szabist 2010

[2] Atkinson, A. C. and Donev, A. N. and Tobias Optimum Experimental Designs, with SAS" Oxford University Press (2007). pp. 511

[3]Z Mustafa, B Saeed, N Yusoff, I Amin,I Haq "Laser Cutting problem Modeling using Statistical Tools" MS Computer Science Szabist 2010

[4] Ning Qian "On The Momentum Term In Gradient Descent Learning Algorithms" Neural Networks 12, 1999, 145-151

[5] S Russell, P Norvig "Artificial Intelligence: A Modern Approach, second edition" Prentice Hall (2003), p. 733

[6] Rumelhart, David; D Zipser, J L McClelland, "Parallel Distributed Processing" Vol. 1. MIT Press(1986), pp. 151–193
[7] M.H Khosravi, S Barghinia, P Ansarimehr "New Momentum Treatement For Levenberg Marquardt Neural Network Training Used In Short Term Load Forecasting", 21st International Power System Conference 2006, 1782-1788
[8]A Mostafa, S Yasir, "Machines That Learn From Hints", Scientific American Vol 272 issue 4 1995
[9]M Conforth, Yan Meng, "Reinforcement Learning For Neural Networks Using Swarm Intelligence",IEEE Swarm Intelligence Symposium 2008