In Proceedings of the *IASTED International Symposium on Expert Systems and Neural Networks*, pp. 41-44, 1989.

Neural Network Applicability: Classifying the Problem Space

Tony Martinez

Professor, Computer Science Dept.
230 TMCB, Brigham Young University
Provo, Utah 84602

Abstract

The tremendous current effort to propose neurally inspired methods of computation forces closer scrutiny of real world application potential of these models. This paper categorizes applications into classes and particularly discusses features of applications which make them efficiently amenable to neural network methods. Computational machines do deterministic mappings of inputs to outputs and many computational mechanisms have been proposed for problem solutions. Neural network features include parallel execution, adaptive learning, generalization, and fault tolerance. Often, much effort is given to a model and applications which can already be implemented in a much more efficient way with an alternate technology. Neural networks are potentially powerful devices for many classes of applications, but not all. However, it is proposed that the class of applications for which neural networks are efficient is both large and commonly occurring in nature. Comparison of supervised, unsupervised, and generalizing systems is also included.

Introduction

The tremendous current effort to propose neurally inspired methods of computation forces closer scrutiny of real world application potential of these models. This paper discusses features of applications which make them efficiently amenable to neural network methods. Limitations and features of computational models, together with a categorization of application classes according to three basic features is first given. This is followed by a discussion of some functional differences of current neural network models.

Computation and Applications

As scientists endeavoring to propose and implement computational mechanisms which will improve our ability to fulfill applications which are considered to enhance our lifestyles, it is important that we understand just what it is that machines can do. I submit that machines do *deterministic mappings* of input to output. The functionality of any machine can be represented by a true mathematical function

$$\boldsymbol{F(x) = y},$$

where F is the mapping function, x is the total environmental input, and y is the output. The input x includes all factors affecting the mapping, which include not only the typical input data, but also the temperature of the machine, any current defects it has, etc. The function F can be changed through programming (although this can be considered as input to a meta-function), but at any given time its mapping is purely deterministic (i.e. given an input x, the output must be the specific value y). Although we consistently use abstract languages which accept a statement such as *if x then do y <u>or</u> z*, this can not be implemented on a machine. We must somehow program the machine and tell it to do y or z when x is true. We can simulate nondeterminism through random number generators, etc., but the seed to the generator is part of the input to the overall mapping function F for that machine, and the mapping remains deterministic.

Initially, this might sound simplistic and not overly powerful. However, we have found that our computational machines can be very powerful and useful in solving many important applications. Given that machines do deterministic mappings, the interesting question is how that mapping takes place. In theory, all functional mappings could be done by a single RAM circuit, where address lines represent input values, if we could (or would want) to make one big enough. Yet, we have always found it desirable to trade space for time and implement mappings using iteration and other algorithmic techniques. Computational machines differ in the ways in which they accomplish this mapping. A *good* machine is one which accomplishes the mapping under the constraint of certain features. Some of the features currently considered important are:

Speed - computational latency and throughput

Cost Effectiveness - implementation, usability

Fault Tolerance - Robustness, Graceful Degradation

Ease of Use - programming, learning, user-friendly

Applicability - Useful in real world applications

Generality - Special, General purpose

These are features of computation, not to be confused with application features which we use to categorize applications below. Neural networks (NN) show potential to fulfill most of the above features desired in computational machines. NN attain speed through massive parallelism, potential cost effectiveness with improvements in VLSI, optics, and new technologies, fault tolerance through implicit algorithmic mechanisms, and ease of use through self-organization and learning by example. The applicability of NN is the subject discussed in the rest of this article. The feature of generality may not be one of its strong points. The capabilities of the human brain, although quite broad, perform quite poorly on a number of interesting applications, including accurate memory, numerical mathematics, etc. It appears that to have high speed execution of applications it may be necessary to create different classes of computational machines for different classes of applications.

Often in the field of neural networks much effort and energy is spent on proposing applications of models, where the application can already be fulfilled in a more effective manner with an alternate technology and/or algorithm. For example, arbitrary mappings of boolean input and output vectors, where the majority of input states are defined and important, coupled with a guaranteed high speed learning mechanism, can be efficiently accomplished with a normal RAM chip. Learning is accomplished simply by presenting an input state (the address) and an output state (which is placed in the contents of the memory address). This type of problem is observed when researchers use as test applications such problems as *n-bit parity, encoders,* etc. These types of applications appear to be unwieldy for both artificial and natural nervous systems, while being reasonably easy to implement using alternate technologies. It *is* important that we understand the limitations of connectionist mechanisms, but it is just as critical that we understand the target application domains for which this promising technology is suited.

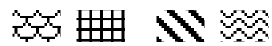
In order to give an initial view of the types of applications for which neural networks have substantial potential, I first introduce three simple features of applications.

Adaptive vs. Fixed: A fixed mapping (application) is one where the mapping function F does not change in time. An adaptive application is one for which the mapping function F does change in time. In other words, in an adaptive application the mapping of a given input may change to a new output as a result of learning, programming, etc. (It is realized the all functions could be considered as fixed, as mentioned above, if we consider programming, feedback, etc. as part of the function input, thus controlling the adaptive mapping. For this discussion we assume the view that a function can adapt.) An adaptive application is typically one that is learned over time and which can continue to change upon receipt of new inputs. An example of an adaptive application would be a rule based expert system which could both receive and infer new rules over time. An example of a fixed application would be accounts payable. (Even though the parameters, number of people, salaries, etc. do change over time, the computational mechanism of the accounts payable system would not change).

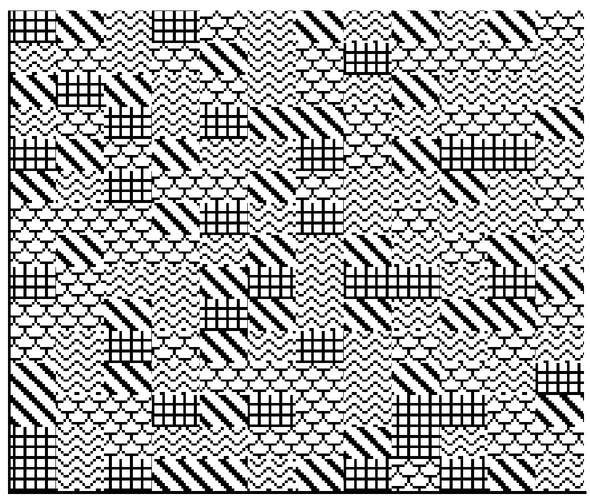
Parallel vs. Sequential: This differentiates between applications which are *naturally (efficiently)* parallel and those which are more cost effective to do sequentially. All applications can be done in a parallel fashion at the lowest level (since all can be represented by a simple mapping function). However, this would not always be efficient. For example, a table-lookup or a connectionist architecture for integer multiplication is not cost-effective for large numbers, because the table is exponential in size. Rather, an iterative technique is used which requires more time but is far more efficient overall. As research continues, more basic applications which initially appear sequential, are being solved with creative parallel techniques [10].

Complete (fragmented) vs. Incomplete (generalizable): This is considered to be a *crucial*, yet not often considered feature of applications. A given application (function) is called complete if a specific output mapping is required for all (or a majority) of the permutations of the input variables. An example of this is integer multiplication. In this case it is necessary to have a well-defined output for each possible value of the inputs. These systems occur often in man-made abstractions such as mathematics, storage devices, etc. An incomplete application is one for which specific outputs need only be defined for a subset of the input permutations, or where large similar classes of inputs map into the same output space. This is the case for generalization where groups of similar inputs map to a similar output. Incomplete functions appear to occur very often in the natural world. Two primary causes of this are the notions of *impossible* and *don't care* states. For example, assume an input, which represents a naturally occurring object or event, which is made up of 30 boolean variables, each representing a feature of the input. Of the billion possible permutations of the 30 input variables, only a subset of those actually occur in the real world. The others are called impossible states. Of the input states which can occur, many are dominated by a few *critical* variables. This is the notion for example, that if one touches a hot stove, arm retraction is immediately initiated regardless of most of the other impinging inputs. The output mapping in this case is defined in terms of some critical variables, without regard to the other *don't care* variables. The notion of complete vs. incomplete is typically demonstrated in numeric vs. symbolic environments. Rule based applications are typically examples of incomplete functions, since any rule only supplies critical input variables to decide an output (the others are don't care) and only specifies rules for interesting input states (the others are either impossible or don't care).

Figures 1 and 2 give graphical examples of the incomplete vs. complete feature. Two continuous variables *x* and *y* are considered and the functional mapping of the two variables into four possible output states is shown by the different patterns on the graphs.

Output States: 

X



Y

Figure 1 - Complete Mapping

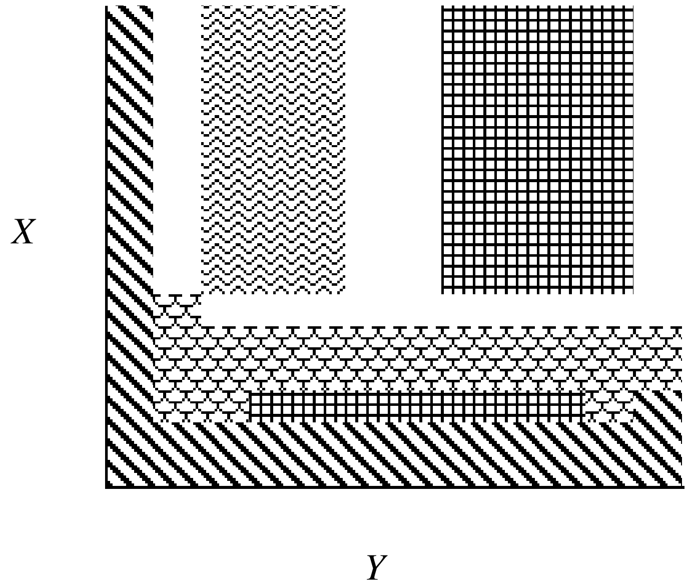


Figure 2 - Incomplete Mapping

The white areas in figure 2 represent impossible states. Because of the potential groupings of the incomplete mapping, it is possible to come up with a parallel mechanism of computation which is both fast and cost efficient, using smaller numbers of units which generalize the mapping.

Examples of applications in the incomplete domain include rule based systems, pattern recognition, control and decision systems, etc. Complete applications include arithmetic, bit mapped graphics, etc. There is a spectrum between complete and incomplete applications. For example, pattern recognition of normal visual objects is incomplete, while pattern recognition of random raster patterns on a CRT is complete. In general, neural networks attempting to do complete applications require exponential amounts of computing ability (whether in number of nodes, number of links, resolution of weights, etc.). In contrast, minimization and generalization techniques can be used for incomplete applications allowing an efficient neural network representation.

Note (referring to figures 1 and 2) that an adaptive function can graphically be considered as one where the arrangement of the output mappings changes over time dependent on inputs and feedback.

Figure 3 is a simple mapping of the applications falling into the 8 permutations of the above 3 application features. At the leafs is mentioned the computational machinery currently considered most effective for that class of applications.

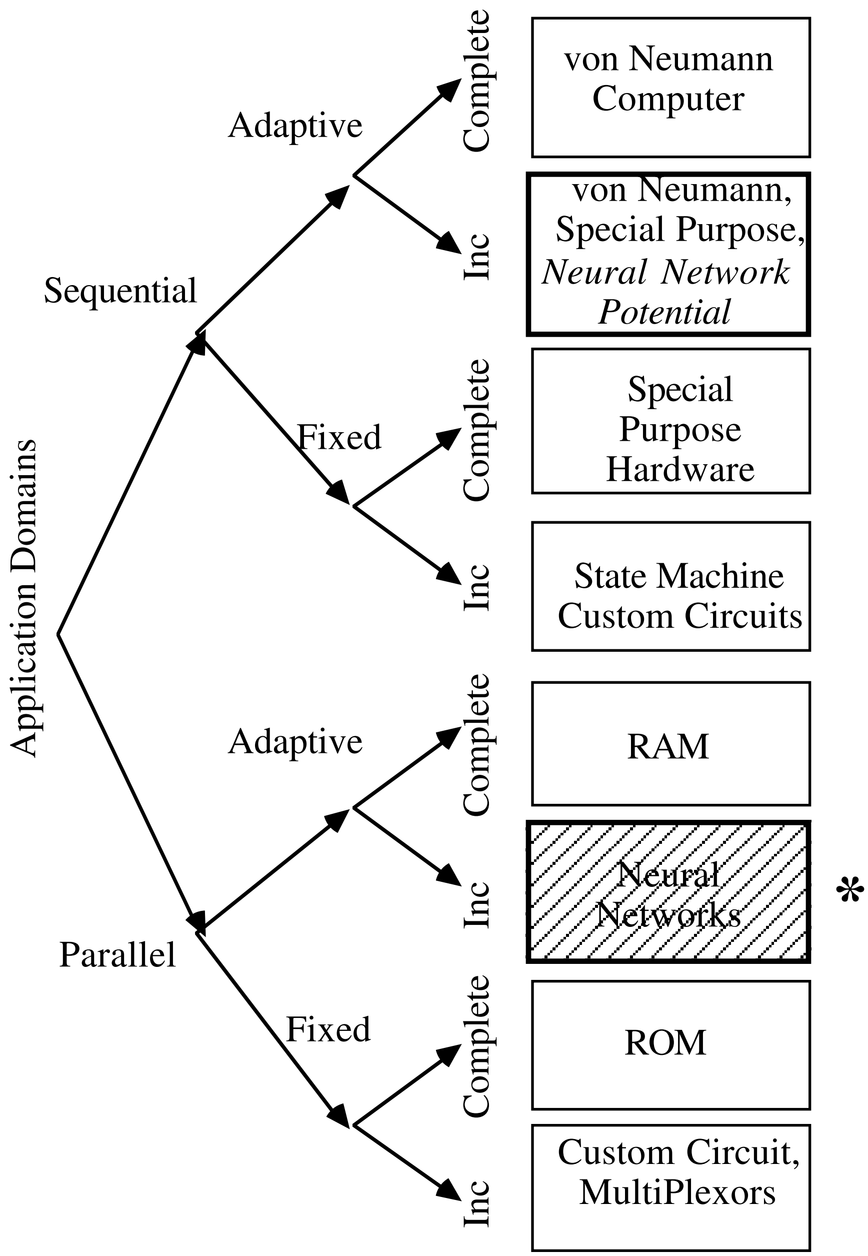


Figure 3 - Classification of Applications

Note that the class of applications for which neural network models seem currently most effective (parallel, adaptive, and incomplete), is exactly that class of applications which occurs more often in nature, rather than in artificial environments. Although most current application efforts of neural networks in the parallel, adaptive, incomplete domain, both precedent from natural nervous systems and application needs point to sequential, adaptive, incomplete applications as another fertile area for neural networks. In these cases, parallelism would still be highly used, but the actual mapping would be temporal (i.e. the control of walking, speech, etc.).

Neural Network Models

The class of parallel, adaptive, incomplete applicatiokns can, of course, be furthur broken down into subclasses. Current neural network models target different categories within these classes. I briefly mention two: supervised vs. unsupervised learning, and generalization.

Connectionist systems and applications differ in their needs and goals. Effective high level processing may require an integration of separate models, each processing in its domain of strength, in order to acquire an emergent multi-purpose mechanism. Although a simplistic and incomplete dichotomy, I classify current artificial neural systems into *categorization (unsupervised)* and *decision* (supervised) systems. Examples of categorization systems include adaptive resonance theory [1] and self-organizing feature maps [4]. These systems do unsupervised learning of categories and classifications of arbitrary inputs. They seek solutions in the important domain of building up reasonable categories from an unknown input environment. However, they are limited to a restricted type of classifying, mapping similar things into similar classes.

A contrasting application is that of taking mapped inputs, such as those from a categorization system, and further mapping them into outputs (motor actions) or inferences. In this case, many inputs which are very dissimilar may need to be mapped into the same output class, while similar inputs could be scattered into various output classes. This is the realm of decision systems. These systems are required to do more arbitrary mappings and require some type of supervised learning (teacher, environment, reward, etc.) Attempts at this type of system include backpropagation techniques [9] and Boltzmann machines [2]. There is certainly overlap between categorization and decision systems, and classifying a model as one or the other is a matter of degree. Many times direct comparison of categorization and decision systems are made. Since their functionality and target applications are quite different, these comparisons seem empty. Perhaps more general systems will include a categorization portion which does unsupervised classification of inputs into usable classes, and which feeds its outputs to a decision making system which more arbitrarily controls motor or other output units (figure 4).

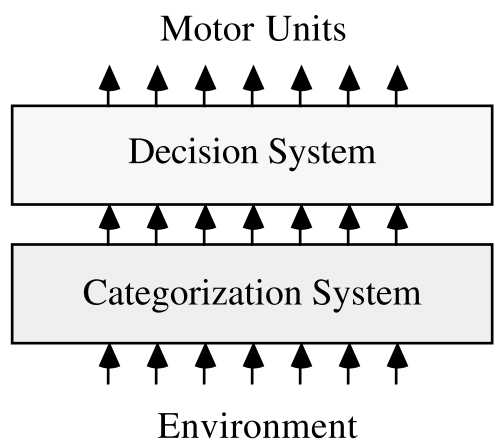


Figure 4 - Integrated categorization and decision systems

Generalization is the ability of a neural network, which has seen a portion of the input environment, to have a high probability of correct output for inputs which it has not yet seen. Note that the most intuitive forms of generalization follow from the *incompleteness* of applications. Many current neural models do generalization by similarity of inputs (hamming distance, etc.) [3]. This is but one type of generalization, and the type most beneficial will depend on the application. One of the apparent powers of natural nervous systems is the ability to discriminate the current critical inputs from a massive barrage of total inputs. This leads to generalization based on extracting critical variables from an environment and treating the others as don't care variables. This type of generalization differs from similarity, but is quite pervasive in the real world. The *ASOCS (Adaptive Self-Organizing Concurrent System)* [5-8], currently under investigation at Brigham Young University, is a supervised learning model which uses this later form of generalization. Rule input in ASOCS allows a natural mechanism to specify critical variables for specific situations. Generalization takes place in that many possible input vectors impinging during processing can match the single rule which has specified a higher order combination of critical variables.

Conclusion

This paper has sought to delineate the classes and features of applications for which neural networks appear most applicable. Currently, the class of adaptive, parallel, incomplete applications best fit neural network utility. An understanding of the target goals of neural networks will aid researchers in measuring the potential and ability of current and proposed models.

Bibliography

1. Carpenter, G.A., and Grossberg, S., A massively parallel architecture for a self-organizing neural pattern recognition machine, *Computer Vision, Graphics, and Image Processing,* **37,** pp. 54-115, (1987).

2. Hinton, G., Sejnowski, T, and D. Ackley, *Boltzmann Machines: Constraint Satisfaction Networks that Learn,"*Tech. Rep CMU-CS-84-119, CMU, Pittsburgh, PA, (1984).

3. Hopfield, J.J., Neurons with graded response have collective computational properties like those of two-state neurons, *Proc. Natl. Acad. Sci*., Vol. 81, pp.3088-3092 (1084).

4. Kohonen, T., *Self-organization and associative memory*, Springer Verlag, New York, (1984).

5. Martinez, T. R., *Adaptive Self-Organizing Logic Networks*, Ph.D. Dissertation, Technical Report - CSD 860093, University of California, Los Angeles, CA (May 1986).

6. Martinez T. R., Models of Parallel Adaptive Logic, *Proceedings of the 1987 IEEE Systems Man and Cybernetics Conference*, pp. 290-296, (October, 1987).

7. Martinez, T. R. and J. J. Vidal, Adaptive Parallel Logic Networks, *Journal of Parallel and Distributed Computing*, Vol. **5**, pp. 26-58, (1988).

8. Martinez, T. R., Adaptive Self-Organizing Concurrent Systems, in *Progress in Neural Networks*, Ablex Publishing, 1989.

9. Rumelhart, D. and McClelland, J., *Parallel Distributed Processing:*, Vol. I, pp. 318-362, MIT Press, (1986).

10. Waltz, D., Applications of the Connection Machine, *Computer*, (Jan. 1987).