MATERIAL HANDLING: A CONSERVATIVE DOMAIN FOR NEURAL CONNECTIVITY AND PROPAGATION

H. Van Dyke Parunak, James Kindrick, Bruce Irish

Industrial Technology Institute
P.O. Box 1485
Ann Arbor, MI 48106
(313) 769-4000

Abstract: Two important components of connectionist models are the connectivity between units and the propagation rule for mapping outputs of units to inputs of units. The biological domains where these models are usually applied are nonconservative, in that a single output signal produced by one unit can become the input to zero, one, or many subsequent units. The connectivity matrices and propagation rules common in these domains reflect this nonconservativism in both learning and performance.

CASCADE is a connectionist system for performing material handling in a discrete parts manufacturing environment. We have described elsewhere the architecture and implementation of CASCADE [PARU86a] and its formal correspondence [PARU86c], [PARU87a] with the PDP model [RUME86]. The signals that CASCADE passes between units correpond to discrete physical objects, and thus must obey certain conservation laws not observed by conventional neural architectures.

This paper briefly reviews the problem domain and the connectionist structure of CASCADE, describes CASCADE's scheme for maintaining connectivity information and propagating signals, and reports some experiments with the system.

1. The Domain of Material Handling

Primitive factory operators fall into two classes, Material Handling and Processing. CASCADE formalizes and extends previous approaches to Material Handling in the context of a connectionist architecture.

1.1. Manufacturing = Processing + Material Handling

Algebraically, a factory making discrete goods applies a series of state-changing operators to inventory. Some state components, such as shape, hardness, and color, reflect the part's specification, and make up its *functional state*. Other components of state, such as a part's location in the plant or the length of time it has been there, are irrelevant to its function, and make up its *non-functional state*.

For each state component, there is a primitive operator that changes only that component. Most processing machines implement complex operators that correspond to the composition of several primitive operators. For example, a painting robot changes a part's color, and also its size, its weight, and (because of the time consumed by the operation) its age.

"Processing" is the set of all primitive operators that change a part's functional state. "Material Handling" is the set of all primitive operators that change a part's non-functional state. Material handling thus includes the traditional functions of moving material between workstations (changing its location), and storing it in a warehouse (changing its age). It also includes transportation and aging that occur as components of a complex operator, such as drying paint by moving a part through a heated tunnel. Thus material handling occurs in almost every machine in a plant, as parts and requests for parts move back and forth.

This model of manufacturing represents only one view of a complex enterprise ([PARU87b], [FOX83]), but an important view economically. If material does not reach machines fast enough to keep them busy, productivity suffers, but if excessive work-in-process inventory (WIP) accumulates, carrying costs, response time to customer orders, and scrap due to engineering changes all increase. Monitoring and controlling WIP is not managed well by raw human intelligence, and represents a major locus of interest in the manufacturing community.

1.2. Previous Work

CASCADE is inspired by the Japanese KANBAN system of inventory flow control [HALL81], [SCHO82], [SUGI77]. In KANBAN, a set of modular workstations request parts from one another by passing tickets back and forth. The factory relaxes into a steady state of production determined by the performance of the workstations and the number of tickets in circulation.

KANBAN performs well for a factory with a stable production schedule, where the flow of parts remains in a steady state for a long period of time. Its behavior rapidly deteriorates, though, when the loading and product mix of the shop change frequently.

KANBAN resembles a neural net model in which machines correspond to neurons, transport links correspond to connections, and parts and requests correspond to neural impulses. CASCADE formalizes the connectionism implicit in KANBAN, and extends it to support the changeability of a flexible manufacturing environment.

2. A Model for Material Handling

We generally follow the *Parallel Distributed Processing* (PDP) model [RUME86]. [PARU86c], [PARU87a] relate CASCADE to PDP formally.

2.1. Objects in CASCADE

CASCADE manipulates three basic kinds of objects.

A *container* represents a parcel of material that can be moved and stored as an entity. Containers are strongly typed on the basis of the functional state of their contents, and two containers of the same type are interchangeable in the manufacturing process is concerned.

A *unit* is a collection of containers that are geographically and functionally related. There are two types of units, corresponding to two types of relations between containers.

1. If all the containers in a unit have the same type, the unit is a *TOMP*. No functional processing can take place in such a unit, otherwise incoming containers would differ in type from outgoing ones.

2. If the containers in a unit are participants in the same functional operation, the unit is a *process*. A process consumes containers of zero or more types and produces containers of zero or more types. It differs from a TOMP in two ways. Its behavior is stochastic rather than deterministic (at least from the perspective of the material handling system), and its output can be of a different type than its input.

Every container belongs to exactly one unit at a time, and moves from unit to unit as the system operates.

An *aggregate* is a collection of units that are in the same geographical area. There are two types of aggregates, corresponding to the two types of units.

1. A *mover* is a collection of TOMP's, and models a single geographically limited material handling module, such as a conveyor loop, a zone of an AGV (automatic guided vehicle) system, or an ASRS (automatic storage and retrieval system).

2. A *workstation* is a collection of processes that run on a single geographically local and functionally integrated set of machines (typically, a machine tool with associated transfer and inspection mechanisms).

Every unit belongs to exactly one aggregate, and retains this association unless the system is reconfigured.

2.2. Messages Among Units

Each TOMP has a maximum and a minimum capacity for containers of its type, and is connected to one or more other units on adjacent aggregates. If its population (the number of containers it contains) exceeds its maximum capacity, it seeks to spill the excess to a neighboring unit. If its population falls below its minimum capacity, it seeks to fill up to the minimum from its neighbors. Thus the units form a network through which containers and requests for containers propagate.

This behavior results in a mechanism that is a superset of KANBAN. We can set capacities to make TOMP's behave like links in a traditional KANBAN system [PARU86a]. However, because we can push as well as pull material through a CASCADE net, we can distribute material ahead of time to anticipate changes in production requirements, and thus avoid KANBAN's problems when production is not steady-state.

The requests and acquisitions that result from fills and spills are local messages, allowing one TOMP to propagate its constraints to its nearest neighbors. The TOMP's correspond to neurons in a neural net, while requests and containers correspond to inter-neural impulses and capacities correspond to neural threshholds.

Experiments with CASCADE [PARU86c] show that it does control WIP levels and reduces waiting time for parts at machines.

3. Connectivity and Propagation in CASCADE

One motive for developing the parallel between material handling and connectionist models is the adaptive nature of such models. These models learn by modifying the weights in their connectivity matrices. Weights can be modified locally, on the basis of the experience of an individual unit, without invoking a global "learning module" that knows the state of the entire system. Such a local learning scheme offers great promise in coping with the complexity of a large material handling network in a flexible manufacturing environment, where part types, quantities, and distribution are continually changing.

This section develops the notion of conservation in propagation rules, describes CASCADE's implementation of such rules, and reports some simple experiments with the system.

3.1. Conservation in Propagation Rules

In the PDP model, signals travel from unit to unit through connections. A propagation rule determines whether the output from one unit reaches the input of another. In the simplest propagation rules, the vector that represents the inputs to each unit is computed as the product of the output vector and a connectivity matrix, and a single output can contribute to many inputs, or to none. That is, neural propagation does not conserve impulses. It can effectively multiply a single impulse by routing it to many inputs, or destroy an impulse by not passing it anywhere.

A system can differ from this standard model in two ways.

1. It might require *quantitative* conservation of signals, so that the total output at one time step equals the total strength input at the next. This constraint can be implemented by normalizing each column in the connectivity matrix so that total output equals total input.

2. It might require *qualitative* conservation, in which signals are discrete packets that must be propagated intact. This constraint can be implemented by interpreting the weights, not as shares into which signals are divided, but as the probability that a packet from one unit arrives at another.

CASCADE exhibits both quantitative and qualitative conservation. If a container leaves one unit, the same container must arrive at precisely one unit. Thus the weights in each column of the connectivity matrix for containers in CASCADE sum to one, and each is interpreted as the probability that a container from the source will go to the target.

Requests are not subject to the same physical constraints that containers are. In our system, though, each request results in the delivery of a container. If propagation multiplies requests, the system as a whole will send many more containers than needed toward the node that initiated the request, and the rest of the network will starve for that type of container. Thus, we require propagation to conserve requests as well as containers.

3.2. Implementing Connectivity and Propagation

One can model connectivity in CASCADE as weight matrices interpreted probabilistically, as outlined above. Containers and requests have distinct matrices, reflecting different connectivities. In our implementation, each unit stores its column of each of the matrices, $\vec{c}$ for container connectivity and $\vec{r}$ for request connectivity. c_i is the probability that the next container spilled

from this unit will be sent to unit i, and r_i is the probability that the next request for a container will be sent to unit i. The total number of units is n, and the probability interpretation requires

$$\sum_{i=0}^{n-1} c_i = \sum_{i=0}^{n-1} r_i = 1$$

Entries in $\vec{c}$ and $\vec{r}$ can be nonzero only if physical connections exist between the associated aggregates, and in most installations these connections do not vary dynamically, so in practice these vectors are sparse and only their nonzero elements need be manipulated.

These vectors are the locus both of propagation decisions and of monitoring changes in the environment.

3.2.1. Propagation Decisions

When a unit is ready to spill or fill, it generates a random number $0 < r \leq 1$, and uses it to select an element from the appropriate vector. This element then becomes the target of the request or container being output. For instance, a spilling unit sends its excess container to c_j such that

$$\sum_{i=0}^{j-1} c_i < r \leq \sum_{i=0}^{j} c_i$$

3.2.2. Modifying the Vectors

Some principles for managing the connectivity vectors in a unit are the same for both container and request connectivities, and apply in general to any connectionist architecture that requires a conservative propagation rule.

- Bayesians will set the initial values in the connectivity vectors for a unit on the basis of their *a prioris*. Others will probably set them to $1/n$, dividing the probability equally among them.

- As the system learns, certain weights change, and the remaining values in the vector must be adjusted in the opposite direction to keep the total at 1. One computationally simple strategy is to adjust the desired entry by a, then divide every element in the vector by $1 + a$.

- As individual weights approach zero, the associated units are for all practical purposes disconnected from the sending unit. Once "forgotten" in this way, they will never be selected. It is easy to show that if the desired containers exist in the system, allowing connection weights to attain zero value insures that requests will be satisfied in bounded time. In applications where this irrevocable forgetting is undesirable, we set maximum and minimum limits beyond which a unit's weight is not adjusted.

The actual adjustments made to individual weights are determined differently for containers and requests. The protocols outlined here reflect the semantics of our application, and may be different for a different application.

The container vector $\vec{c}$ for a unit records the probability of selecting each of that unit's neighbors as the recipient of a spilled container. The weights reflect our judgment that a given neighbor desires containers. Two events affect this judgment.

1. When a neighbor requests a container, we know that it has some interest in receiving containers. So when a unit receives a request, it augments the container weight for the requesting neighbor (say, by R).

2. When one unit spills a container to another, the receiving neighbor is less likely to need one than it was before the spill. So its weight in the sending unit should be decremented (say, by S).

The sizes of adjustments for receiving a request or delivering a spill are tuning parameters. Their ratio R/S reflects the impact of a single request. If this ratio is greater than one, the spilling unit interprets a single request as expressing a relatively long-term interest in receiving containers. If it is less than one, the spilling unit attaches much less significance to the long-term implications of a single request. The average value of R and S reflects how quickly the spilling unit shifts its attention to a requesting unit.

The request vector $\vec{r}$ for a unit records the probability of selecting each of that unit's neighbors as the recipient of a request issued by the unit. The weights reflect our judgment that a given neighbor has a container in stock, or has access to a container from one of its neighbors. We modify these weights on the basis of the neighbors' cost of effort (COE) in filling past requests. In the present implementation, the COE is the number of units that were searched to find a container. Each time a neighbor successfully satisfies a request, we augment its weight (in the current implementation, by U/COE, where U is a constant). Each time a neighbor fails to satisfy a request, we decrement its weight (in our implementation, by a constant V). Again, the average value of U and V determines the stability and rate of learning of the system, while their ratio controls whether success or failure has more impact on the learned behavior.

This back-propagation error-correction algorithm is similar to that used in the perceptron convergence theorem [ROSE62]. It differs from the classical procedure in two main ways.

1. Traditional systems distinguish the learning and performance phases. In the learning phase, the system converges to a stable set of connection weights that produce a desired output pattern by modifying those weights using back-propagation. During the performance phase, weights are not modified and the network is no longer adaptive. Our problem domain requires continual adaptive behavior, merging the two phases by modifying connection weights during performance. This strategy is necessary since the desired output pattern is not fixed, but continually varies as containers move through the system.

2. The classical approach modifies connection weights proportional to the magnitude of error, the difference between output produced and desired output. In our system the desired output pattern is variable, so the traditional notion of error is not well defined. The same activation level may be an error during one trial and correct during another. Therefore we apply a constant negative reinforcement on error. Our system propagates back the cost of achieving a success rather than the degree of failure. The magnitude of connection weight adjustment is proportional to this COE of success.

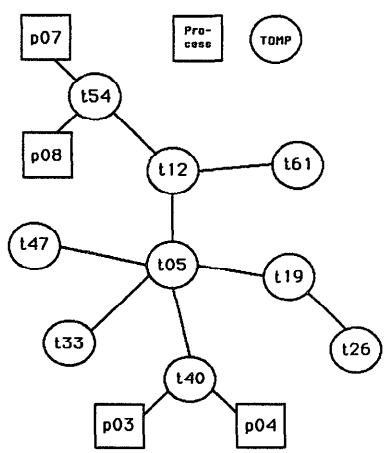
The depth to which search proceeds before reporting failure, and the number of trials that a unit makes before reporting failure, are

parameters of the model. In the implementation described here, the search proceeds depth-first, and each unit tries only one neighbor before reporting success or failure.

3.3. Experimental Results

Figure 1 shows the connectivity of some of the units (TOMP's and processes) in ITI's Advanced Manufacturing Center, a working CIM cell in which CASCADE is implemented. These particular units manipulate empty boxes into which processes *p07* and *p08* pack finished parts. To demonstrate the system, we load TOMP *t26* (an ASRS) with 15 empty boxes, and begin retrieving from *p07* and *p08*. When the 15 boxes from *t26* are gone, we load 15 more at *p04* (a process that loads inventory at a manual workstation). For each retrieval we record the cost of effort (COE), the number of units that CASCADE searched to find the requested part. The minimum COE possible is 6 from *t26* and 5 from *p04*.

Figure 1: "Empty Box" Units



As a control run, we assign probability $1/n$ to each connection in each TOMP, and do not vary these weights as the system operates. Thus later retrievals do not learn from the success or failure of earlier ones. Figure 2 shows the COE as a function of retrieval. The variability results solely from the stochastic search process. The median COE to retrieve a box is 22.5, the inter-quartile spread is 38.5, and the standard deviation is 28.9.

Figure 2: Retrievals Without Learning

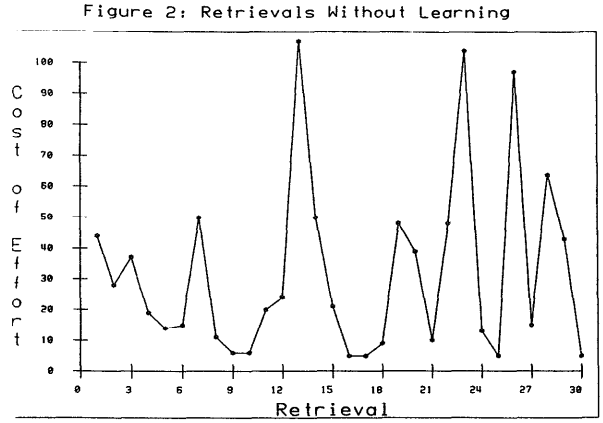
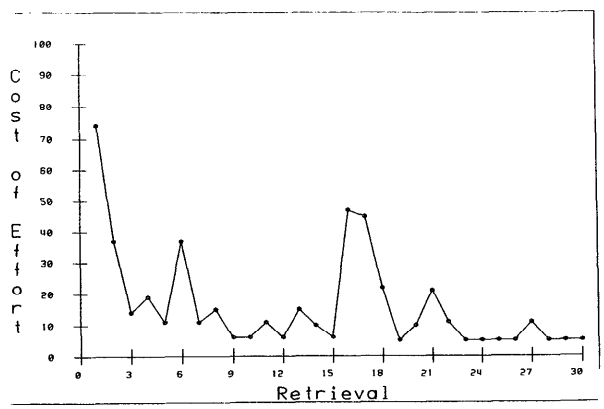


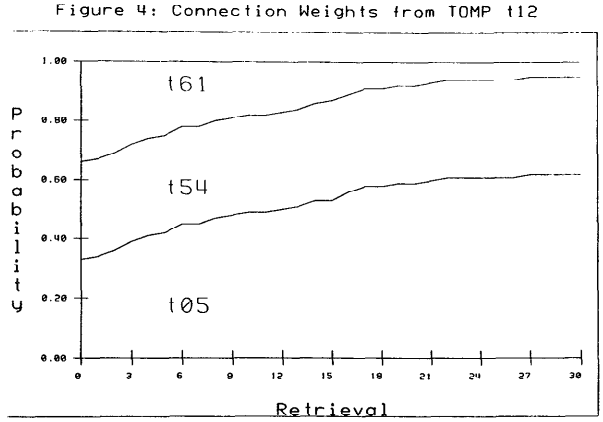
Figure 3 shows COE as a function of retrieval when the weights are allowed to vary in response to success and failure. For this trial, we augment the weight of a successful neighbor by 0.1/COE, and decremented the weight of an unsuccessful one by 0.01. The COE drops as the system learns to go to *t26* for boxes. Limited exploration of other units continues at trials 11, 13, and 14, even after focusing on *t26*. The COE rises when the boxes run out at *t26*, then drops as the system discovers the new source at *p04*. Over the entire run, the median COE is 11, the inter-quartile spread is 13, and the standard deviation is 16.

Figure 3: Retrievals With Learning



Figures 4 and 5 show how the probability assigned to each neighbor of TOMP's *t12* and *t05*, respectively, changes during the 30 retrievals described in Figure 3. These TOMP's are the major decision points in the system, and the weights of the connections between them and their neighbors are the main locus of learning in this experiment. For each retrieval, the vertical space between the x axis and the line at $y = 1.00$ represents unit probability, and is divided into as many bands as the TOMP under consideration has neighbors. The relative width of each band shows the weight of the connection to the associated neighbor.

Figure 4: Connection Weights from TOMP t12



For instance, Figure 4 shows three horizontal bands, one each for *t54*, *t05*, and *t61*. In our experiment, requests all enter from *t54*, so it never succeeds or fails, and its probability remains constant. The weight of the connection to *t05* increases through normalization when *t61* fails, and increases through augmentation when it succeeds, so it increases monotonically. Similarly, the weight of the connection to *t61* falls monotonically. Since *t05* is on the route to both sources of empty boxes, it grows both before and after the switch from *t26* to *p04*.

Figure 5: Connection Weights from TOMP 105

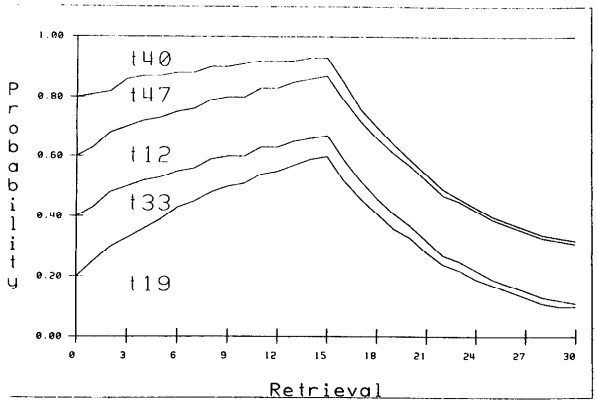


Figure 5 tells a similar story. Now the weight of *t12* remains constant, since it does not participate in the competition. As long as *t26* has boxes, *t19* (which leads to it) gains weight. When we switch to *p04*, *t19* rapidly loses weight and *t40* (now on the correct route) becomes prominent.

4. Summary

CASCADE uses a connectionist model to manage the distribution and movement of inventory in a discrete manufacturing environment. It differs from traditional neural models in conserving its signals as they propagate between units. We have described a scheme that adjusts the connectivity of such a network dynamically and propagates signals with the required conservation.

Ongoing research is probing several directions.

- This system modifies weights as it uses them, and so merges learning and performance. Because of the conservation characteristics of the application, propagation of requests and containers is strongly serial, and a failed request is an expensive way to learn. Under some circumstances, it may be advantageous to add a separate parallel search of the network to set weights from time to time. In the multiprocessor environments for which CASCADE is intended, such a phase reduces the cost of futile requests.

- A flexible manufacturing environment favors integration of learning and performance for material handling because the distribution of supply and demand for inventory varies continuously. Probably, though, certain distribution states recur periodically, due to repeated runs of the same parts and customer order cycles. We can store the weights periodically as a function of an estimator of system state, and then retrieve them to shorten the adaptation time.

- Units in CASCADE can learn by modifying their threshholds as well as their connectivity.

Some of the original ideas in CASCADE originated in discussions with Bob Judd. The work described in this report was financed by a grant from the Kellogg Foundation.

References

[FOX83] Fox, M., 1983. "Constraint-Directed Search: A Case Study of Job-Shop Scheduling." Carnegie-Mellon University: Robotics Institute CMU-RI-TR-83-22; Computer Science Department CMU-CS-83-161.

[FOX85] Fox, B.R.; and K.G. Kempf, 1985. "Complexity, Uncertainty, and Opportunistic Scheduling." *Second IEEE Conference on Artificial Intelligence Applications*, Miami, FL, 487-492.

[HALL81] Hall, R.W., 1981. *Driving the Productivity Machine.* American Production and Inventory Control Society.

[PARU85a] Parunak, H.V.D.; B.W. Irish; J. Kindrick; and P.W. Lozo, 1985. "Fractal Actors for Distributed Manufacturing Control." *Proceedings of the Second IEEE Conference on AI Applications.*

[PARU86a] Parunak, H.V.D.; P.W. Lozo; R. Judd; B.W. Irish; J. Kindrick, 1986. "A Distributed Heuristic Strategy for Material Transportation." *Proceedings of the 1986 Conference on Intelligent Systems and Machines*, Oakland University, Rochester, MI.

[PARU86b] Parunak, H.V.D.; J.F. White; P.W. Lozo; R. Judd; B.W. Irish; J. Kindrick, 1986. "An Architecture for Heuristic Factory Control." *Proceedings of the 1986 American Control Conference.*

[PARU86c] Parunak, H.V.D., and James Kindrick, "A Connectionist Model for Material Handling." Presented at the Seventh DAI Workshop, Gloucester, MA, Oct. 1986.

[PARU87a] Parunak, H.V.D., James Kindrick, and Bruce W. Irish, "A Manufacturing Application of a Neural Model." Submitted to IJCAI-87.

[PARU87b] Parunak, H.V.D., and John F. White, "A Framework for Comparing CIM Reference Models." Spring 1987 Meeting, International Purdue Workshop on Industrial Computer Systems, April, 1987.

[REES86] Rees, L.P.; and P.R. Philipoom, 1986. "Dynamically Adjusting the Number of Kanbans in a Just-In-Time Production System." 1986 S.E. AIDS (prepublication draft).

[ROSE62] Rosenblatt, F., 1962. *Principles of Neurodynamics.* New York: Spartan.

[RUME86] Rumelhart, D.E.; G.E. Hinton; and J.L. McClelland, 1986. "A General Framework for Parallel Distributed Processing." In Rumelhart and McClelland, eds., *Parallel Distributed Processing*, Cambridge: MIT Press, 1.45-76.

[SCHO82] R.J. Schonberger, 1982. *Japanese Manufacturing Techniques.* New York: The Free Press.

[SUGI77] Sugimori, Y.; K. Kusunoki; F. Cho; and S. Uchikawa, 1977. "Toyota production system and Kanban system: Materialization of just-in-time and respect-for-human system." *International Journal of Production Research* **15:6**, 553-564.