Chapter 4

Limits of Connectionism

4.1. Introduction

Let us distinguish two parts of a programming language. First, its **framework***, which gives the overall rules of the system, and second, its* **changable parts***, whose existence is anticipated by the framework but whose particular behavior is not specified by it.*[1]

The connectionist paradigm is much like a programming language, in that it provides a framework for constructing computational models which anticipates various changeable parts. This chapter examines the connectionist framework and its underlying assumptions, and focuses on certain limitations which recur frequently in connectionist modeling efforts. Based on the elements used to construct a Turing Machine within this framework, it is argued that some of the default assumptions need to be rethought.

4.2. The Connectionist Framework

Feldman and Ballard (1982) provide a particularly concise framework for connectionism, which anticipates both logical and numeric computation within the same

[1] (Backus, 1978) p. 617.

models:

> *A* **unit** *is a computational entity comprising:*
>
> *{q} - a set of* **discrete states**, *< 10*
> *p - a continuous value in [-10,10], called* **potential**
> *(accuracy of several digits)*
> *v - an* **output value**, *integers* $0 \le v \le 9$
> **i** - *a vector of* **inputs** $i_1, \cdots i_n$
>
> *and functions from old to new values of these*
>
> $$p = f(\mathbf{i},p,q)$$
> $$q = g(\mathbf{i},p,q)$$
> $$v = h(\mathbf{i},p,q)$$
>
> *which we assume to compute continuously. The form of the f, g, and h functions will vary, but will generally be restricted to conditionals and simple functions.*[2]

These units are arranged into networks, where the input vectors for units consist of previous output values of other units. Particular functions that Feldman & Ballard mention include linear combinations of the inputs with weights, and sums of products.

A similar framework is described by Rumelhart, Hinton & McClelland (1986). According to them, there are eight major aspects of a Parallel Distributed Processing (PDP) model:

[2] (Feldman & Ballard, 1982) p. 250.

- *A* **set of processing units**
- *A* **state of activation**
- *An* **output function** *for each unit*
- *A* **pattern of connectivity** *among units*
- *A* **propagation rule** *for propagating patterns of activity through the network of connectivities*
- *An* **activation rule** *for combining the inputs impinging on a unit with the current state of that unit to produce a new level of activation.*
- *A* **learning rule** *whereby patterns of connectivity are modified by experience.*
- *An* **environment** *within which the system must operate*[3]

There are many assumptions underlying these connectionist frameworks, leading to constraints which are often violated, but, nonetheless, operate as a normative force on a rapidly changing field. For example:

(1) *Neurons are not born*: The number of units in a connectionist network is fixed, and they are not dynamically created.

(2) *Synapses change slowly*: The weights in a connectionist network change slowly over time, modified by experience.

(3) *Connectivity is limited*: Units have multiple inputs but only a single output, which may be used as input to multiple neighbors.

(4) *Computation is simple*: Units perform a simple (constant-time) function on their inputs, rather than an involved computation. Furthermore, units do not get to examine all their inputs individually, rather they see some kind of sum or average of them.

(5) *Communication is local and limited*: Units only see the outputs of their immediate neighbors. There is no global memory or blackboard, with inherent synchronization problems. Furthermore, the information circulated is limited in its complexity; numeric values rather than complex messages are passed around.

[3] (Rumelhart et al., 1986a), p. 46.

(6) *100 Cycles to Intelligence*: Since the quickest neural reaction time is on the order of 10 milliseconds, ‘‘intelligent’’ functions need to be computed in 10’s to 100’s of cycles.

These assumptions and constraints, which have something of a religious flavor, originate from a confluence of what is *almost* known about the actual architecture of the brain, along with a hearty dose of computational, philosophical and psychological concerns. For example, (1) through (3) are essentially *architectural* assumptions which arise from knowledge of the structure of real neural networks: Neurons are not dynamically created as needed (at least for adults), synapses do change slowly, and neurons have multiple inputs (synapses on dendrites) but only a single output (an axon), which can serve as inputs to many other neurons. Assumption (4) is a *computational-philosophical* constraint, for who would want to imagine that a single neuron is capable of complex information processing -- i.e., can a sponge think? Number (5) is a deferral to computational reality: By insisting on strict locality, parallelism is possible. Feldman and Ballard (1982) are responsible for (6), which is very compelling, but is a *functional* assumption, which spans levels and fields from neurons, to cycle time, to reaction time.

The connectionist framework is not as fixed as the framework of a programming language, and the rules can be easily broken. The usual attack on heretics is that what they’ve done is not ‘‘neurally plausible.’’ Neuro-fundamentalism aside, there are severe computational and cognitive issues which are stressed by blind faith to architectural assumptions. The sections below examine some of the pros and cons of connectionism, in an attempt to understand how the limiting assumptions need to be reformed.

4.3. The Promises of Connectionist Models

Much of the current interest in these models arises because of their potential for as a good substrate for both cognitive modeling and advanced computational applications. The heralded promises of connectionist networks include:

Effective Use of Parallel Computers

Given that a connectionist network strictly adheres to a principle of local computation, then it is possible to implement such networks on the new generation of parallel computers which are now becoming available. In fact, there have already been several successful efforts in this direction (Blelloch & Rosenberg, 1987; Plate, 1987; Pomerleau et al., 1988).

Soft Constraints

When logic is continuous instead of discrete, decisions can be made ‘‘smoothly’’ over time, rather than through serialized backtracking. One use of activation levels is as confidence values, and the collective input to a unit, either positive or negative, is used as evidence to increase or decrease confidence respectively. If many knowledge sources communicate in the same language (i.e., numbers), then the problem of translating information between different knowledge domains goes away.

Graceful Degradation

This feature is usually associated with distributed representations (Hinton, 1984). If the elements of a representational system are ‘‘coarsely coded,’’ whereby each representational element is represented by a population of units, and each unit represents a particular set of elements, then the incapacitation of individual units will have little impact on a system’s effectiveness.

Learning

One of the dreams of connectionism is that systems can be built with mostly random connections, and that powerful, local learning algorithms can adjust the connections resulting in powerful dedicated machines. The earliest approach to this was the *perceptron convergence procedure* (Rosenblatt, 1962), but Minsky & Papert (1969) showed that single-layered perceptron systems (on which this procedure might be successful) were extremely limited. Recently several techniques have been developed which seem to overcome the limits of perceptrons by adjusting weights in several layers including the

Boltzmann machine (Ackley et al., 1985) and back-propagation of errors (Rumelhart et al., 1986b).

4.4. The Pitfalls of Connectionist Models

On the other hand, connectionism introduces some extreme limitations as well. Many of these problems only arise when connectionism is applied to higher-level cognitive functions such as Natural Language Processing. These problems have been described in various ways, including: Recursion, variable-binding, and cross-talk, but they seem to be just variations on older problems, for which entire fields of research have been established.

Generative Capacity

Despite the promise of connectionism, its paradigmatic assumptions can lead to language models which are strictly finite-state. Several parsers have been built which parse context-free grammars of bounded length — i.e., regular grammars. The term ‘‘generative capacity’’ is due to Chomsky, who used it as a measure of the power (capacity) of particular classes of formal grammars to generate natural language sentences; regular grammars are the weakest in this respect. Although the history of science is riddled with paradigm revolutions and counter-revolutions, returning to computational linguistic systems without generative power is not the kind of counter-revolution one would consider as progress.

For example, as an adjunct to his model for word-sense disambiguation, Cottrell (1985b) proposed a fixed-structure local connectionist model for length-bounded syntactic processing.

In a well-circulated report, Fanty (1985) describes the automatic construction of a connectionist network which parses sentences using a context-free grammar. In what is essentially a time-for-space tradeoff, his system can rapidly parse bounded-length sentences, when presented all lexical items at once. The number of units needed for his network to parse sentences of length n rises as $O(n^3)$.

Selman (1985) also reports an automatic construction for networks which can parse sentences using a context-free grammar. His system is stochastic, and based on the Boltzmann Machine notions of (Ackley et al., 1985). Selman's system is yet another machine for parsing sentences of bounded length. Finally, the connectionist constraint of limited processing cycles is ignored and a parse may take several thousand cycles to ‘‘anneal.’’

And even the newer crop of research in this area suffers from the same fixed-width problem. For example, (Hanson & Kegl, 1987) use a three-layer back-propagation network to encode sequences of up to 15 lexical categories. Allen (1987) demonstrates several applications of three-layer back-propagation to such tasks as pronoun resolution (indicate which word in a 15-word sentence is referred to by a pronoun) and machine translation (from 10-word english sentences to 11-word spanish sentences). And McClelland & Kawamoto (1986) assign case-roles, such as agent, object, and instrument, to three- or four-word sentences.

Representational Adequacy

Closely related to the problem of generative capacity is the problem of representational adequacy. One must be careful that a model being proposed can actually represent the elements of the domain being modeled.

Localist network representations, such as the one used in our work in chapter 2, and throughout the early 1980's connectionist revival, where each unit has some concept ascribed to it, are notoriously lacking in this regard. Basically a diminutive form of semantic networks, local representations are even missing the relational labels on links which give semantic networks their limited representational power.

Fully distributed representations, such as feature- or microfeature-based systems, such as the one used by (Kawamoto, 1985), also suffer. If the entire feature system is needed to represent a single element, then attempting to represent a structure involving those elements cannot be managed in the same system. For example, if all the features are needed to represent a *Nurse*, and all the features are needed to represent an *Elephant*, then the attempt to represent a *Nurse riding an elephant* will result in a common representation of a *white elephant* or a *rather large nurse with four legs*.

One obvious solution to this problem of superimposition versus concatenation involves using separate ‘‘pools’’ of units to represent elements of propositional triples, such as Agent, Action, and Object. In each pool would reside a distributed representation filling these roles, such as ‘‘Nurse,’’ ‘‘Riding,’’ and ‘‘Elephant.’’ Because of the dichotomy between the representation of a structure (by concatenation) and the representation of an element of the structure (by features), this type of system cannot represent recursive propositions such as ‘‘John saw the nurse riding an elephant.’’

There are more sophisticated distributed representations than features, but they do not yet solve the representational adequacy problem. Hinton (1984) set out the notion of ‘‘coarse-coded’’ representations as an alternative to too local or too distributed representations. The essence of ‘‘post-modern connectionism,’’ his main notion is that instead of dedicating a single unit to every element of a representation (the ‘‘Unit/Value principle’’ of (Feldman & Ballard, 1982)) one ‘‘coarsely codes’’ these elements by having each represented by a subset of units and by representing multiple elements with a single unit. The effect is that the activation value of an element becomes the percentage of its representing units which are ‘‘on’’ and that each unit is very ambiguous so that its meaning can only be determined by the entire state.

For example, Touretzky devised a coarse-coding of symbolic triples from a finite alphabet which has been used in a production system (Touretzky & Hinton, 1985), in a Sisyphean effort to build a connectionist system capable of CONSing (Touretzky, 1986b), and in a combination of the two which implements a phrase-structure transformation system (Touretzky, 1986a). The coarse-coding is as follows: The 15,625 possible triples of 25 symbols (A - Y) are represented by the activity pattern of 2000 units. Each unit has a ‘‘receptive field’’ of 216 triples, generated by the cross product of three groups of 6 symbols. The semantic interpretation of this system is that a particular triple is considered on when a majority of the approximately 28 units which represent it are on. This coding can simultaneously represent a small number of symbolic triples, which can be interpreted as, say, a binary tree structure.

In their perceptron model for learning the past tenses of verbs, Rumelhart & McClelland (1986) use an even more complicated distributed representation to represent a sequence of phonemes. Rather than trying to elucidate their actual scheme, it is instructive to view it abstractly as follows:

Consider representing a sequence of tokens, $(i_0, \cdots, i_n)$ as an unordered set of overlapping subsequences (each of length k) of tokens, $\{(i_0, \cdots, i_k), (i_1, \cdots, i_{k+1}), \cdots, (i_{n-k}, \cdots, i_n)\}$. Call this an implicit sequential representation of breadth k. Given a distributed representation whose primitive elements are these subsequences, and the ability to simultaneously represent several of these elements, it is clear how a well-formed set of overlapping subsequences can be interpreted as a longer sequence.

It is also clear that the number of representational elements needed for such a system is exponential, I^k, where I is the number of different possible tokens. Rumelhart & McClelland used an implicit sequential representation of breadth 3 over 27 possible tokens. However, through sheer ingenuity, and by taking account of the internal structure of their tokens, they packed a potential 19,683 elements into a distributed representation using only 460 units. Each triple is represented by 16 units, and each unit has a receptive field of about 685 triples.

Each of these systems start with a finite population of representational elements, and then distribute these elements over a smaller number of processing units. Sequence and structure are derived from collections of elements under some external interpretation, and the limits of the representation appear when substructures are duplicated. For example, fixed coarse-coded systems, such as Touretzky's, run into limitations when there is too much overlap between elements. Touretzky pointed out that the memory exhibits a phenomena, which he called local blurring, ‘‘whereby if we store many closely related triples such as (F A A), (F A B), and (F A C), etc., it becomes increasingly difficult to determine whether a related triple such as (F A L) is present’’ (Touretzky, 1986, p. 524).

And Rumelhart & McClelland's system would not be able to represent words with duplicate phonetic triples such as *Banana*; but they did not face this problem for the verbs used in their model.

Distributed representations are clearly an improvement over local ones, in terms of the ratio of the number of representation elements to the number of units representing them, but the notion of a fixed set of representational elements is still in combat with the infinite generative capacity, and thus, the representational needs of language. The issue of distributed representations is considered further in Chapter 6.

Task Control

A final problem is that many connectionist models use every allowable mechanism they have to do a single task. This leaves no facility for changing tasks, or changing the size of tasks, except massive duplication and modification of resources.

For example, in the Stroop model of Chapter 1, changing the task from color-naming to word-naming requires some external modification of the network. In the past-tense model (Rumelhart & McClelland, 1986), there is no obvious means to conjugate from, say, past to present tense, without another 200,000 weights. In the Traveling Salesman network (Hopfield & Tank, 1985), there is no way to add a city to the problem without configuring an entire new network.

4.5. Towards Effective Connectionist Computation

These recurrent problems stem from the unquestioning acceptance in the neurally-inspired constraints discussed earlier. In particular, if units cannot be created and if links cannot be changed during a computation, then such networks are totally static, and cannot adapt to different tasks. If Feldman & Ballard's constraint, that units only have a few possible states, is taken seriously, then such static networks are just fancy finite-state machines, and so it is not surprising that people keep coming up with parsers for finite-state languages.

The ability to reconfigure a network dynamically is clearly necessary for modeling higher level cognitive faculties, such as language. In some current work on making connectionist networks perform symbol-processing types of computation, for example, Touretzky (1985, 1986) has had to resort to external programs which ‘‘gate’’ various connections. And in our own previous work we have used ‘‘normal’’ computer programs such as a chart parser (Kay, 1973) or production systems to hook up our networks on the fly. However, the use of normal computer procedures leads to the sticky problems of power discussed at the end of Chapter 2.

In fact, the whole issue of computational power in neural or connectionist models is somewhat murky. It is usually taken for granted that a connectionist architecture can compute as effectively as any other machine. This truism dates back to McCullogh &

Pitts (1943). What they actually said was:

> *One more thing is to be remarked in conclusion. It is easily shown: first, that every net, if furnished with a tape, scanners connected to afferents and suitable efferents to perform the necessary motor-operations, can compute only such numbers as can a Turing machine; second, that each of the latter numbers can be computed by such a net; and that nets with [feedback] circles can compute, without scanners and a tape, some of the numbers the machine can, but no others, and not all of them.*[4]

In other words, a neural network, *when attached to a tape, head, and motor,* is a universal computer. While the primitive computations of modern connectionist models are more along the general lines of the perceptron, using linear combinations and thresholds, their ability to compute general logical functions (such as AND, OR, and NOT) is still thought to be enough. According to Rumelhart and McClelland:

> *As we have already seen, one can make an arbitrary computational machine out of linear threshold units, including, for example, a machine that can carry out all the operations necessary for implementing a Turing machine; the one limitation is that real biological systems cannot be Turing machines because they have finite hardware. In Chapter 14, however, we point out that with external memory aids (such as paper and pencil and a notational system) such limitations can be overcome as well.*

In other words, after 43 years, connectionism is still in the same place with respect to computation. They go on to say:

> *We have not dwelt on PDP implementations of Turing machines and recursive processing engines because we do not agree with those who would argue that such capabilities are of the essence of human computation...*[5]

The last paragraph distinguishes our computational approach from the more psychological PDP school. We believe that, besides being of the essence of intelligence, effective computation and recursion are crucial features for any sophisticated application of connectionism.

Since biological systems (i.e., humans) do not need paper and pencil to understand language, it would seem that examining a connectionist construction of a Turing Machine would yield some useful insights for overcoming the 4 limitations discussed

[4] (McCulloch & Pitts, 1943), p. 129.
[5] (Rumelhart & McClelland, 1986b), p. 119.

earlier.

The Neuring Machine

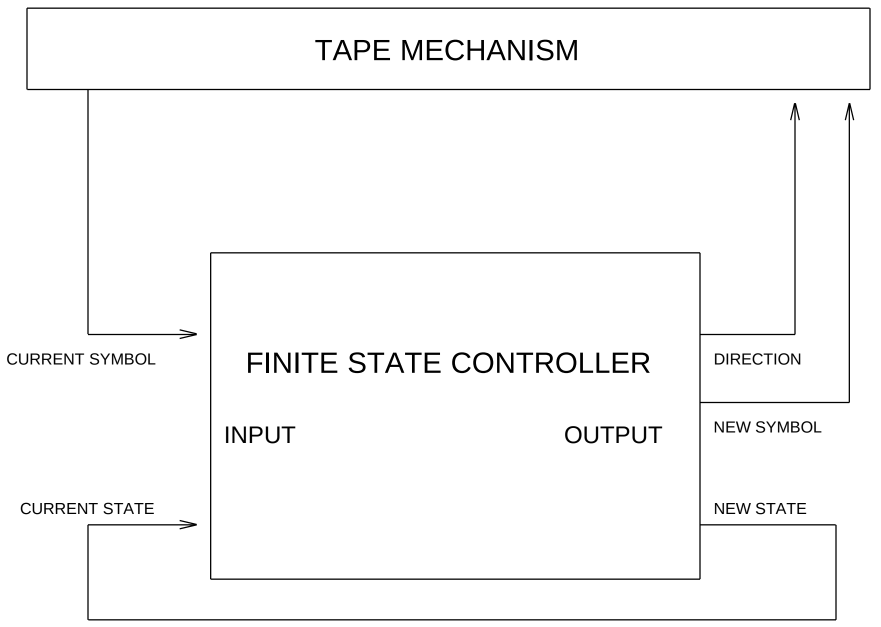

Figure 1. *Block diagram of a Turing Machine, consisting of a finite state controller and a tape mechanism. The tape mechanism produces an input symbol for the controller, which outputs a symbol to write on the tape and a direction to move, as well as a new state for itself.*

A Turing machine consists of a potentially unbounded linear memory called a ‘‘tape,’’ and a finite-state controller which can move, read, and write symbols on the tape. The symbols are elements of a finite alphabet, which for simplicity we will assume to just contain the two symbols 0 and 1.[6]

A block diagram of a Turing Machine as two interacting ’’black boxes’’ is shown in Figure 1. At any time, the controller is in a particular state and receives a symbol from the tape. It produces its own next state, a new symbol to write on the tape, and a direction (either left or right) to move the tape, which then produces the next symbol for the controller, and so on, until the controller reaches a distinguished ‘‘halting’’ state.

[6] Although a TM is usually defined with any finite set of symbols, it is easily shown that 2 are enough.

Implementing the finite-state controller is particularly trivial in almost any computational medium. A simple network of linear threshold units is sufficient, and will not be detailed here. The real problem involves getting a tape *into* the machine. Since no physical system can actually have the unbounded memory of a Turing machine, that unbounded memory becomes just a theoretical device, a ‘‘tape factory’’ which can create memory as needed. But because the units in a connectionist model represent hypothetical neurons, and because neurons are not born, but only die, it is a very un-connectionist idea to have a ‘‘neuron factory.’’[7]

Once the idea of manufacturing new units is eliminated, the only place left to store the tape is in the states of a finite set of units. Indeed, in proofs of the power of stored-program register machines, the assumption is made that registers can hold arbitrarily large integers[8]. Similarly, in order to build a ‘‘Neuring Machine’’ it will be assumed that the output of a unit can be a fraction, between 0 and 1, with arbitrary resolution.

This idea is in clear violation of at least two of the fundamental constraints on connectionist models. As mentioned earlier, Feldman & Ballard (1982) constrained the number of states a unit could have to 10, but we are assuming a potentially infinite number of states. Furthermore, sending an unbounded amount of information across a link clearly violates the constraint on limited communication. But because our purpose is to find a sufficient set of connectionist primitives for effective computation, this assumption should not be considered from a practical perspective.

To implement the tape mechanism, we use two tricks. Unbounded resolution fractions, and true analog gating. To represent the tape, we use two stacks of bits. Consider that a stack of bits, $(s_1, s_2, \cdots)$, can be represented as a fraction, S, between 0 and 1:

$$S = \sum_i s_i 2^{-i}$$

To ‘‘pop’’ a bit off this stack, simply compare it to 0.5 (using a threshold logic unit) and subtract the result from twice the original value. To ‘‘push’’ a bit, add half the stack and half the new bit (using a linear combination). Figure 2 shows these two operations.

[7] Since a neuron includes both memory and control, a neuron factory would manufacture control as well as memory, which is quite different from a tape factory.

[8] For a detailed construction of this sort see (Minsky, 1972), especially chapters 10 & 11.

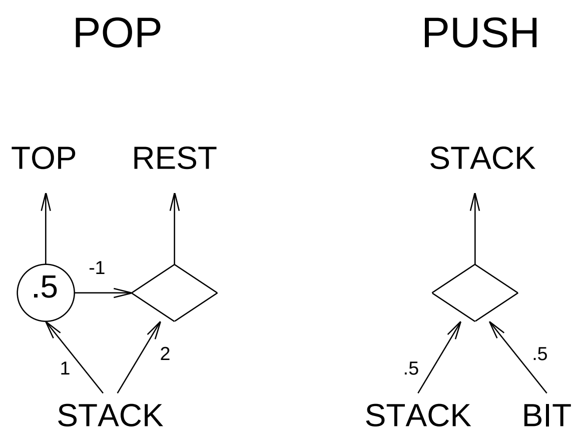

Figure 2. *Simple mechanisms for a stack of bits represented as binary-coded fractions between 0 and 1. The diamonds are linear combination units, and the circles are threshold logic units, which return 0 or 1 depending on whether the inputs are greater or equal to the labeled threshold.*

The tape can be represented by two such stacks, representing its left and right portions. The controller emits two signals to the tape mechanism, a new symbol to write, and a direction to move. If the direction is LEFT, the symbol must be pushed onto the right stack, and the top bit of the left stack is output to the controller; if the direction is RIGHT, the symbol must be pushed onto the left stack, and the top bit of the right stack is output to the controller.

Both the new representations for the stacks and the next input symbol to the controller are gated by the direction signal. The next input symbol, being just a bit, can be gated using threshold logic. However, the new stacks are fractions, which cannot be gated accurately using thresholds, except through an interactive procedure. The ‘‘numeric’’ way to accomplish this gating is through multiplication, by generalizing the the sum of products from logic to arithmetic.[9]

Figure 3 shows the entire tape mechanism, which uses 6 linear combination units (diamonds) and 5 threshold units (labeled circles). The previous push and pop mechanisms are in the bottom of the figure, used by both left and right portions, to get the top bits and to push the new symbol into the stacks. Given two direction signals, threshold

[9] Logically, gating bits A or B by C can be done by the sum of products $AC+B\bar{C}$; when A and B are rational numbers between 0 and 1, the logical product must be replaced with an arithmetic product.

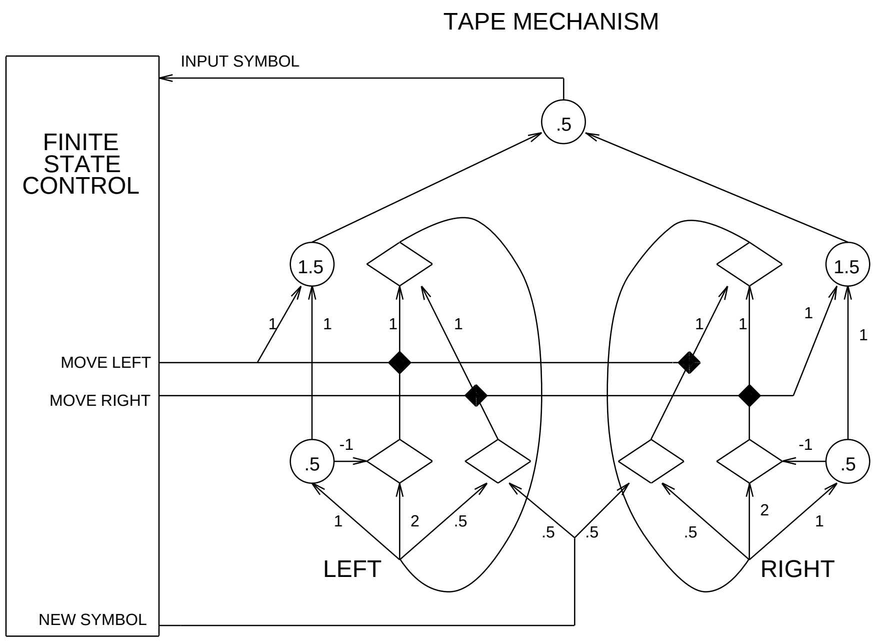

Figure 3. *A tape mechanism which uses two bit-stacks to represent the left and right portion of a TM tape. Depending on the direction to move, the stacks are either popped or have the new symbol pushed, and this action is selected by multiplicatively gated connections shown as small diamonds.*

logic is used to select the input symbol for the next move of the machine. These two signals are also used to multiplicatively gate (shown by small diamonds) the representations of the left and right stacks.

4.6. Multiplicative Connections

The purpose of the the exercise was to find a set of ‘‘changeable parts’’ which could achieve effective computation in the connectionist framework. Taken together, fractional values, linear combinations, thresholds, and multiplicative connections are sufficient for computing. The first three elements are quite standard connectionist fare, but the multiplicative connections are a rarely used element in such models.

The need for these multiplicative connections, arises because of a Turing machine’s ability to move both left and right on the tape -- Turing machines which move in only one direction are only finite-state transducers.

Finally, it should be noted that arbitrarily resolved fractional values are not very practical. It is not suggested that such precise numbers, or the particular single-bit stack mechanism, be widely adopted in connectionist models. We believe, however, that even when a more reasonable bound on analog values is imposed, multiplicative connections remain a critical, and underappreciated, component for neurally-inspired computing.

In simple terms, a normal connectionist network may be thought of as having a state vector $V_i(t)$ which evolves over time as a function of the fixed weights in the system, W_{ji}:

$$V_i(t+1) = f(W_{ij} V_j(t))$$

A multiplicative system uses a three-dimensional array of weights, and involves multiplying the state vector together:

$$V_i(t+1) = f(W_{ijk} V_k(t) V_j(t))$$

which can be rewritten as a system whose configuration (weights) change dynamically over time:

$$W_{ij}(t) = W_{ijk} V_k(t)$$

$$V_i(t+1) = f(W_{ij}(t) V_j(t))$$

So multiplicative, or higher-order, connections are useful both for the gating of analog values, and for getting a form of dynamic reconfiguration into a connectionist model. Various researchers have proposed that the state of one unit can be used to continuously modify the weight between another pair of units. For example, (Hinton, 1981b) proposed using multiplicative connections in a system for object recognition independent of size or rotation; (Feldman & Ballard, 1982) described them as *variable mappings* which reduce the number of conjunctive connections in certain models; (McClelland, 1985) proposed such a system for reducing the amount of resource duplication in a model of letter perception; and (Lapedes & Farber, 1985) have applied them to a master-slave topology of Hopfield networks.

Unfortunately, the complexity of programming or controlling such a network is very difficult. In a simple, static network of n units, one only has to ‘‘program’’ or ‘‘learn’’ at most n^2 weights. But in a system where the states of the n units can contribute to the n^2 weights, one has to deal with n^3 weights. This is perhaps why many researchers have given up on using multiplicative connections (McClelland, Personal Communication).

We believe that the current state of connectionist architectures is akin to the first general-purpose computers of the 1940’s. Because programs were developed on patchboards, there was no connection between ‘‘program space’’ and ‘‘data space’’ so languages, interpreters, and compilers could not exist. The major breakthrough came with the advent of stored-program computers, where the program could be stored in the same memory as data, and thus could be manipulated.

Multiplicative connections add this element to connectionist architectures, a short-circuit between the program space of weights and the data space of (activation) states.

This analogy can be extended further. The effective power[10] that this merging of spaces allowed was immediately subject to the abuses of self-modifying code and branching into data; practices that led to unstable systems. Connectionist networks using multiplicative connections are also unstable, because it is difficult to relax when your environment is dynamically changing.

The abuse of power ultimately creates restrictions on its use. In the stored program computer, languages were developed which both restricted branching by allowing only structured programming, and removed the possibility of self-modification. These crucial powers are only granted to operating systems, interpreted languages (such as LISP and APL), and loaders. Similarly, restrictions on the use of multiplicative connections will be necessary in order to build powerful, but stable, general purpose connectionist systems.

The next chapter describes experiments with one such ‘‘programming convention.’’

Ackley, D. H., Hinton, G. E. & Sejnowski, T. J. (1985). A learning algorithm for Boltzmann Machines. *Cognitive Science, 9*, 147-169.

Allen, R. (1987). Several Studies on Natural Language and Back Propagation. In *Institute of Electrical and Electronics Engineers First International Conference on Neural Networks*. San Diego, II-335-342.

Backus, J. (1978). Can programming be liberated from the von Neumann style? A functional style and its algebra of programs.. *Communications of the Association for Computing Machinery, 21*, 613-641.

Blelloch, G. & Rosenberg, C. (1987). Network Learning on the Connection Machine. In *Proceedings of the Tenth International Joint Conference on Artificial*

[10] We use the term ‘‘effective power’’ since a non-stored-program machine is just as theoretically powerful as a stored program machine.

Intelligence. Milan, 323-326.

Cottrell, G. W. (1985). Connectionist Parsing. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*. Irvine, CA.

Cottrell, G. W. (1985). A Connectionist Approach to Word-Sense Disambiguation. TR154, Rochester: University of Rochester, Computer Science Department.

Fanty, M. (1985). Context-free parsing in Connectionist Networks. TR174, Rochester, N.Y.: University of Rochester, Computer Science Department.

Feldman, J. A. & Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science, 6*, 205-254.

Hanson, S. J. & Kegl, J. (1987). PARSNIP: A connectionist network that learns natural language grammar from exposure to natural language sentences. In *Proceedings of the Ninth Conference of the Cognitive Science Society*. Seattle, 106-119.

Hinton, G. E. (1981). Implementing semantic networks in parallel hardware. In G. E. Hinton & J. A. Anderson, (Eds.), *Parallel models of associative memory*. Hillsdale: Lawrence Erlbaum Associates.

Hinton, G. E. (1981). A Parallel Computation that Assigns Canonical Object-Based Frames of Reference. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*. Vancouver, B.C., 683-685.

Hinton, G. E. (1984). Distributed Representations. CMU-CS-84-157, Pittsburgh, PA: Carnegie-Mellon University, Computer Science Department.

Kawamoto, A. H. (1985). Dynamic Processes in the (Re)Solution of Lexical Ambiguity. Doctoral Dissertation, Providence: Department of Psychology, Brown University.

Kay, M. (1973). The MIND System. In Rustin, (Ed.), *Natural Language Processing*. New York: Algorithmics Press.

Lapedes, A. S. & Farber, R. M. (1985). A self-optimizing, nonsymmetrical neural net for content-addressable memory and pattern recognition. LA-UR-85-4037: Los Alamos National Laboratory.

McClelland, J. L. (1985). Putting Knowledge in its Place. *Cognitive Science*, *9*, 113-146.

McClelland, J. & Kawamoto, A. (1986). Mechanisms of Sentence Processing: Assigning Roles to Constituents. In J. L. McClelland, D. E. Rumelhart & the PDP research Group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, Vol. 2. Cambridge: MIT Press.

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*, 115-133.

Minsky, M. (1972). *Computation: Finite and Infinite Machines*. Cambridge, MA: MIT Press.

Minsky, M. & Papert, S. (1988). *Perceptrons*. Cambridge, MA: MIT Press.

Plate, T. (1987). A design for the simulation of Connectionist models on coarse-grained parallel computers. MCCS-87-106, Las Cruces: Computing Research Laboratory, New Mexico State University.

Pomerleau, D. A, Gusciora, G. l., Touretzky, D. S. & Kung, H. T. (1988). Neural Network Simulation at Warp Speed. In *Proceeedings of the Institute of Electrical and Electronics Engineers 1988 Conference on Neural Networks*. San Diego.

Rosenblatt, F. (1962). *Principles of Neurodynamics*. New York: Spartan.

Rumelhart, D. E., Hinton, G. E. & McClelland, J. L. (1986). A General Framework for Parallel Distributed Processing. In D. E. Rumelhart, J. L. McClelland & the PDP research Group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, Vol. 1. Cambridge: MIT Press.

Rumelhart, D. E., Hinton, G. & Williams, R. (1986). Learning Internal Representations through Error Propagation. In D. E. Rumelhart, J. L. McClelland & the PDP research Group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, Vol. 1. Cambridge: MIT Press.

Rumelhart, D. E. & McClelland, J. L. (1986). On Learning the Past Tenses of English Verbs. In J. L. McClelland, D. E. Rumelhart & the PDP research Group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, Vol. 2. Cambridge: MIT Press.

Rumelhart, D. E. & McClelland, J. L. (1986). PDP Models and General Issues in Cognitive Science. In D. E. Rumelhart, J. L. McClelland & the PDP research Group, (Eds.), *Parallel Distributed Processing: Experiments in the Microstructure of Cognition*, Vol. 1. Cambridge: MIT Press.

Selman, B. (1985). Rule-Based Processing in a Connectionist System for Natural Language Understanding. CSRI-168, Toronto, Canada: University of Toronto, Computer Systems Research Institute.

Touretzky, D. S. & Hinton, G. E. (1985). Symbols among the neurons: details of a connectionist inference architecture. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*. Los Angeles, CA.

Touretzky, D. S. (1986). Representing and transforming recursive objects in a neural network, or ‘‘trees do grow on Boltzmann machines’’. In *Proceedings of the 1986 Institute of Electrical and Electronics Engineers International Conference on Systems, Man, and Cybernetics*. Atlanta, GA.

Touretzky, D. S. (1986). BoltzCONS: Reconciling connectionism with the recursive nature of stacks and trees. In *Proceedings of the 8th Annual Conference of the Cognitive Science Society*. Amherst, MA, 522-530.