Learning in the Cerebellum with Sparse Conjunctions and Linear Separator Algorithms

Harlan D. Harris and Jesse A. Reichler
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{*hharris,reichler*}*@uiuc.edu*

Abstract

This paper investigates potential learning rules in the cerebellum. We review evidence that input to the cerebellum is sparsely expanded by granule cells into a very wide basis vector, and that Purkinje cells learn to compute a linear separation using that basis. We review learning rules employed by existing cerebellar models, and show that recent results from Computational Learning Theory suggest that the standard delta rule would not be efficient. We suggest that alternative, attribute-efficient learning rules, such as Winnow or Incremental Delta-Bar-Delta, are more appropriate for cerebellar modeling, and support this position with results from a computational model.

1 Introduction

Many who have studied the cerebellum have concluded that it functions as a general-purpose supervised learner that regulates other circuits, particularly in motor control [1, 2]. The cerebellum is viewed as a vast collection of repeating modules, within which Purkinje cells, acting as feature detectors, are trained by climbing fibers to recognize patterns in the activations of their inputs.

Most models of the Purkinje cell system have been developed to account for particular motor skills [3]. In almost all of these, the Purkinje cell has been modeled as a Perceptron (or sometimes a linear unit), trained using some variant of the delta rule. Although these models have been successful at modeling toy versions of behaviors, little work has been done that looks at the abstract computational power of the cerebellum as a learning system. In order to develop a complete model of cerebellar function, important theoretical issues such as scalability, learning speed, and noise sensitivity are relevant. In particular, recent work in Computational Learning Theory has shown striking contrasts in efficiency among weight update algorithms for Perceptrons, particularly when the input is a high-dimensional input vector with a small proportion of relevant inputs. We discuss the applications of these results toward understanding the computational function of the cerebellum.

2 Cerebellum

The cerebellum receives numerous inputs from sensory systems and motor areas, and to a lesser extent, from higher-level areas. Based on neuroanatomical studies and observations of neurological patients, the cerebellum has long been thought to be involved in the tuning of motor control for precise and well-timed actions. In the discussion that follows, we describe the human cerebellum, although the structure is very similar in all mammals.

The cerebellum's inputs arrive directly from *mossy fibers* (MF), and indirectly from *climbing fibers* (CF) projected from the *inferior olive* (IO) in the brainstem. The output of the cerebellum is mostly via the *deep cerebellar nuclei* (DCN), which primarily project to motor control areas (M). Many researchers view this structure as a supervised learning system, with MFs providing input to the learner, and CFs providing the training signal. (Others disagree, eg. [4])

The microanatomy of the cerebellum (Figure 1) is well known, although the details and functions of the connections are controversial. The MFs project to the numerous *granule cells* (GC). Each GC receives synapses from only a small, random selection (say, 3 to 5) of the potential nearby MF inputs.

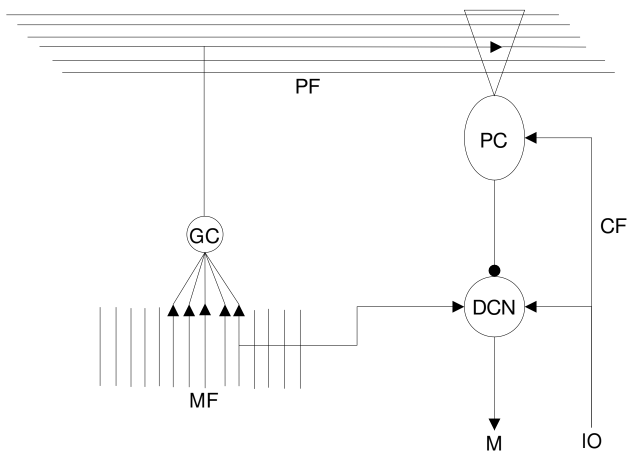

Figure 1: Microanatomy of the cerebellum, simplified.

GCs project axons which form long *parallel fibers* (PF), each contacting a large number of *Purkinje cells* (PC). A particular PC will usually have one synapse with a given PF, but will have perhaps 200,000 synapses with the many nearby PFs.

One CF generally wraps around each PC and its dendrites, with about ten PCs being contacted by each CF. Additionally, there are connections from MFs and CFs to the DCN, plus other cells (basket, Golgi, stellate) that receive inputs from GCs and elsewhere and inhibit the PCs and GCs. These other cells are often ignored in discussions of the processing of the PC, and we do so here as well. Finally, PC axons terminate on DCN cells, which project outside the cerebellum.

A number of computational models of cerebellar learning have been proposed [5, 3]. One of the first models of the cerebellum was introduced by Marr [1], who suggested that a simple associative learning process took place at the PF-PC synapses, with which the cerebellum could identify sensory patterns and activate motor programs in response. While subsequent research has contradicted some aspects of Marr's model, its description of the cerebellum as a pattern recognizer was influential.

A similar model of the cerebellum, the Cerebellar Model Arithmatic Computer (CMAC), was introduced by Albus [2]. A key element of the CMAC model is that each GC is viewed as a hard-wired feature detector and recoder, responding to a specific pattern of sensory signals arriving on different MFs. The GCs together form a basis for later learning by the PC. PCs are treated as simple linear units, which compute a linear weighted sum of their PF inputs. Training in CMAC is performed with a version of the delta rule, using training signals from CFs. Although some details of the CMAC model have been superseded by models that more accurately reflect known neurochemistry and behavior, it has remained popular in the robotic control community [6].

A notable model that combines realistic simulation of anatomy with unrealistic simulation of learning is described by Tyrrell and Willshaw [7]. Their comprehensive simulation of cerebellar structure follows Marr's 1969 theory faithfully, using binary weights and a one-shot associative learning rule. Although they use their full-scale model to investigate theoretical representation and learning issues, it is in the context of Marr's weights and learning methods, and is inconsistent with more recent results.

A factor which has not been well examined by existing models is the question of scalability; the extent to which the learning systems proposed for the cerebellum remain useful when presented with large-scale problems, with realistic quantities and complexities of inputs and outputs. Scalability questions and their theoretical and experimental answers may be necessary to be fully confident that a particular learning system is appropriate for modeling the cerebellum.

3 On-line Learning

The Machine Learning and Computational Learning Theory communities have looked at learning from several points of view, including the well-known Probably Approximately Correct (PAC) framework. In PAC learning, the goal is to characterize the number of examples needed to learn a concept within specified tolerances, assuming the examples are selected randomly and are supplied to the learner all at once. In the contrasting on-line supervised learning framework, it is assumed that the learner can eventually learn a concept accurately, but the number of training errors must be minimized. At each step, the learner receives an unlabeled example, classifies it based on the current hypothesis, and then receives the correct classification. The total number of misclassifications during training is the error rate to be minimized.

When analyzing the mistake bounds of many concept-learning algorithms, it was shown that the number of inputs, including irrelevant inputs, strongly affected the mistake bounds. Specifically, when learning small disjunctions, the Perceptron rule makes a number of errors that grows linearly with the number of irrelevant inputs, while algorithms that make very rapid weight changes, such as Winnow, make numbers of errors that grow only logarithmically with the number of irrelevant inputs [8].

3.1 Algorithms

Linear separators categorize their input vector, $\mathbf{x}$, by multiplying it by a weight vector, as $\mathbf{x} \cdot \mathbf{w}$, and outputting 1 if that value is greater than a specified threshold. The familiar algorithms for Perceptron training learn linear separators by doing gradient descent in weight space, as in the delta rule: $w_i = w_i - \eta x_i \delta$. The Winnow algorithm and its variants also learn linear separators, but do so by multiplicative, instead of additive, updates to the weights: $w_i = w_i e^{-\eta x_i \delta}$. (The exponential function is only necessary to prevent multiplying by 0 or a negative number. There are variants of the algorithm that avoid the exponential by multiplying or dividing by a constant.)

When learning monotone (negation-free) disjunctions of k literals, Winnow makes $O(k \log N)$ mistakes, where N is the width of the input vector. In contrast, the delta rule may make in N a nearly linear number of mistakes, perhaps as many as $(N - k + 1)/2$ [9]. If N, the number of attributes, is much larger than k, the number that are relevant to the concept being learned, then Winnow performs much better, while if k is near N, the delta rule may learn faster. Note that there is no claim of representational power; both algorithms can learn to represent the same concepts (those that are linearly separable). The only issue is the speed at which this learning can take place.

Intuitively, Winnow works by rapidly shrinking weights on inputs that are active during false positives, and increasing them on inputs that are active during false negatives. Using multiplicative instead of additive updates means that fewer changes (and thus examples) are needed to reach appropriate weight levels.

A rather different algorithm with similar goals is the IDBD (Incremental Delta-Bar-Delta) algorithm [10], an elaboration of Jacobs' Delta-Bar-Delta algorithm [11]. In IDBD, the weights are modified additively, but the learning rate varies on a per-weight basis, effectively performing gradient descent in learning-rate-space. Essentially, if the weight changes are consistently in the same direction, then the learning rate is increased, since this input's weight appears not to have converged (it is either high or low). If the weight changes appear random, then the learning rate is decreased, as this weight appears to have converged (that is, the weight is oscillating around the correct weight). In both IDBD and Winnow, weights that are not relevant stay near 0, and weights that are relevant are helped to quickly attain their appropriate value. The update rules, somewhat more complex than Winnow or Perceptron, are as follows:

$$w_i = w_i - \eta_i x_i \delta \tag{1}$$

$$\eta_i = e^{\beta_i} \tag{2}$$

$$\beta_i = \beta_i + \theta x_i \delta_i h_i \tag{3}$$

$$h_i = h_i[1 - \eta_i x_i^2]^+ + \eta_i x_i \delta \tag{4}$$

Note the per-input learning rate, η_i, in equation 1. Also, observe that β is updated using the delta rule, using θ as a learning rate, with the addition of h_i, which represents the recent summed history of the weight changes.

3.2 Relevance to Cerebellum

With this background, our argument can be stated more precisely: *If the cerebellum is using some procedure at the granule cells to generate a sparse expansion of the cerebellar input, and is using a supervised linear learning algorithm at the parallel fiber-Purkinje cell synapses, and if the target concepts depend on only a small subset of the expanded inputs, so that the vast majority of the parallel fibers carry irrelevant information, learning in the cerebellum is likely to resemble a feature-efficient algorithm like Winnow or IDBD, rather than the delta rule.*

This general approach was first suggested by Schweighofer and Arbib (S&A) [12], who proposed that an IDBD-derived algorithm, CARL (Cerebellar Adaptive Rate Learning), could be used to model cerebellar learning, and that its attribute-efficient properties could be important. S&A do not, however, discuss the motivation for generating huge numbers of combinations of the input at the GC layer, and the target concepts in their model are simple disjunctions. Certainly, if it were sufficient for the cerebellum to learn successfully given the raw input represented in the MFs, without the need for the wide, conjunctive representation in the GC layer, then evolutionary processes would likely not have been so wasteful. We suggest that one likely motivation is that the cerebellum must learn concepts that are not linearly separable (such as exclusive-or, parity, or arbitrary boolean expressions), which could not be represented only by the weighted synapses of a single PC. Another possibility is that the

PFs may be involved in temporal processing [13], and that temporal variation might be one of the dimensions of the expansion. We suggest, then, that the target concepts learned by PCs are more complex than could be learned by a linear separator, and that the expansion onto GCs is a basis of non-linear features, from which more-complex concepts can be learned [14, 2, 15].

4 Experiments

The contents of the MF and CF inputs to the cerebellum are a matter of current debate, and the cerebellum's outputs are only poorly understood. Therefore, we believe that it is premature to attempt to investigate computational issues in the cerebellum by modeling the function of the cerebellum in detail. Instead, we present a proof-of-concept experiment, making no claim to accurately portray the cerebellum, its inputs, or its outputs. The architecture of our model is inspired by and analogous to the architecture of the cerebellum, but is greatly simplified so that the computational questions can be examined as precisely as possible. We believe that the concepts learned by the cerebellum for motor and other tasks are relatively complex, and our simulations are therefore designed to accurately reflect the level of complexity in the biological system.

With this in mind, the computational question we investigate is: *When learning t-literal k-DNF*[1] *concepts from a relatively small subset (n) of the possible random m-conjunctions of b boolean inputs, do irrelevant-attribute-efficient algorithms learn faster?* That is, when learning fairly complex boolean expressions (not merely linearly-separable disjunctions, as in S&A) given a sparse expanded basis similar to that generated by the GCs in the cerebellum, how does the choice of update rule affect the number of mistakes made during learning? The n random m-conjunctions of b boolean inputs define the basis, where b is the width of the input, and m-conjunctions are simply conjunctions of size m.

There are two common ways of learning a non-linearly-separable concept with simple Perceptrons. One is to chain linear threshold units into a multiple layer network, and train them with back-propagation or a similar algorithm. The other is to generate the first layer of a two-layer network automatically, and learn the second layer's weights using a simpler (and more biologically plausible) rule. In a brute-force approach, the first layer might consist of all possible combinations of the inputs, allowing the output layer to then learn a disjunction over this basis layer. Here we allow the learner to represent arbitrary linear separations instead of the less-expressive class of disjunctions, and we generate only a small subset of the polynomially-many possible combinations of input, in a manner similar to the MF-GC system in the cerebellum.

The learning system we simulate is quite simple. Let I be a boolean input vector of size b [2]. I^E is the expanded vector, generated by selecting (without replacement) m random elements of I, and calculating their logical conjunction. I^E is of width n. W is the weight vector, also of width n. O is the output, generated by a linear threshold: $O(I^E) = h(I^E \cdot W)$, where $h(x)$ is 1 if $x > 0$. The training signal T is 1 if the target concept is 1 for the current input, and 0 otherwise.

For this experiment, $b = 10$, $n = 1000$, and $m = 4$ [3]. Target concepts were 4-DNF expressions with 4, 8, or 10 literals. We compared IDBD, the Perceptron delta rule, and Winnow, using learning rates of 0.1, 0.01, and 2, respectively. For IDBD and Winnow, there was a fixed bias weight of $\frac{n}{2}$, while the delta rule version used a trainable bias weight with a fixed input of -1. 5000 examples were given, each expanded into a 1000-element basis (containing less than 25% of possible conjunctions), trained, and the resulting errors were counted.

Figure 2 shows the results for ten averaged runs of each algorithm. IDBD and Winnow performed similarly, while Perceptron (delta rule) converged poorly. The baseline, to always respond "false," makes around 34% (1700) mistakes for this problem.

The results suggest several things. First, a concept as complex as a 10-literal 4-DNF may be represented by a combination of a linear separator and a sufficiently wide but still sparse basis. This is clear from the nearly-flat mistakes line in the graph (for some individual trials, the line was actually flat). Second, the Perceptron algorithm has difficulty learning these concepts, as il-

[1] A t-literal k-DNF expression is a disjunction of conjunctions, where each conjunction is of length k (or less), and the total number of literals is t (eg. $(x_1x_2x_3x_4)\vee(x_5x_7x_{12}x_{54})\vee(x_{56}x_{312})$ is a 10-literal 4-DNF).

[2] To avoid creating negations of inputs, and allow learning over just monotone functions, we actually use $2b$ inputs, where the second half are negations of the first half. We do this because Winnow is only able to represent monotone functions, as its weights are always positive.

[3] Although $m = 4$, each generated conjunction actually had a 10% chance each of being of size 1, 2, or 3, and a 70% of being of size 4. This was done to allow the learner to construct missing terms present in the concept out of the random terms in the basis. For example, given $ABCE$ and $D\bar{E}$, $ABCD$ can be constructed.

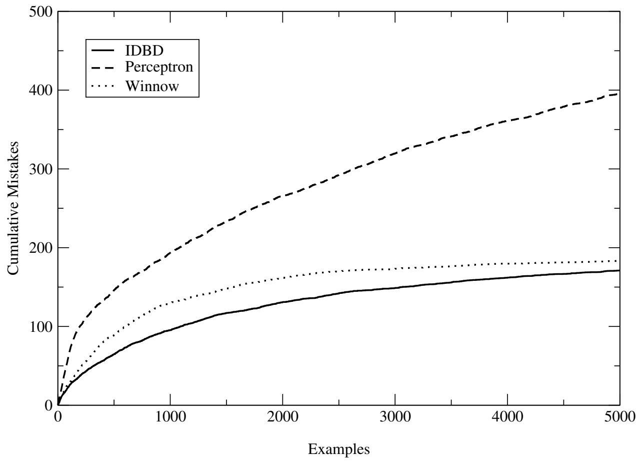

Figure 2: Results, comparing algorithms on 10-literal k-DNF over a size-1000 4-conjunct basis.

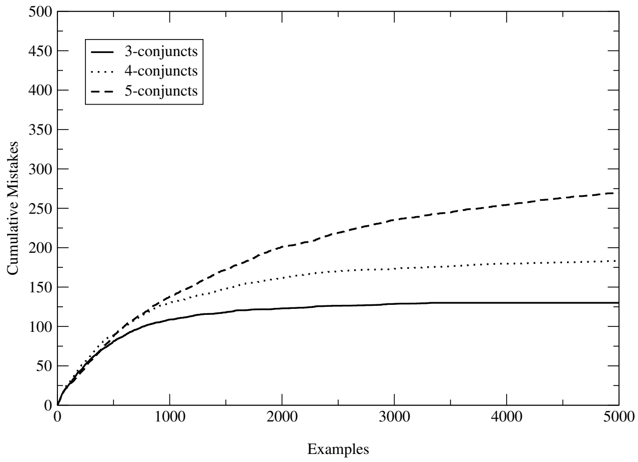

Figure 3: Effect of varying size of basis conjunctions for Winnow.

lustrated by the moderate positive slope in cumulative mistakes. Third, IDBD and Winnow, despite their very different learning rules, make similar numbers of mistakes.

If the same experiment is done with a trivial 1-term 4-DNF (ie., a 4-conjunction), the differences become even more stark. IDBD and Winnow converge very rapidly, making around 20 mistakes, while Perceptron converges imperfectly, making nearly 200 mistakes. Clearly, the theoretical sensitivity of the delta rule algorithm to irrelevant inputs is reflected in this experiment.

One potential issue is that the size of the terms in the concept were the same as the size of the terms in the basis conjunctions (four). This might be a key advantage, or disadvantage. To investigate this, the same 10-literal 4-DNF concepts were learned, but the the random conjunctions were experimentally varied to be of size 3, 4, or 5. For Perceptron and IDBD, the results closely resembled the curves shown in Figure 2. However, for Winnow, the differences were clearly significant, as shown in Figure 3. Is is not clear to us why Winnow should be so sensitive to this variation, although these results imply that the inductive biases of IDBD and Winnow are distinctive.

5 Discussion

We would like to comment on how Winnow and IDBD compare as models for the learning in the cerebellum. Our results show that both are efficient given a large number of irrelevant input attributes. However, their methods and details are very different. S&A [12] note the parallels between IDBD/CARL and the exponentiated gradient (EG) learning rule [16], the regression case of Winnow, but reject it as an alternative. They say that EG "might be difficult to implement in a biologically plausible manner because of the use of the exponential function at each synapse and the need for equal distribution of a global quantity that depends on the sum of the weight changes over the cell." (p. 426) To this, we point out that versions of Winnow using extremely simple learning rules, such as just doubling/halving weights, with no exponential or normalization, have proved effective in use [17]. We disagree that a biologically plausible version of Winnow or EG is impossible. As to which feature-efficient algorithm should be used for modeling cerebellar function, we suggest that continued biological investigation of learning in the cerebellum may be the best approach to answering that question. We would also like to note that other feature-efficient algorithms exist, such as the new p-norm class of learners [18], and that although their biological plausibility is questionable, they may provide insights into the nature of learning with large, relevancy-sparse basis vectors.

The main points of this paper are as follows. First, when modeling the cerebellum, and the brain in general, it is important to investigate how postulated learning algorithms scale up. Second, in the specific case of learning in the cerebellum, the commonly-used Perceptron/delta rules are substantially less efficient than alternative rules, given the expanded input space generated by the GCs. Third, when learning complex concepts, Winnow and IDBD represent a class of attribute-efficient linear-separators that are effective when combined with a sparse set of conjunctive basis features. And fourth, although Winnow and IDBD have different properties, we see no reason yet to exclude either as a possible abstraction of the actual cerebellar learning processes.

Acknowledgments. Thank-you's to Tom Anastasio, Gary Dell, Dan Roth and the Artificial Neural Networks and Computational Brain Theory group for helpful discussion of these ideas. This work was supported by NSF grant SBR 98-73450 and NIH grant DC-00191, and the Research Board of the University of Illinois and NSF Grant BAC 92-16562.

References

[1] David Marr. A theory of cerebellar cortex. *Journal of Physiology*, 202:437–470, 1969.

[2] J. S. Albus. The theory of cerebellar function. *Mathematical Bioscience*, 10:25–61, 1971.

[3] James C. Houk, Jay T. Buckingham, and Andrew G. Barto. Models of the cerebellum and motor learning. *Behavioral and Brain Sciences*, 19:368–383, 1996.

[4] J. L. Krichmar, G. A. Ascoli, L. Hunter, and J. L. Olds. A mode lof cerebellar saccadic motor learning using qualitative reasoning. *Lecture Notes in Computer Science, Artificial and Natural Neural Networks*, 1240:134–145, 1997.

[5] James C. Houk and Andres G. Barto. Distributed sensorimotor learning. In G. E. Stelmach and J. Requin, editors, *Tutorials in Motor Behavior II*, pages 71–100. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, 1992.

[6] Young H. Kim and Frank L. Lewis. Optimal design of CMAC neural-network controller for robot manipulators. *IEEE Transactions on Systems, Man, and Cybernetics Part C: Applications and Reviews*, 30(1), 2000.

[7] Toby Tyrrell and David Willshaw. Cerebellar cortex: its simulation and the relevance of marr's theory. *Philosophical Transactions of the Royal Society of London, B*, 336:239–257, 1992.

[8] Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[9] J. Kivinen, M. K. Warmuth, and P. Auer. The perceptron algorithm versus winnow: linear versus logarithmic mistake bounds when few input variables are relevant. *Artificial Intelligence*, 97:325–343, 1997.

[10] Richard S. Sutton. Adapting bias by gradient descent: An incremental version of delta-bar-delta. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 171–176. MIT Press, 1992.

[11] R. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988.

[12] Nicolas Schweighofer and Michael A. Arbib. A model of cerebellar metaplasticity. *Learning and Memory*, 4:421–428, 1998.

[13] Valentino Braitenberg, Deflek Heck, and Fehad Sultan. The detection and generation of sequences as a key to cerebellar function: Experiments and theory. *Behavioral and Brain Sciences*, 20:229–277, 1997.

[14] T. M. Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-14:326–334, 1965.

[15] J. S. Albus. A new approach to manipulator control: The cerebellar model articulation controller (CMAC). *Trans. ASME J. Dynamic Systems, Measurement, and Control*, 97:220–227, 1975.

[16] Jyrki Kivinen and Manfred K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 10 January 1997.

[17] Andrew R. Golding and Dan Roth. A Winnow based approach to context-sensitive spelling correction. *Machine Learning*, 34:107–130, 1999.

[18] Claudio Gentile and Nick Littlestone. The robustness of the p-norm algorithms. In *COLT 1999*, 1999.