Self Trained Artificial Neural Network

Vikramaditya Reddy.J.

Student Member AAAI, IEEE, ISOC, CSI, ISTE.
Plot no 22, K.R.N.Colony, Bowenpally, Secunderabad, A.P-500011, India.
vikramadityareddy@gmail.com,v_jakkula@hotmail.com

Abstract

Traditional supervised neural network trainers have deviated little from the fundamental back propagation algorithm popularized in 1986 by Rumelhart, Hinton, and Williams. Typically, the training process begins with the collection of a fixed database of input and output vectors. The operator then adjusts additional parameters such as network architecture, learning rate, momentum, and annealing noise, based upon their past experience in network training. Optimizing the network's generalization capacity usually involves either experiments with various hidden layer architectures or similar automated investigations using genetic algorithms. Along with these often-complex procedural issues, usable networks generally lack flexibility, beginning at the level of the individual processing unit. Normally, the user finds him or her confined to a limited range of unit activation functions, usually including linear, linear threshold, and sigmoidal analytical forms whose partial derivatives with respect to net input are definable through a similar continuous, analytical expression. Generally, the demand is for a more flexible and user friendly system that will not only lessen the technical confusion for non-connectionist end-users, but also create expanded utility for those demanding more architectural freedom and adaptability in their artificial neural networks or ANNs.

Introduction

By implementing an ANN within a spreadsheet environment, using only relative cell references and supplied functions, we create a quasi-parallel computational scheme quite distinct from past algorithmically based neural network simulations and addressing the above-mentioned needs. By allowing one such spreadsheet network to serve as training supervisor to another, we create an adaptive neural network cascade that need only be physically juxtaposed with the database to learn any of its contained patterns. Viewed as a whole, the combination of untrained network and its supervising counterpart represent a self-training artificial neural network, learning by parallel processing and without the assistance of algorithmic code. Providing some means to stream data past this self-training artificial neural network, or "STANN" (patent-pending), this agent "sees" exemplars and spontaneously self-organizes to generalize the represented knowledge domain. Such a network implementation is well suited for monitoring and learning from data streams supplied to the Excel environment, either from external devices, human input, or algorithmic codes. Furthermore, because such networks function in a parallel fashion, we may train multiple STANNs simultaneously, allowing us to gain multiple perspectives on the influx of data.

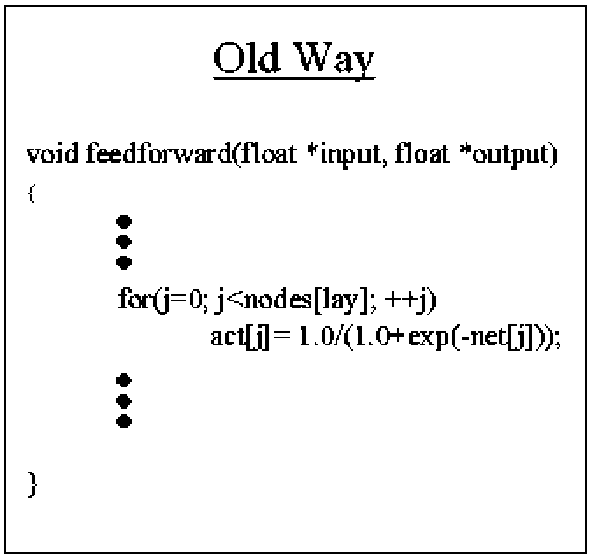
Because such self-trainers numerically calculate the needed partial derivatives, the functional form of activation need not be simple. Instead, each processing unit's activation function may be arbitrarily complex, perhaps taking the form of an entire neural network. We thereby achieve the capacity to create self-training neural network cascades, with each component network emulating the behavior of any given analog or digital device. Training of such compound networks would allow any combination of electrical, mechanical, or optical devices to self-organize and interconnect to achieve any desired input-output relation, in turn allowing us to rapidly prototype and invent a wide variety of hardware devices. Herein we elaborate on the basic operation of the self-training artificial neural network and then expand on its potential applications and derivative concepts.

Principles and Applications of the Self-Training Artificial Neural Network Object

We typically think of neural network simulations as the sequential evaluation of activation states of neurons within a network, using some algorithmic language such as C or C++. Within such schemes, individual activation levels are only momentarily visible and accessible, as when the governing algorithm evaluates the sigmoidal excitation of any given neuron (Figure 1). Except for its fleeting appearance during program execution, a neuron's excitation becomes quickly obscured by its redistribution among downstream processing elements.

Exploiting the many analogies between biological neurons and cells within a spreadsheet, we may evaluate the state of any given network-processing unit through relative references and resident spreadsheet functions (Figure 2).

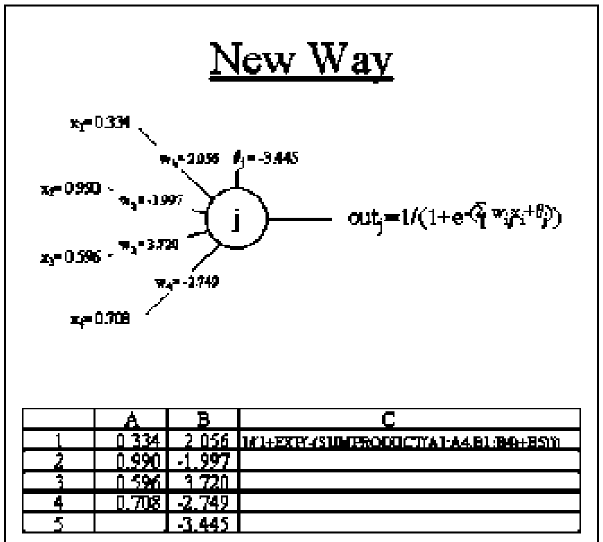
By referencing the outputs of such spreadsheet neurons to the inputs of other similarly implemented neurons, we may create whole networks or network cascades. Unlike the algorithmic network simulation, all neuron activations are simultaneously visible and randomly accessible within this spreadsheet implementation. More like a network of virtual, analog devices, we consider this simulation to be a distributed algorithm, with all neurons equilibrating with one another following each wave of spreadsheet renewal.



Old Way

```
void feedforward(float *input, float *output)
{
        •
        •
        •
        for(j=0; j<nodes[lay]; ++j)
                act[j] = 1.0/(1.0+exp(-net[j]));
        •
        •
        •
}
```

Figure 1. Neuron activation within a given network layer is evaluated in C-code.



	A	B	C
1	0.334	2.056	1/(1+EXP(-(SUMPRODUCT(A1:A4,B1:B4)+B5)))
2	0.990	-1.997	
3	0.596	3.720	
4	0.708	-2.749	
5		-3.445	

Figure 2. Neuron activation within a given network layer is by relative reference within a spreadsheet environment.

As a further benefit of the spreadsheet implementation, the user now has a convenient graphical interface for constructing and experimenting with ANNs. For instance, we need only build a template neuron once, using simple copy and paste commands to build and connect whole networks from a prototypical processing unit. We may repeat these procedures on larger scales to move networks into position and link them into larger cascade structures. The compound neural networks that result are transparent in operation and easily accessible for modification and repair. No longer confined to simple feed forward architectures, we may readily introduce recurrences and all manner of neural network paradigms, including IAC, Boltzmann Machine, Harmonium, Hopfield nets, and self-organizing maps.

Originally intended to serve as an expedient means to produce spreadsheet network cascades we have discovered many additional advantages to this graphical network implementation. Probably the most important development to emerge from this scheme is the patent-pending concept of a ‘self-training artificial neural network’ or "STANN." Consisting of one network training another, we may view the STANN as a spin-off of the so-called "Creativity Machine" paradigm (Yam, 1995). Such Creativity Machines consist of a chaos-driven network, trained within a given knowledge domain, and supervised by a second network. As the level of internal perturbation is slowly ramped up in the former net, the second supervisory net monitors its output, alert to any emerging vectorialized ideas that fulfill some predetermined need or goal. Notable exercises of this paradigm have included the generation of myriad; esthetically pleasing musical hooks (Thaler, 1994) as well as the discovery of new ultra hard materials (Thaler, 1995). Because we may rapidly train both component networks by example and then control the chaotic network’s "egress" from the chosen knowledge domain (by progressive weight perturbation), we may rapidly prototype and refine these systems. The technique offers the significant advantage of reduced development time over past rule- and model-based attempts at discovery and creativity (Rowe & Partridge,1993). Viewed in this context, the STANN is comparable to an ‘inverse’, spreadsheet-implemented Creativity Machine. Rather than subjecting the connection weights of the first network to randomizing influences, we apply weight updates obtained from the supervising network. These weight adjustments ultimately correct the first net’s initially randomized connection weights to achieve the desired mapping (Figure 3). Important to emphasize is that the supervising network calculates all weight updates from the observed errors, using a distributed algorithm built upon the traditional back propagation paradigm. (There are no sequential algorithmic routines corresponding to the partial derivatives, error terms, and updates of the prevalent network trainers.) The result is that such a spreadsheet-implemented network cascade quite literally need only be shown data to train. For each exemplar presented to the STANN, a self-correcting wave propagates through the composite network, beginning with a spreadsheet calculation of network error and terminating at the network's updated connection weights.

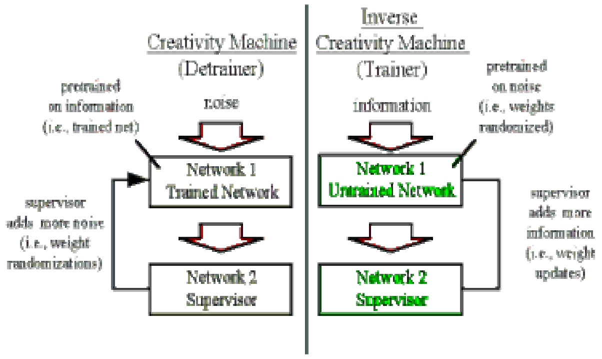

Creativity Machine
(Detrainer)
noise
pretrained on information (i.e., trained net)
Network 1 Trained Network
Network 2 Supervisor
supervisor adds more noise (i.e., weight randomizations)
Inverse Creativity Machine
(Trainer)
information
pretrained on noise (i.e., weights randomized)
Network 1 Untrained Network
Network 2 Supervisor
supervisor adds more information (i.e., weight updates)

Figure 3. Considering noise and information to be complementary, the inverse of a Creativity Machine is a self-training neural network

Viewing the untrained net and its supervisor as a single, composite net, adapting to data external to itself, the boundary between supervised and unsupervised learning becomes diffused. After all, except for the repeated application of the intended input-output to the net, there is no supervisory algorithm per se, simply a single composite network internally adapting to its external (spreadsheet) environment. The distinction between 'supervised' and 'unsupervised' totally dissipates when we purposely use an auto associative network, and consider training on exemplars containing identical input and output vectors. Training then consists of applying these vectors to both inputs and outputs simultaneously while hidden layer neurons connect themselves to attain some classification scheme. The resulting process bears a close resemblance to the unsupervised methods characteristic of Kohonen (1982) self-organizing maps.

STANN Structure and Operation

Using only relative references and resident spreadsheet function, I have built several prototype STANNs. Whereas, we have implemented such STANNs in myriad ways using these underlying principles, their general construction (Figure 4) usually involves clearly differentiable modules corresponding to the individual stages of the back propagation paradigm (Rumelhart, Hinton & Williams, 1986). All of these modules communicate through cell reference thereby binding them as a unit, embedded alongside the spreadsheet data. The distributed algorithm then estimates the partial derivative of each neuron's activation with respect to its net input by adding some small increment, d, to the input vector's components and submitting them to a replica network contained in module 2. The STANN then numerically calculates the derivative in module 3 for all neurons by noting the changes in both its activation and its net input between modules 1 and 2. Modules 4 and 5 handle the back propagation of error, within similar cell reference networks, with the resulting corrections simultaneously appended to all weights and biases in both the parent and replica networks in modules 1 and 2.

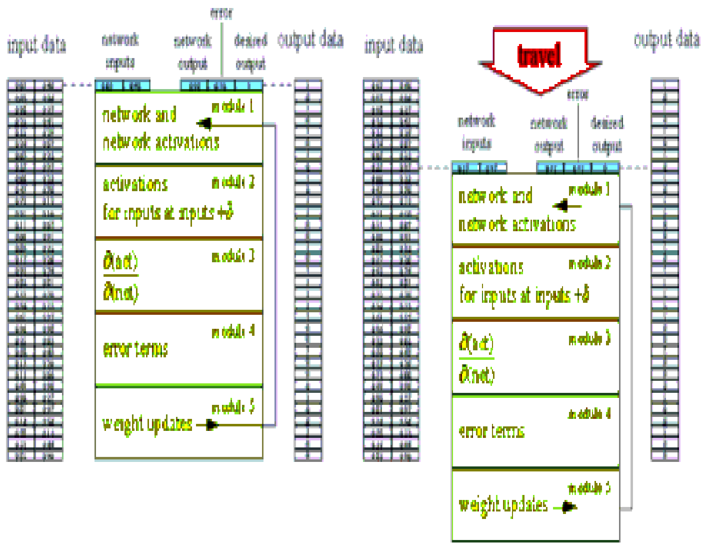


Figure 4. Module structure of a simple STANN (left) consisting of two inputs and one output, immersed within a database. The STANN mobilizes itself by automated cut and paste commands to move through the data to learn any contained patterns.

Important to note is that the STANN "sees" its training data by relative referencing. Thus any input data to the left of the STANN's network input cells appears duplicated within those cells. Similarly any output data to the STANN's right appears reproduced simultaneously in the designated output cell. Viewing these "interface" cells between the external spreadsheet "environment" and the STANN's internal network as metaphorical "retinas," we see that data may appear as a training progression to the STANN if data and STANN move relative to one another.

We achieve such relative movement between the STANN and the data by either (1) movement of the data or (2) movement of the STANN. In the former mode, data streams through the Excel columns by dynamic data exchange as the STANN observes a fixed group of spreadsheet cells. Alternately, STANN motion may occur by allowing the STANN to command an algorithm (i.e., Visual Basic macros) that performs cyclic cut and paste commands to the totally self-contained network cascade, as depicted in Figure 4.

The former training technique may find immense utility in the real-time patrol of data streams from sensors in various hardware or laboratory systems. Such STANNs learning in real time, may provide appropriate outputs to control various system actuators. In another application, separate VBA macros running algorithmic models may generate streams of model exemplars to a STANN to convert various rule- or model-based algorithms into their corresponding neural network models. Extensions of this concept may vastly facilitate the conversion of expert systems to their connectionist counterparts.

In the latter training technique, STANNs may move and train within static or periodically updated spreadsheet databases. There, the added dwell time allows the STANN to make many passes through the data (e.g., training epochs) between database updates. This concept finds added utility in advanced STANNs that are able to move in two or three spreadsheet dimensions, choose their own perspective on the database, and elect their own trajectory through the spreadsheet workbook. Such mobile STANNs or "data bots" now can exhaustively search through either fixed or dynamic databases in search of subtle patterns of interest or unexpected discoveries.

Dynamic Data Exchange to STANNs

One exciting application of the self-training artificial neural network object is in the area of real time training of sensor data fed directly to spreadsheet environments by dynamic data exchange or 'DDE.' That is, we may elect with various newly developed software packages such as National Instrument's LabView^{TM} and Measure^{TM} to pass sensor output to selected spreadsheet cells that double as the inputs and outputs of a STANN. The self-training net may then simultaneously train on any desired input output relationship as data originates from the system of interest. This development now paves the way for totally adaptive control systems, entirely implemented within Microsoft Excel. In theory, it should be possible for a STANN to observe the action of human operators in restoring various production systems back to their intended set points. Then, after sufficient 'apprenticeship' period, they may step in to control that process by outputting the necessary DDE control parameter adjustments.

System Prototyping and Self-Organization

As mentioned above, one of the most powerful features of the STANN is that the transfer function of each processor element is free to take a wide variety of forms. Therefore, it need not be restricted to an analytical and differentiable squashing functions such as the sigmoids popularly used in ANN construction (see for instance, Skapura & Freeman, 1991). Individual processing units may even take the form of whole ANNs representing the input-output response of various analog and digital devices. Training of a collection of such composite ANN systems would be tantamount to the discovery of just how to connect such subsystems to attain some desired global response. Thus the trained connection weights between various electrical devices such as diodes, transistors, etc. would represent connecting resistances to attain some overall electronic function. This device prototyping concept is extendible to other digital electronic applications involving such components as flip-flops and counters. It would likewise be applicable to a wide range of optical, mechanical, and hydraulic systems.

As an example of such device prototyping I have performed preliminary experiments in which I have coupled various ANNs representing various harmonic generation devices in parallel, driven by a single sinusoidal signal generator to produce some predetermined waveform such as a square wave. The compound network self-organizes itself to allow conversion of the sinusoidal input into a rectangular pulse output. We note that during training of this cascade, connection weights assume values proportional to the Fourier coefficients. In principle, this ANN cascade could represent the prototype of an electrical or optical waveform synthesis device.

Potential Applications

In the spirit of object-oriented programming, it is my intention to build Self-Training Artificial Neural Network Objects possessing a variety of innovative properties and methods. Some very broad and powerful applications of such "STANNOs" will include:

Autonomous Knowledge Base Agents - One of the unique advantages of the STANN is that multiple networks can simultaneously train. Therefore, multiple STANNs may patrol the dynamic data exchange to an Excel spreadsheet, each taking their own unique perspective on the data stream. STANNs or STANN cascades may in turn take action upon that data to extract or manipulate the resulting database.

Adaptive Creativity Machines - Until now, Creativity Machines have been static in the sense that we have already trained all of its component networks prior to its discovery function. Using the STANN's ability to train on a dynamic influx of data, a Creativity Machine may simultaneously train and create. Presently, I have implemented such schemes allowing multiple STANNs to concomitantly train on the data stream, with a master algorithm periodically copying and pasting the updated networks into their respective positions within a spreadsheet-resident Creativity Machine.

Adaptive Hardware and Instrumentation - Smart rooms, furniture, vehicles, etc. may utilize the self-trainer to adapt to changing requirements and conditions. Thus an automobile equipped with STANNO control would adapt fuel injection within the driving context to attain the highest fuel economy. Such software could likewise learn driver idiosyncrasies in real time and automatically compensate for disadvantageous motoring habits.

Artificial Life - Expanding on the metaphor of a "retina," implemented through relative referencing, we may provide various levels of visual processing, analogous to those in human vision, and subsequently, databot reaction to that sensory information. Autonomy derives from the encapsulation of data and function in a manner reminiscent of class objects in object-oriented programming. Further augmenting the databot's organism-like capabilities (e.g., locomotion, reproduction, etc.) the integration of Creativity Machine architectures within the virtual creature's neural cascade allows for its own autonomous movement planning, attention-windowing, and course of action when manipulating its database environment.

Cooperative Spreadsheet Organisms - Implementing a communication scheme between such spreadsheet objects, it may be possible to achieve cooperative behaviors among databots to build larger neural cascades. Having achieved prototypical structures of this kind in the autonomous construction of Creativity Machines, it is conceivable that much more generalized self-constructing cascades may spontaneously organize within Excel to produce highly sophisticated neural processing structures.

Conclusion

The Self-Training Neural Network Object represents a major paradigm shift from the notion of algorithmic code training an algorithmic neural network simulation. Here, one distributed algorithm trains another, without the intervention of a programmer or sequential computer code. Viewed as a whole, these two reciprocal nets constitute a single, self-training spreadsheet object that need only be physically cut and pasted within the data to learn any patterns contained therein. If data streams through a spreadsheet application by DDE, such STANNOs may similarly glean trends from external agencies and instrumentation. With the appearance of Excel spreadsheets as a convenient medium of dynamic data exchange from instrumentation, satellite feeds, and the Internet, the STANNO may represent a means for sensing, detection, and prediction, all in real time. Used in conjunction with other cascaded networks and algorithms, such self-trainers will allow for the bi-directional sensing and control required in a number of applied and experimental areas.

References

Freeman, J.A. and Skapura, J.A. (1991*). Neural Networks, Applications, and Programming Techniques*, (Reading: Addison-Wesley), 98-99.

Kohonen, T. (1982). *Self-organized formation of topologically correct feature maps, Biological Cybernetics* 43, 59-69.

Rowe, J. and Partridge, G. (1993). *Creativity: a survey of AI approaches, Artificial Intelligence Review*, 7, 43-70.

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. 1986). *Parallel distributed processing: Explorations in the microstructure of cognition, Vol. 1* (eds Rumelhart, D.E. & McClelland, J.L.) (Cambridge: MIT).

Thaler, S.L. (1994) *Musical Themes From Creativity Machine*, U.S. Copyright Pau1-920-845.

Thaler, S.L. (1995) *An Autonomous Discovery Machine for Ultra hard Materials*, *Bulletin of the American Physical Society*, Program of the March Meeting, Series II, 40(1), 592.

Yam, P. (1995). *Scientific American* 272(5), 24-25.