Research Report AI–1991–01
SALMON: a TEMPERAMENTAL Program that Learns

Gregg H. Rosenberg

Artificial Intelligence Programs
The University of Georgia
Athens, Georgia 30602 U.S.A.

February 2, 1994

Abstract

A new approach to machine learning is introduced that utilizes semantic information in a connectionist network. The approach is implemented in a program that learns to act appropriately in the dynamic environment of a children's game of tag. The model is interesting in several respects including the ability to begin with no connections and then make and break them according to its experience, the ability to adjust the weights on its connections and the ability to interact with its environment.

1 Introduction

This thesis introduces a new approach to machine learning that uses aspects of semantic and connectionist systems. The name of the programming

technique is **TEMPERAMENTAL** programming. TEMPERAMENTAL stands for **T**he **E**ffective **M**anagement of **P**rocess **E**volution and **R**esponse in an **A**ssociative **M**emory **E**mulating a **N**eural **T**ype **A**ction **L**anguage. The algorithm has been implemented successfully as a program named **SALMON** (**S**emantic **A**ction **L**earner **M**odeled **O**n **N**eurons) that learned an appropriate strategy for a simulated game of tag from watching the other participants. The main features of interest in a TEMPERAMENTAL program are:

1. The system is a self-organizing learner which uses experience in a dynamic environment to learn how to react appropriately in it.

2. The learner begins with only unconnected semantic nodes then makes and breaks connections between them based on what occurs in the environment.

3. The learner adjusts the weights of connections to conform to its experiences in the environment.

4. Both the nodes and connections have semantic meaning, but the information passed through the connections is numerical Π la the connectionist paradigm.

5. Despite its semantics, the algorithm is potentially as massively parallel as other connectionist schemes.

Sections two and three describe TEMPERAMENTAL programming. Section two contains information concerning the components of a TEMPERAMENTAL system, and how TEMPERAMENTAL programs learn. Section three details how activation is spread in a TEMPERAMENTAL network and how the results are interpreted. Sections four through six deal with different aspects of the implementation, section seven analyzes the results obtained from the implementation, and section eight is a general discussion of issues raised by TEMPERAMENTAL programming and possible directions for future work.

2 TEMPERAMENTAL programming

TEMPERAMENTAL programming is a hybrid scheme which uses the interpretive power of semantics and the pattern recognition and parallelism of connectionist strategies to learn about a dynamic environment. It connects nodes with semantic meaning using links that also have semantic meaning, but weights the links numerically and has them activate the nodes in parallel. Once trained, the activation and the semantics cooperate by combining their knowledge to decide upon an action given the current circumstances. The key ideas taken from connectionism are parallelism and communication through numeric activation rather than semantic message passing. The key ideas taken from semantics are consistent interpretation of symbols and procedural interaction with an environment.

2.1 Node Types

There are four types of nodes in a TEMPERAMENTAL system: objects, dynamic attributes, actions and action attributes.

1. Objects — each object in the environment is assigned a node with a unique label. Objects have a full repertoire of connectionist behaviors, these being the reception, accumulation, and passing along of activation energy.

2. Dynamic Attributes — Each environment has a set of attributes, like a particular location, that an object may or may not possess. These are called dynamic attributes and a node with a unique label is assigned to each attribute. Dynamic attributes act as conduits for activation from objects to other nodes but do not accumulate activation themselves.

3. Actions — An action is a change to an attribute or the value of an attribute of one or more objects. Each action in the environment is given a node with a unique label. Like objects, action nodes have the full repertoire of connectionist behaviors including reception, accumulation

and transference of activation. In addition, action nodes have a procedure and a function associated with them. The function accepts an object and determines if the action is enabled with respect to that object. The procedure contains the semantic instructions for performing the action when the node is activated.

4. Action Attributes — Many actions can be performed in more than one way. For instance, a player in a game of tag must decide not only to flee, but also in which direction to flee. The direction can be thought of as an attribute of the action. One node with a unique label is assigned to each possible attribute of each action defined. The attribute is explicitly bound to the action it is responsible for helping to perform. One attribute may belong to many actions. The action attributes accumulate activation but do not transfer it.

2.2 Connection Types

Connectionist networks normally define their connections by the type of activation that they convey. Thus the connections are either inhibitory or excitory. Semantic systems usually define their connections by the type of relationship that they represent. Thus, we have IS-A links, HAVE links, MEMBER links and so forth. TEMPERAMENTAL programs do both. Each connection is assigned a semantic meaning used in making the connection and in adjusting its weights. It is also assigned a role in spreading activation that defines it as either a companion, excitory, or enemy connection. Finally, each connection is assigned a directionality. Being a companion or an enemy connection in a TEMPERAMENTAL system is not the same thing as being inhibitory or excitory in a normal connectionist network. Companion connections between two nodes represent, with a certain strength, the possibility that the two nodes will co- activate. Enemy connections represent the possibility, with a certain strength, that two nodes will not co-activate. Therefore, whether a connection passes along an inhibitory or an excitory signal depends not only on what kind of connection it is, but also on what kind of activation the node received. If a node receives an excitory signal then it would send an excitory signal along its companion connections. But if the signal received

by the node was inhibitory, then it would send inhibitory signals along its companion connections. In general we can think of the activation received by a node as containing a message. Excitory activations contain the message that, "You are my friend." Inhibitory activations contain the message that, "You are my enemy." The goal is not to pass the form of the message along intact, but rather the content. Therefore, we can follow these old proverbs:

```
"The friend of my friend is my friend."
"The enemy of my friend is my enemy."
"The friend of my enemy is my enemy."
"The enemy of my enemy is my friend."
```

After running experiments with the system I found that it was absolutely necessary for there to be strictly excitory connections as well as companion connections. The justification can be understood by imagining this situation: Let's say that a friend of yours has the peculiar habit of walking backwards every once in a while. After he walks backwards he always walks forwards again. Now, of course you will get an expectation that he will walk backwards from seeing him walk backwards, but from seeing him walk forwards again you also will get a negative expectation (called an exclusive connection in TEMPERAMENTAL programming) that he will walk backwards (these connections are talked about soon). Further, he always walks forwards after he walks backwards so you get a very strong positive expectation that this will be the sequence of events. Now, since he does not walk backwards very often, your negative expectation for his walking backwards grows very strong, stronger than your positive expectation that he will walk backwards. When wondering whether your friend will walk forwards or backwards, which will you say? Intuitively, you will say that he will walk forwards. But imagine that the expectation for the sequencing is a companion connection as outlined above. There is a very strong negative expectation that gets fed to walking backwards, and walking backwards has a very strong companion connection to walking forwards through sequencing. This companion connection is a path that will convey this strong negative expectation about walking backwards to walking forward. Because the weight of the sequence connection is so strong the negative energy is not diminished very much as it moves along the path, and it may even be enough to cause you to have more belief that

he will walk backwards than you do that he will walk forward.

That is all wrong! Just because you don't believe something will happen doesn't mean that you also believe what usually follows it won't happen. You remain neutral regarding what will follow it. If the preceding action A occurs, then it raises your expectation that the following action B will occur, but the unlikelihood of A merely means that B will have to get activation from some other source if it is to compete with your other expectations. Therefore, your sequencing expectations are not companion but strictly excitory. Another way of putting it is to remember that companion connections represent co-activation, but sequences are serial activations, not co-activations. Therefore, sequence expectations are not companions.

In the following discussion the terms "giver node" and "giving node" refer to the node which passes activation along in the connection. The terms "receiver node" and "receiving node" refer to the node which acquires activation from the connection. In two-way connections both nodes are givers and also receivers.

1) Expectation Connections — Expectation connections are formed between objects and the actions that they perform. The heuristic is, if an object is performing an action then establish an expectation connection between the object and the action. Expectation connections are two-way and companion.

2) Exclusive Connections — Of course, sometimes attributes an object possesses make it less likely to perform an action. Exclusive connections try to capture these kinds of relationships. Formation of exclusive connections is motivated by an object changing the action it is performing. When an object stops performing an action, make an exclusive connection between each attribute the object had at the moment it stopped and the action no longer being performed. Also make a connection between the object and the action. The idea is that the attribute responsible will eventually emerge as the connection is re-enforced through the attribute's continual presence when the action is not being performed and absence when it is.

3) Focus Connections — The idea of a focus is currently ill defined. Many actions seem to have them, though. If one object is pursuing another, the

focus of the action is the object being pursued. However, a focus relation is not symmetric. A zebra may flee a lion even though the lion is not concerned with the zebra at all; thus the lion is the focus of the zebra's action but the zebra is not the focus of the lion's action. Vaguely, we can think of the focus of an action as being the object which motivates it. The focus object of giving flowers is a loved one, the focus object of driving to work is the place of employment, etc.

In the performance of any action with a focus there are at least three nodes involved, the performer, the focus object and the action. The TEMPERAMENTAL system makes a focus connection between the performer and the focus object. The connection is two-way companion. The TEMPERAMENTAL system also makes a connection from the action to the focus object. The connection is two-way companion.

One reason the system makes separate connections for the action and the performer is because it is naive. It does not know if this relationship holds because of the focus object's relationship to the performer or because of some special relationship to the action, or both. For instance, the net must be able to capture fixations one object might have for another without contaminating the connection for the action in general. By separating the focus connections we are able to maintain a strong relationship between the performer and the focus object without the focus object becoming a fixation of the action also.

Another reason for making the connections separately is to reduce the number of connections the system has to make. The number of connections that need to be made are roughly exponential to the number of nodes involved in the connection. Full connectivity between four objects and five actions in triplet form would require 80 ($4\times5\times4$) connections; however, in pairs of doubles the same set of associations is represented with only 36 connections (4×4) + (5×4). Finally, representation as pairs of doublets can be defended on the grounds of ineffectiveness. More complicated connections would work too well. A learner that remembered in triplets, or quadruplets, or whatever is necessary would always know exactly what to expect. Why even have a scene represented by sets of connections? Why not have the entire scene captured in one big beautiful connection? The reason is that no learner ever actually does learn that well. The doublets crystallize the correct ex-

pectation or response much of the time, but we cannot be sure they will do so.

4) Using Connections — Many actions require tools or use objects in some other way. The TEMPERAMENTAL system should make connections to these tools. Therefore, a two-way companion connection is made between the object performing the action and the tool being used. Also, the system makes a using connection from the action being performed to the object being used. The connection is two-way. The connection from the user object to the used object, and the connection from the action to the used object are separated for the same reasons given for separating the focus connections.

5) Sequence Connections — If an object is performing an action A one moment and an action B the next, then form a sequence connection from action A to action B. The sequence connection is one-way and strictly excitory from A to B.

6) Dependency Connections — Action attributes depend on the attributes of the objects involved. Therefore, in a situation where object O1 is performing action A1, using O2 with focus O3, make dependency connections between A1's current attribute and the attributes of O1, O2 and O3. The dependency connections are from dynamic attributes to action attributes, and the connection-making strategy is naive. For each object involved, make a dependency connection from each attribute of the object to each current attribute of the action.

7) Cooperative Connections — Each action has a function attached to it that decides if any given object is able to perform an action. Cooperative connections are formed based on the following rules.

a) If the action possesses attributes, then the object is enabled or disabled with respect to action/attribute pairs, and not just the action.

b) If an action (or an action/attribute pair) is disabled with respect to an object one moment and enabled the next, then make a cooperative connection from the disabled object to the action the object was just performing. Again, the connection making strategy is naive and the connection is one-way companion from the newly enabled action to the actions which the object

was involved with.

8) Competitive Connections — Competitive connections are the opposite of cooperative connections. If an action was enabled with respect to some object and then becomes disabled, make competitive connections between the newly disabled action and the action the object was just performing. The connection is one-way enemy from the newly disabled action to the action suspected.

Cooperative and competitive connections are meant to convey information from an action to other actions about the likelihood that the other action's occurrence will enable/disable the action sending the activation. In real environments, in addition to actions being disabled and enabled by other actions an object performs, they may also be affected by actions that the object is a focus of or used by, and also by circumstances arrived at indifferently to the object. Presently, TEMPERAMENTAL programming does not take into account this complication.

9) Connections from dynamic attributes to other nodes — Whenever a connection is made between an object O and another node, then a connection is also made between each dynamic attribute of O and the node.

2.3 Learning By Making and Breaking Connections

As mentioned, all the strategies for making the different kinds of connections are naive. They will each result in many spurious connections between nodes. This is done on principle. A connection is made between nodes only if one does not already exist, and if a connection does not already exist then the system has no way of knowing, *a priori*, that it is spurious. TEMPERAMENTAL programming balances its naiveté with methods for discovering and breaking the spurious connections as it gains experience. Additionally, the technique for spreading activation includes mechanisms to safeguard against undiscovered spurious connections achieving an inordinate amount of influence. Finally, the TEMPERAMENTAL system has more than one learning mode, and the rules for making connections are applied differently in each.

The learning modes are characterized as "Beginner" and "Expert," but do not take the names to mean that they model how beginners or experts actually learn. They just denote that one method is used early and the other later in the learning cycle for the TEMPERAMENTAL program. Originally, they differed in two important respects. They differed first concerning the level at which they consider a connection to have degraded and gone bad, and secondly they differed in how they treat a connection that has degraded but is a candidate for being reconsidered. After running experiments I found that a rudimentary ability to discover and remove irrelevant connections that maintain strong weights was desirable. The system was extended to incorporate this ability in expert mode.

After a connection is made, the weight on it is continually being adjusted to reflect the system's experience in its environment. Experience then causes some connections to strengthen and others to weaken. When a connection becomes too weak it is considered to have degraded and is broken, but it is saved by the system in a list of "bad" connections. In Beginner mode, the system sets a relatively low level for a connection to be broken. In addition, if the TEMPERAMENTAL system subsequently has a reason to form the connection again, it is automatically taken off of the "bad" list and re-established at its previous strength level.

In Expert mode the system is more skeptical about its connections. The level at which a connection is considered degraded and broken is higher and it is not as easy to retrieve a connection from the "bad" list. A confidence function is defined, and a connection is only retrieved from the "bad" list if the system has confidence in it. Confidence is based on the experience the system has working with the receiver node of the connection. Specifically, the confidence function is:

$$c = \text{Bias} \times \text{Signal}$$

If $c > 1$ and the signal was excitory, then the system has confidence in the connection, or if $c < 1$ and the signal was inhibitory, then the system has confidence in the connection; otherwise, the system does not have confidence in the connection. The signal is the value received by the giver node and

that is being considered for transference to the receiver node. The Bias is given by:

$$\text{Bias} = \log_{10}(\text{CurrentTotal})/\log_{10}(\text{HighestTotal})$$

where CurrentTotal is the total number of opportunities on the weight records of the connection under consideration, and HighestTotal is the highest number of opportunities from other connections from the same kind of node (object, action, dynamic_attribute, etc) that are also currently relevant to the receiver node. The $\log_{10}$ of these totals is taken to flatten the shape of the bias. Only large differences should play a significant role in shaking confidence. Differences within an order of magnitude do not matter a great deal.

For example, in the game of tag we may define a dynamic attribute for objects that specifies their current distance from the tagged player. A dependency connection from the dynamic attribute of distance to flee's action attribute for direction will be made whenever an object is fleeing from that distance. We may represent the connection this way:

```
connection(dependency(da, aa),
           distance(21, 23),
           [flee, change(minusminus),
           [[frequency, 3, 3], [recency, 1]]).
```

`Distance(21, 23)` means the player is between 21 and 23 units from the `it`, `[flee, change(minusminus)]` is the receiver node and means that the `flee` action should be in the minus X direction and the minus Y direction, and the last spot contains the information for calculating the weight of the connection. The frequency means that for 3 out of 3 times that an object has been at that distance and fled, it has fled in the `minusminus` direction.

Now let us imagine there is another connection currently relevant to the `flee` action. This signal comes from `direction(plusplus)` and goes to the attribute `change(plusplus)`. The weights are `[[frequency, 450, 578],`

`[recency, 1]]`. Naively, a connection of 450 out of 578 is weaker than a connection of 3 out of 3. The question is, should we accept the weights naively? The confidence function does not. The fact that one connection has only had 3 chances to be related to the flee attribute while the other has had 578 chances should be taken into account. It is likely that the perfect correlation between the former weight was just coincidence resulting from the smaller sample size. We use the larger total as a standard to bias the multiplier. Therefore, we obtain a bias

$$\text{Bias} = \log_{10}(3)/\log_{10}(578) = 0.172$$

The bias is used to obtain the confidence for the activation signal. Say the signal is 1.4, then

$$c = 1.4 \times 0.172 = 0.24$$

Since 1 is the neutral signal, the bias has changed the signal from excitory to inhibitory and there is no confidence in the connection.

Experiments turned up the need for a test for spuriousness independent of the strength of the connection or the confidence in it. Some connections from attributes to other nodes will remain strong because there are only a small number of actions to be performed, and the attribute has no effect on performance. Since the attribute has no effect on performance, the connection simply records the independent tendency of the system to perform the particular action and this tendency may be very high. Therefore, the connection never becomes weak and never degrades. Additionally, it may be an attribute that the objects possess quite frequently, thus making its `opportunities` high, enabling it to pass the confidence test. To catch these kinds of connections the TEMPERAMENTAL system implements a relevancy test for keeping a connection when in Expert mode. Because it was added late, the mechanism is quite crude and only tests for the relevancy of connections between dynamic attributes and actions. However, I believe that natural extensions to the present theory would smoothly allow more so-

phisticated tests[1]. The relevancy test hinges on the variation of connection strength between different values of a dynamic attribute and the action being considered. A connection between a dynamic attribute's value and an action is relevant if it satisfies at least one of these criteria:

1. The frequency on the connection for this value of the attribute varies significantly from the average frequency for connections between the other values of the attribute and the action.

2. a) Some values of the attribute are not connected to the action,
   b) the unconnected values possess *some* connections to *some* actions, AND
   c) the system has confidence in at least one of these connections.

3. The dynamic attribute has only one value.

The explanation for 2 is simple. If only some values of an attribute are connected to the action, then the system must decide if this implies that the connection from the attribute value under consideration to the action under consideration is, indeed, significant. To decide, it checks to see if the unconnected values have been encountered before. If they have been encountered frequently enough that the system has gained confidence in one of the connections from them, then the system assumes that the reason the unconnected value is unconnected is because the value of the dynamic attribute does, indeed, make a difference. On the other hand, if it has no confidence in any connections to the unconnected value (meaning that it does not have much experience with it), then it makes the assumption that if the connection existed, then it would not make a difference to the final average. If a connection is deemed irrelevant, it is put on a list of irrelevant connections and can never be retrieved.

[1]The extensions that I have in mind include establishing category nodes for dynamic attributes and relations between dynamic attributes. The category nodes would help control the flow of activation and are something I planned to add to a more sophisticated system well before considering the relevancy issue. It is fortuitous that they should be able to help with relevancy also.

To recap, in Beginner mode the level for degradation of connections is set relatively low and connections are retrieved from the "bad" list readily. In expert mode, the level for degradation is raised and a connection is retrieved from the "bad" list only if the system has confidence in it, and a test for the irrelevancy of connections between dynamic attributes and actions is implemented. These two strategies applied serially are generally enough to preserve the good connections and to remove the spurious ones.

2.4 Learning by Adjusting Weights

Unlike many other connectionist systems, the weights on connections never settle into a learned state but are continually recalculated from moment to moment to reflect the learner's experiences in the environment. The weights are calculated from the average of frequency and a recency function, or

```
w = (fq + f(rc))/2
```

The frequency portion of the weight also consists of two measures. The first measure is of the frequency of success of the connection, and the second measure is of the number of opportunities to succeed that the connection has had. Remember, each connection has a semantic meaning, and the meaning denotes a relationship between the nodes connected (focus, using, expectation, etc). We use this semantic meaning to determine if the relationship between the nodes is satisfied in the current moment. If it is, then we increment the `success` portion of the frequency weight. We can also determine if the current moment contained situations in which there was an opportunity for the relationship to be satisfied. We then increment the `opportunities` portion of the frequency weight for each opportunity found. The frequency is then simply:

```
fq = Successes/Opportunities
```

A connection is degraded if its frequency function falls below a set level. The level is set by the learning mode, Beginner or Expert , that the TEMPERAMENTAL program is in.

While the frequency function reflects the overall trends in the environment, TEMPERAMENTAL systems can also be biased towards the recent past. If a relationship denoted by a connection is currently satisfied, we shall represent this by assigning the connection a recency weight of 1. If the relationship could have been satisfied but was not, then the recency weight will be incremented by 1 from whatever its current level is. Thus, if a relationship is currently satisfied its recency goes to 1, and then if it goes 3 opportunities without being satisfied the recency will be at 4. A limit on the amount recency may be incremented is established, and the value associated with each recency is between 0 and 1. The values are stored in a table with the number 1 having the highest value and the values then decreasing sharply downward. The table used by SALMON had the values

```
recency_table(1, 0.80).
recency_table(_, 0.65).
```

The value given by 1 must be below 1.0 so that perfect correlations in the frequency function do not endlessly cycle through the network. I tried differences greater than 0.15 between the high value and the low value, but they seemed to cause the system to continually repeat it's last action. 0.80 seems to work well, at least for this particular implementation. The other weight was similarly hand-tuned. As it turns out, the rule governing what values should be in the table is the obvious one — they should reflect the effect recency has in the environment to be learned. Since the simulation I wrote to act as an environment for SALMON only biases its players towards their most recent behavior without regard for more than the immediate past, only the value for 1 should be biased in the learner. Allowing many more levels of "memory" by adding to the table produces a bias towards recent behavior not found in the simulation.

2.5 Adjusting Particular Connections

1) The Expectation Connection — The expectation connection is satisfied if object O is performing the action expected. An opportunity occurs if the object is enabled to perform the action.

2) The Exclusive Connection — An exclusive connection between attributes and actions is a success if the object is not performing the action, is enabled to perform the action, and has the attribute. An opportunity for an exclusive connection occurs whenever the object has the attribute, was previously performing the excluded action, and is currently enabled to perform it. Successes and opportunities between objects and actions in exclusive connections are judged similarly, except the judgement is made without regard to attributes the object may possess.

3) Focus Connections — The focus connection between objects is satisfied between O1 and O2 if O2 currently is the focus of O1's action. There is an opportunity if some object is the focus of O1. The focus connection from an action to an object is satisfied if the object is currently the focus of the action. An opportunity exists if the action is being performed. Since an action can be multiply instantiated, the adjustment must take into account each instantiation.

4) Using Connections — A success in a using connection between objects O1 and O2 occurs if O1 is using O2. An opportunity occurs if O1 is using something. A success for using connections between objects and actions occurs for an action A and an object O2 if the action A is currently using O2. An opportunity occurs if the action is being performed and is using something.

5) Sequence Connections — A success for a sequence connection from A1 to A2 occurs if an object is performing A1 one moment and A2 the next. An opportunity occurs if an object was performing A1 at the previous moment.

6) Dependency Connections — A successful dependency connection occurs when an object is performing the action with the specified attribute, and either it has the dynamic attribute or the focus of its action has the dynamic attribute. An opportunity occurs if an object is performing the action and either the object or its focus has the attribute.

7) Cooperative Connections — A success for a cooperative connection occurs if the giving action in the connection becomes enabled with respect to an object after being disabled, and the object just performed the receiving action.

An opportunity occurs simply if the giving node was disabled with respect to an object and the receiving action was performed by the object. An alternative way to count an opportunity would be if the giving node changed states from disabled to enabled. This alternative method captures the likelihood that a change in state is caused by the receiving node, whereas the method used captures the likelihood that the receiving node being performed will cause the change in state. Since the connection is used by the giver to enable itself, the method used was deemed to provide the more valuable information.

8) Competitive connections — A competitive connection is a success if the giving action was enabled with respect to an object, and became disabled after the object was performing the receiving action. An opportunity is occurs if the giving action was previously enabled and then becomes disabled.

3 Spreading Activation

Once the TEMPERAMENTAL learner has had a significant number of moments observing the environment he may participate. Participation in an environment occurs by inputting activation into the connectionist network it built as an observer and letting the activation spread through the nodes. Before spreading activation, all nodes have their activation levels returned to a "start" point. There is no need to save the activation levels from previous computations because the result is captured in the recency weights on the connections. The input nodes into the net are the ones corresponding to the objects presently in the environment and the actions that they are performing. Thus, the nodes input into are not fixed, but rather change with every moment. This is in stark contrast to many other connectionist architectures, particularly those which use back-propagation, which are arranged hierarchically with fixed input and output nodes. Each node passes activation along its connections to other nodes and uses the information on the connections to determine a transfer value for the activation. Following is a discussion of the factors used to calculate the transfer value.

3.1 Relevance of a connection

Nodes form many connections, not all of which are relevant at any given time. Before a connection can transfer a signal from one node to another, it must pass a relevance test. A connection is relevant if one or more of the following are true:

1. At least one of the nodes is the relevant object.
2. The connection is a sequence connection and the relevant object is enabled with respect to the giving node.
3. At least one of the nodes is a dynamic attribute of the relevant object.
4. The connection is not an expectation or exclusive connection and at least one of the nodes is currently the focus of the relevant object.
5. The connection is not an expectation or exclusive connection and at least one of the nodes is a dynamic attribute possessed by an object that currently is the focus of the relevant object.
6. The connection is an expectation or exclusive connection, at least one of the nodes is currently the focus of the learner, and the focus is enabled with respect to the action involved.
7. The connection is an expectation or exclusive connection, at least one of the nodes is a dynamic attribute possessed by an object that currently is the focus of the learner, and the focus is currently enabled with respect to the action involved.
8. The relevant object is currently the focus of at least one of the nodes.
9. At least one of the nodes is an attribute of an object which currently has the relevant object as its focus.

The relevant object is usually the learner but may be changed dynamically to suit the system's purpose. Detailed discussion of tests $4 - 7$ is deferred until section 9. Activation is input into the network as excitory through the nodes

corresponding to each object and each action present in the environment. The point of view of the learner is achieved by biasing the input into the nodes representing itself and the action it is currently performing. The bias multiplies the normal input level.

3.2 The Transfer Function

Each input represents a signal which increases the activation level of the node receiving it. Signals above one are excitory and signals between 0 and 1 are inhibitory. Every time a signal passes through a connection, it is moved closer to the neutral value. Eventually all activation stops as the signals expend their energy moving through connections. The transfer function takes the values stored in the connection as arguments and determines how much energy the signal will expend traversing the connection. When an activation expends so much energy traversing a connection that it falls within a defined neighborhood of 1, then it is exhausted and is not transferred.

All signals are input as excitory. If an excitory signal must pass through an enemy connection, then it becomes inhibitory. If M is the excitory value of the signal, then 1/M is the inhibitory value. Conversely, an inhibitory signal passing through an enemy connection becomes excitory (see section 2.3). If M is the value of the inhibitory signal, its corresponding excitory value is 1/M. Otherwise, the signal maintains its type.

We define for each connection a raw weight. The raw weight of a connection is the average of the frequency function and the recency function given by

```
rw = (fq + f(rc))/2  (see section 2.4)
```

The actual weight of the connection is dependent on the type of signal being transferred. If the signal being transferred is inhibitory then the actual weight is given by

```
actual = 1/rw
```

Otherwise, actual = rw

The signal received is multiplied by the actual weight of the connection, which always moves the signal closer to 1 than it was before passing through the connection. If it becomes too close to 1 then the signal is not transferred. An additional check is made by passing the signal and connection to the confidence function. The confidence function bias may move the activation even further towards the neutral level. Recall, the confidence bias is given by

$$\text{Bias} = \log_{10}(\text{Giver})/\log_{10}(\text{HighestRelevant})$$

where Giver is the opportunity value on the connection the signal is passing through, and HighestRelevant is the highest opportunity of any connection currently relevant to the receiver node, the reciever node is also receiving in the HighestRelevant connection, and the giver node in the HighestRelevant connection is of the same type as the giver node in the Giver connection (See section 2.3 for a detailed discussion of the confidence function). If the signal fails the confidence test then it is not transferred. When signals are passed through object nodes, three things occur.

1. The object's activation level is multiplied by the signal received.
2. The signal is transferred to all relevant nodes connected to the object.
3. The signal is transferred to all relevant nodes connected to the object's dynamic attributes.

When activation is passed through an action, two things occur.

1. Its activation level is multiplied by the signal.
2. The signal is transferred to all relevant nodes connected to it.

Dynamic attributes do not have activation levels. They merely act as conduits for signals from objects to other nodes. The transfer function for a

connection involving a dynamic attribute is exactly the same as for connections involving any other kind of nodes. There is no penalty as the signal moves from the object to its dynamic attributes, but the transfer function is applied as the signal propagates from the dynamic attribute to other nodes.

Currently, a signal passing through a dynamic attribute is finagled sometimes. More than one object may possess any given attribute at any given time. If many objects with relatively weak activation levels possess an attribute, the nodes connected to that attribute receive a great deal of activation. Generally, this is enough to overpower the effect of any dynamic attributes represented by only a single object. However, this group effect is often not desirable, as the system is really concerned only with the effects associated with individuals and does not care how they are grouped. To correct this grouping effect, whenever a signal is passed through a multiply instantiated dynamic attribute the system biases the signal in the following way — divide the activation level of the object the signal is emanating from by the highest activation level of the other objects with the dynamic attribute, and then multiply the signal by this ratio. That is, if object O is passing a signal M through its attribute DA, then

$$\text{GroupingBias} = \text{activation level(O)}/\text{activation level}(O_2)$$

where O_2 is the most highly activated object possessing that attribute.

In principle, a multiply instantiated attribute could still greatly affect the result obtained in the network, but in practice activation levels quickly diverge so only the activation from the most highly activated possessor has much influence. Of course, sometimes a group effect is appropriate. This solution is temporary and, admittedly, a compromise. A better solution would be for the system to determine for itself when a group effect matters, and choose its strategy for spreading activation appropriately.

Action attributes merely collect activation. They do not transfer it. So when a signal is received by an action attribute then the attribute's activation level is modified and the activation stops there.

3.3 Choosing an Action

Once activation has been input and spread into the network, then the learner decides on the course of action to be taken. There are no thresholds in TEMPERAMENTAL programs. Competition between nodes is strictly the accumulation of wealth, wealth being represented by the activation level. The procedure for choosing an action is as follows:

1. Choose the action with the highest activation level.
2. If the learner is enabled with respect to this action, go to choosing an attribute.
3. Else choose the action with the next highest activation level.
4. Go to step 2.

Once an action has been chosen, then the system must decide on an attribute. The procedure is as follows:

1. If the action has no attributes, perform the action.
2. Else, choose the attribute with the highest activation level.
3. If the learner is enabled with respect to this action/attribute pair then perform the action with that attribute.
4. Else, input activation into the highly activated node and spread the activation only through the cooperative and competitive connections, then go to "Choosing an action."[2]

[2]The algorithm does not call for the system to search for the next most highly activated attribute because, generally, this is not reliable. It is not unusual for only the appropriate attribute to have been activated. Therefore, choosing the next highest would be random. Spreading the energy from the cooperative and competitive brings out the appropriate response.

If the action requires a focus, then choose the highest activated object in the environment which the semantics say is appropriate and available. If the action requires a tool, then choose the highest activated object in the environment which the semantics say is appropriate and available.

The execution of the action consists of running a procedure which affects the properties or relationships of objects in the environment. These semantics are responsible for specifying and choosing a focus if one is needed, and also any tools. Additionally, it is responsible for making any changes in the tools, focus or learner that are necessary. In execution of it's duties, the semantics are empowered to spread activation through the learner in appropriate ways, including simulating other points of view by altering which node the input bias is used on and sending signals to nodes which are not currently present in the environment. This is how the system answers "What if?" questions, and, also, can daydream. The daydreaming ability is realized by allowing activation to be input into the network from within the TEMPERAMENTAL program as well as by the environment. In principle, the TEMPERAMENTAL system can activate any arbitrary combination of nodes and see what consequences follow. This capability can be used to theorize, daydream, imagine counterfactuals, etc. Its greatest advantage is that, harnessed and used in a sophisticated manner, it could help the system learn.

4 SALMON

A TEMPERAMENTAL system has been implemented to learn the game of tag by observing the execution of a discrete event simulation of the game. The system was implemented in Quintus Prolog version 2.0 on a SUN Sparcstation. The name of the implemented program is SALMON. The simulation outputs a game state from moment to moment which is then translated by an interface module into a predicate language that SALMON and his TEMPERAMENTAL learning mechanism can understand.

The only objects in the environment are the players. The number of players varied from 4 to 7. The number of players does not affect SALMON's performance after he has learned, but during learning a high number of players

gives better results. Two factors are responsible. First, more players give a statistically better sampling so we would expect better behavior. The second reason has to do with the way the simulation is programmed. If a player is being chased, it always `flees`. Therefore, if you have a small number of players then one is always tagged and of the remaining players at least one is always fleeing. Therefore, the sample becomes biased towards fleeing, especially the sequence connections which will record the large number of sequences that the chased player flees consecutively without the less determined sequences of the other players tempering it. After the learning period, SALMON can be inserted as a player and participate in the game. Upon insertion, the simulation simply transfers control to SALMON whenever it is time to decide what his next move will be. After SALMON has decided and executed his decision, control passes back to the simulation.

Six actions were defined for the game:

1. `Flee` — An untagged player flees by increasing the distance between himself and the player currently tagged.
2. `Tease` — An untagged player teases by decreasing the distance between himself and the player currently tagged.
3. `Chase` — The tagged player chases another player by decreasing the distance between himself and the player being chased.
4. `Count` — After a tag has been made, the newly tagged player must stand still and count for a specified number of moments.
5. `Make-tag` — If an untagged player is within one step of the tagged player, then the tagged player may make a tag on the untagged player. Making a tag consists of transferring the tagged attribute.
6. `Stayput` — A player may, at any time, stand still.

Three different kinds of dynamic attributes were defined which objects (players) in the environment can possess.

1. `Being tagged` — SALMON was informed who was currently tagged by making tagged a dynamic attribute. In real life there are versions of the game of tag, such as ball tag where the `it` carries a ball allowing being tagged to be physically observable. We can think of this game of tag as being like ball tag.

2. `Distance from the it` — Since the currently tagged player is the focal point of the game, SALMON was informed of the distance each player was from the `it` at each moment. This distance was presented as a dynamic attribute of the players.

3. `Direction from the it` — The `it` was also considered the origin (0,0) of a cartesian graph, and the direction that each player was from the `it` was attached to objects as a dynamic attribute. The directions were (in (X Y) notation): (plus plus), (plus minus), (plus same), (minus plus), (minus minus), (minus same), (same plus), and (same minus).

The action attributes were the general direction which any of the movement actions could take. They were mapped onto the direction from the it dynamic attributes with the addition of (same same) for standing still.

5 The Simulation

The simulation was a discrete event simulation in which each player's next set of attributes was determined solely from his current set, mostly without regard to changes in any other player's attributes. The player's strategies were quite straightforward. If a player was tagged he chose another player to chase based on a function of that player's distance from him and the difference in their speeds. Chasing a player meant decreasing the difference between the pursuer's X and Y coordinates so that they converged towards the chased player's coordinates.

Player's who were untagged had a choice of actions: `tease`, `flee`, `and` `stayput`. Functions were also defined for these actions based on a player's current distance from the `it` and the difference in speed between the player

and the it. This function returned a value between 0 and 1. A uniformly distributed random number was generated, and if it was below the value returned by the function then the action being considered was performed, otherwise another action was considered. The functions were defined so that faster or more distant players were more likely to tease or stand still. Slower or nearer players were likely to flee. Fleeing players were not given knowledge about how to use open space or any other sophisticated strategies. The playing field was bounded and untagged players were not allowed to go out of bounds.

6 The Interface to the Learner

SALMON was always provided with the following information about the current moment and the immediately preceding moment. A translator was responsible for asserting

1. clauses telling SALMON what objects were performing what actions, with any attributes included.
2. clauses telling SALMON what the focus of each action and object was.
3. clauses informing SALMON what the dynamic attributes possessed by the objects were.
4. clauses saying which actions were enabled and disabled with regard to each object.

These clauses play the role of a rather sophisticated perceptual system. In particular, informing SALMON of the focus of actions and objects assumes a highly developed instinct. Such recognition in animals requires hardwired instincts for recognizing certain traits in the environment; brightly colored objects, sudden movement, movement in general, diminishing and increasing distances, concepts of personal space, etc. Additionally, such recognition uses inborn and developed abilities animals have to recognize what satisfies and motivates them and allows them to project it to other creatures. For

instance, an animal is satisfied by food and so if it sees another animal stalking prey, killing it and eventually eating it, it is likely that it may know by projection that the killed animal was the focus of all the killer's action. This is because the actions led to a situation it desires, namely, satisfying hunger. When another animal satisfies a similar desire, then the observing animal perhaps feels a titular satisfaction also, and uses this feeling to recognize that it should imitate. These subtle clues in the environment and the internal models that allow animals to pick them out are presently beyond the ability of the simulation to provide or TEMPERAMENTAL systems to simulate. Therefore, it is necessary that focus information be provided to SALMON.

The rules of the game were semantically encoded as enabling conditions on actions. The rules were —

1. untagged players must stay in bounds.
2. only tagged players can chase, count or make-tag.
3. only untagged players can flee or tease.
4. a player may only count if the count is above one.

This information amounted to permission information for the different actions. A great deal was left for SALMON to learn. In particular, it had to discover that `flee` meant moving away from the `it`, and, indeed, what it means to move away. It had to discover that `tease` meant moving closer to the `it` and what moving closer is. It had to induct the functions for deciding when to `flee` or `tease`, meaning it had to tease more often when it was far away or the `it` was slower and `flee` conversely. It had to discover what to do when it reaches a boundary and cannot perform the action it would most like to. It had to discover that it should `make-tag` when it is close to another player. It had to discover that it should `count` after being tagged. Finally, it had to learn to make the rather difficult judgement concerning who to chase when it is tagged and what direction to move in to run the player down. All in all, there was a great deal for the system to learn. Of course, an on-going goal of TEMPERAMENTAL programming as a research effort

is to reduce the semantics that must be provided to the system. Performing actions consisted simply of updating the X and Y coordinates of SALMON, and his tagged attribute if necessary.

7 Results

The results were surprisingly good. The learning session consisted of 500 moments watching the simulation in Beginner mode and 1000 watching in Expert mode. SALMON was then inserted into the game as various players. Scripts were written to files and animated. To the naked eye, SALMON appears virtually identical to the simulated players in an animation. Examination of the scripts and connections reveals a difference between SALMON's behavior and the other players.

Four scripts, inserting SALMON as a different player in each, were run for 200 moments each. Each script was begun from the same point in the action. The simulation was then run from this same point without SALMON's participation. A comparison of the results of each run with SALMON to the run without SALMON turned up some interesting results.

7.1 Analysis

We can list each of the items available for SALMON to learn:

1. Count after being tagged.
2. Chase after counting.
3. Choose a "reasonable" player to chase.
4. Move in the direction of the player chosen for the chase.
5. Tag players it is chasing when it gets close to them.

6. Flee when close to the `it`.
7. Flee more often if the `it` is faster than SALMON.
8. When fleeing, move away from the `it`.
9. Tease when further away from the `it`.
10. Tease slower players more often.
11. When teasing, move towards the `it`.
12. Choose an appropriate move when at the boundaries.
13. Stand still sometimes.

SALMON eventually learned to do 1–12 almost flawlessly. Still, finding a training set that provided good `tease` behavior proved difficult. In no training set would SALMON stand still. SALMON's difficulty in picking up both these behaviors is related. `Flee` is performed more often by untagged players than either `tease` or `stayput`. As the disparity grows larger from `flee` to `tease` to `stayput` SALMON has more and more difficulty deciding when the less frequently performed actions are appropriate, and defaults to the global tendency. Especially apparent was the tendency for SALMON to continue with whatever action it was performing. Of course, the `sequence` connection was responsible for this tendency, especially in the case of `fleeing` which quite often follows itself in time. In a situation in which `tease` has only a small advantage over `flee` for the other connections, the sequence connection tended to make a huge difference in favor of `flee` if SALMON was already fleeing. Eventually, I was able to coax perfect behavior from SALMON but it took a great deal of experimentation. The final result was not caused from an ad hoc change, but rather resulted from the discovery of general faults with the earlier versions of the system and corrections to them. The changes were:

1. The criteria which determines the momentary relevance of connections must direct the spread of activation adequately. Specifically, signals from the `sequence` connections should not be spread unless SALMON

is enabled for the giver node in the connection. It makes no sense for SALMON to consider what follows from actions that it cannot perform.

2. In the same vein, expectation and exclusive connections to an object which is SALMON's focus at the moment should not be spread unless the focus object is enabled with respect to the action in the connection. For example, if SALMON is focusing on the player that is the `it`, SALMON should not spread activation to the expectation that the `it` will flee.

3. The values in the recency table should accurately reflect the effect of recency in the environment to be learned. Improper deviation's can cause SALMON to either care too much for the recent past or disregard it totally.

4. The `sequence` connections should be purely excitory rather than companion.

These corrections to the original algorithm are not fully general, but they are not completely *ad hoc* either. Rather, they are moves in the right direction that an expanded TEMPERAMENTAL model would have to improve upon. The first three problems can be considered generally as controlling the spread of activation (1 & 2) and correctly tuning the system's parameters (3). More detailed discussion of these problems is deferred until section 10. Correction 4, the redefinition of the `sequence` connection, occurred because the natal stage of the theory made it easy to incorrectly define connections, not from fundamental problems with the ideas behind the theory. Basically, the terms need to be formalized. I did not realize that sequence connections should not be companion because I was working with a vague, intuitive idea of it. All the connections, nodes and other elements of the system need more formal, precise definitions.

The weight adjusting algorithm was pathologically bad in the case of `stayput`. It is unusual for a player in the simulation to perform `stayput` for more than 1, maybe 2, consecutive moments. `Expectation` connections were strengthened whenever an object performed an action it was enabled to perform, and weakened whenever an object did not perform an action it was enabled

to perform. Oppositely, `exclusive` connections were strengthened whenever an object was not performing an action that it was enabled to perform and weakened whenever an object was performing the action. Since objects generally had the same attributes (because they stood still) when they stopped performing the `stayput` action as they had when they started, the system strengthened the `exclusive` connection to `stayput` nearly every time it strengthened the `expectation` connection! With these adjustments nearly always canceling one another out, the system never really built any power to excite `stayput`. I would like to add however, as a caveat, that TEMPERAMENTAL programming really was not designed to pick up on patterns like `stayput` behavior and I never expected the system to do it. The system is designed to be able to detect and mimic caused changes, whereas the `stayput` behavior by the simulation's players is not only the least frequently performed, but also the most uniformly distributed action and the one involving the least amount of change.

On the positive side, the system nearly always had the correct direction for whatever action it performed. It always counted after being tagged, acted reasonably at the boundaries and made the tag when it got close to a player it was chasing. What was, to me, more impressive was the good behavior it showed when it had to `chase`. I considered the `chase` decision to be the most difficult it had to make and actually expected to get something more akin to a random walk than a chase. Below is a typical excerpt from a script. In it, SALMON is player number 4, the `it`. The first number in the clause is just an index for the simulation to use in its memory management. The second number is the player number. The next two are the X and Y coordinates, and, finally, the value of the tagged property and action for each player. It is informative to study the script from moment to moment. I provide comments to point out the interesting facets of the chase.

moment(Index, Player Number, X, Y, Property Value, Action)

1 — The simulation is programmed so that the player being chased always flees, so we know immediately that SALMON must be chasing players 1, 7, or 2 since they are the only players fleeing.

moment(1,4,22.1559,13.5921,tagged,chase)

moment(1,1,19.0,12.5,untagged,flee)
moment(1,7,17.9619,7.61374,untagged,flee)
moment(1,2,17.0,3.25,untagged,flee)
moment(1,5,39.8124,8.29486,untagged,tease)
moment(1,3,41.8862,6.32001,untagged,stayput)
moment(1,6,22.8083,9.09282,untagged,tease)

2 — SALMON has continued to chase (as opposed to stayput, flee, count, tease, make-tag, etc... we should remember it has to make a choice at every moment) and has decreased both its X and Y coordinates. Unfortunately, all the candidate players for being chased have lower X and Y coordinates than SALMON so this doesn't help us discover which it is chasing.

moment(2,4,20.7559,12.1921,tagged,chase)
moment(2,1,17.5,11.0,untagged,flee)
moment(2,7,16.6619,6.31374,untagged,flee)
moment(2,2,15.75,2.0,untagged,flee)
moment(2,5,38.4624,9.64485,untagged,tease)
moment(2,3,43.1661,5.04001,untagged,flee)
moment(2,6,22.8083,9.09282,untagged,stayput)

3 — SALMON has once again lowered his X and Y coordinate. Player 7 has decided to tease, so we know that it must be chasing 1 or 2. Notice that SALMON does not chase in a direction where there are no players, nor does it chase in a direction where the closest player is far away, like player 5 in the +X, +Y direction.

moment(1,4,19.3558,10.7921,tagged,chase)
moment(1,1,16.0,9.5,untagged,flee)
moment(1,7,17.9619,7.61373,untagged,tease)
moment(1,2,15.75,3.25,untagged,flee)
moment(1,5,37.1124,10.9948,untagged,tease)
moment(1,3,41.8861,6.32001,untagged,tease)
moment(1,6,21.2582,10.6428,untagged,tease)

4 — SALMON is consistent. It continues on its course and does not take a random walk around the playing field.

moment(2,4,17.9558,9.39204,tagged,chase)
moment(2,1,16.0,8.0,untagged,flee)
moment(2,7,16.6619,6.31373,untagged,flee)
moment(2,2,15.75,2.0,untagged,flee)
moment(2,5,35.7623,9.64484,untagged,tease)
moment(2,3,40.6061,7.60001,untagged,tease)
moment(2,6,22.8082,9.0928,untagged,flee)

5 — SALMON appears to be closing in.

moment(1,4,16.5558,7.99204,tagged,chase)
moment(1,1,16.0,6.5,untagged,flee)
moment(1,7,15.3619,5.01373,untagged,flee)
moment(1,2,15.75,3.25,untagged,flee)
moment(1,5,34.4123,8.29483,untagged,tease)
moment(1,3,39.326,8.88,untagged,tease)
moment(1,6,24.3582,7.5428,untagged,flee)

6 — It will be a difficult choice who to finish off among the three players all bunched up together.

moment(2,4,15.1558,6.59204,tagged,chase)
moment(2,1,16.0,5.0,untagged,flee)
moment(2,7,15.3619,3.71373,untagged,flee)
moment(2,2,15.75,2.0,untagged,flee)
moment(2,5,33.0623,6.94483,untagged,tease)
moment(2,3,38.046,7.60001,untagged,tease)
moment(2,6,25.9082,5.9928,untagged,flee)

7 — SALMON inexplicably raises its Y coordinate, perhaps distracted by player's 3 and 5, two of the slower players, both in that direction. A simulation player would never do this, so SALMON has not learned perfectly. Actually, I considered this good. I wanted heuristic learning methods that gave good behavior but not perfect behavior. I don't know any perfect biological learners.

moment(1,4,16.5558,7.99203,tagged,chase)
moment(1,1,17.5,3.5,untagged,flee)
moment(1,7,16.6618,2.41373,untagged,flee)
moment(1,2,15.75,2.0,untagged,stayput)
moment(1,5,31.7122,5.59483,untagged,tease)
moment(1,3,39.326,8.88,untagged,flee)
moment(1,6,27.4582,4.44279,untagged,flee)

8 — It seems back on track. SALMON has moved +X, −Y, so obviously it is going after player 7. Player 1 is also in that direction but it is teasing so can't be the player being chased. Player 1 is pretty fast, but player 7 is slow. The tease by player 1 should tempt SALMON.

moment(2,4,17.9557,6.59203,tagged,chase)
moment(2,1,16.0,5.0,untagged,tease)
moment(2,7,17.9618,1.11373,untagged,flee)
moment(2,2,15.75,2.0,untagged,stayput)
moment(2,5,33.0622,4.24483,untagged,flee)
moment(2,3,38.046,7.60001,untagged,tease)
moment(2,6,29.0082,2.8928,untagged,flee)

9 — SALMON is continuing after player 7. This is a good choice because player 1 is faster than SALMON.

moment(1,4,19.3557,5.19203,tagged,chase)
moment(1,1,16.0,3.5,untagged,flee)
moment(1,7,19.2618,1.11373,untagged,flee)
moment(1,2,15.75,3.25,untagged,flee)
moment(1,5,31.7122,5.59483,untagged,tease)
moment(1,3,36.7659,6.32001,untagged,tease)
moment(1,6,30.5582,1.3428,untagged,flee)

10 — This is a tough situation for SALMON. The player it has been chasing, player 7, is in the (plus minus) direction, but two players in the (minus minus) direction appear to be only a step or two away from being caught also. I am not sure what the simulation would do in this circumstance.

moment(2,4,17.9557,3.79203,tagged,chase)
moment(2,1,16.0,2.0,untagged,flee)
moment(2,7,17.9618,1.11373,untagged,flee)
moment(2,2,15.75,2.0,untagged,flee)
moment(2,5,30.3622,4.24483,untagged,tease)
moment(2,3,35.4859,5.04001,untagged,tease)
moment(2,6,32.1082,1.3428,untagged,flee)

11 — SALMON goes after the two players and appears ready to finish off the chase, decreasing its own X and Y coordinates for the second moment in a row. I wonder if its decision here was affected by the grouping? It appears close enough to tag either player 1 or player 2.

moment(1,4,16.5557,2.39203,tagged,chase)
moment(1,1,16.0,3.5,untagged,flee)
moment(1,7,19.2618,1.11373,untagged,flee)
moment(1,2,15.75,3.25,untagged,flee)
moment(1,5,31.7122,5.59483,untagged,flee)
moment(1,3,34.2059,3.76001,untagged,tease)
moment(1,6,33.6581,1.3428,untagged,flee)

12 — SALMON decides to make the tag. Remember, the semantics do not force it to do so; the system was free to continue chasing indefinitely.

moment(2,2,15.75,3.25,tagged,count)
moment(2,4,16.5557,2.39203,untagged,make_tag)
moment(2,1,16.0,5.0,untagged,flee)
moment(2,7,20.5618,1.11373,untagged,flee)
moment(2,5,30.3622,4.24483,untagged,tease)
moment(2,3,32.9258,2.48001,untagged,tease)
moment(2,6,32.1081,2.8928,untagged,tease)

Sequences such as this are typical — and impressive. SALMON was quite consistent in producing this kind of behavior when tagged. When untagged, it also always chose a proper direction, the one real problem, as mentioned earlier, is that it never stood still.

8 Issues Raised

As it stands, SALMON and TEMPERAMENTAL programming cannot make any strong claims about learning and cognition. Their most useful contribution is as an exercise for determining just what abilities a hybrid system built along these lines might have, what limitations, and what problems need to be addressed. This section is a discussion along those lines.

SALMON shares one limitation with human beings that is rather interesting. As it tries to "think" about more than one issue simultaneously its performance on all the issues deteriorates. This is a consequence of using a bias on the input to simulate a point of view. If only one bias is used SALMON performs very well. One bias is like asking it "What is object X (where X is the object receiving the bias) going to do next?" But if you wish to ask it to consider two or more situations simultaneously then the ordering of activation levels becomes less reliable. The highest activated objects and actions sometimes may be appropriately paired together, but finding the appropriate pairing for the second highest activated object and action is more difficult. It is not a simple mapping. Since the brain is highly parallel I have always thought it was mysterious that we must focus so often on a single issue in order to think clearly about it. The answer to how a limitation like this can arise in a parallel network is not obvious. Therefore, I find it interesting that this kind of "concentration" which also is a property of animal intelligence should emerge from the highly parallel TEMPERAMENTAL algorithm.

Neurons have been characterized simplistically as having all or nothing firing behaviors. Accordingly, many connectionist nets use thresholds and have nodes with all or nothing firing behavior. It is significant that TEMPERAMENTAL systems diverge from this behavior. The divergence can be defended on two grounds, one from the point of view of connectionist systems and the other from the point of view of semantic systems —

1) As a hybrid system, SALMON's nodes represent complex concepts. These concepts would likely be represented in the brain by many millions of neurons and not a single neuron. With any given input, some neurons in this representation will fire as activation spreads, even though the whole neu-

ral ensemble encoding the action or object may not. The various biases in the TEMPERAMENTAL system, the confidence function and connection weight, can be thought of as representing the proportion of neurons in the ensemble representing the giving concept that will fire into the receiving concept given a certain level of activation passed into the whole ensemble[3]. The highes activation levels at the end then represent the ensemble which had the highest number of individual neural firings during computation.

2) On mentalistic grounds, spreading activation through a transfer function can be defended on the basis of communication between concepts. The TEMPERAMENTAL system must reason through time, and therefore must take into account the future. Let us imagine that action nodes had thresholds and the firing of a node represented a decision to perform the action. In this case, the action could not have input from the future about whether or not it should fire until after the decision was made! Admittedly, a more complicated interpretation could be put on the firing of a node, but that would also require a more complicated algorithm for spreading activation. It is better to use a transfer function and have nodes "fire" only once, when computation is finished.

Philosophically, SALMON gains expectations from the simplest possible induction over experience. It is interesting that a calculation as simple as the successes/opportunities ratio is enough to make the correct induction when magnified and balanced in a network, even in SALMON's simplified world. Additionally, perfect correlations lead to very strong expectations. SALMON's highest action was often count regardless of the context. Only

[3]Of course, I am not saying that the *actual* strength is captured by the weights on the connection. All I mean to imply is that, in the brain, any ensemble of neurons will have constituents receiving signals at any given time. Each of these individual neurons participates in other ensembles and some will fire, sending signals here and others over there. Some pairs of ensembles share more interconnections between constituent neurons, and share more neurons, than others. We can think of the TEMPERAMENTAL weights as vaguely representing the degree to which neurons are interconnected or outright shared between concepts. Therefore, the entire concept does not have to "fire" to signal another concept, but only has to pass along some proportion of the signals that its constituent neurons send out from the signals that they receive. This is what the transfer function does and explains why there are no thresholds on nodes.

the semantic enabling conditions kept it from always counting[4]. This is because the count action had a perfect expectation. It was always performed whenever it was enabled. This seems interestingly like how we might take for granted propositions like, "The sun will rise tomorrow morning." SALMON would not think of questioning the truth of that statement, and neither, usually, do we.

8.1 Problems Exhibited by SALMON

Besides the grouping effect already written about, the model exhibited several other troublesome tendencies. For instance, there are the spurious connections SALMON cannot discover simply by having them degrade. For example, whenever an untagged player performs a flee or tease action, a connection is made not only to the player but also to the dynamic attributes possessed. One of the dynamic attributes the player possesses will be the direction he is from the it. So, an expectation connection is made, say, that looks like this

```
direction(minusplus) ------> flee
```

meaning that if an object has the direction(minusplus) attribute than it should be expected to flee. This is spurious. However, currently there is no way SALMON could discover this. The problem is that, on the whole, players perform the flee action nearly 60are enabled to. The frequency function on the weight is then nearly 60if he had a degradation level above 60simply eliminate too many good connections. Therefore, SALMON must use the relevancy test to discover these connections. However, I don't feel the current

[4]Because so much emphasis has been put on explaining the learning components of the system in this thesis, it may be tempting to think of SALMON's semantic information as a kind of cheating, but that is inappropriate. TEMPERAMENTAL programs are, by design, *hybrid* systems with the semantic components in on way taking a back seat to the knowledge contained in the connections. The whole point is that they can work together to do things neither could do alone, or could only do with much more difficulty. It would be just as easy to think of the connectionist components as cheating to make up for what the semantics cannot do.

relevancy test is very general and TEMPERAMENTAL systems need more sophisticated (and local) ways of discovering these kinds of connections.

Of course, this really points out a much broader problem. Namely, there is a great art to defining a dynamic attribute. For instance, if we had other objects like trees or fire hydrants or sidewalks or bushes in the playing field they would also have directions and distances relative to the `it`. Still, we would not tell SALMON this because it would throw off all its connection weights. The programmer must make this choice for SALMON and the choice is really on pragmatic grounds and not principle. SALMON should be able to tell when an attribute is relevant for an object without being spoon-fed good descriptions of the phenomena. Just as SALMON cannot tell if an attribute is irrelevant despite its high correlation to an action, he cannot tell if an attribute is not even worth noticing.

The result of this inability to distinguish correctly between relevant and irrelevant attributes independently of connection strength can be a bias in SALMON to perform the overall action tendencies of the simulation at the expense of context. In plain English, this means SALMON never stood still. Another problem is that there are many parameters in the model and the quality of the connections is very sensitive to some of them. Particularly, the parameter which sets the lowest weight allowed on a connection before breaking it is crucial. If it is too high nonsense behavior can result. If it is too low the number of connections increases rapidly and learning and decision making slow appreciably.

I have already mentioned that the system is sensitive to the weights in the recency table. If these weights do not reflect the effect of recency in the actual environment then SALMON will not learn as effectively. In principle, the model is not too sensitive to the strength of the input signals, as long as the bias is 3 or greater. However, when input is too large the activation level of the nodes goes to infinity very quickly. On the positive side, ultimately it may not be necessary for the programmer to hand-tune the parameters. When the TEMPERAMENTAL theory is extended to include feedback from the environment it is likely that the system will be able to tune its own parameters. Also, even though activation levels become very high with strong input signals, the activation levels on SALMON's nodes were usually irreversibly

ordered long before signals were fully spent. In general, the result obtained after 500 or so signals had been passed produced an ordering that did not change if signals were allowed to continue propagating until completely spent. If something like this is generally true of TEMPERAMENTAL systems, and I think it likely is, then the system can stop computation at this signaling limit and not necessarily when signals are spent. Since the limit is low relative to the number of signals that would be passed if all were allowed to expire, the system can probably avoid activation levels spiraling towards infinity even if the input signals are fairly strong. Most likely, the number of signals that need to be sent is proportional in some way to the number of relevant connections involved in the computation.

Finally, the nodes in the model compete with each other primarily indirectly, by gathering connections. I feel there must be a place in the learning for more enemy or even purely inhibitory connections. The problem is how to make them. We can intuitively see that some actions compete with one another even though they do not disable one another. For instance, the `flee` and `tease` actions in a game of tag somehow "compete" in the sense that they are exclusive options in a single scenario. But it is not at all obvious to me how to provide a general heuristic for establishing these enemy connections and, if necessary, adjusting their weights.

8.2 Areas to develop

Each problem listed above suggests an area of the model which needs more development. In addition, there are areas not even addressed by the model that should be tackled. The problem of discovering spurious connections could be alleviated considerably if SALMON could learn by experimenting as well as observing. SALMON has the ability, as he stands, to make predictions about any aspect of his environment. There should be a way to make him predict, check his results against what actually happened and then try to discover the culprit(s) responsible for any incorrect predictions. Of course, the problem of how to incorporate environmental feedback in learning is universal in both semantic and connectionist systems.

SALMON's problems arise not only from bad connections, but also from a relatively naive way of spreading activation. He could possibly be provided with introspective mechanisms for thinking about his connections, their meaning, and then noticing and correcting deficiencies in how activation is spread through them. Along these lines, an activation level could be provided to each node for each kind of connection. Thus, it would have a focus activation, a performance activation and a using activation. Multiple activation levels would provide a smoother interface to the semantics when more complicated actions have to be performed. The drawback would be that signals between nodes would have to carry more than just numerical information. In general, the nodes could move towards being something like frames with slots for each type of connection. Each slot has multiple default values, the connections attached to them, and the weight on the connection represents the strength of the default assumption by the system. Activation is spread as in a normal connectionist network and the slots become filled with the highest activated nodes connected to them.

SALMON should be able to categorize attributes and objects. Ideally, it should connect up attributes to objects in such a way that the psychological prototypes investigated by Rosch (1978) emerge. This would require that a TEMPERAMENTAL system be able to establish and retract nodes as well as connections. These nodes would then have to play some role in the reasoning process, perhaps conducting or filtering activation for their members the way actions direct the choosing of attributes. Also, frequency tabulations on the category nodes can help in deciding if a strong connection is, nevertheless, irrelevant.

TEMPERAMENTAL systems should have goals and priorities that bias certain actions and direct attention, and hence input. The goals should compete with one another and establish perspectives in the system. They should help in organizing the categories discussed in the previous paragraph and the organization should support planning. Incorporating goals will probably require meta-connections (connections to connections) and defining "types" for groups of connections that can be discovered and classified. The system could then activate the "type" just as if it were another node, thus bringing into prominence the connections associated with it. Just as semantic procedures are associated with action nodes, learned parameters would be associated

with meta-connection nodes and become dominant when the node is active. This is all still very speculative, though.

The game of tag would be just one environment out of many a full TEMPERAMENTAL system experienced. Upon entering or considering an environment, the system would need to "crystallize" the appropriate set of responses. The effect would be analogous to calling up a script, but the process would be analogous to having a fluid super-saturated by many elements and knowing the proper way of stirring up the solution so that just the proper one falls out. This may be as difficult as it sounds but meta-connections may be the key.

SALMON uses its dynamic attributes to react inside its environment, but it does not have any idea that the attributes themselves are systematically related. For instance, the attributes distance(1,3) and distance(3,5) are just two nodes to SALMON. It has no concept of before or after for them, and consequently could never purposefully move from distance(1,3) to distance(3,5). Heuristics for organizing spatial and other attributes systematically through connections need to be developed.

Overall, SALMON was a learning experience, not a technological or theoretical advance. As it stands, neither SALMON nor the TEMPERAMENTAL method are either useful nor theoretically significant. However, there is also much promise for the system. The success of SALMON in his domain, although not perfect, was encouraging. The problems that occurred are problems that were anticipated, and I do not think their solutions entail significant revisions to the model as it stands but rather an expansion of its capabilities into a larger domain of mentality.

References

[1] Agha, Gul A. (1988) *Actors:A model of Concurrent Computation in Distributed Systems.* The MIT Press, Cambridge, Mass.

[2] Arbib, Michael A. (1989) *The Metaphorical Brain 2.* John Wiley and Sons, New York.

[3] Brachman, Ronald J. (1985) "'I Lied About the Trees", Or, Defaults andDefinitions in Knowledge Representation.' *The AI Magazine*, 6.3, Fall : pp. 80–93.

[4] Carpenter, Gail A. and Grossberg, Stephen. (1987) 'Invariant Pattern Recognition and Recall By an Attentive Self-Organizing ART Architecture in a Nonstationary World.' *Proceedings of the IEEE International Conference on Neural Networks*. Vol. 1.: pp. 736–742.

[5] Carpenter, Gail A.; Grossberg, Stephen. (1987) 'ART 2: Self-Organization of Stable Category Recoginition Codes For Analog Input Patterns.' *Proceedings of the IEEE International Conference on Neural Networks*. Vol. 1.: pp. 727–735.

[6] Churchland, Patricia. (1989) *Neurophilosophy*. The MIT Press, Cambridge, MA.

[7] Davis, Lawrence and Steenstrup, Martha. (1987) "Genentic Algorithms and Simulated Annealing: An Overview." *Genetic Algorithms and Simulated Annealing*. Morgan Kaufmann Publishers, Inc., Los Altos, CA.: pp. 1–11.

[8] Holland, J.H. (1975) *Adaptation in Natural and Artifical Systems*. University of Michigan Press, Ann Arbor, MI.

[9] Hofstadter, Douglas. (1983) "The Architecture of Jumbo." *Proceedingof the 2nd Machine Learning Workshop*: pp. 161–170.

[10] Johnson, R. Colin, (1988) *Cognizers*. Chapel Brown, New York.

[11] Lee, Y.C., ed. (1988) *Evolution, Learning, and Cognition*. World Scientific, New Jersey.

[12] Minsky, M. (1975) "A Framework for Representing Knowledge." *The Psychology of Computer Vision*. P.H . Win ston (ed.) McGraw Hill , New York: 211–277.

[13] Minsky, M. (1986) *The Society of Mind*. Simon & Schuster, New York.

[14] Reeke, George N, Sporns, Olaf, and Edelman, Gerald. (1989) "Synthetic Neural Modeling: Comparisons of Population and Connectionist Approaches." *Connectionism in Perspective*. Pfeifer, R; *et al* (eds). North-Holland , New York: pp 113–139.

[15] Rosch, Eleanor and Lloyd, Barbara. (1978) *Cognition and Categorization*. Lawrence Erlbaum Associates, Publishers, New York.

[16] Rumelhart, D.E., McClelland, J.L. and the PDP Research Group. (1986) *Parallel Distributed Processing. Vol. 1: Foundations*. MIT Press, Cambridge, MA.

[17] Schmidhuber, Jürgen. (1988) "The Neural Bucket Brigade" *Connectionism in Perspective*. Pfeifer, R; et al (eds). North-Holland, New York: pp 429–437.

[18] Von Seelen, W., Shaw, G. and Leinhos, U.M. (eds.) (1988) *Organization Of Neural Networks*. VCH, Federal Republic of Germany.