I have the Answer but remind me what the Question was

Tapan P Bagchi[1]
Milan A Joshi[2]

Abstract

This paper presents a novel and effective technique based on Artificial Neural Net (ANN) technology to reverse map from process response to the input (control) variables when the input-response relationships are *nonlinear, complex or intractable by theory*. This is often the problem when the control variables in manufacturing or prototype development must be set such that the response hits a *specified target*. The ANN technique avoids countless empirical searches in the decision space and thus minimizes the expenditure on R&D or production resources. Additionally, this paper illustrates how one may build an ANN model with top-flight performance in Microsoft Excel® when a commercial neural net software is unavailable. The paper includes a learning-oriented reworking of a well-established example from the response surface literature. In conclusion, it indicates room for further research on training data collection methods.

Keywords: *Process modelling, Artificial neural networks, Connection weights, Optimization, Reverse mapping, Regression*

[1] *Indian Institute of Technology, Kharagpur, India*
[2] *NMIMS University, Shirpur, India*

1. Process Optimization using Empirical methods

This paper could be retitled, for it asks "I have a process response ('the answer') that is now sitting at its desired target value. But how did the engineer reach it? *What settings* of the design or process control variables did she use ('the question') to deliver the response on that target?" One must note that this is often a critical question in engineering design and process control.

Frequently, we run into situations where we wish to *control* a phenomenon that is observed as a response, and we have speculations about what factors might be *causing* it. It is often intended that we deliberately avoid the issue of establishing such causality. We assume that this has already been done by suitable experimentation (Montgomery, 2007; Bagchi, 2012; Gershenson, 2018). The goal presently is to drive the response to some desired target—by manipulating the enlisted causes or predictors. For this discussion, we assume that both - the set of predictors and the response - are all quantitative and measurable. This is the premise of Chapters 10 and 11, *Fitting Regression Models* (Montgomery, 2007)—premises which we unconditionally adopt.

Montgomery marshals the situation where optimization or process control is the focus. He explores settings when the *true* functional or dependency relationship between (a) y—the response—and (b) the predictors x_1, x_2, ...x_k is unknown. Montgomery subsequently expounds the machinery required to develop an empirical model relating the two, under some assumptions that make the approach discussed statistically valid. The data required to track such relationships may come from well-planned statistical experiments, or from unplanned experiments involving the observation of uncontrolled phenomena. The model proposed hinges on the hypothesized relationship between y and {xi}, assuming it to be a *linear* function of unknown model parameters β_0, β_1, β_2, ... β_k. With k regressor (independent or predictor) variables, the "response" model takes the form

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots \beta_k x_k \quad (1)$$

The terms "x_1", "x_2", etc. need not be the predictors directly; in general, these can be even nonlinear functions of $\{x_i\}$, such as $x_2x_3^2$, $\log(x_4)$, etc. Thus, this model allows the response y to be nonlinear functions of the regressors. But the model remains strictly linear with respect to the model parameters or regression coefficients $\{\beta_i\}$. This gives the name *linear regression* to this approach of empirical model building from the $\{x_i, y\}$ data.

Montgomery next presents an estimation of parameters and their statistical significance. A key aspect of regression modelling is hypothesis testing, which, when incorporated, allows one to use even randomly collected "input – response" data and to test statistically if the regressor-response dependency hypothesized as in model (1) is wholly or partially acceptable. In Chapter 11, Montgomery discusses a subsequent issue: the use of the "response model" thus developed to optimize the response. An example that we evoke (without its associated elaboration) from Montgomery is one in which reaction time (time) and reaction temperature (temp) are to be manipulated so as to maximize the yield of a certain process (Example 11-2, Montgomery, 2007). The data was collected by following a Central Composite Experimental Design (CCD), partially reproduced in Table 1.

The final regression model that Montgomery obtained is

$$\text{yield} = -1430.52285 + 7.80749\ \text{time} + 13.27053\ \text{temp} - 0.055050\ \text{time}^2 - 0.040050\ \text{temp}^2 + 0.010000\ \text{time temp} \quad (2)$$

We refer to Montgomery (2007) which expands the explanations and the procedure to obtain model (2).

Model (2) is a second order equation and it would be quite possible using standard methods now to optimize the process being studied. The goal would likely be to find the optimum time and temperature settings to maximize yield. Such exercises are numerous. In industrial processes in many domains where theory is unavailable or not sufficiently advanced, such empirical optimization is standard practice.

Table 1: Experimental data of Montgomery's CCD example*

Experiment #	Natural variables		CCD Coded variables		Observed response
	Reaction Time	Reaction Temperature	X1	X2	Y (yield)
1	80	170	-1	-1	76.5
2	80	180	-1	1	77
3	90	170	1	-1	78
4	90	180	1	1	79.5
5	85	175	0	0	79.9
6	85	175	0	0	80.3
7	85	175	0	0	80
8	85	175	0	0	79.7
9	85	175	0	0	79.8
10	92.07	175	1.414	0	78.4
11	77.93	175	-1.414	0	75.6
12	85	182.07	0	1.414	78.5
13	85	167.93	0	-1.414	77

*Example 11-2, Montgomery (2007), page 441

Suppose we reverse the question now. If we are given a target yield, can we trace back to the corresponding values used for the input variables? This is sometimes possible. Model (2) may allow us to develop contour plots or response surfaces when the number of independent process variables is small. This can thus localize the search for optimization. However, if this number is large or the dependency relationship is complex, this *reverse mapping* from a desired target response to the input variables would be infeasible. It might entail solving constrained equations that are intractable in reverse (Bagchi 2012). Luckily recent innovations can greatly help here. A similar situation with injection moulding is described by Manjunath, P G C and P Krishna (2012). This present study illustrates one such effective approach based on a *trained* Artificial Neural Network (ANN). Our focus here would be to test the effectiveness of a trained ANN to achieve such reverse *response-to-input* mapping.

2. Artificial Neural Networks (ANN)

Ingenious emulation (modelling) of the human mind by computer scientists have led to several types of constructs and intelligent behaviour of computer programs and such artefacts that, except spontaneous thinking, nervous systems in living organisms regularly carry out. These artefacts—called Artificial Neural Nets (ANN)—given some hardware and logic, can sense, process, learn and activate internal and external actions. Intelligence is one's ability to acquire and apply knowledge and skills. Wikipedia expands this statement as one's capacity for logic, understanding, self-awareness, learning, emotional knowledge, planning, creativity, and problem-solving. Not all of this is entirely possible yet for machines. Still, a lot is now possible where smart (not just automatic) actions can be expected of machines without human intervention. This section tackles an effort-intensive problem from the manufacturing world and shows with a completely worked out example of how the task can be effectively delegated to a trained intelligent machine, here a neural net. Artificial neural networks have been shown to be able to reproduce the behaviour of complex and nonlinear relationships between a set of input factors and another set of dependent responses to an arbitrary degree of closeness, often much better than classical dependency modelling methods (Hornik, Tinchdombe and White, 1989). We briefly recall the basics of ANN's design, training and use in a situation that would otherwise require in-depth engineering and computational exercise and sometimes intractable analysis—every time a new production technology is to be evaluated for its utility, for example. We also describe the steps in doing this in embedded technical notes. One special facility we extend to the reader: In order to keep mysterious "black boxes" doing the needed computations behind the scene at the minimum, we do all our work using Microsoft Excel®. To aid the learner, we purposely show here the steps to build simple yet very effective ANNs in Microsoft Excel®. See also Choong (2009) and Kendrick, Mercado and Amman (2006).

2.1 ANN—why study them?

Advanced manufacturing technologies such as EDM (Electrical Discharge Machining), CNC (Computer Numerical Control) and robotics in mechanical industries have made it possible for us to produce top-class products, often in a shorter time and at a lower cost—but the production processes are now more complex. The same challenge exists in fabricating VLSI (Very Large-Scale-Integration) or memory chips, processing pharmaceuticals, or in alloy steel production. Frequently we wish to optimize such processes; however, traditional analytical methods aren't always suitable to manipulate or model intricate manufacturing processes—to help control or improve them. Process variables today are many and the relationships in the problem may be *nonlinear*. Textbook theory also often reaches its limit and one then resorts to empirical approaches, the notable exception being electrical and electronic sciences. Artificial neural networks sometimes provide an effective way in such cases and a number of its successful implementations have been seen in the past two decades. Deep learning, a sophisticated empirical modelling and decision making approach, is an advanced form of the ANN technology to help recognize faces, script, and voice, and we may not be aware, but banks now regularly sift each of our credit card transactions through neural nets to check fraud.

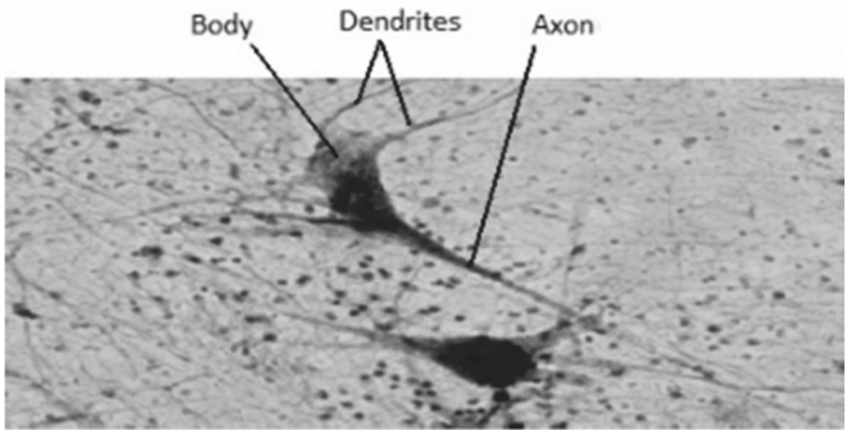


Figure 1: *A natural neuron showing its body (soma), dendrites that connect it to other neurons and axon which carries the activation signal to muscles*

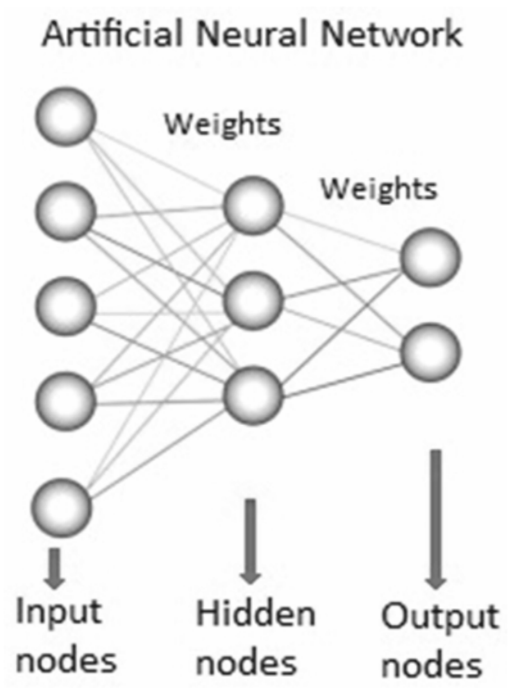


Figure 2: *The Input-Hidden nodes-Output Architecture of a typical ANN*

Ostensibly, this is an area that may have a high potential to help optimize or improve many complex *manufacturing processes* also. To set the stage, this paper outlines the steps in building ANN models in general and a production system example for illustration, in particular, to replace or simplify much of that slog. For the uninitiated, we include discussions on ANN architecture, training the selected net, testing it, and then using it for prediction of the production system's performance. The goal would be to raise the system's productivity, the quality of its output, or cut unit cost, etc.

The human brain learns, remembers, decides and activates the right muscles or organs to let us live. It has approximately 100 billion neurons, which communicate through electro-chemical signals. The neurons are connected through junctions called synapses. Each neuron receives thousands of connections with other neurons, constantly receiving incoming signals to reach the cell body (Figure 1). If the resulting sum of the signals surpasses a certain threshold, a response is sent through the axon. The ANN attempts to recreate the computational mirror of the biological neural network, although it is not comparable since the number and complexity of neurons and the connections used in a biological neural network is significantly greater than those in an ANN. Good introductions to ANN are given by Gershenson (2018) and Tutorials Point (2018).

Character and image recognition and natural language processing are two areas in which ANN has made unbelievable strides already. Both use a new approach called *Deep Learning*. ANN now automatically translates English into Greek and guides physicians in diagnostics. Manufacturing system design, scheduling, interpreting non-random behaviour of a process, substitution for complex statistical experiments and needless runs of simulation models to discover optimum process conditions are instances where perhaps our own intelligence and ingenuity to exploit ANN has become wanting. Like many other similar tools and artefacts, ANN empowers analytics. Thus, ANN is rapidly knocking out and moving ahead of conventional problem-solving.

ANN comprises a network of *artificial* neurons (also known as "**nodes**"). By programming, these nodes are connected to each other, and by training the strength ("**weight**") of their connections to one another, is assigned a value that may *inhibit* (minimum often being -1.0) or *excite* (maximum being say +1.0) the receiver. If the value of this weight is high, then it indicates a strong connection. Further, within each node's design, a transfer or **activation function** is built in—to help it to act on nonlinearities in input-output relationships. The nodes are of three types—**input**, **hidden** and **output**. Presently we consider only feed forward ANN.

2.2 How does learning occur?

Figure 2 indicates the flow of information in the ANN. The input nodes take in information in a form which can be numerically expressed. The information is presented as **activation values**, where each node is given a number; the higher the number, the greater the activation. This information is then passed throughout the network. Based on the connection strengths (weights), inhibition or excitation, and transfer functions, the activation value is passed from node to node. Each node sums the activation values it receives; it then modifies the value based on its transfer function. The activation flows through the network, through hidden layers, until it reaches the output nodes. The output nodes then reflect the input in a meaningful way to the outside world.

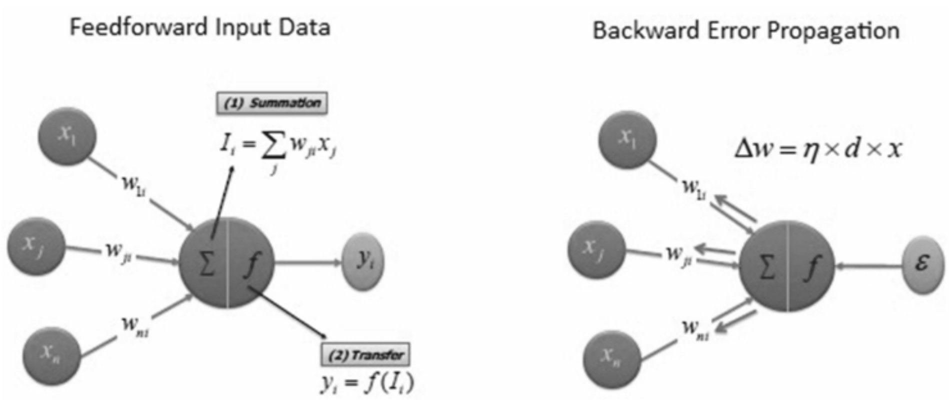


Figure 3: *The forward flow of input data and the backward propagation of error*

During training, the *dierence* between predicted output value and the actual output value (the **error**) will be propagated backward by apportioning them to each node's weights according to the amount of this error the node is responsible for. The gradient descent algorithm may be used here. Figure 3 shows the forward flow of input information and the backward flow of the error in the predicted value. A key job in training the ANN is to optimize the weights between their connections to have a minimum error in its prediction.

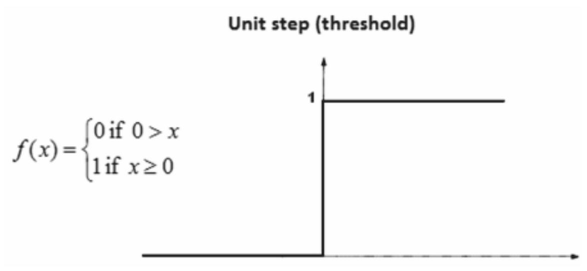
An alternative to using the back propagation procedure is to use the conjugate-gradient optimization method to globally optimize the weights. This method is built into Microsoft Excel® Solver® as the GRG (Generalized Reduced Gradient) nonlinear macro. In the present use of building an ANN model in Microsoft Excel® we used this built-in optimizer to train the neural net to determine the optimum weights. Choong (2009) has also demonstrated the effectiveness of Microsoft Excel®'s GRG nonlinear macro.

2.3 Transfer (Activation) Functions

It is the difference in their transfer (**activation**) functions that makes one ANN different from another, changing the way they internally process nonlinear information and help learning. The transfer function translates the input signals to output signals. Activation functions commonly used are Unit step (threshold), sigmoid, piecewise linear, tanh, ReLU and Gaussian. Activation functions introduce non-linearity in the network so it would be capable of learning complex relationships between the input and the output.

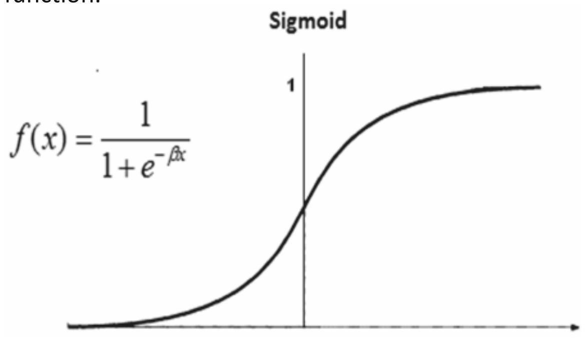
Unit step (threshold)

The output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value.



Sigmoid or Logistical Sigmoid

The Sigmoid function consists of two functions, logistic and tangential. The values of the logistic function range from 0 to 1 and -1 to +1 for tangential function. It can handle a broad range of positive input numbers all the way to infinity and squash the calculation into an output between 0 and 1. This is sometimes required because we need decision making systems that are capable of reasoning within uncertain environments. Linearity is not how the real world always works. This makes the Sigmoid the most common activation function.



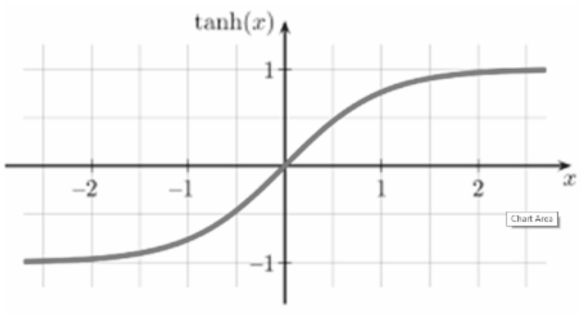
Piecewise Linear

The output is proportional to the total weighted output. As may be noted, it does minimal calculations except squashing large inputs into ±1.

Tanh

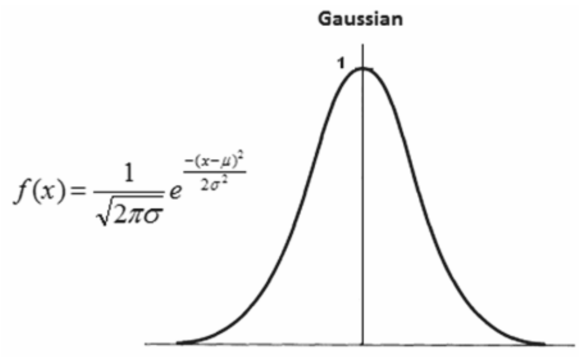
Though the logistic Sigmoid has a nice biological interpretation, it turns out that the logistic Sigmoid can cause a neural network to get "stuck" during training. An alternative to the logistic Sigmoid is the hyperbolic tangent or Tanh function. Like the logistic Sigmoid, the Tanh function is also Sigmoidal ("s"-shaped), but instead outputs values that range (±1). This function thus squashes its output into a range of numbers between -1 and 1, a much more flexible approach to consider because it accounts for both positive and negative values. Tanh remains a favourite function to use while one designs neural networks. We have used it in the present work.

$$f(z)=\tanh(z)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}},$$



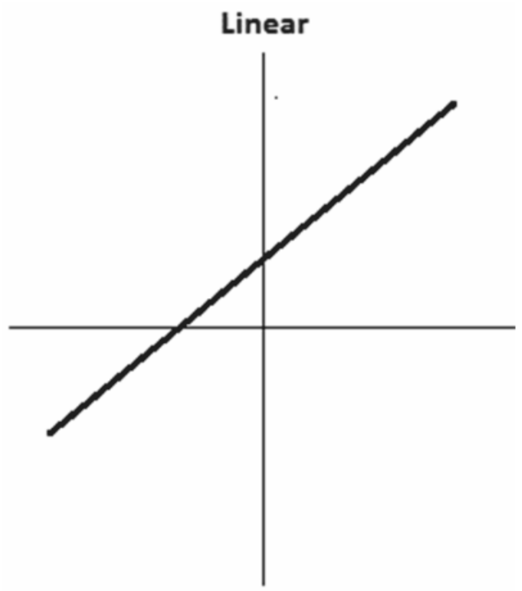
Gaussian

Gaussian functions are bell-shaped curves and continuous. The node output (high/low) is interpreted in terms of class membership (1/0), depending on how close the net input is to a chosen value of average.



Linear

Like a linear regression, a linear activation function transforms the weighted sum inputs of the neuron to an output using a linear function. It passes the input forward without any change in it.



3. How do we train a Neural Net? We use Algorithms to adapt weights

Learning is well known to be a process. Neural nets can be configured in manners that adapt to the situation at hand so as to minimize their prediction errors. There are dierent types of neural networks, but they are generally classied into *feed-forward* and *feed-back* networks.

A **feed-forward network** is a non-recurrent network which contains inputs, outputs, and hidden layers; in it the signals can only travel in one direction. Input data is passed onto a layer of processing elements where it performs calculations. Each processing element makes its computation based upon a weighted sum of its inputs and its activation function. The new calculated values then become the new input values that feed the next layer. This process continues until it has gone through all the layers and determines the ANN's output. A threshold transfer function is sometimes used to quantify the output of a neuron in the output layer. Often used in data mining, feed-forward networks include Perceptron (linear and non-linear) and Radial Basis Function networks. The present study uses a feed forward ANN.

A **feed-back network** has feed-back paths meaning they can have signals traveling in both directions using loops. All possible connections between neurons are allowed. Since loops are present in this type of network, it becomes a non-linear dynamic system which changes continuously until it reaches a state of equilibrium. Feed-back networks are often used in associative memories and optimization problems where the network looks for the best arrangement of interconnected factors.

3.1 Neural networks for modelling manufacturing processes

ANNs have been used by several researchers in manufacturing. A signicant application of ANNs in manufacturing, to judge by the volume of publications, is in the area of process monitoring and control, where ANNs have provided an alternative to the traditional Statistical Quality Control (SQC) charting methods. ANN is used here to identify the appearance of ''special causes'' (unnatural process faults which may adversely affect quality) by recognizing abnormal patterns in the process outputs. Typically, using this approach, the network is trained (using process outputs as its inputs) to recognize specic abnormal patterns and associate them with specied fault conditions. The trained network is then used to monitor the process outputs. It is expected that once trained, the network will respond with an output indicating the particular type of special cause, when a similar pattern reappears in the process outputs. Zorriassatine and Tannock (1998) provide a review of this approach.

The use of ANN for process modelling must be distinguished from their use for process monitoring and control. When modelling the process, the network is trained using process parameters (e.g., process settings or in-process measurements) as its inputs, and process response/outputs (e.g., quality characteristics) as the network outputs. The intention is that the network should behave exactly like the process in respect to its response to parameters and conditions, hence providing a model of the process, which can then be used for experimentation and process optimization. Su and Hsieh (1998) compare the optimization of a semiconductor manufacturing process by using Taguchi's approach with the ANN. Coit, Jackson and Smith (1998) suggest some practical aspects of constructing and validating ANNs for manufacturing process modelling. Cook, Ragsdale and Major (2000) combined ANNs together with genetic algorithms to model and optimize a critical strength parameter in a particleboard manufacturing process. Tong, Lee-Ing and Hsieh (2000) optimized multiple quality response characteristics (both qualitative and quantitative) in IC manufacturing using ANNs. A noted benefit is that some of these modelling applications allow experiments to be conducted directly with the trained ANN model to explore planned what-if conditions, instead of manipulating the real process.

Substantial progress has been made in ANN training strategies even though we are still learning the use of this new tool. But according to a survey of business applications by Vellido, Lisboa and Vaughan (1999), a disadvantage of ANN for process modelling is the lack of guidance and background that still prevails in selecting ANN architectures. Various techniques to help developers select the optimum ANN topology are periodically reported. For example, Williamson (1995) used Genetic Algorithms for selecting the optimal ANN topology while Khaw, Lim, and Lim (1995), Macleod, Dror and Maxwell (1999), and Lin and Tseng (2000) preferred Taguchi experimental design methods. Today deep learning—immensely effective in image and voice recognition—stands at the frontier of this technology (Patil et al 2019).

4. Training neural networks for the modelling of complex processes

This section outlines the steps for the training of ANNs to model complex industrial processes. Often the decision to utilize the ANN methodology is prompted by the *complexity* of the process being studied, for no coherent link between the several inputs and the outputs may be apparent. Complex processes are considered to be those where there are multiple process parameters and/or environmental conditions which are thought to affect process behaviour. There may also be multiple process responses/outputs, and a full theoretical understanding of the process operation may be missing.

The five steps typically used in training the ANN are:

- process parameter and process response identification,
- training data collection,
- training and testing set preparation,
- training and testing the network,
- training using hints

4.1 Process parameters and process response identification

The idea of using ANNs to model the processes is to create networks that take process parameters as inputs and produce process responses (such as quality, cycle time, yield, etc.) as outputs. One way to do this is to assign all available process parameters as network inputs, and then let the network adjust itself during training so that the connection of any insignificant process parameters becomes weak. Another approach is to be more selective—and use prior experience or knowledge to assign as inputs only those parameters that are believed to influence the process outputs. The first approach has been termed the ''global network'' while the second is called the ''focused network'' (Wilcox and Wright 1998). These authors showed that, when modelling the same process, the focused network performed better than the global one, suggesting that process parameters should be carefully selected to improve the performance of a network. However, one must note that for either approach, if a significant process parameter input is missing, ANN's performance will be compromised, as the variation in the available input data will not be sufficient to explain the variation in the process output characteristics. This is what makes the selection of input factors critical.

Process response used for ANN training should be an appropriate measurement of the successful value-add to the process in focus. The selection of a suitable response requires an understanding of the process and what it lacks. The ANN, one may further note, can be trained to predict *both* a single response and multiple responses—with only marginal added effort.

4.2 Training data collection

Training an ANN usually requires a substantial amount of good data, of which one part will be used for training and the rest will be used for testing the goodness of the trained network. Having selected the significant process parameters and assigned them to network inputs, the next key step is to acquire the required process parameter (the control factors) and response (output) data for training. It is important that the parameter data be correctly associated with the *consequent* process output data. Hence, data collection is an important step to ensure the sufficiency and integrity of the data used to train the network, as the network performance can only be as good as the training data.

It is not possible to say how many data items are appropriate, because this depends on the complexity of the process modelling problem. Generally, the training data should be representative of the *entire population* of data items, unless there is a good reason to resort to stratification or data blocking.

The proportion of training in testing data has varied considerably in the published research. For example, Nascimento, Guidici and Guardani (2000) vary the ratio from 1:1 to 3:1; Tong, Lee-Ing and Kun-Lin Hsieh (2000) use 2:1 and Coit, Jackson and Smith (1998) use 4:1. Training and testing data can be acquired in a number of ways, as described in the following paragraphs. Note that Josh and Shah (2019) provide a sampling procedure to help collect training data.

4.2.1 Simulated data

When large amounts of data are not available from the actual manufacturing process, simulated data can be substituted. Simulated data includes data prepared from statistical models (such as the normal distribution), Monte Carlo runs and computational or numerical simulations (such as finite-element analysis methods, or the ball bearing technology selection example of Bagchi (2012)). Networks trained using this type of data can be used to capture information from a process model and to eventually *replace* the numerical, quasi-theoretical, or even a simulation model's equations. The trained network might then be used to estimate optimal settings for process parameters by conducting a grid search, Taguchi-type experiments, hill-climbing, etc. in the region of interest. (We shall attempt this optimization using Microsoft Excel® Solver® and the data in Table 1.)

Still, one gains several advantages of using simulated data. First, simulated data can be noise-free (unless one intends noise to be part of the model). The random uncontrollable variation, which affects real process data, is thus eliminated. Hence, it should be easier to train the neural network with this data. Second, it is usually easier, cheaper and faster to train a network, and run an experiment, using simulated data. However, the simulated data may have a serious drawback, that is the statistical or computational models providing the data might itself not be a good fit to the real production process, which might lead to inaccurate results from the trained ANN when using real process data to predict the output. To reduce this effect, several researchers have combined simulated data with some actual process data for training.

4.2.2 Actual process data

Raw process data may also be used for training —whenever available, for this is coming straight from the "voice" of the actual process! Many manufacturing companies already have available both process parameter data and quality inspection/product test data deposited in their ERP systems or on the process engineer's Microsoft Excel® sheets. In order to use them for ANN training, it must be possible to relate the two for an individual product—in other words, here *product traceability* is required. This is frequently the case, for example, in the aerospace and silicon wafer industries where good data logging systems exist. However, these data might be collected as 'quality control' records rather than for use in process modelling, and hence, the process parameters set might not cover the full range possible (a necessity for training) and might not include the optimum conditions. Also, the data collection and storage may not be tightly controlled; this will result in data integrity problems. However, such historical data reflect a wide variety in real process conditions and if they are available, no further data collection cost will be incurred. The authors have experienced that this is generally possible in factories.

If large amounts of raw data are available, using all of it might be unnecessary and time-consuming. Random selection of the required training and testing sets from the available data can be employed, but it is important that the selected data cover the *entire* population of interest—evenly—to be representative of the domain of the process. (We visit this point in our example below.)

4.2.3 Process data collected from designed experiments

Data in this category are actual process data that are not collected directly from the normal production conditions but are obtained from Designed Statistical Experiments (DOE) conducted on the process, often using the Taguchi approach (Rajagopalan and Rajagopalan, 1996). The investigator, in this case, has strategic control over data collection, and since unwanted effects can be blocked in these controlled experiments, unwanted noise effects are reduced. Generally, data from designed experiments are more carefully collected in a controlled environment. Furthermore, literature suggests that well-designed experiments such as the Central Composite Designs (CCD), described in Montgomery (2007), should be used here. CCD should be able to cover a wider range of process parameters, which should target to include second order nonlinearities and factor effect interactions and possible, yet unknown optimum factor setting within those regions. The undesirable aspects of DOE data are that it is costly to collect, often confusing to technicians and might still not reflect real process conditions. For the most part, CCD attempts to estimate only up to second order nonlinearity—one should remember! If you have resources, go for higher resolution experiments.

In this study, for illustration's sake, we used the experimental data provided by Montgomery (2007) obtained in a CCD exercise cited. Note however, that the CCD used by Montgomery (2007) has a big inherent limitation. The sampling it does is good only for building a *second order linear regression model*, not necessarily a near exact sampling of the real response that might have higher order terms and factor effect interactions.

4.3 Training and testing data set preparation

The ANN can only work with data within certain ranges and in specified formats. Hence, the data usually have to be pre-processed before being presented to the network. Studies by Su and Hsieh (1998) provide guidance for the preparation of the process data as follows.

It is essential to check **data integrity**, especially when using raw process data collected from a quality control database. Errors such as incorrectly entered data, duplicate and missing data must be corrected, for the quality of the network training can only be as good as the quality of the training data set, especially in medical work.

When used, the Sigmoidal activation function should receive input in the range ±5. If the input is out of the acceptable range, it might cause neurons to saturate and stop learning. *Data scaling* ensures that each input contributes to the same proportion to the adaptation of network weights during training. Without scaling, an input that varies (say) from 10,000 to 100,000 could have a significantly greater effect than the one that only varies between 1 and 10. *Data coding* can also be used to reduce the effect of noise in the data, before it is presented to the network, thus allowing training convergence to occur more readily.

5. Building simple neural networks by Microsoft Excel®

For learning to play with ANN, it is very much possible to use Microsoft Excel® to build simple neural networks. We show that in this section. The goal of this section is to let you dip your toes in this new and powerful approach to quantify existing and known cause-effect relationships. But working with ANN is not a statistical procedure whereby you may establish *causality*. Still, you will be able to model here unknown complex nonlinear relationships between the observed values of a few independent (control) variables and some dependent variables (responses) using the ubiquitous tool— Microsoft Excel®, and a problem—to explore ANNs with. As you go through this paper, you may wish to try the steps hands-on. We provide you the data in this exercise, reminding you only that the procedure is empirical and the starting point will be to have with you a representative sample of *actual input-output process data*, as in Table 1.

We used Montgomery's CCD data—displayed in Table 1—to build a simple 2-9-1 ANN with its architecture comprising two neurons to receive two independent inputs (time and temperature, appropriately normalized), a hidden layer with nine neurons (nodes), and a single neuron in the third (output) layer to deliver the yield. The network we shall presently build will not attempt classification; rather it will receive numerical data as input, process it using *weighted connections* and then process the resulting information using selected *activity functions*—to deliver a numerical output. It will train itself by using a set of input-response data pairs (the training data). The weights will be optimized by Microsoft Excel®'s built-in Solver® macro to produce answers as close to the actual outputs provided as possible.

Upon training, Microsoft Excel® will display the actual responses as fed for training it, alongside the net's predicted (computed) responses for the input data entered. The difference between the net's predicted response and the actual response is the *error*. The net's weights will be optimized by the conjugate-gradient based GRG macro, by minimizing the *square* of these errors. Each of these steps can be coded in Microsoft Excel® in a relatively straightforward manner as shown in this section.

Some general aspects of neural nets are to be recalled. ANNs are typically *non-linear*, with the power to capture relationships much more complex than second order regression equations. It has been shown that even simple ANNs are able to model or reproduce almost *any* nonlinear function (the "responses" {**y**}) of independent variables {**x**} to an arbitrary degree of accuracy, provided the ANN has been properly trained (Rocca, 2018). This performance is dependent on the array of neurons used in its several layers, architecturally how they are connected, the activation functions used to emulate the nonlinearity, and the training strategy.

In advanced ANNs, the number of hidden layers is increased and such ANNs are used not for simple prediction, but for what is called *deep learning*, a feature particularly useful in image recognition and other advanced applications of neural nets. Deep learning uses include zip code recognition from a handwritten script and voice translation. Still, it is well recognized that a biological neural network is far bigger and considerably more complex than today's man-made computerized neural networks.

As for training strategies, learning can be *supervisory* when the ANN is shown the correct output {**y**} for each specified member in the array {**x**} of input and it optimizes its connecting weights. Such ANNs are used generally for function approximation, prediction and forecasting.

Some ANNs can learn on their own—given input-output examples repeatedly. Learning, in this case, is *unsupervised*, often employed by classifier ANNs. Presently, however, we shall use supervisory learning to reproduce nonlinear relationships between a set of numerical input variables and numerical responses.

5.1 The Task at Hand

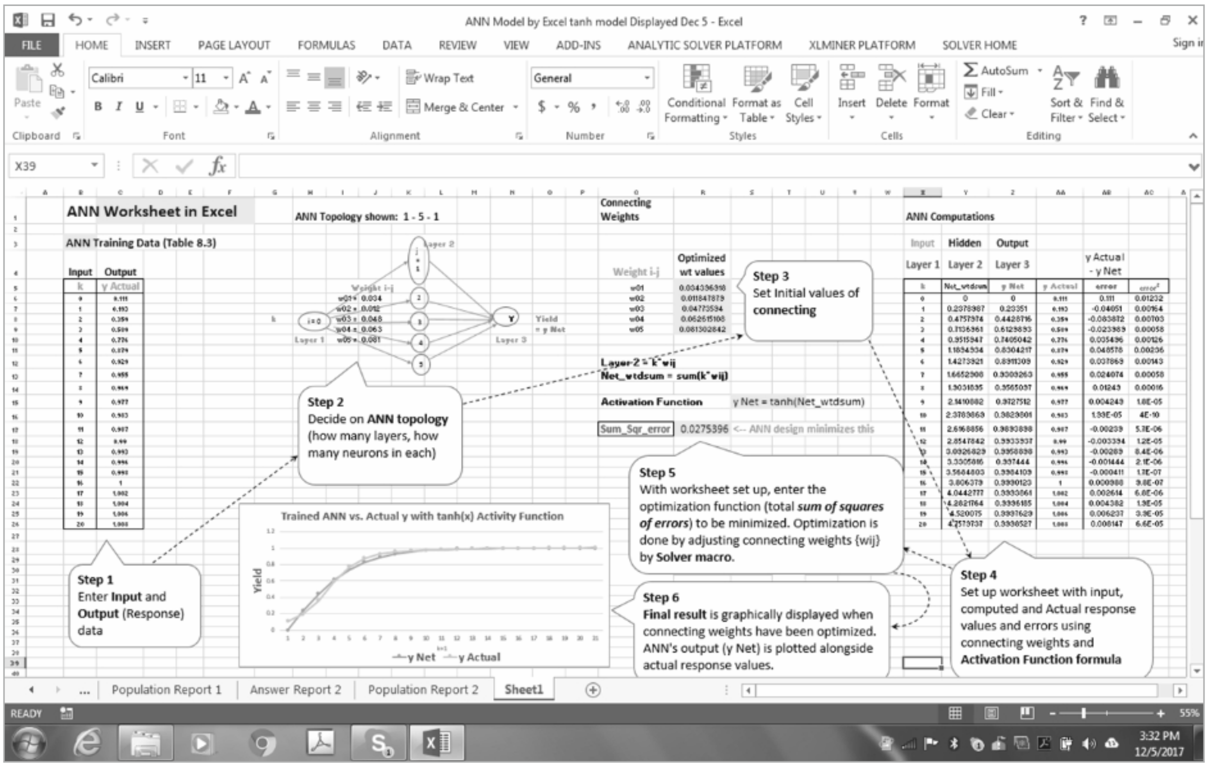
A representative complex set of interdependent time-temperature-yield observations were experimentally produced as quoted by Montgomery (2007), page 442. The analytical solution to relate this interdependent set of data was not deemed feasible nor possible to produce an expression like *yield* = *f*(*process* and *design factors*). But following the principles of statistical design of experiments, the pertinent data values capturing such interdependency could be experimentally obtained by the judicious setting of the control factor level combinations and the resulting yield observed. Montgomery's experimental design was central composite or CCD, quite effective in generating a *second order response surface* for the process under investigation. This is how the training data (Table 1) for this illustration of using Microsoft Excel® for ANN development was generated. At the juncture cited by Montgomery, the investigators wished only to be able to *predict* yield given a set of specified input factor values of interest, not by running the manufacturing process physically in a search mode under various conditions to find the yield numbers. Their goal was to explore the process and design conditions that would lead consistently to high yield, not insisting to obtain the *best ever possible* yield for this chemical process.

5.2 The issue of a small number of observations (instances) in the Training Set

As with statistical inference, larger the sample's size, the more precise will be the estimates. This principle applies to build multi-factor linear regression models as well (Montgomery 2007). Two issues are prominent in the strategy to collect data for model building in statistics. The first is the quality of the estimated model parameters. The second is the capture of nonlinearities in the input-response relationship, generally modelled using factor interaction, and the use of higher degree terms in regression. Theory in this regard is well developed. Being a young domain, regression model building based on ANN does not yet have a comparable rooting in theory, though due to the availability of powerful processors and large inexpensive memory, kernels and other methods have been devised that enable it to capture nonlinearities with amazing accuracy and precision (Rocca, 2018; Bagchi, 2013). The ANN technique also infers ("learns") the input-response dependency by estimating its own model parameters—the connection weights {wij} between the neuron layers (Figure 3). However, even with the computational machinery at hand, frequently one is unable to find a sufficiently large set on input-response instances. As in statistical inference, one recourse here is bootstrapping (Paass, 1993). As an analog to the statistical notion of *bootstrapping*—drawing many secondary samples with replacement from a primary sample of data of limited size to produce empirical estimates of parameters of interest of the population—is often feasible (Efron, 1979).

Paass (1993) discusses how the same may be done to train an ANN. It regards the ANN to be a nonlinear or nonparametric regression model which defines the relation between vectors **X** and **Y** of input and output variables. The procedure he uses generates the training dataset by repeated sampling of the primary dataset on a limited quantity of input-response instances whose effectiveness he demonstrates. Several other studies have now used bootstrapping to develop training data in machine learning (Neumann and Schiller, 2000). Paass (1993) asserts that the approach is valid for a large number of linear, nonlinear and even nonparametric regression problems. He states that bootstrapping has the potential to model the distribution of estimators (for ANN the connection weights) to a higher precision than the usual normal asymptotics based on the original data.

Some critical observations are now in order. To achieve good fidelity, since model building in ANN is empirical, experts urge that ANN's training dataset must attempt to span its entire range of interest of input conditions in which the ANN will be used for prediction. The *extent* of such spanning, if that primary data is to be used for bootstrapping, must approach this condition. In this study, we chose to use Table 1, extracted from Montgomery's cited CCD experiments. Note that the data collection procedure used (the CCD experiment) had a different goal—by statistical design, it aimed to aid the estimation of only a *second order statistical regression model*, the likeness being model (2). For higher order models, hopefully, to deliver a regression model of superior predictive capability than (2), the domain spanned by the process control (i.e., input) variables would be sampled extensively, using observation points suitably spread and scattered in the decision space (Box, Hunter and Hunter, 1978).


ANN Model by Excel tanh model Displayed Dec 5 - Excel
FILE HOME INSERT PAGE LAYOUT FORMULAS DATA REVIEW VIEW ADD-INS ANALYTIC SOLVER PLATFORM XLMINER PLATFORM SOLVER HOME
X39
ANN Topology shown: 1 - 5 - 1
Layer 1
Layer 2
Layer 3
Weight i-j
w01 = 0.034
w02 = 0.012
w03 = 0.048
w04 = 0.063
w05 = 0.081
Yield = y Net
Trained ANN vs. Actual y with tanh(x) Activity Function
Yield
y Net
y Actual
Population Report 1
Answer Report 2
Population Report 2
Sheet1
READY
3:32 PM 12/5/2017

ANN Worksheet in Excel

ANN Training Data (Table 8.3)

Input	Output
k	y Actual
0	0.111
1	0.193
2	0.359
3	0.589
4	0.776
5	0.879
6	0.929
7	0.955
8	0.969
9	0.977
10	0.983
11	0.987
12	0.99
13	0.993
14	0.996
15	0.998
16	1
17	1.002
18	1.004
19	1.006
20	1.008

Connecting Weights

Weight i-j	Optimized wt values
w01	0.034336318
w02	0.011847879
w03	0.04773594
w04	0.062615108
w05	0.081302842

Layer-2 = k*wij
Net_wtdsum = sum(k*wij)

Activation Function y Net = tanh(Net_wtdsum)

Sum_Sqr_error 0.0275396 <-- ANN design minimizes this

ANN Computations

Input Layer 1	Hidden Layer 2	Output Layer 3		y Actual - y Net	
k	Net_wtdsum	y Net	y Actual	error	error²
0	0	0	0.111	0.111	0.01232
1	0.2378987	0.23351	0.193	-0.04051	0.00164
2	0.4757974	0.4428716	0.359	-0.083872	0.00703
3	0.7136961	0.6129893	0.589	-0.023989	0.00058
4	0.9515947	0.7405042	0.776	0.035496	0.00126
5	1.1894934	0.8304217	0.879	0.048578	0.00236
6	1.4273921	0.8911309	0.929	0.037869	0.00143
7	1.6652908	0.9309263	0.955	0.024074	0.00058
8	1.9031895	0.9565037	0.969	0.01243	0.00016
9	2.1410882	0.9727512	0.977	0.004249	1.8E-05
10	2.3789869	0.9829801	0.983	1.99E-05	4E-10
11	2.6168856	0.9893898	0.987	-0.00239	5.7E-06
12	2.8547842	0.9933937	0.99	-0.003394	1.2E-05
13	3.0926829	0.9958838	0.993	-0.00289	8.4E-06
14	3.3305816	0.997444	0.996	-0.001444	2.1E-06
15	3.5684803	0.9984109	0.998	-0.000411	1.7E-07
16	3.806379	0.9990120	1	0.000988	9.8E-07
17	4.0442777	0.9993861	1.002	0.002614	6.8E-06
18	4.2821764	0.9996185	1.004	0.004382	1.9E-05
19	4.520075	0.9997629	1.006	0.006237	3.9E-05
20	4.7579737	0.9998527	1.008	0.008147	6.6E-05

Step 1
Enter **Input** and **Output** (Response) data

Step 2
Decide on **ANN topology** (how many layers, how many neurons in each)

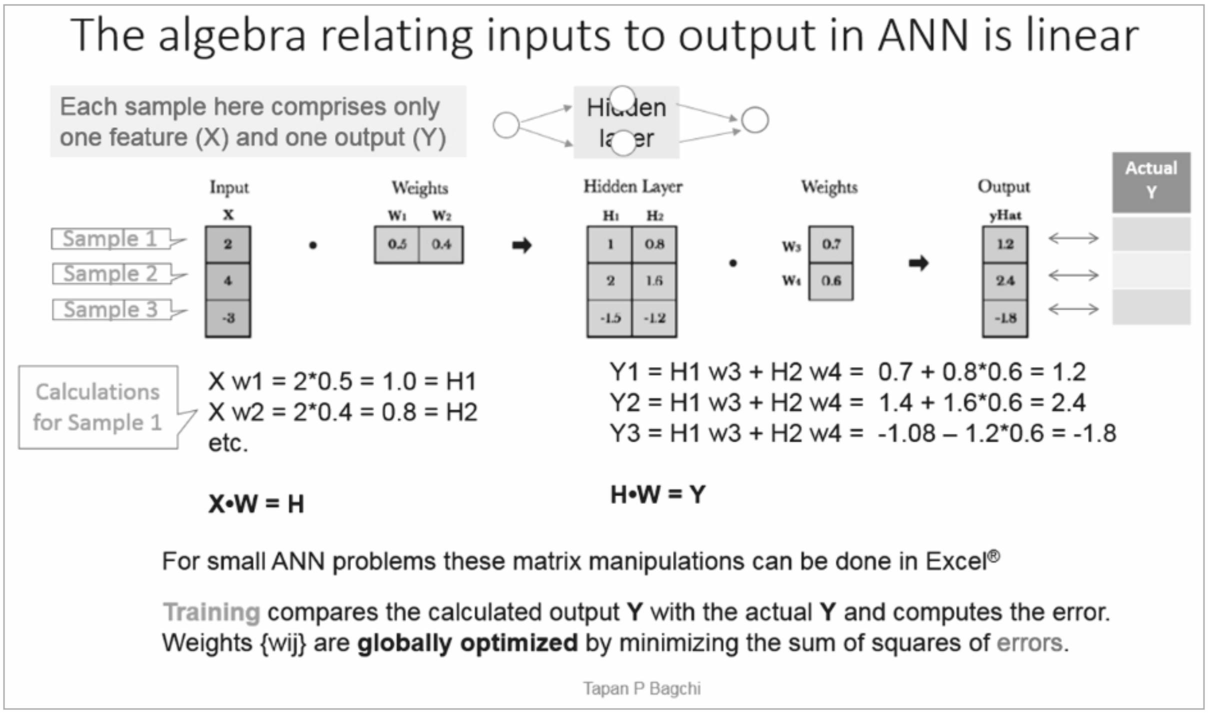
Step 3
Set Initial values of **connecting**

Step 4
Set up worksheet with input, computed and Actual response values and errors using connecting weights and **Activation Function formula**

Step 5
With worksheet set up, enter the optimization function (total ***sum of squares of errors***) to be minimized. Optimization is done by adjusting connecting weights {wij} by **Solver macro.**

Step 6
Final result is graphically displayed when connecting weights have been optimized. ANN's output (y Net) is plotted alongside actual response values.

Figure 4: *Overview of the typical steps in ANN building by Microsoft Excel® employed by authors*



The algebra relating inputs to output in ANN is linear

Each sample here comprises only one feature (X) and one output (Y)

Input X		Weights W1	Weights W2		Hidden Layer H1	Hidden Layer H2		Weights		Output yHat
2	•	0.5	0.4	➡	1	0.8	•	W3: 0.7	➡	1.2
4					2	1.6		W4: 0.6		2.4
-3					-1.5	-1.2				-1.8

X w1 = 2*0.5 = 1.0 = H1
X w2 = 2*0.4 = 0.8 = H2
etc.

X•W = H

Y1 = H1 w3 + H2 w4 = 0.7 + 0.8*0.6 = 1.2
Y2 = H1 w3 + H2 w4 = 1.4 + 1.6*0.6 = 2.4
Y3 = H1 w3 + H2 w4 = -1.08 – 1.2*0.6 = -1.8

H•W = Y

For small ANN problems these matrix manipulations can be done in Excel®

Training compares the calculated output **Y** with the actual **Y** and computes the error.
Weights {wij} are **globally optimized** by minimizing the sum of squares of errors.

Tapan P Bagchi

Figure 5: *The inside view of data manipulation in ANN to develop output from input. Computed H and yHat are reformed by activation Φ(.) to seize nonlinearities. This scheme was implemented by the authors.*

The limited hypothesis that led to the empirical estimation of model (2) speculated that a second order response would be sufficient to capture nonlinearities due to reaction time and reaction temperature for the optimization being attempted (by RSM, Ch 11, Montgomery 2007), and also that the effect of any *other* factors would be negligible. We wish to point out that the 13 CCD data instances observed (Table 1) do not all reveal identical factor effect information. Experiments 1 to 5, and 10 to 13 manifest the effect on yield of varying time and temperature, and the effect of any *uncontrolled* “input” variables, whereas replicated experiments 5 to 9 with time and temperature held constant reveal the effect primarily of the *uncontrolled* variables.

All the 13 observations also reflect any nonlinearity in the input-response relationship. In this study, we intended to use these very 13 observations to build an ANN and compare its predictive quality with model (2). No other data acquisition strategy would be used (we couldn't do any more experiments!): The ANN to be built would use *all, and only*, these 13 input-output observations as our primary data. The only extra step we took, by invoking the work of Paass (1993), was to *bootstrap* Table 1 by a simple procedure—we computationally bootstrapped the 13 data points seven times to generate a training dataset with 104 observations—13 primary and 91 secondary (Paass 1993). Subsequently, as the goal of this study, if the resulting ANN would evidence good fidelity, we would explore the possibility of testing the idea of '*reverse mapping*' with it.

The typical steps for building the ANN by Microsoft Excel®—only for illustrative purpose—are displayed in Figure 4. (This figure displays a model built to predict yield in a ball bearing manufacturing process.)

Most of the ANN-related data were kept in Microsoft Excel® in matrix form and manipulated using built-in functions including TRANSPOSE and MMULT. The actual data manipulations followed operations of linear algebra, a glimpse of which is given in Figure 5.

The connection weights were globally optimized by Microsoft Excel® Solver® by minimizing the sum of squared errors—the difference between the predicted and the actual response values noted in Table 1. We note in passing that subsequent development of software tools based on programming in Python have created extensive matrix manipulation libraries of tools, the popular ones being TensorFlow® and Keras®. These reduce the developer's coding a great deal.

Table 2: *Comparison of observed yield and the ANN's predictions as obtained by authors*

CCD Input x1	CCD Input x2	Yield Observed Actual	Real x1	Real x2	Normalized x1	Normalized x2	Normalized Actual Yield	Predicted real Y by Tanh in ANN
-1	-1	76.5	80	170	0.14639321	0.14639321	0.191489	76.5006472
-1	1	77	80	180	0.14639321	0.85360679	0.297872	76.9997737
1	-1	78	90	170	0.85360679	0.14639321	0.510638	78.000603
1	1	79.5	90	180	0.85360679	0.85360679	0.829787	79.4997472
0	0	79.9	85	175	0.5	0.5	0.914894	79.9399325
0	0	80.3	85	175	0.5	0.5	1	79.9399325
0	0	80	85	175	0.5	0.5	0.93617	79.9399325
0	0	79.7	85	175	0.5	0.5	0.87234	79.9399325
0	0	79.8	85	175	0.5	0.5	0.893617	79.9399325
1.414	0	78.4	92.07	175	1	0.5	0.595745	78.4001785
-1.414	0	75.6	77.93	175	0	0.5	0	75.6004931
0	1.414	78.5	85	182.07	0.5	1	0.617021	78.4997551
0	-1.414	77	85	167.93	0.5	0	0.297872	77.000694

Reckon that so far, we only *reproduced* by ANN what Montgomery had analytically done in Chapter 11 using his CCD data (that we presently bootstrapped), and little else. His response surface model ((2) cited earlier) would also predict yield—given valid values for time and temperature, albeit considering only up to second order terms. And it would allow one to optimize the process 'easily' since the regression model produced by CCD has a nice closed functional form. It is still noteworthy that unlike ANN, regression model (2) could be built under only the rather restrictive conditions of linearity in parameters ($\{\beta_i\}$ in (1)) and nonlinearities captured only up to second order terms.

7. How did the ANN fare in comparison with Montgomery's CCD model?

Model (2) is the result of building a second order regression model using the standard statistical procedure and assumptions. The method is statistically precise in that it can help one predict yield given values within the span of the process domain explored experimentally, albeit within the limits of the underlying assumptions of ignoring higher order dependencies and other nonlinearities. Such predictive ability of model (2) can lead one to quickly build the response surface of yield, a key step in optimizing yield. In comparison, predictability is also possible for the ANN model built, except that the world of artificial neural networks still remains somewhat of a black box. Yes, we can print out the final optimized weights of the trained ANN. But writing the equation for the response (here yield) in terms of given inputs is generally intractable. Hence, to make a prediction by ANN, we must enter the inputs of interest into the *actual parameterized code* of the *trained ANN*, not an equation.

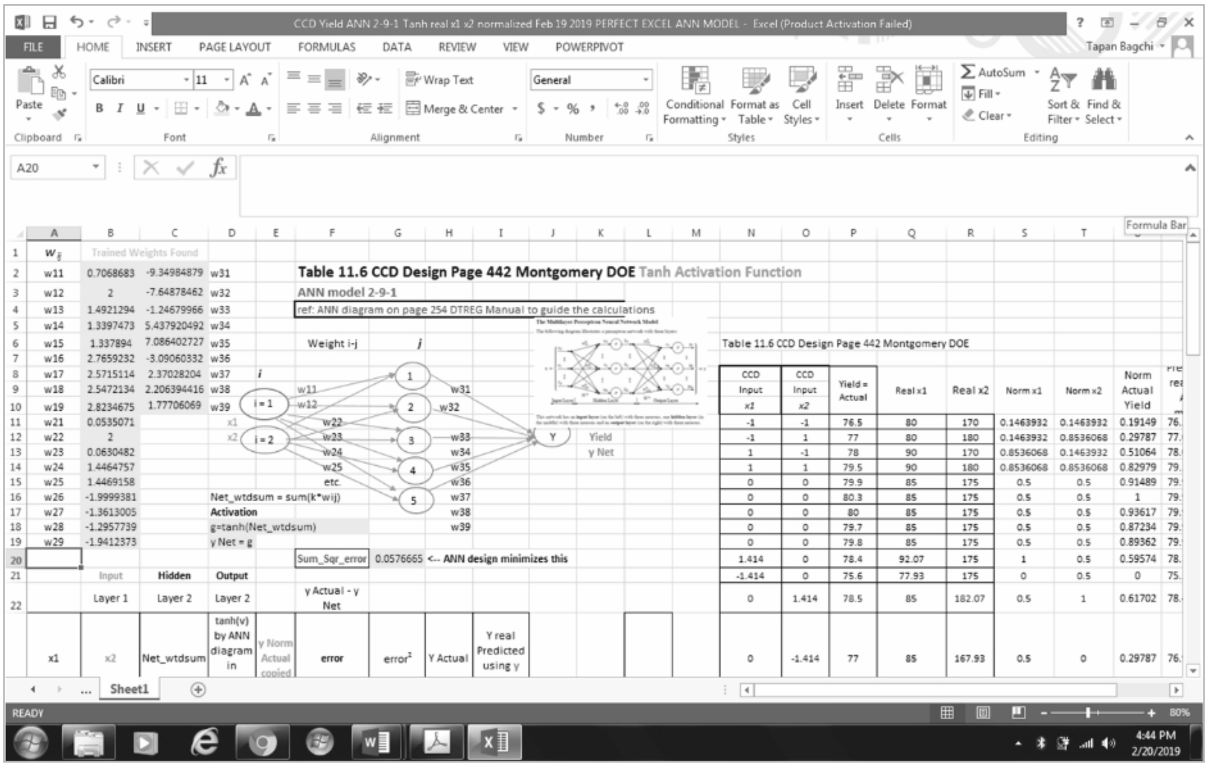


Table 11.6 CCD Design Page 442 Montgomery DOE Tanh Activation Function

ANN model 2-9-1

ref: ANN diagram on page 254 DTREG Manual to guide the calculations

w_{ij}	Trained Weights Found		
w11	0.7068683	-9.34984879	w31
w12	2	-7.64878462	w32
w13	1.4921294	-1.24679966	w33
w14	1.3397473	5.437920492	w34
w15	1.337894	7.086402727	w35
w16	2.7659232	-3.09060332	w36
w17	2.5715114	2.37028204	w37
w18	2.5472134	2.206394416	w38
w19	2.8234675	1.77706069	w39
w21	0.0535071		
w22	2		
w23	0.0630482		
w24	1.4464757		
w25	1.4469158		
w26	-1.9999381		
w27	-1.3613005		
w28	-1.2957739		
w29	-1.9412373		

Net_wtdsum = sum(k*wij)
Activation
g=tanh(Net_wtdsum)
y Net = g

Sum_Sqr_error 0.0576665 <-- ANN design minimizes this

	Input	Hidden	Output					
	Layer 1	Layer 2	Layer 2		y Actual - y Net			
x1	x2	Net_wtdsum	tanh(v) by ANN diagram in	y Norm Actual copied	error	error²	Y Actual	Y real Predicted using y

Table 11.6 CCD Design Page 442 Montgomery DOE

CCD Input x1	CCD Input x2	Yield = Actual	Real x1	Real x2	Norm x1	Norm x2	Norm Actual Yield
-1	-1	76.5	80	170	0.1463932	0.1463932	0.19149
-1	1	77	80	180	0.1463932	0.8536068	0.29787
1	-1	78	90	170	0.8536068	0.1463932	0.51064
1	1	79.5	90	180	0.8536068	0.8536068	0.82979
0	0	79.9	85	175	0.5	0.5	0.91489
0	0	80.3	85	175	0.5	0.5	1
0	0	80	85	175	0.5	0.5	0.93617
0	0	79.7	85	175	0.5	0.5	0.87234
0	0	79.8	85	175	0.5	0.5	0.89362
1.414	0	78.4	92.07	175	1	0.5	0.59574
-1.414	0	75.6	77.93	175	0	0.5	0
0	1.414	78.5	85	182.07	0.5	1	0.61702
0	-1.414	77	85	167.93	0.5	0	0.29787

Figure 6: *The authors' trained ANN to predict yield given time and temperature as inputs*

Let us see another aspect of this two-pronged attempt to understand the underlying chemical process empirically. With little further explanation, we present the final results due to model (2) and by ANN visually in Figures 7 and 8 respectively. Clearly, based on bootstrapped training, Figure 8 reveals more about the process and its nonlinearities; hence, it is likely to be a better predictor. The response surface in Figure 7 may be improved by expanding the exploration of the original process space by going beyond the 13 CCD points of Table 1 (Box, Hunter and Hunter, 1978). Such additionally observed data, preferably randomly experimentally sampled in the time-temperature space, is also likely to further mend Figure 8.

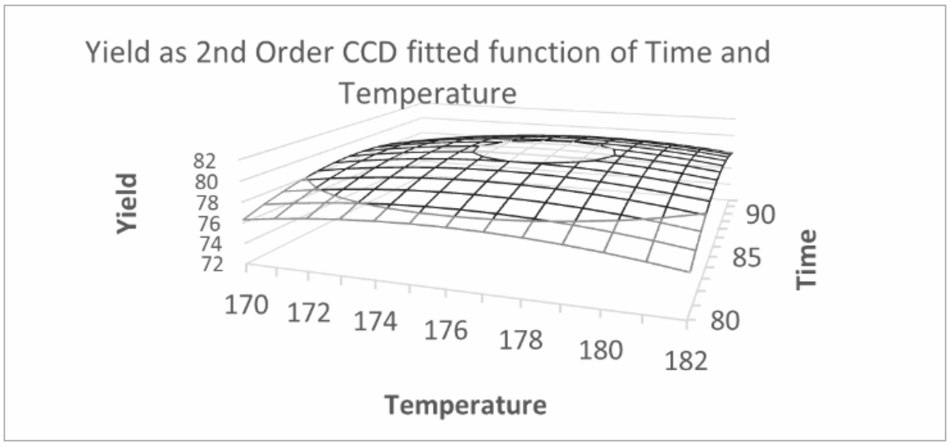


Figure 7: 2nd Order *Response surface fitted using the CCD model (2) obtained by authors*

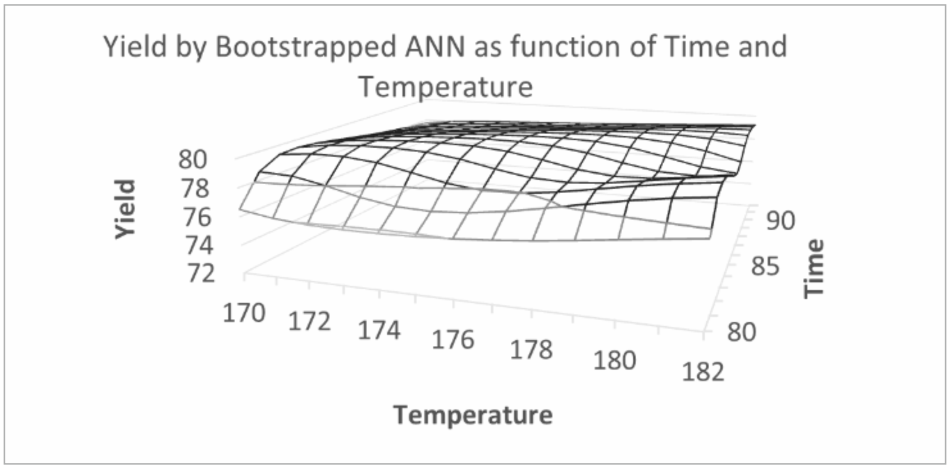


Figure 8: *Response surface generated by the 2-9-1 ANN with Tanh Activation and bootstrapped training as obtained by authors. This trained response surface produced by authors using Microsoft Excel® Solver® is identical to the surface produced separately by SPSS 22 based on the same training data.*

8. However, an intrigue remains!

Elegant as it is, model (2) is yet unable to *find* the time-temperature combination *given* some pre-specified value of yield; for instance, say for yield = 78. Solving multiple regression models of the likes of (2) in reverse, in general, would be intractable. At the outset of this study, we had set out to be able to respond to such a question, yet the RSM method won't answer it without engaging such methods perhaps as goal programming. It's worth noting that this question is one almost trivial to respond to—for the *trained* ANN model sitting on the worksheet behind Figure 6. For this, we *twist* it.

To answer now the process engineer's question such as “What time and temperature do I need to set on the process to get the yield of …?” one would proceed as follows.

Since at this stage the ANN is already *trained*, its input and output are now *rigidly connected*, mathematically and exactly, within the limitation of the training carried out and the linear algebraic equations, optimized weights, and activity functions incorporated in it. Hence, given some particular input value pair (time* and temperature*), the trained ANN would give exactly the *same* output (yield*) *every time*, except for degeneracy.

Being connected end-to-end mathematically, the *reverse relationship* also holds: Given some valid target yield# and the *trained* ANN, we would be handed back the exact same or a valid process factor value pair (time#, temperature#) *every time*. Mathematically this input-output relationship is one-to-one and onto, i.e., *bijective*, or it is *surjective—for every element y in the codomain Y of f there is at least one element x in the domain X of f such that f(x) = y.* It is not required that x be unique; the function *f* may map one or more elements of X to the same element of Y (Wikipedia, 2019). Thus, such an ANN is an amazing facility to help discover input conditions that would lead to some *particular and specified* yield number! Once the ANN is thoroughly trained, if some desired output (yield) is specified, the ANN—*with the same training weights carefully preserved* on the Microsoft Excel® worksheet—will be able to tell us the matching input conditions by reverse mapping. All we have to do is to briefly run the Microsoft Excel® Solver®'s GRG nonlinear optimizer in reverse on the *trained* ANN. Why are we doing this by ANN? Generally speaking, regression models such as (2) can't be easily solved backward—from target output values to their unknown matching inputs. Procedurally we would need to do as follows.

The Microsoft Solver® dialog box must now set yield as the *optimization objective*, *equated* to the desired target value, with time and temperature entered as the *decision variables* in the dialog box whose optimum values will be searched by GRG using the trained ANN. (Be careful to enter valid constraints in the dialog for the search to work correctly!) Table 2 displays some trials. The trained ANN can also generate the response surface to help the analyst visualize the input-response dependency. Figure 8 shows this. This "reverse optimization capability" of a trained ANN is thus a rather clever exploit of neural nets and a huge time saver, even if we continue to loosely say that the ANN works like a black box and "never shows the equations" linking input to output. With a trained ANN or a deep learned network, you will neither need nor will ever know the "equations" linking input to output (Gershenson, 2018; Hornik, Tinchdombe and White, 1989; Kolhe and Bhise, 2019; Siddique et al., 2019).

Note further that it is possible that such tracing back by Microsoft Solver® may lead to degenerate or multiple candidate answers, for end-to-end process paths may not always be unique (see the contours on the yield surface in Figure 8—*many* time-temperature combinations would give *identical* yield). But such situations may be resolved by some secondary selection criteria such as expediency or cost, etc.

Table 2: The trained ANN's *Reverse Predicting* the required Control Variable settings, given desired yield targets, as obtained by authors

	Required process settings discovered by ANN	
Yield desired	**Time**	**Temperature**
76	79.009607	172.92076
77	79.594144	174.151356
78	80.395863	174.579339
79	81.508053	174.735613
Max 79.9709076	85.908794	176.45915

9. Applicability and Limitations of Research, and Future Research Opportunities

This study has successfully demonstrated a most useful capability of a well-trained ANN—it can help *locate or discover process conditions* (inputs) to deliver a *pre-specified response-on-target*. This capability of neural nets generally goes unexploited, though a lot of historical process data with much training value languishes in storage in computer files in process plants. But such intrigues abound in the application of advanced manufacturing technologies such as EDM, CNC, and robotics, fabricating VLSI or memory chips, processing pharmaceuticals, or in alloy steel production. Much of this is currently attacked by exploratory plant trials or by designed experiments as necessitated.

Microsoft Excel® can frequently do a good job for building an ANN

Modelling physical phenomena has been a strongly desirable aspect of human intellectual pursuit—we want to understand and control the phenomena and systems around us—in order to live well. The tools that we have used have been qualitative, and wherever possible, quantitative. Sometimes the goal is to optimize the phenomena or processes in which we have an interest. We have employed logic, abstraction and math, and recently computing machinery—hardware and software—to expand our capabilities. The recent advent of artificial intelligence, the popular objet d'art being the neural net, has greatly enhanced this capability of ours, particularly in getting "almost as close to as we wish" to the true phenomenon with approaches such as deep learning. Just look back at Figures 7 and 8. Thus, this has taken us where classical statistical modelling methods such as multiple regression so far could not. In our research, however, using only simple network architecture, we have demonstrated that ubiquitous tools such as Microsoft Excel® can take us *a long way* in this mission. We can build very effective and useful learning systems using Microsoft Excel® and its Solver® macro and produce predictive process models, even if we have only a limited understanding of the “theory inside.” This work thus opens the door for plant engineers and R&D personnel to start using data from process or lab log books to model systems and processes quite effectively, for most of them now know and regularly use Microsoft Excel®.

Reverse mapping the process from Output to Input is demonstrated

Additionally, we have demonstrated how with a trained neural network, one can effectively discover the required *input* (process or design variables) to give a desired *target output*. This is often a highly desirable aspect of process control and design work. We have shown here how such *reverse mapping* can be done with a trained artificial neural network very effectively. This is frequently a nearly impossible task, when *only* to go from input to output, one has to use a repertoire of complex theories and dependency relationships. This is a singular achievement of this study.

And we need new data collection strategies that go beyond designed experiments—to raise model fidelity

This, in fact, opens the door for considerable further research. In classical statistical modelling, in order to effectively build multiple regression models, one uses appropriately designed experiments (Montgomery 2007; Box et al., 1978). However, beyond a point, such special experiments become burdensome to conduct, and ever theoretically limiting for the phenomena we wish to model and understand. So many of "reality" thus stay out of the data we would collect using conventional statistical experiments such as CCD. Neural nets, on the other hand, make very few assumptions about the input-output data collection procedure. Enough of the nonlinearities are often captured in lab or process plant logbooks, but they remain *buried*! For practical purpose, even randomly but widely spread data, or if possible, data collected in a grid-like manner, can suffice to help produce a decent prediction model. And one can bootstrap data for training (Paass, 1993). However, it would behove data scientists and statisticians to strategize this data collection process in order to produce networks with even higher predictive capability.

This aspect we feel should be formally studied.

10. Concluding Remarks

This study has successfully demonstrated a most useful capability of a well-trained ANN—it can help *locate process conditions* (inputs) to deliver a *pre-specified response-on-target*. This capability of neural nets generally goes unexploited, though a lot of historical process data with much training value languishes in storage in computer files in process plants. But such requirements abound in the application of advanced manufacturing technologies such as EDM, CNC, and robotics, fabricating VLSI or memory chips, processing pharmaceuticals, or in alloy steel production. Much of this is currently probed by exploratory plant trials and direct experiments as necessitated.

To operating, engineering and R&D personnel, the ANN approach of process modelling offers an exceptional capability that is entirely possible to acquire and exploit—as shown in this paper—with access only to Microsoft Excel® and some carefully preserved historical process data logs or trials and experiments. Such data can serve as the golden repository of complex process knowhow—typically buried in records in the organization. The larger the amount of such historical process data, the better it would be able to catch the nuances of nonlinear input-output dependencies in the process. This thus has the potential to materially raise the likelihood of superior R&D and effective process control and optimization at the plant level. This can also guide one to decide what the configuration of the next prototype design or process should be. Note that the entire exercise presented in this study utilized only 13 data points from Montgomery's CCD example. Still, planned statistical experiments for such knowledge acquisition is *not* always a precondition to produce valuable returns that a trained ANN can provide. Wisely spread random acquisition of input-output pairings in a relatively stable environment could often be enough to serve well for the net's training (see Cross Validated 2019).

SAS® (2016) has noted that machine learning can become an important part of the company's innovation strategy, for machine learning allows for data-driven solutions. By contrast, conventional approaches typically attempt analytical solutions to model processes, an action which slows down getting to answers quickly. Machine learning models are also able to shoulder more of the intellectual work that humans would do conventionally, allowing many decisions to be made more directly from observed data. However, SAS® remarks that ANN and similar models must be managed and their performance continually monitored. Typically, such models have been trained on static temporal snapshots of data causing their predictions to become less accurate over time as conditions captured in the training data change. In this light, the prediction error rate needs to be monitored to check if it surpasses a predefined threshold on new data. These are rather useful pointers to heed, irrespective of the well-proven capability of ANNs to help comprehend situations where observations can easily be made, but rigorous analytical modelling is not feasible.

To sum up, building a well-trained ANN model for systems involving nonlinear complex relationships certainly seems to hold a special promise for process control and optimization, as regression and RSM methods have done in the past. Machine intelligence already appears to be sufficiently matured to serve also as an important aid in analytics. Even without moving on to Deep Learning, a well-trained ANN developed using a representative sample of process input-output data, it therefore appears, can help a great deal as a stand-in model when nonlinear relationships are suspected among input and response variables in manufacturing and other enterprises. Our work has illustrated this capability of artificial neural net models using only Microsoft Excel®. We successfully demonstrated effective reverse mapping from a desired target output to the corresponding input variable values, using the forward-trained ANN with its connection weights preserved—here on the Microsoft Excel® worksheet. With Microsoft Solver®'s capabilities available at hand, this did not require the reverse output $\rightarrow$ input training. And we did not anywhere invoke RSM, the classical process optimization method increasingly being contrasted with ANN (Patel and Brahmbhatt, 2016). Our only suggestion to the process engineer is for her to strive for the acquisition of representative training data to build the ANN, and not be restricted to experimental designs with low statistical resolution (Box et al., 1978). For a variety of such reasons, use of ANN in manufacturing is rapidly growing. Instances may be now found in Huang and Zhang (1994), Serio, Facchini and Mummolo (2018), and many other emerging studies.

References

- Bagchi, Tapan P (2012). Justifying Six Sigma Projects in Manufacturing Management, NMIMS Management Review, April-May 2012, ISSN 0971-1023.
- Bagchi, Tapan P (2013). Refining Al Methods for Medical Diagnostics Management, NMIMS Management review, October 6.
- Bernard, GA (1982). "Causation", Encyclopedia of Statistical Sciences, VOL 1, EDS. S KOTZ, N JOHNSON AND C READ, JOHN WILEY, 387-389.
- Box, G E P, W G Hunter and J S Hunter (1978). Statistics for Experimenters, Wiley.
- Choong, Joe (2009). Build Neural Network with MS Excel®, http://www.xlpert.com/ebook/Build_Neural_Network_With_MS_Excel_sample.pdf Accessed February 19, 2019.
- Coit, D. W., Jackson, B T and Smith, A. E. (1998). Static neural network process models: considerations and case studies. International Journal of Production Research, 36(11), 2953-2967.
- Cook, D. F., Ragsdale, C. T. and Major, R. L. (2000). Combining a neural network with a genetic algorithm for process parameter optimization. Engineering Applications of Artificial Intelligence, 13, 391-396.
- Cross Validated (2019). Neural Network Modeling Sample Size https://stats.stackexchange.com/questions/78289/neural-network-modeling-sample-size
- Efron B (1979). Bootstrap methods: Another look at the jackknife. Annals of Statistics; 7:1-26
- Franke, J and M H Neumann (1998). Bootstrapping Neural Networks, https://kluedo.ub.uni-kl.de/frontdoor/deliver/index/docId/511/file/gelb_38.pdf
- Gershenson, Carlos (2018). Artificial Neural Networks for Beginners, https://arxiv.org/ftp/cs/papers/0308/0308031.pdf Accessed February 19, 2019
- Holland, P (1986). Statistics and Causal Inference, Journal of American Statistical Association, Vol 81 No. 396, 945-960.
- Hornik, C, Maxwel Tinchdombe and Halbert White (1989). Multilayer Feedforward Networks are Universal Approximators, Neural Networks, Vol. 2, pp. 35Y-366.
- Huang S and Zhang, H-C (1994). Artificial neural networks in manufacturing: Concepts, applications, and perspectives, IEEE Transactions on Components Packaging and Manufacturing Technology Part A, July.
- Josh, N and Seema Shah (2019). A Survey of Resource Management in Containerized Cloud, NMIMS Engineering and Technology Review, Volume I Issue 1, January.
- Kendrick, David A, P Ruben Mercado and Hans M Amman (2006). Neural Nets in Excel, Computational Economics, Ch 2, Princeton University Press.
- Khaw, J F C, Lim, B S and Lim, L E N (1995). Optimal design of neural networks using the Taguchi method. Neurocomputing, 7, 225-245.
- Kolhe, Abhay K and Archana Bhise (2019). Automatic Road Extraction using Modified Local Vector Pattern, NMIMS Engineering and Technology Review, Volume 1 Issue 1, January.
- Lin, T Y and Tseng, C. H. (2000). Optimum design for artificial neural networks: an example in a bicycle derailleur system. Engineering Applications of Artificial Intelligence, 13, 3-14.

- Luigi A, L Serio, F Facchini and G Mummolo (2018). ANN Modelling to Optimize Manufacturing Process, in book Advanced Applications for Artificial Neural Networks, Intech, Chapter 11, Publisher: InTech, Editor Adel El Shahat.
- Macleod, C, Dror, G. and Maxwell, G. (1999). Training artificial neural networks using Taguchi methods. Artificial Intelligence Review, 13, 177-184.
- Manjunath, P G C and P Krishna (2012). "Prediction and Optimization of Dimensional Shrinkage Variations in Injection Molded Parts Using Forward and Reverse Mapping of Artificial Neural Networks", Advanced Materials Research, Vols. 463-464, pp. 674-678.
- Montgomery, D C (2007). Design and Analysis of Experiments, 5th ed. (2007); 8th ed. (2012), Wiley.
- Nascimento C A O, Guidici R, Guardani R (2000). Neural network based approach for optimization on industrial chemical processes. Comput Chem Eng, 242: 303-2314.
- Neumann, M H and F Schiller (2000). Bootstrapping Neural Networks, Neural Computation, September, https://www.researchgate.net/publication/12368224_Bootstrapping_Neural_Networks?enrichId=rgreq-c3cc88ca184e77f4821bb6973acf027a-XXX&enrichSource=Y292ZXJQYWdlOzEyMzY4MjI0O0FTOjIyNzk4MDQwMDEzMjEwMEAxNDMxMzY2MTcxOTU0&el=1_x_2&_esc=publicationCoverPdf
- Paass, G (1993). Assessing and Improving Neural Network Predictions by Bootstrapping Algorithms, German National Research Center for Computer Science, https://papers.nips.cc/paper/659-assessing-and-improving-neural-network-predictions-by-the-bootstrap-algorithm.pdf
- Patel, Kiran A and P K Brahmbhatt (2016). A comparative study of the RSM and ANN models for predicting surface roughness in roller burnishing, Procedia Technology, 23, 391 - 397.
- Patil, S, Vaishali Kulkarni and Archana Bhise (2019). Comprehensive Assessment of Dental Cone Beam Computed Tomography (CBCT): A Systematic Approach, NMIMS Engineering and Technology Review, Volume I Issue 1, January.
- Rajagopalan, R and Purnima Rajagopalan (1996). Applications of Neural Network in Manufacturing, Proceedings of the 29th Annual Hawaii Int'l Conference on System Sciences. IEEE, 447-453.
- Rocca, Joseph (2018). A gentle journey from linear regression to neural networks, https://towardsdatascience.com/a-gentle-journey-from-linear-regression-to-neural-networks-68881590760e
- SAS (2016). The Evolution of Analytics: Opportunities and Challenges for Machine Learning in Business, authors Patrick Hall, Wen Phan, and Katie Whitson, O'Reilly.
- Siddique Mohd Umair, Abhishek Priyam, Sher Afghan Khan, Ravi Terkar and Rajesh Patil (2019). On Numerical Investigation of Nusselt Distribution Profile of Heat Sink Using 107 Lateral Impingement of Air Jet, NMIMS Engineering and Technology Review, Volume I Issue 1, January.
- Su, C and Hsieh, K L (1998). Applying neural network approach to achieve robust design for dynamic quality characteristics. International Journal of Quality and Reliability Management, 15(5), 509-519.
- Sukhotmya, W and James Tannock (2005). The training of neural networks to model manufacturing processes, Journal of Intelligent Manufacturing, 16, 39-51, 2005.
- Tong, Lee-Ing and Kun-Lin Hsieh (2000). A novel means of applying neural networks to optimize the multiresponse problem, Quality Engineering 13(1):11-18, September 2000.

- Tutorials Point (2018). Artificial Intelligence, https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_tutorial.pdf Accessed February 19, 2019.
- Vellido, A, Lisboa, P J G and Vaughan, J (1999). Neural networks in business: a survey of applications. Expert Systems with Applications, 17, 51-70.
- Wikipedia (2017). Activation Function; one to one and onto https://en.wikipedia.org/wiki/Activation_function Accessed on February 19, 2019.
- Wilcox, J A D and D T Wright (1998). Towards pultrusion process optimization using artificial neural networks. Journal of Materials Processing Technology 83: 131-141.
- Williamson, A G (1995). Refining a neural network credit application vetting system with a genetic algorithm. Journal of Microcomputer Applications, 18(3), 261-277.
- Zorriassatine, F and Tannock, J D T (1998). A Review of Neural Networks for Statistical Process Control. J Intelligent Manufacturing, 9(3), 209-224.

***Tapan P Bagchi**, B Tech (IIT Kanpur), MASc and Ph D (Toronto) and D Sc (IIT Kharagpur) has over 40 years' experience split between industry (EXXON-Mobil USA) and academics (the IIT system). Published widely in operations management, stochastic processes, quality engineering, microeconomics, project management, and metaheuristic optimization, Bagchi has authored seven texts in these areas. He is a registered professional engineer in Ontario. He can be reached at tapan.bagchi@gmail.com*

***Milan A Joshi**, Assistant Professor at NMIMS University Shirpur, holds B Sc, M Sc and Ph D—all in mathematics. He specializes in harmonic analysis, machine learning and data science. A member of Indian Mathematical Society, he is certified in statistical machine learning, data science, MySQL, SAS and Teradata. He has published extensively in topology, Banach algebra, SVM-based classification, and Cancer classification. He can be reached at mlnjsh@gmail.com*