Appears in the Journal of Adaptive Behavior (Vol. 1, No. 1, pp. 3-37)

A Model of Primate Visual-Motor Conditional Learning

Andrew H. Fagg Michael A. Arbib

ahfagg@pollux.usc.edu arbib@pollux.usc.edu

Center for Neural Engineering and Computer Science Department

University of Southern California

Los Angeles, California 90089-2520

TR # USC-CNE-92-01

Abstract

Observations of behavior and neural activity in premotor cortex of monkeys learning to pair an arbitrary visual stimulus with one of a set of previously learned behaviors are modelled with a network comprising a large number of "motor selection columns." Reinforcement-learning is used to recognize new visual patterns and acquire the appropriate visual-motor conditions. The architecture employs a distributed representation in which a single pattern is coded by a small subset of columns. A column is initially able to respond to many different inputs; as it learns to trigger a motor program, its responses become more narrowly defined. Each column's output is a set of votes for the various motor programs. The votes for each program are collected by "selection units" which drive a winner-take-all circuit to determine whether or not a particular motor program is executed. The model is successful in reproducing the sequence of behavioral responses given by the subjects, as well as a number of phenomena that have been observed at the single-unit level. Finally, we offer a comparison to the backpropagation learning algorithm that demonstrates key principles that have been designed into our algorithm.

Key Words: Conditional Learning, Reinforcement Learning, Columns, Distributed Representations, Premotor Cortex, Primates.

Running Head: Visual-Motor Conditional Learning

1. Introduction

Mitz, Godshalk and Wise (1991) examine learning-dependent activity in the premotor cortex of two rhesus monkeys required to move a lever in a particular direction in response to a specific visual stimulus. Figure 1 shows the protocol and expected response for one such trial. The monkey is initially given a ready signal, which is followed by a visual stimulus (instruction stimulus, IS). The monkey is then expected to wait for a flash in the visual stimulus (trigger stimulus, TS), and then produce the appropriate motor response. The four possible motor responses are: move the handle left, right, down, or no movement. When a correct response is produced, the subject is rewarded and a stimulus is picked randomly for the next trial. On the other hand, when an incorrect response is produced, no reward is given and the same stimulus is kept for the next trial.

Figure 1 about here

During the initial training phase, the two subjects were trained to perform the task with a fixed set of visual stimuli. This phase taught the protocol to the subjects, including the four appropriate motor responses. Through the second phase of learning, which we model here, the subjects were presented with novel stimuli and were expected to produce one of the four previously-learned motor responses. It was during this phase that single-unit recordings were taken from neurons in the primary- and premotor cortices.

Figure 2 demonstrates the results of a typical set of second-phase experiments. The left-hand column shows the correct response, and each row of the right-hand column shows the monkey's response over time. Two features of this figure are particularly interesting. First, there are a number of cases in which the monkey exhibits an incorrect response, and even though it does not receive the positive feedback, it will continue to output the same response for several additional trials. In most of these cases, the no-go response is given, which appears to be the "default" response. The second interesting feature, demonstrated in almost half of these response traces, is that once the monkey exhibits the correct response, it may give one or more improper responses before producing the correct response consistently.

Figure 2 about here

This behavior may be captured at a high level by considering a separate decision "box" for each stimulus. [1] A box maintains a measure of confidence that each motor output is correct, given its particular input stimulus. When the system is presented with a stimulus, the appropriate box is chosen, and a motor output is selected based upon the confidence vector. When the monkey exhibits an incorrect response, positive reinforcement is not given. Therefore, the likelihood of the last response should be reduced slightly, while the probability of picking one of the other motor responses increases. When a correct response is given, the confidence value for the exhibited response is rewarded by a slight increase. Our challenge is to construct a neural implementation that is both distributed in nature and is capable of identifying novel stimuli as they are presented. The following data gives some hint as to how the implementation might look.

Mitz et al. recorded primarily from cells in the premotor cortex. A variety of cell types were identified. *Anticipatory* cells tend to fire between the ready signal and the IS. *Signal* cells respond to the presentation of a movement-related stimulus, whereas *set-related* cells fire after the IS, in preparation for a particular motor response. *Movement-related* cells respond to the presentation of the TS and in some cases stay on for the duration of the movement. Most cells exhibit multiple response properties (e.g. combined set- and movement-related responses). Signal- , set-, and movement-related cells typically fired in correlation with a particular motor response. Thus, for any particular visual stimulus, only a small subset of cells fired significantly during the execution of the corresponding motor program. As learning progressed, some cells were seen to increase in their response activity towards a stimulus, while others decreased in their response.

[1] A more formal treatment of these computing elements (stochasitic learning autom
found in Bush (1958) and Williams (1988).

Figure 3 shows normalized activity and performance curves for one experiment plotted against the trial number. The normalized activity is computed for a particular stimulus by looking at the activity of the ensemble of units that show an increase in activity over the course of learning. The performance curve is computed as a sliding window over a set range of trials.[2] It is important to note that the performance curve precedes the activity curve in its sudden increase.

Figure 3 about here

Mitz et al. identified a number of key features of learning-dependent activity in these experiments :

a. The increase in cell activity (for those cells that increased their activity over the learning period) was closely correlated with, but was preceded by, the improvement in performance. Similar relations were seen in signal-, set-, and movement-related units.

b. Activity of a particular unit for correct responses was, in most cases, higher than that during incorrect responses in the same movement direction.

c. Activity for correct responses during times of good performance exceeded that at times of poor performance.

d. When multiple sets of novel stimuli were presented to the monkey, similar learning-dependent responses of the signal-, set-, and movement-related cells were observed for stimuli that yielded the same motor response.

e. The activity pattern resulting from a familiar stimulus closely correlated with the activity due to novel stimuli (after learning), although this correlation was not a perfect one. This and (d) demonstrate that a similar set of premotor neurons are involved in responding to all stimuli mapping to the same motor output. From this, we can conclude that the pattern discrimination is probably not happening within the premotor cortex. If this were the case, one would expect separate groups of cells to respond to different stimuli, even if these stimuli mapped to the same motor output.

2 Mitz et al. do not go into detail as to the exact formulas used to compute these me

This set of experimental results presents a set of modelling challenges. We here list both those that we meet in the present model, and those that pose challenges for future research.

1. Our neural model is capable of learning the stimulus/motor response mapping, producing qualitatively similar response traces to those of Fig. 2:
   a. The appropriate number of trials are required to learn the mapping.
   b. Incorrect responses are sometimes given several times in a row.
   c. Correct responses are sometimes followed by a block of incorrect responses.

The model can generate the variety of response traces, with the network starting conditions determining the actual behavior.

2. The model produces realistic normalized activity/performance curves (Fig. 3) :
   a. The performance curve leads the activity curve by a number of learning trials.

Challenges that remain for future work include:

3. A complete model will also reproduce the temporal activity of various neurons in the premotor cortex: anticipatory units, signal-related units, set-related units, and movement-related units.

2. Model Overview

Much of neural network research has concentrated upon supervised learning techniques, such as the generalized delta rule (backpropagation; e.g., Rumelhart, Hinton, and Williams, 1986). In our modelling efforts, we have chosen to explore other algorithms within an architecture that can be related (at least at a high level) to the biological architecture, while perhaps also offering greater computational capability.

Backpropagation suffers from the problem of global representation — in general, every unit in the network, and thus every weight, participates in a single input-output mapping. As a result, the gradient in weight space due to a single pattern will contain a component for almost every weight, and therefore learning can become rather slow. A related problem is that, in order to maintain an older memory for at least some amount of time, the learning of a new memory cannot alter the older memory to all but a very small degree. This is difficult to accomplish if all units are participating in every computation and all weights are altered as a result of learning.

With these problems in mind, we have sought *distributed representations* in which a single pattern (or task) is coded by a small subset of the units in the network. Although different subsets of units are allowed to overlap to a certain degree, interference between two patterns is minimized by the non-overlapping components. Inspired by the cell activities observed by Mitz et al., we see a unit that has not learned to participate in a motor program as being able to respond to a wide range of different inputs. As learning progresses within this unit, its response increases significantly for some stimuli, while it decreases for the remainder.

Figure 4 about here

The Architecture of the Model

The primary computational unit in the proposed model is the *motor selection column*, each consisting of two neurons: the *feature detector unit* and the *voting unit* (Figure 4). The overall network is composed of a large number of these columns, each performing a small portion of the stimulus-to-motor program mapping.

The feature detector recognizes small portions (micro-features) of the input stimulus. Due to the distributed construction of the circuit, a particular signal unit is not restricted to recognize patterns from a single stimulus, but may be excited by multiple patterns, even if these patterns code for different responses. A particular signal unit is physically connected to only a small subset of the input units. This enforces the constraint that only a small subset of the columns will participate in the recognition of a particular pattern. As will be discussed later, this reduces the interference between patterns during learning.

Figure 5 about here

The voting unit receives input from its corresponding feature detector as well as from a noise process and the threshold modulator. Based upon the resulting activity, the voting unit instantiates its "votes" for one or more motor programs. The strength of this vote depends upon the firing rate of this neuron and the strength of the connection between the voting unit and the motor program selector units. As shown in Figure 5, the votes from each column are collected by the *motor program selection units*, labeled "Left", "Right", "Down", and "No-Go". The final activity of these units determines whether or not a particular motor program is activated, and thus executed. The winner-take-all circuit (Didday 1976) ensures that at most one motor program will be activated at any one time. This is accomplished through the inhibitory neuron ("S"). When more than one motor program selection unit becomes active, this unit sends an inhibitory signal to the array of motor program selection units. The result is that all of the units will begin turning off, until only one is left (the unit receiving the largest total of votes from the motor columns; Amari and Arbib 1977). At this point, the one active unit will cue the execution of its motor program.

The reception of the trigger stimulus (TS) causes the execution of the selected motor program. Although only a single motor program selection unit will typically be active when the TS is received, two other cases are possible : none active, and more than one active. In both cases, the No-Go response is executed, irrespective of the state of the No-Go motor program selection unit. Thus, the No-Go response may be issued for one of two reasons : explicit selection of the response, or when the system is unsure as to an appropriate response by the time the TS is received.

The global threshold modulator and the local noise processes play an important role in the search for the appropriate motor program to activate. When a new visual stimulus is presented to the system, the feature detector units will often not respond significantly enough to bring the voting units above threshold. As a result, no voting information is passed on to the motor program selection units. The threshold modulator responds to this situation by slowly lowering the threshold of all of the voting units. Given time (before the TS), at least a few voting units are activated to contribute some votes to the motor program units. In this case, a response is forced, even though the system is very unsure as to what that response should be.

Noise processes have been used as an active element of several neural models. Noise is used in Boltzmann machines as a device for escaping local minima and as a way of breaking symmetry between two possible solution paths (Hinton and Sejnowski 1986). Although the problem of local minima is not a concern in this work, the problem of choosing between two equally desirable solutions is a considerable one and so a small amount of noise is injected to randomly bias solutions so that a choice is forced within the winner-take-all (WTA) circuit. There are some cases in which two motor program selection units receive almost the same amount of activity. Due to the implementation of the winner-take-all circuit (see the Appendix for the equations), this situation may send the system into oscillations, where it is not able to make a decision. The added noise coming into the voting units helps to bias one of the motor programs, to the point where a decision can be made quickly. Moreover, rather than always selecting the motor program that has the highest incoming feature support, the system is enabled by the noise to choose other possibilities. This keeps the system from prematurely committing to an incorrect solution, maintaining diversity during the search process (Barto et al. 1983). Thus, the amount of time dedicated to the search process can be significantly decreased.

The details of the equations for each of the units in Figures 4 and 5 are given in the Appendix A.

Learning Dynamics

Learning in this model is reinforcement-based, and is implemented by modifying two sets of synapses: the sensory input to feature detector mapping and the voting unit to motor program selection unit mapping, i.e., the weight matrices **Win,feature** and **Wvote,motor** corresponding to the fan-in and fan-out of Fig.4, respectively (see Appendix A). Only those columns that participate in the current computation adjust their weights. In the experimental setup, positive reinforcement is given when the monkey exhibits a correct response, but not otherwise. Similarly, in the model, a scalar quantity called **reinforcement** is set by the teach to +1 if the selected motor program is correct, and to -1 otherwise.

However, a special case occurs when the system is unable to make a decision within the allotted time (causing the "No-Go" response to be selected). Two possible situations have occurred: no motor program selection units are active, or more than one are active. In the first case, the reinforcement term is set to +1 by the system itself, regardless of the teacher feedback. Therefore, the currently active columns are rewarded, ensuring that the next time the pattern is presented, these columns will yield a greater response. Thus, they will have a greater chance of activating one of the motor program selection units. Without this additional term, negative reinforcement from the teacher is disastrous. The negative reinforcement further decreases the response of the already poorly responding columns, further decreasing their response. The result is a self-reinforcing situation that can never discover the correct response.

In the second situation, where more than one motor program selection unit becomes active at one time, the reinforcement term is set by the system to -1. This decreases the response of all columns involved, adjusting the input to the two (or more) motor program selection units until one is able to achieve threshold significantly before the other(s). It is at this point that the symmetry between the two is broken.

When positive reinforcement is given, the weights leading into the feature detector units are adjusted such that the feature detector better recognizes the current sensory input. In the case of negative reinforcement, the weights are adjusted in the opposite direction, such that the current input is recognized by the feature detector unit to an even lesser degree. Note that this reinforcement depends on whether or not the overall system response was correct, not on the output of any individual motor selection column. We thus have:

$$\text{lgain} = \begin{cases} \text{negative\_factor\_f} & \text{if reinforcement} < 0 \\ 1 & \text{if } 0 \leq \text{reinforcement} \end{cases}$$

$$\Delta W_{\text{in,feature}} = \text{reinforcement} \cdot \text{lgain} \cdot \text{lrate}_f \cdot (\text{Input} \cdot \text{Voting}^T) \quad (1)$$

where **reverse** is a scalar function that is typically returns +1, but returns -1 in the case described below; **lrate_f** is the learning rate coefficient for the stimulus-to-feature and **$(\text{Input} \cdot \text{Voting}^T)$** is the outer product of the Input and Movement vectors. The **lgain** factor is simply used to scale the effect of negative reinforcement relative to positive reinforcement. In this case, the effect of negative reinforcement on the weights is intended to be less than that of positive reinforcement. This is done because negative reinforcement can be very devastating to columns that are just beginning to learn the appropriate mapping.

To simultaneously weaken those weights that are not strengthened by reinforcement, we then set

$$W_{in,vote} = \text{Normalize}(W_{in,feature} + \Delta W_{in,feature} \wedge \text{W\_in\_feature\_mask}) \quad (2)$$

where **Normalize** is a function that L1-normalizes the vector of weights leading into each feature detector unit to length 1.[3] **W_in_feature_mask** is a matrix of ones and zeros that determines the existence of a weight between the corresponding voting and motor program selection units. The elements of this matrix are point-wise multiplied with those of $\Delta W_{in,feature}$ to mask out weight deltas for weights that do not exist (see Appendix A for further details).

Equation (2) produces a competition between the weights associated with a particular unit. Thus, the weights are self-regulating, forcing unneeded or undesirable weights to a value near zero. If a column continues to receive negative reinforcement (as a result of being involved in an incorrect response), then it becomes insensitive to the current stimulus, and is reallocated to recognize other stimuli.

The voting unit to motor selection mapping is adjusted similarly. Positive reinforcement increases the weight of the synapse to the correct motor program. When negative reinforcement is given, the synapse is weakened, allowing the other synapses from the voting unit to strengthen slightly through normalization. Thus, more voting power is allocated to the other alternatives:

$$\Delta W_{vote,motor} = \text{reinforcement} \cdot \text{lgain} \cdot \text{lrate}_v \cdot (\text{Voting} \cdot \text{Motor}^T) \quad (3)$$

$$W_{vote,motor} = \text{Normalize}(W_{vote,motor} + \Delta W_{vote,motor}) \quad (4)$$

where $\mathbf{lrate_v}$ is the learning rate coefficient for the voting-to-motor response mapping. A similar type of reinforcement learning is utilized in Barto et al. (1983, see later discussion).

$W_{in,feature}$ and $W_{vote,motor}$ are initially selected at random (See Apendix A for more details). When a response is generated, learning is applied to each of the columns that are currently participating in the computation. The learning objective of an individual column is to recognize particular patterns (or subpatterns) and to identify which of the possible motor programs deserves its

---

[3] Y = Normalize(X) $\Rightarrow$ $Y_i = \dfrac{X_i}{\sum_j |X_j|}$

votes. Equation (1) attempts to create feature detectors that are specific to the incoming patterns. As these feature detectors begin to better recognize the correct patterns, the activity of the signal units will grow with respect to the pattern, thus giving the column a larger voting power. The feature detecting algorithm is related to the competitive learning of von der Malsburg (1973) and Grossberg (1976) (discussed further in Rumelhart and Zipser 1986). Individual columns learn to become feature detectors for specific subpatterns of the visual stimulus. However, a column does not recognize a pattern to the exclusion of other patterns. Instead, several columns participate in the recognition at once. In addition, a column is responsible for directly generating an appropriate motor output. Therefore, the update of the feature detector weights not only depends upon recognition of the pattern (as in competitive learning), but also upon whether or not the network generates the correct motor output. In the case of a correct response, the feature detector weights become better tuned towards the incoming stimulus, as in the von der Malsburg formulation. For an incorrect response, the weights are adjusted in the opposite direction, such that recognition is lessened for the current input.

Note that in this scheme, all of the columns that participate in the voting are punished or rewarded as a whole, depending upon the strength of their activity. Thus, a column that votes for an incorrect choice may still be rewarded as long as the entire set of votes chose the correct motor program. This method works, in general, because this "incorrect column" will always be active in conjunction with several other columns that do vote appropriately and are always able to overrule its vote. This scheme is similar to that used by Barto et al. (1983) in that one or more elements may correct for errors made by another element. In their case, however, the correction is made sequentially through time, rather than in parallel.

It should be noted that there is a tradeoff in this algorithm between the speed of learning and the sensitivity to noise. Because this protocol always gives the correct feedback and the possible motor outputs are finite and discrete, this tradeoff is not quite as evident. Imagine the case where learning is very fast and the reinforcement function occasionally makes a mistake (as can easily be imagined in real-world situations). If the system has discovered the correct response, but is then given no positive reinforcement for the correct response, extremely rapid learning would cause this response to lose favor completely. Likewise, if an incorrect behavioral response is positively rewarded, a high learning rate would cause the incorrect response to rise quickly above the alternatives.

3. Simulation Results

The following experiment utilized a variant of the winner-take-all (WTA) algorithm in the simulation, referred to as first-past-the-pole WTA. Rather than requiring that the network settle down into a stable state ($\frac{d\,Motor_{mem}}{dt} \approx 0$ for all motor program selection units), the first unit that achieves a membrane potential above the threshold is declared the winner. In the case that more than one unit activated at the same instant, the standard winner-take-all circuit is used to squelch the activity of all but one. Using this particular algorithm allows for a faster simulation, since more time is required if the units must settle down to equilibrium.

During the testing/learning trials, a pattern is randomly presented to the system. The overall control system waits until a single selection is made (after the TS is presented), before moving on the the next trial. When the network produces an incorrect answer, the same pattern is presented on the next trial (as in the primate experiments). This protocol allows for much quicker learning, as opposed to a completely random sequence of stimulus/response pairs (see later experiments).

Primary Experiments

Figure 6 shows the behavioral traces resulting from a single experiment. In two of the three traces, the network produces a correct answer, and then attempts other choices (given the identical pattern). This happens due to the fact that the voting strengths are influenced by the noise process. Even though a correct answer has been given, there is still a probability that another answer will be output at a later time. Eventually, however, the learning biases the correct motor program to a level sufficiently above the noise. After this point, the correct motor program is always chosen.

Figure 6 about here

For the same experiment, the pattern of activity for a particular motor response and the behavioral performance were compared to those of the monkey. The overall activity of the resulting voting unit response is measured by using the final voting unit activity pattern (i.e. after learning) as a reference. The overall activity is defined as the dot product between this reference and the voting unit activity pattern from every trial in the learning sequence.[4] As in the results reported by Mitz et al., the normalized activity and performance curves for a single motor response are plotted together. The performance is computed by low-pass filtering the performance value (where 0 corresponds to incorrect response and 1 to correct; see Appendix A).

Figure 7 shows the resulting set of curves for one such experiment. The solid curve corresponds to the activity measure and the dotted line is the behavioral performance of the subject. These values are plotted over the number of trials. In this, as in several other experiments, the performance begins its steady increase 3 to 4 trials before the activity measure becomes significant.

Figure 7 about here

When a naive network is first tested, the presentation of a pattern causes some random set of columns to be active, as determined by the initial weight values from the input units to the feature detector units. Based on the strength of the pattern match, the corresponding voting units may not immediately become active, but instead have to wait for the Threshold Modulator to lower the threshold to an appropriate level. Given time, this function forces the system to vote for some response, even though it is not very sure about what the correct response might be.

With respect to identifying the correct response, positive reinforcement gives the system more information than does negative reinforcement. For the case of positive reinforcement, we are telling the

[4] Mitz et al. define overall activity as the degree of activity change for those units t
increase of activity during learning. Without knowing more detail about the actual
that was used, we believe that the dot product captures the essence of what has been

system what the correct response is (specific feedback), but negative reinforcement only tells the system that the correct response is one of three choices (nonspecific feedback).

Because the system is essentially guessing on these initial trials, the performance is very poor at first. Therefore, the system is primarily receiving negative reinforcement, keeping the overall response activity at a low level. An occasional correct response, in combination with the negative feedback for other choices, begins to bias the voting unit output towards the correct motor program selection unit. In turn, this effect begins to increase the probability of selecting the correct motor response.

Once the performance of a set of columns begins to increase, the positive feedback becomes significant enough to reward the correctly responding feature detector units on average, thus switching over from nonspecific to specific feedback information (in the weight update equations, the **reinforcement** term becomes +1 for the most cases). In Figure 7, as in other experiments, the overall activity does not begin to rise significantly until the performance passes the 0.5 mark. This also appears to be the case in most of the graphs provided by Mitz et al. Once the performance is correct on average, the activity of the feature detector units belonging to the "correct" set of columns increases. This increase comes from the fine tuning of the feature detector weights towards the incoming pattern. As a result, we see an overall increase in activity in response to the learned stimulus, and, most importantly, we see this increase **after** the increase in performance.

Figure 8 about here

In addition to looking at the overall activity of the network, it is also possible to examine an individual column's response to input stimuli as learning progresses. Figure 8A shows the response activity of one voting unit from the previous experiment. In this particular case, the unit initially responds equally well to two different stimuli. As learning progresses, however, the response to one stimulus grows to a significant level. Ultimately, this unit becomes allocated to the recognition of the stimulus pattern that maps to the Left response.

Figure 8B represents the same unit's orientation towards a particular motor program, as measured by the weight from the unit to the motor program selection unit. Initially, the unit supports the four motor program selection units almost equally, but within 12 trials, the weight corresponding to the Leftward motor unit begins increase above the others possibilities. After learning has completed, this weight completely dominates the others.

Figure 9 about here

Changes in Protocol

In initially examining the protocol described by Mitz et al., we found it interesting that when the monkey responded incorrectly to a particular stimulus, the same stimulus was presented for the next trial. This repetition was continued until the monkey produced the correct response. The question that came immediately to mind was why a totally random presentation sequence was not used. We presented this question to our model through a simple modification of the protocol. The results shown in Figure 9 represent a typical behavioral trace under this new protocol. In this case, the system requires almost twice as many trials before it begins to perform the task perfectly. This is especially evident in the Rightward response.

This effect can best be explained by looking at the competition between the different stimuli. The degree of competition is determined by the amount of overlap between the sets of columns that are activated by each of the stimuli. In addition, certain stimuli may activate their set of columns more strongly than other stimuli, due to the initial random selection of weights. This activity difference can give the stronger stimulus a slight advantage in the learning process, since a weight update is related to the degree of activation of the voting unit. Therefore, given that a significant overlap exists between groups of columns, as well as an activity bias towards one or more stimuli, the learning induced by the stronger patterns can often cancel out any learning caused by the weaker stimulus. In the original protocol, this interference is not as much of a problem, since incorrectly mapped stimuli are allocated a larger number of consecutive trials. Within the new protocol, the probability of a favorable set of trials is relatively low.

Figure 10 about here

Figure 11 about here

Figure 10 shows the overall activity curve corresponding to the Right response in the above experiment. It is interesting to note that the activity curve increases prior to the performance curve. This can be explained by looking closer at the individual unit participation for the Rightward mapping. In this case, only a single column takes on the task of performing this particular mapping. During the early stages of learning, the network quickly learns the other three mappings. This particular column initially responds to both the Rightward and Downward stimuli (Figure 11 A). When the Rightward stimulus is presented, the support to the columns is so weak that the system does not make a decision in the allotted time. Therefore, the input/feature weights are adjusted to maintain recognition of the Rightward stimulus (see equation (1) and equation (2)). As shown in Figure 11B, the system finally discovers the correct motor program to output at about trial 45. At this point, though, it still significantly supports the Downward response, but not enough to make incorrect decisions.

Figure 12 about here

Reversal Experiments

Another set of experiments performed on the model asked about the system's behavioral and neural responses after a reversal takes place. In this experiment the network is presented with the standard set of four novel stimuli. After a given number of trials, the teaching system switches the mapping of two responses. In this case, the stimulus that originally mapped to the No-Go motor response, now maps to the Down motor response, and vice versa. In looking at this experiment, we are interested in seeing how quickly the network is able to recover from the change in mapping and in understanding the underlying neural basis for this change. Figure 12 shows the behavioral results of one such experiment.

After 26 trials, the visual/motor mapping had been learned perfectly for all cases. The first few responses that are generated after the reversal correspond to the original mapping. The system requires only a few trials of negative reinforcement to the Left and Right responses before the original mappings lose their dominance. At this point, the system continues its search as in the other experiments.

Figure 13 about here

Figure 14 about here

The activity/performance curve for the Left response is shown in Figure 13. Recall that the activity curve is computed by taking the dot product between current activity of the voting unit vector and the same vector of activity after the learning is complete. The sudden jump in the activity curve indicates point at which the reversal takes place. This jump happens because although the column continues to respond to the same stimulus, the stimulus is now supposed to map to the No-Go response (which has been plotted). This and Figure 14 A demonstrate that the column maintains its mapping to the specific stimulus. Figure 14 B (the output weights from the same column) demonstrates that it is these weights that are adjusted to deal with the new mapping. Note in this figure, the reversal takes place over just a few trials (both in the reduction of the Downward weight and the increase of the No-Go weight.

Comparison to Other Algorithms

In measuring the performance of a new learning algorithm, a natural question is how well this algorithm compares to others. Here, we briefly present a comparison to the backpropagation learning algorithm (Rumelhart, Hinton, and Williams, 1986), with the intention of illustrating one of our main computational themes: distributed but sparse coding of a single input/output mapping.

As discussed earlier, the backpropagation algorithm suffers from the fact that, in general, every hidden unit participates in every input/output mapping that the network represents. Thus, almost every weight is affected by the learning of a single pattern. As a result, different input/output mappings often work against each other as learning proceeds. This is especially evident as the task complexity increases.

Vanilla backprop (Rumelhart, et al.) provided the basis for this comparison. Slight modifications were made to simulate the limited reinforcement information provided by the teacher. Further details are supplied in Appendix C.

The overall performance of the networks are measured by examining the time required to learn the mapping. This is computed by taking the sum of the number of trials required to learn each individual mapping. The number of trials to learn a mapping consists of the trials up to and including the first trial from which the performance is perfect (after which no incorrect responses are given).

The overall time required by our algorithm for the case demonstrated in Figure 6 is 17. On average, the algorithm requires 24 trials (5 experiments with different starting conditions). In the case of backpropagation, learning rates were chosen to maximize the network performance as best as possible. The result was that an average of 80 trials were required (average over 10 experiments).

Although this is an encouraging result, it is most important to consider the performance of the networks as the task becomes more difficult. To demonstrate an advantage, it must be shown that the performance of one algorithm degrades more rapidly than the other. To test this, we presented the networks with a sequence of training sets, each more difficult than the one before. For our experiments, the training patterns began to overlap to greater and greater degrees, making it more difficult to distinguish them. One example of this is given in Appendix B.

Figure 15 shows the result of our experiments. Here, the learning time is plotted against task difficulty for three cases : backprop (lrate = 1.2), backprop (lrate = 1.4), and our algorithm. Backpropagation demonstrates a clear degradation in performance, especially for the most difficult tasks. Our algorithm, on the other hand, shows only a slight linear increase in learning time with task difficulty. It is also important to note the great variability in performance for the backpropagation algorithm. This is a strong indication of the algorithm's extreme dependence on initial conditions. This is not an effect that is as visible with our algorithm.

Figure 15 about here

It is not our intention here to claim that our algorithm provides a superior learning capability to backpropagation. In fact, much is left to be understood, especially regarding the point of saturation of the network. We do, however, maintain that this theme of distributed, but sparse, coding is an important one as we begin to formulate the learning algorithms that will carry us to a new generation of neural network research.

4. Discussion

Our model has primarily addressed the computational issues involved in learning appropriate stimulus/motor program mappings. However, we believe that the functional role of voting units within our network may be related to that of set units within the premotor cortex. The actual visual/motor mapping is considered to be taking place further upstream from premotor cortex (within the $W_{in,feature}$ weights in our model). We believe this to be the case, due to the fact the Mitz et al. observed similar set unit activity patterns in response to *different* visual stimuli that mapped to the same motor response.

The motor programs, themselves, are most likely stored in regions further downstream from premotor cortex, as is the circuitry that chooses a single motor program to execute (motor program selection units and the winner-take-all circuit of our model). We believe that the cerebellum may play a primary role at this level.

This model was successful in meeting a number of the challenges set forth earlier It produces a behavior similar to that which was seen in the monkey experiments (goals 1a - c), and also produces normalized activity/performance curves that are qualitatively similar to the experimental data. Although neither of these two challenges (goals 2a and b) were explicitly designed into the neural algorithm, the two features dropped out of the original formulation of the model. Finally, the model produces neuronal activity phenomena that are representative of those observed by Mitz et al. (Mitz challenges a-c).

The primary computation within the model was performed using distributed coding of the information, thus demonstrating that not all of the relevant information need be present at a single location to perform a complex task. Rather, a distributed set of computers, each acting with a limited set of information, is capable of producing a global decision through a voting mechanism, although in this model votes were cast in a more centralized manner than is appropriate for a more faithful model of the brain's circuitry.

The concept of the column served to bind together a minimal set of computational capabilities needed to perform the local computation. This structure was then replicated to solve the more global computation. The claim here is not that a cortical column in the neurophysiological sense consists strictly of feature detector and voting units, but that a local organization is sufficient to perform a significant part of the computation. Allowing all neurons to connect to all other neurons is not practical from a hardware standpoint, and may impede the learning process.

The learning algorithm was a local one. Except for the reinforcement signal, the update of a particular weight only used the information available locally (the activation of the presynaptic and postsynaptic neurons, and the surrounding weights that shared common dendritic tree). This feature adds to the biological plausibility of the process, and may also have important consequences such as easy implementation in VLSI. In addition, the learned function was stored in a local manner (any particular column was active for only a subset of the inputs). This type of representation can limit the amount of interference between different input patterns, and thus the learning may be faster and more effective in achieving its goal. A first hint of this is demonstrated in our backpropagation experiments.

The model, however, does not attempt to account for the different types of units observed within the premotor cortex (goal 3). In particular, Mitz challenges d and e are not in general satisfied by the model (multiple stimulus patterns that map to the same motor response do not necessarily activate the same set of columns). This is due to the normalization operation that is performed on the input to the feature detector units. Again, in the premotor cortex of monkey, one would expect a set unit to continue participating in the same motor program after a reversal has taken place, rather than responding continually to the same input stimulus. This would be due in part to the fact that the monkey has already created and solidified its motor programs in memory (during the first stage of learning). Because the mapping from visual stimulus to motor program is transient (and/or most recent), the synaptic changes should be more likely taking place at this location.

Finally, the behavior of the model under different experimental conditions may yield some predictions as to the monkey's behavior under similar conditions. As discussed earlier, the use of a completely random sequence of stimuli (as opposed to repeating trials in which the incorrect response was given) significantly hindered the system's ability to learn the visual-motor mapping. From this observation, we would like to posit that the monkey would suffer a similar fate given the completely random trial presentation. This is not meant to say that the monkey would necessarily be unable to learn the task, but that the learning would at least be significantly more difficult. The degree to which this is true can ultimately feed back to future work on this model, since it would tell us something about the degree of interference between the different mappings.

The next generation of models will be developed with two goals in mind :

1. To increase the biological verisimilitude of the present model of visual-motor conditional learning, e.g., to refine the description of cell types to reproduce the temporal activity of various neurons in the premotor cortex: anticipatory units, signal-related units, set-related units, and movement-related units. This involves looking at the monkey's generation of the time course of activity in relation to the instruction stimulus and trigger stimulus, rather than the simple pairing of stimulus and response. We will also study distributed, neurophysiologically plausible implementations of the winner-take-all circuit, and seek to determine their localization in the primate brain.

2. We will begin to approach the problem of how the legal motor responses are learned. In order to accomplish this, we must look more deeply at the first stage of learning described in the protocol used by Mitz et al. In general, when a monkey is taught a task of this difficulty, the experimenter will take the monkey through a series of learning stages. The first of these learning stages trains the monkey to perform very simple subtasks. In successive stages, the monkey creates more complex motor programs either by augmenting the previously-learned programs with additional capabilities or by combining several programs together. The investigation of such capabilities will extend the functional analysis of Arbib (1990) to a neural network model of the interactions of many brain regions.

Appendix A: Model Details

Our model has been implemented in the Neural Simulation Language (NSL; Weitzenfeld 1991), and executes on Sun workstations. Both NSL and this model are available via anonymous ftp from the Brain Simulation Laboratory at USC. For further information, email may be sent to ahfagg@pollux.usc.edu.

Model Dynamics

The computational model is implemented using the leaky-integrator model (e.g., Arbib 1989), in which each neuron is represented by a membrane potential and a firing frequency (Figure A). The dynamics of the generalized neuron are defined by the following :

$$\tau \frac{dmem}{dt} = - mem - threshold + \Sigma \; inputs$$

firing = f(mem)

where :

τ is the time constant of integration.

mem is the membrane potential of the neuron.

threshold is the neuron's internal threshold.

inputs is the set of external inputs into the neuron.

firing is the firing rate of the neuron.

f() is a function (typically non-linear).

Figure A about here

In the following set of equations, neural states are represented as vectors. Two vectors are connected through a set of weights through either a one-to-one connection scheme, or a fully-connected scheme (all neurons of the input layer are connected to each of the neurons of the output layer). The first case is represented by a vector addition, while the second case is represented by convolving the vector I with a "mask" of synaptic weights W to yield W*I.

The state of the feature detector units are described by the equations :

$$\tau_f \frac{d\,Feature_{mem}}{dt} = - Feature_{mem} - Threshold_f + W_{in,feature} * Inputs$$

$$Feature = NSLramp(Feature_{mem})$$

where :

τ_f is the time constant (scalar) of membrane potential change.

Threshold$_f$ is the internal threshold of the feature detector units (a scalar).

Feature$_{mem}$ is a vector of membrane potentials for the set of feature detector units. The initial condition (at the beginning of a trial) of this vector is $Feature_{mem} = -Threshold_f$ for all elements of the vector.

W$_{in,feature}$ is the weight matrix between the input stimulus and the feature detector units. These weights are updated by learning (see below).

Inputs is the vector of stimulus inputs.

Feature is the vector firing rates of the feature detector units.

$$\mathbf{NSLramp(x)} = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } 0 \leq x \end{cases}$$

The behavior of the voting units is governed by the equations :

$$\tau_v \frac{d\,Voting_{mem}}{dt} = - Voting_{mem} - Threshold_v(t) + Feature + Noise$$

$$Voting = NSLsat(Voting_{mem})$$

where :

τ_v is the time constant of the voting units.

Voting$_{mem}$ is the membrane potential of the voting units (vector). The initial conditions are $Voting_{mem} = - Threshold_v(0)$ for all units.

Threshold$_v$(t) is the time-dependent threshold determined by the threshold modulator (a scalar).

Feature is the vector of feature unit firing rates. Each voting unit receives input **only** from its corresponding feature unit.

Noise is a low-amplitude noise process, that changes slowly relative to τ_v (vector).

Voting is the firing rate vector.

Depending upon the state of the voting units, the motor program selection units / winner-take-all circuit attempts to choose a single motor program to activate. This selection process is governed by the following equations:

$$\tau_m \frac{d\,Motor_{mem}}{dt} = - Motor_{mem} - Threshold_m + W_{vote,motor} * Voting - S + Motor + Motor\_Noise$$

$$Motor = NSLthresh(Motor_{mem})$$

$$\tau_S \frac{d\,S_{mem}}{dt} = - S_{mem} - Threshold_S + \sum_i^N Motor[i]$$

$$S = NSLramp(S_{mem})$$

where :

τ_m is the motor selection unit time constant.

Motor$_{mem}$ is the membrane potential of the motor selection units (a vector). The initial conditions are $Motor_{mem} = - Threshold_m$ for all elements in the vector.

Threshold$_m$ is the scalar threshold of the motor selection units.

W$_{vote,motor}$ is the weight matrix representing the projection from the voting units to the motor selection units.

S is the firing rate of the inhibitory neuron. The initial condition of this neuron is S = 0.

Motor is the firing rate vector. Initially, Motor = 0 for all elements.

Motor_Noise is a low-amplitude noise process, that changes slowly relative to τ_m (a vector).

S$_{mem}$ is the membrane potential of the inhibitory neuron (a scalar, since there is only one). The initial condition for this neuron is $S_{mem} = - Threshold_S$.

Threshold$_S$ is the threshold of the inhibitory neuron.

$$\textbf{NSLthresh(x)} = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } 0 \leq x \end{cases}$$

A selection is considered to have been made when only one motor program selection unit is firing.

The dynamic equations controlling the noise process are described as follows. At each simulation step, a decision is made as to whether or not a new noise state is to be selected.

```
if(random(0, 1) < change_probability)
    Noise = noise_gain * random(-1, 1)
```

where :

random(0, 1) represents a value selected from a uniform distribution between 0 and 1 (a scalar).

change_probability is the probability of change for the simulation step (a scalar).

Noise is a vector of current noise values.

noise_gain is a scalar that determines the amount of noise that is injected into the system.

random(-1, 1) is a vector of values selected from a uniform distribution between -1 and 1.

For a particular column, the result is a low-amplitude noise process that changes at a low frequency.

A similar vector of noise processes act as input to the motor program selection units :

```
if(random(0, 1) < motor_change_probability)
    Motor_Noise = motor_noise_gain * random(-1, 1)
```

where :

motor_change_probability is the probability of change for the simulation step (a scalar).

Motor_Noise is a vector of current noise values.

motor_noise_gain is a scalar that determines the amount of noise that is injected into the system.

Weight Initialization

As discussed in the text, only a limited number of weights are allowed to take on values other than zero. The existence of a weight is represented in a matrix of zeros and ones of the same dimensionality as the weight matrices. When the network is initialized, these matrices are first computed as the following :

W_in_feature_mask = compute_connections(W_in_feature_probability)

W_vote_motor_mask = compute_connections(W_vote_motor_probability)

where :

compute_connections(prob) returns a matrix of appropriate size, which contains zeros and ones. The probability of any element being assigned to one is determined by the parameter (<prob>).

W_in_feature_probability and **W_vote_motor_probability** are network paramters.

The initial weights are computed as :

W_in_feature = Normalize((input_weight_bias + randomize_weights()) ^ W_in_feature_mask)

W_vote_motor=Normalize((voting_weight_bias+randomize_weights()) ^ W_vote_motor_mask)

where :

randomize_weights() returns a matrix of weight elements (matrix dimensions are of appropriate size). Each element is chosen from a uniform distribution over [0,1].

input_weight_bias and **voting_weight_bias** are network parameters (scalars).

Performance Measures

Overall activity (ACT) at a particular instant of time is computed by examining the sequence of correct and incorrect behavioral responses.

$$\mathrm{ACT}(0) = 0$$

$$\mathrm{ACT}(t+1) = \alpha\ \mathrm{ACT}(t) + (1 - \alpha)\ \mathrm{Performance}(t+1)$$

where :

ACT(t) is the overall activity after t trials.

α is the low-pass filter gain parameter.

$$\mathrm{Performance}(t) = \begin{cases} 0 \text{ if incorrect on trial } t \\ 1 \text{ if correct} \end{cases}$$

Appendix B: Simulation Setup

The following section outlines the set of parameters used to produce the results presented in this paper. Table 1 shows the complete list of paramters and the values used in the simulation.

Table 1 about here

A number of parameters play a crucial role in the behavior of the network. These are further discussed here :

W_in_feature_probability determines how likely that a connection exists between an input unit and a feature unit. For this work, it was important to keep this parameter at a low value (0.3). This serves to minimize the number of columns that will respond at all to an input stimulus, thus minimizing the interference between columns. If set too low, not enough columns will react to a particular input.

input_weight_bias determines the distribution of weight values for those weights that do exist. A high value forces the existing weights synapsing on a particular feature to be very similar. On the other hand, a low value causes the weights to be more randomly distributed. In the case of our simulation, this value is set to 1.0 (a low value), yielding a reasonable distribution that allows different columns to respond differently to an individual stimulus. Thus, the weight initialization procedure biases the symmetry breaking between stimuli that goes on during the learning process.

noise_gain determines the magnitude of noise injected into the voting units. It is important that this value is significantly less than **init_threshold_v**. Otherwise, the voting unit may fire spontaneously (without feature unit support) before the threshold is lowered.

noise_change_probability is set such that the noise value changes slowly relative to the the time constant of the voting unit (τ_v). When the noise changes at this time scale, on average, the effects of the noise are allowed to propagate through the system before the noise value changes again. Thus, in the early stages of learning, different groups of voting units may fire given the same input stimulus, allowing the system to experiment with what the appropriate set of voting units might be. If the noise changes too quickly, then the average affect will be very little noise injected into the system. Therefore, all eligible columns will fire together, and not in different subsets.

The constraints on **motor_noise_gain** and **threshold$_m$** are similar.

lrate$_f$ determines how much effect that one trial will have on the weight matrix that maps from the input units to the feature units (the value used in these simulations was 0.4). When set too low, the slope of the overall activity curve begins to decrease and the system will take longer before it achieves perfect performance. On the other hand, setting this parameter too high will amplify the interference between the various weights (this is critical during the early stages of learning). Thus, the learning of one pattern may erase (in one trial) the information associated with another pattern.

lrate$_v$ is the learning constant for the vote-to-motor weight matrix (the value used was 0.035). Setting this constant too high, will cause the system to very quickly commit columns to particular motor responses. The result is that the network is able to learn the mapping much quicker than in the cases discussed in this paper. Although it appears to be advantageous to use a higher parameter value, we would move away from the behavioral results seen in the Mitz experiments. In addition, the network may become more sensitive to interference, a problem that will show itself as the task difficulty is increased.

negative_factor_f scales the effect of negative reinforcement on the network. When this value approaches 1, the effect of a negative signal can be devastating to the network (see discussion of learning dynamics). In general, we found that too high of a value will decrease the slope of the overall activity curve (evident when the network begins to produce the correct answer, but then tries other responses).

Training Patterns

The patterns shown in Table 2 were used to train the network for most of the above experiments. The right-hand column denotes the expected motor response.

Table 2 about here

For this case, the input patterns are orthogonal. Other training sets that were used for the comparison with backpropagation included overlapping patterns. One such training set is shown in Table 3.

Table 3 about here

Appendix C: Backpropagation Setup

Vanilla backprop serves as the basis for comparison with our algorithm, with a few minor modifications. The network parameters that were used are shown in Table 4.

Table 4 about here

On a single trial, an input is first presented to the network. The activity is passed through the hidden units to the output units. The output unit with the highest activation is considered to be the winner, and thus determines the output motor response. Internally, the desired (expected) output is represented as a vector that is the same dimensionality as the output unit vector (4 units). All elements of this vector are 0, except the element corresponding to the desired response, which is set to 1. It is this vector that is compared to the outputs to determine the errors at each of the output units.

The fact that the teacher provides only a limited amount of teaching information to the network is simulated by only providing feedback information on the unit that responds maximally. This is accomplished by setting all elements of the error vector to 0, except for the element that corresponds to the winning motor program. This value is left as the difference between the desired and expected output. It is this modified error vector that is propagated back through the network to compute the appropriate weight updates.

As errors are backpropagated through the network, the error is typically computed at each node by the equation :

$$\delta_i = f'(net_i) * \sum_k \delta_k \cdot w_{ik}$$

where :

δ_i is the error at node i.

net_i is the net input into unit i.

$f'(net_i)$ is the derivative of the activation of the unit (i.e $f(net_i)$ = activity of the unit). When f() is the sigmoid, $f'(net_i) = f(net_i) * (1 - f(net_i))$.

$\sum_{\mathbf{k}} \boldsymbol{\delta}_{\mathbf{k}} \cdot \mathbf{w}_{\mathbf{ik}}$ denotes the propagation of errors from the units to which unit i feeds activity.

(see Rumelhart et al. for more details)

Very often, a unit's activity ($f(net_i)$) will near either 0 or 1, causing $f'(net_i)$ to approach zero. The result of this is a considerable slowing down of the learning process. Thus, we have chosen to use the following formula for single unit error instead :

$$\delta_i = (f'(net_i) + \text{delta\_increment}) * \sum_k \delta_k \cdot w_{ik}$$

In general, the result is a considerable decrease in learning time, since δ_i is less likely to approach 0.

In making our comparison, we felt it important to give our network the best challenge possible, rather than selecting "any old learning algorithm" to which to compare. In the spirit of this, we also spent some time experimenting with different learning rates, and found that 1.2 and 1.4 appeared to perform the best. Because the weights were updated after every trial, the momentum term was set to zero. This is done because the learning performed for one trial does not necessarily have any relation to the weight update for the following trial.

Acknowledgements

This work was supported in part by a fellowship from the Graduate School, the School of Engineering, and the Computer Science Department of the University of Southern California (to A.H.F.), and in part by a grant from the Human Frontiers Science Program (to M.A.A.). We thank Steven Wise for correspondence which formed the basis for this project, and George Bekey for his help in the shaping of this document. In addition, we would like to thank Rob Redekopp for his aid in performing some of the backpropagation experiments.

References

Amari, S., and Arbib, M. A., 1977, "Competition and Cooperation in Neural Nets," in *Systems Neuroscience* (J. Metzler, Ed.), Academic Press

Arbib, M. A., 1989,*The Metaphorical Brain 2 : Neural Networks and Beyond*, John Wiley & Sons

Arbib, M.A., 1990, "Programs, Schemas, and Neural Networks for Control of Hand Movements: Beyond the RS Framework," in *Attention and Performance XIII. Motor Representation and Control* (M. Jeannerod, Ed.), Lawrence Erlbaum Associates

Barto, A.G., Sutton, R.S., Anderson, C.W., 1983, "Neuron-like Adaptive Elements That Can Solve Difficult Learning Control Problems," *IEEE Transactions on Systems, Man, and Cybernetics*,, SMC-13, pp. 834 - 846

Bush, R. R., Mosteller, F., 1958, *Stochastic Models for Learning*, New York, Wiley.

Didday, R.L., 1976, "A Model of Visuomotor Mechanisms in the Frog Optic Tectum," *Math. Biosci.* 30: 169-180.

Grossberg, S., 1976, "A Theory of Visual Coding, Memory, and Development: Part 1. Parallel Development and Coding of Neural Feature Detectors," Biological Cybernetics, 23, pp. 121-134

Hinton, G. E., Sejnowski, T. J., 1986, "Learning and Relearning in Boltzmann Machines," in *Parallel Distributed Processing : Explorations in the Microstructure of Cognition, Volume 1: Foundations* (J. L. McClelland and D. E. Rumelhart, Eds.), Bradford Book/The MIT Press, Chapter 7

Mitz, A.R., Godshalk, M., and Wise, S.P., 1991, "Learning-dependent Neuronal Activity in the Premotor Cortex. Activity during the Acquisition of Conditional Motor Associations," *Journal of Neuroscience*, June, 11 (6): 1855 - 1872

Rizzolatti, G., 1987, "Functional Organization of Inferior Area 6," in *Motor Areas of the Cerebral Cortex* (CIBA Foundation Symposium 132), Wiley, Chichester, pp. 171-186

Rumelhart, D.E., Hinton, G.E., and Williams, R.J., 1986, Learning Internal Representations by Error Propagation, in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (Rumelhart, D., and McClelland, J., Eds.), The MIT Press/ Bradford Books, Vol.1:318-362.

Rumelhart, D. E., Zipser, D., 1986, "Feature Discovery by Competitive Learning," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition Volume 1: Foundations* (J. L. McClelland and D. E. Rumelhart, Eds.), Bradford Book/The MIT Press, Chapter 5

Taira, M., Mine, S., Georgopoulos, A.P. , Murata, A., and Sakata, H., 1990, "Parietal Cortex Neurons of the Monkey Related to the Visual Guidance of Hand Movement," *Exp. Brain Res.*, 83:29-36

von der Malsburg, C., 1973, "Self-organizing of Orientation Sensitive Cells in the Striate Cortex," *Kybernetik*, 14, pp. 85-100

Weitzenfeld, A., 1991, "NSL : Neural Simulation Language, Version 2.1," Center for Neural Engineering, University of Southern California, Technical Report 91-05, July

Williams, R. J., 1988, "Toward a Theory of Reinforcement Learning Connectionist Systems," TR NU-CCS-88-3, July

Figures

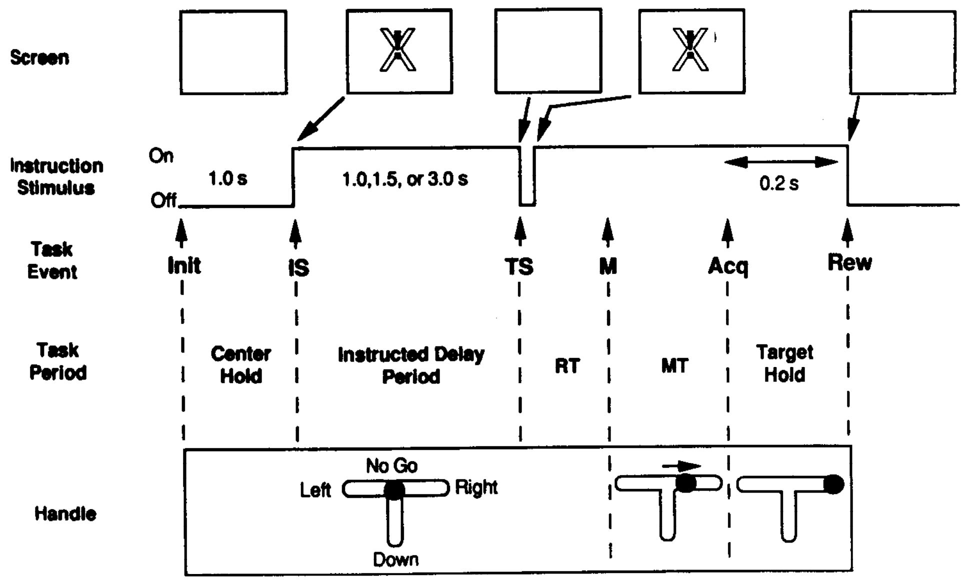
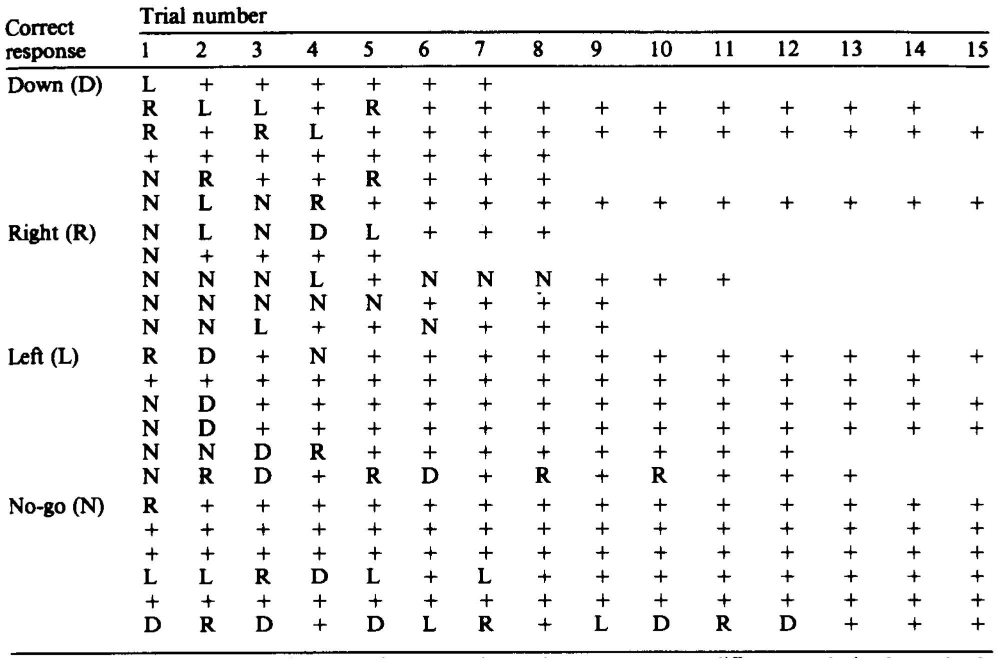


Figure 1: Top row : visual stimulus as seen on the video screen. Second row : temporal trace of the visual stimulus. Third and fourth rows : Primary events and periods of the experimental trial. Fifth row : expected motor response. (From Mitz et al., Figure 1; reprinted by permission of the Journal of Neuroscience.)



Correct response	Trial number 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Down (D)	L	+	+	+	+	+	+								
	R	L	L	+	R	+	+	+	+	+	+	+	+	+	
	R	+	R	L	+	+	+	+	+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+							
	N	R	+	+	R	+	+	+							
	N	L	N	R	+	+	+	+	+	+	+	+	+	+	+
Right (R)	N	L	N	D	L	+	+	+							
	N	+	+	+	+										
	N	N	N	L	+	N	N	N	+	+	+				
	N	N	N	N	N	+	+	+	+						
	N	N	L	+	+	N	+	+	+						
Left (L)	R	D	+	N	+	+	+	+	+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+	+	+	+	
	N	D	+	+	+	+	+	+	+	+	+	+	+	+	+
	N	D	+	+	+	+	+	+	+	+	+	+	+	+	+
	N	N	D	R	+	+	+	+	+	+	+	+			
	N	R	D	+	R	D	+	R	+	R	+	+	+		
No-go (N)	R	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	L	L	R	D	L	+	L	+	+	+	+	+	+	+	+
	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
	D	R	D	+	D	L	R	+	L	D	R	D	+	+	+

Figure 2: Samples of responses to novel stimuli given example specific expected motor responses. Each row represents only those trials from an experiment that correspond to a specific desired motor response. Correct answers are indicated with a '+'. (From Mitz et al., Table 1; reprinted by permission of the Journal of Neuroscience.)

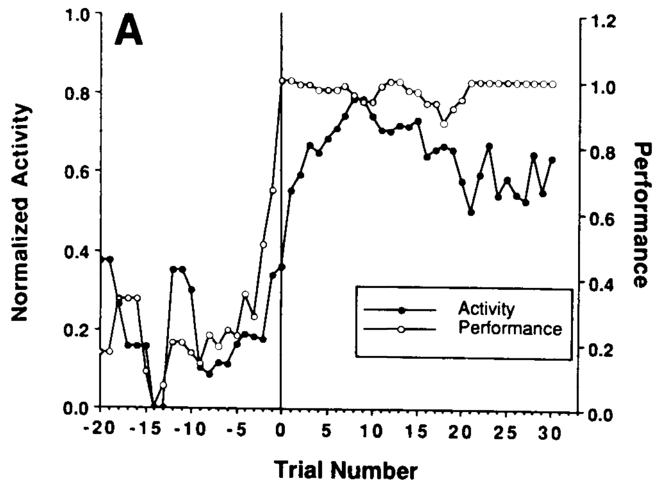


Figure 3: Normalized activity and performance curve plotted as a function of trial for the presentation of a novel stimulus. The rise in overall performance precedes that of cellular activity by about 3 trials. (From Mitz, Figure 3; reprinted by permission of the Journal of Neuroscience.)

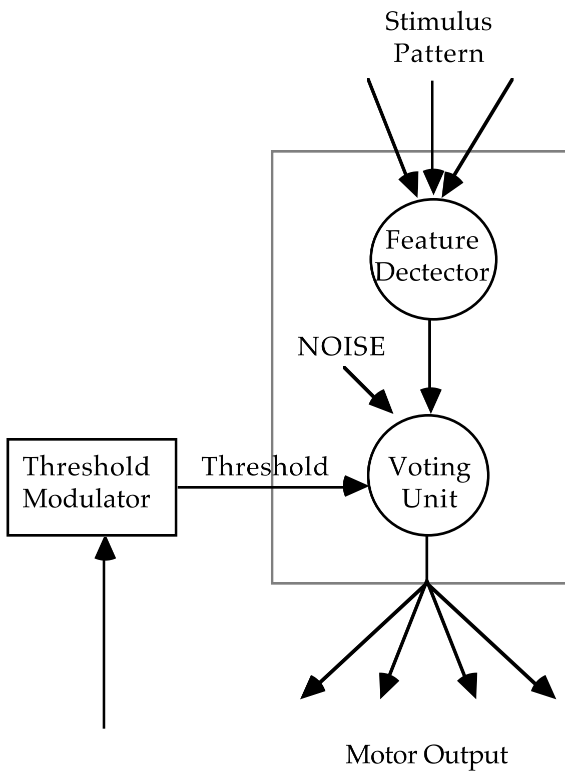


Figure 4 : The motor selection column model. The feature detector detects specific events from the sensory input. The voting unit produces a vote for an appropriate set of motor programs. This unit, along with the noise and the threshold modulator, implements a search mechanism.

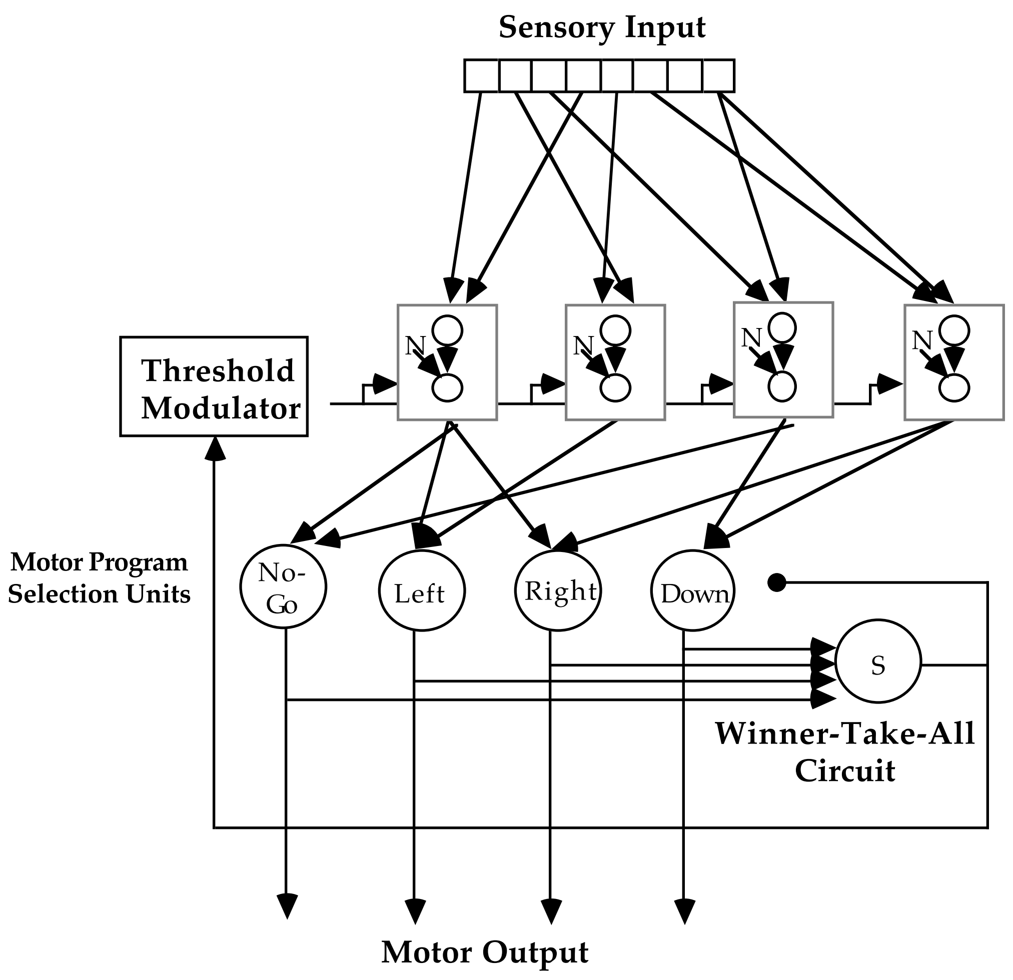
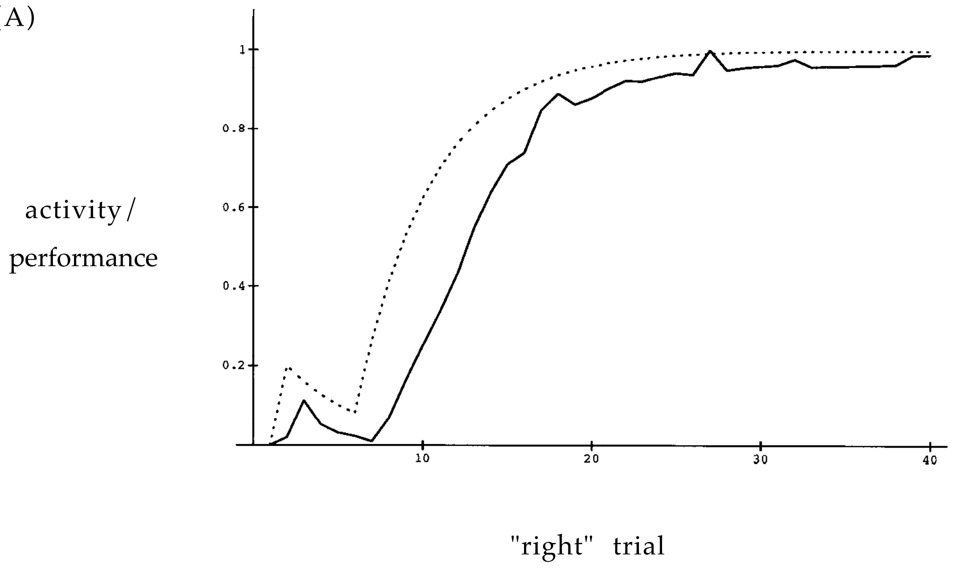


Figure 5 : The motor program selection model. A set of motor selection columns votes for the motor responses. The votes are collected by units representing the activity of the schemas for each of the legal motor responses. The winner-take-all circuit ensures that only one motor program is selected.

```
N : + + + + + + + + + + + + + + + + + + + + + + + + + + + + + +

L : N + R + + + + + + + + + + + + + + + + + + + + + + + + + + +

R : N + L D L N + + + + + + + + + + + + + + + + + + + + + + + +

D : L L N N + + + + + + + + + + + + + + + + + + + + + + + + + +
```

Figure 6: The behavioral responses of one experiment broken into sequences corresponding to a particular stimulus/motor output pair. +'s indicate correct responses, where letters indicate an incorrect response of a particular type.

(A)



(B)

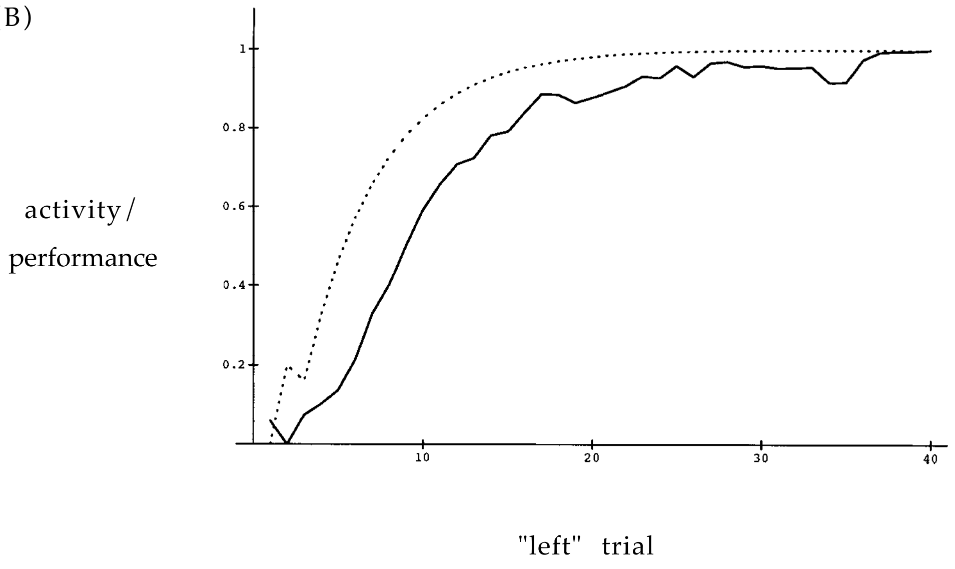
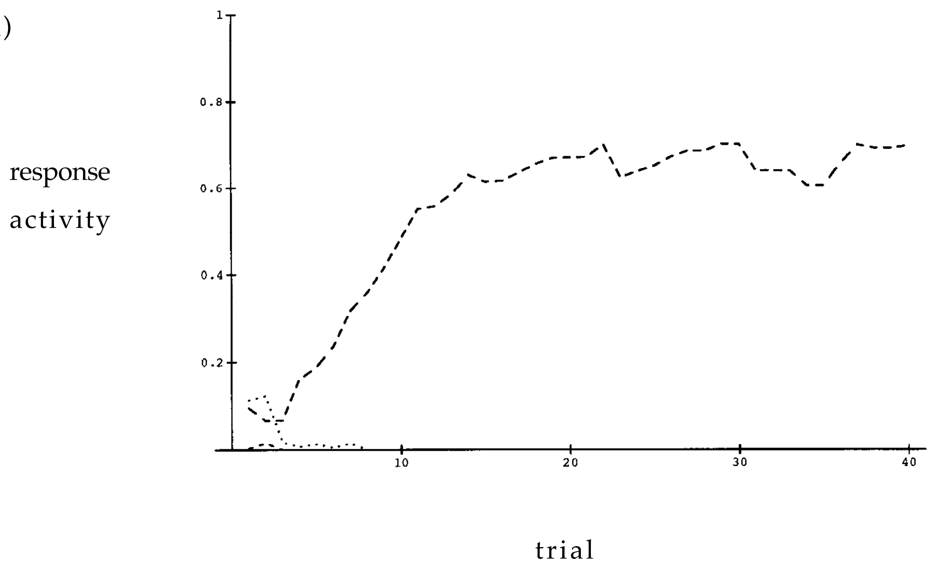


Figure 7: Overall activity (solid) curve and performance (dotted) curve plotted against trial for the (A) "Right" and (B) "Left" responses. As in the experimental curves, the performance curve begins to increase prior to the increase in the activity curve. Note that the trial axis represents only those trials for which the "Right" and "Left" responses are expected, respectively.

(A)



(B)

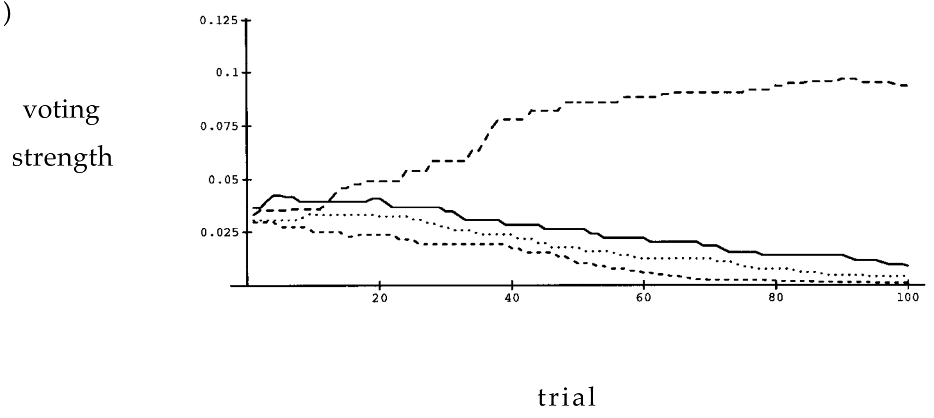


Figure 8: (A) Activity of a single voting unit plotted against trial number. The four curves represent responses to each of the four input stimuli (solid = no-go, long dash = left, dotted = right, and short dash = down). Note that trial number *N* corresponds to the Nth occurrence of each of the four stimuli, and does not necessarily correspond to the same moment in time. (B) Evolution of the voting strength of the same unit. The four curves (designated as above) represent the voting strength to each of the four motor program selection units. Note that in this graph, the trial axis represents all trials.

```
N : D R D L + + R D + + + + + + + + + + + + + + + + + + + + + +

L : + D + + + + + + + + + + + + + + + + + + + + + + + + + + + +

R : L D D N N N N N N N N N N + + + + + + + + + + + + + + + + +

D : L + + + + + + + + + + + + + + + + + + + + + + + + + + + + +
```

Figure 9: The behavioral responses of one experiment with a completely random sequence of stimulus presentations. Under these conditions, the task requires more trials of learning.

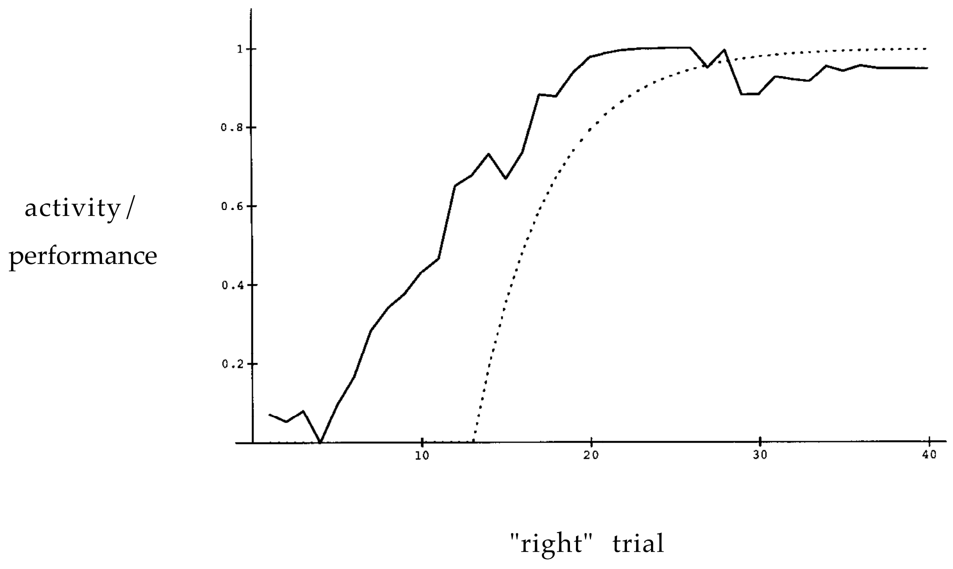
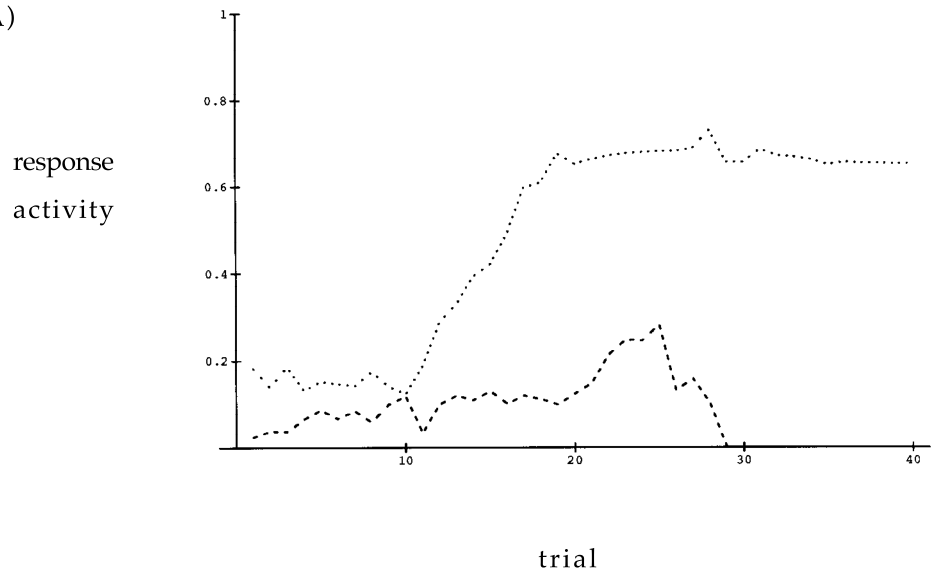


Figure 10: Overall activity response to the stimulus coding for the "Right" response (corresponding to Fig. 9). In this case, the increase in activity leads that of the performance (see discussion).

(A)



(B)

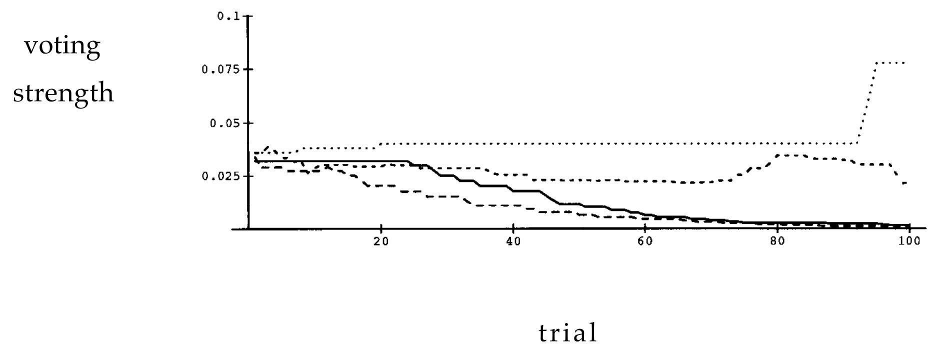


Figure 11 : (A) Activity of the one voting unit that learned to perform the mapping plotted against trial number. Because this is the only unit that learns the mapping, it determines the curve of Figure 10. (B) Evolution of the voting strength of the same unit. Only towards the end of the experiment (95th trial) does the unit discover the correct motor program selection unit.

```
N : + +                  D D D D D D R + + + + + + + + + + + + + +

L : N + R + + + +        + + + + + + + + + + + + + + + + + + + + +

R : N + L D L N + +      + + + + + + + + + + + + + + + + + + + + +

D : L L N N + + + + +    N N R + + + + + + + + + + + + + + + + + +
```

Figure 12: Behavioral response during a reversal task. The break in the strings indicates the point at which the reversal (between the No-Go and Down responses) takes place.

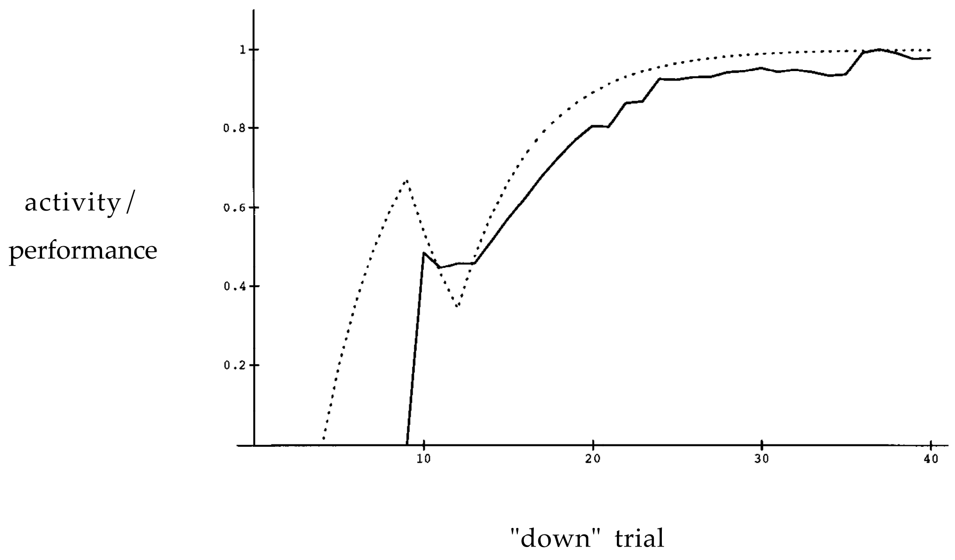
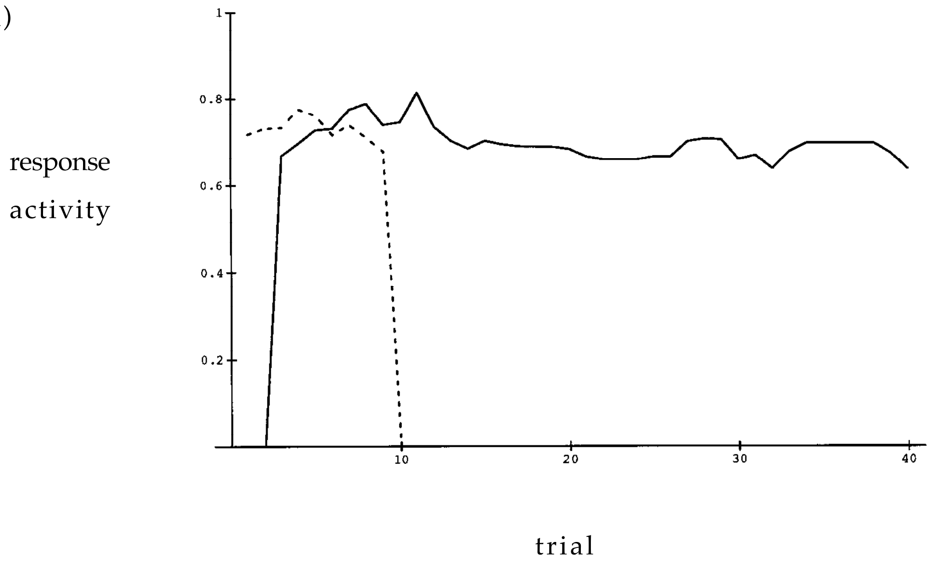


Figure 13: activity/performance curve for the reversal case ("Down" motor mapping). The solid (activity) curve corresponds to the overall activity in response to the stimulus that maps to the "Down" motor response, which switches between the 9th and 10th trials. The drastic change in overall activity after the reversal indicates that two separate sets of columns are being used to process the two different stimuli (recall that overall activity is measured by comparing the current activity pattern of the voting units to their activity pattern after learning is complete, in this case, trial 40). This also shows that the column continues responding to the same input before and after the reversal.

(A)



(B)

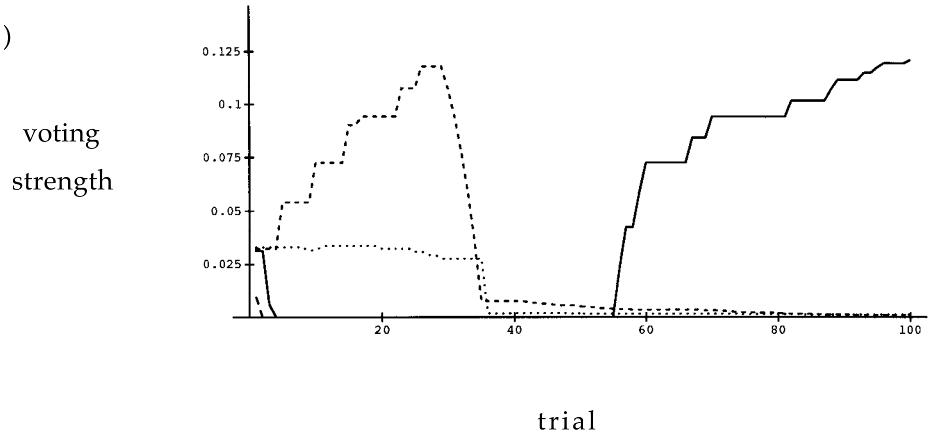


Figure 14: (A) A single unit's response to the various input stimuli over time. The solid curve represents the unit's response to the Nth occurrence of the stimulus that maps to the "No-Go" response. Likewise, the dashed curve represents the unit's response to the Nth occurrence of the stimulus that maps to the "Down" response. The drastic increase of the solid curve and decrease of the dashed curve indicate the point of reversal (after the 2nd occurrence of the stimulus that maps to "No-Go", and the 9th occurrence of the stimulus that maps to "Down, respectively). Note that the unit continues to respond to the same stimulus **after** the reversal, although the stimulus now maps to a different motor program.

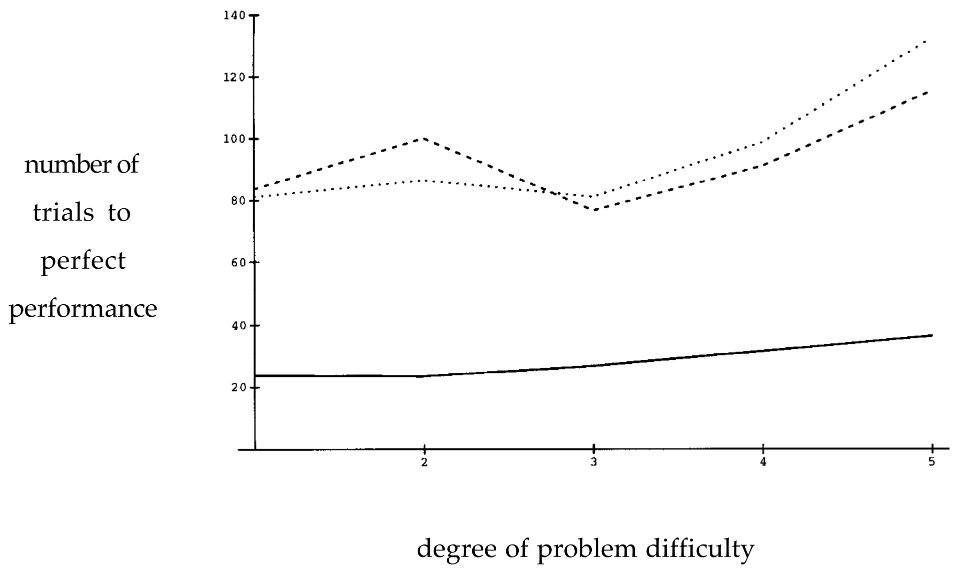


Figure 15: Learning performance versus degree of pattern overlap for the newly developed algorithm and backpropagation (solid = new algorithm, dashed = bp with lrate = 1.2, dotted = bp with lrate = 1.4). Performance is measured as the number of trials required for each pattern before perfect behavioral performance is reached. As the degree of pattern overlap increases, the patterns become harder to differentiate, and thus the problem becomes harder to learn. Note that the new algorithm requires significantly fewer trials to learn the problem, and as the problem difficulty increases, the performance does not degrade as quickly as the backpropagation experiments.

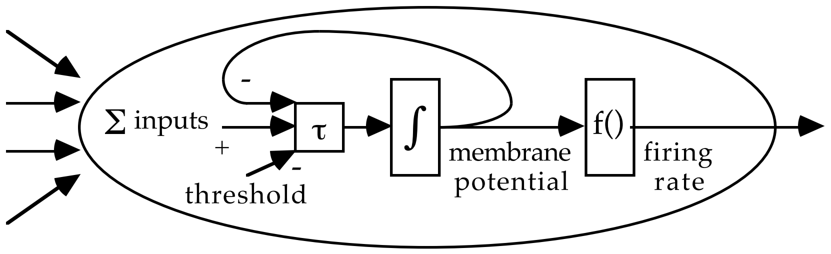


Figure A: The leaky integrator model of the neuron. The membrane potential of the neuron is affected by the current set of inputs, the neuron's threshold, and the current state of the neuron. The firing rate is a non-linear function of the membrane potential.

Tables

Table 1 : Model Parameters

Network Parameters		
num_columns	30	Number of columns in the middle layer.
num_inputs	14	Number of inputs into the columns.
num_choices	4	Number of motor program selection units (no-go, left, right, down)
NSL Parameters		
delta	0.01	Integration step
Weight Initialization		
input_weight_bias	1.0	Constant added to random weight value (see weight initialization)
W_in_feature_probability	0.3	Probability that a particular weight will exist.
voting_weight_bias	4.0	
W_vote_motor_probability	1.0	
Feature detector parameters		
Threshold_f	0.1	Threshold
τ_f	0.05	Time constant
Threshold Modulator		
init_threshold_v	0.2	Initial threshold (threshold is determined by the threshold modulator).
τ_{tv}	4.0	Time constant of threshold modulator.
Voting Unit		
τ_v	0.05	Time constant
noise_gain	0.045	Injected noise.
noise_change_probability	0.01	Determines how often the injected noise term changes value.
Motor Program Selecion Unit		
τ_m	2.0	Time constant
motor_noise_gain	0.05	Injected noise.
motor_noise_change_prob.	0.01	Determines how often the injected noise term changes value.
threshold_m	0.035	Motor program unit threshold.

Table 1 (cont) : Model Parameters

S (Inhibitory Unit)		
τ_S	0.5	Time constant of membrane potential.
threshold_S	0.1	Unit threshold.
Learning Parameters		
lrate_v	0.035	Voting/motor program selection unit lrate
lrate_f	0.4	Input/feature detector unit lrate
negative_factor_f	0.25	Input/feature factor for negative reinforcement
Protocol Parameters		
first_pole_mode	1	1 indicates first-passed-the-pole mode is turned on.
repeat_mode	1	1 indicates stimuli are repeated when an incorrect response is generated by the system.
max_time_counter	200	Maximum number of time steps alotted to the system for making a decision.
Analysis Parameters		
α	0.8	Used to compute average performance.

Table 2: Input patterns and expected motor responses.

Training Pattern	Expected Response
1 1 1 0 0 0 0 0 0 0 0 0 0 0	No-Go
0 0 0 0 0 0 0 0 0 0 1 1 1 0	Left
0 0 0 0 1 1 1 0 0 0 0 0 0 0	Right
0 0 0 1 0 0 0 1 0 1 0 0 0 0	Down

Table 3: Input patterns and expected motor responses (more difficult case). Each of the patterns overlaps at least one other pattern.

Training Pattern	Expected Response
1 1 1 0 0 0 0 0 0 0 0 0 0 0	No-Go
1 0 0 0 0 0 1 0 0 1 0 0 0 0	Left
0 0 0 1 0 1 1 0 0 0 0 0 0 0	Right
0 0 0 1 0 0 0 1 0 1 0 0 0 0	Down

Table 4 : Backpropagation parameters.

inputs	14	The number of input units.
hidden	30	Number of hidden units.
outputs	4	Number of output units.
update mode	individual	Weights are updated after every presentation of a pattern.
lrate	1.2, 1.4	Learning rates used for the results reported in this work.
momentum	0	Because update mode = individual, momentum has little meaning.
delta_increment	0.1	Added to f'(x) (See below).