WASHINGTON UNIVERSITY
SEVER INSTITUTE OF TECHNOLOGY

ABSTRACT

SUPERVISED COMPETITIVE LEARNING:

A TECHNOLOGY FOR PEN-BASED ADAPTATION IN REAL TIME

by Thomas H. Fuller, Jr.

ADVISOR: Professor Takayuki Dan Kimura

December, 1994
Saint Louis, Missouri

The advent of affordable, pen-based computers promises wide application in educational and home settings. In such settings, systems will be regularly employed by a few users (children or students), and occasionally by other users (teachers or parents). The systems must adapt to the writing and gestures of regular users but not lose prior recognition ability. Furthermore, this adaptation must occur in real time not to frustrate or confuse the user, and not to interfere with the task at hand. It must also provide a reliable measure of the likelihood of correct recognition.

Supervised Competitive Learning is our technology for the recognition of handwritten symbols. We demonstrate that it satisfies the following requirements:

1. *Pen-based technology*: digitizing display tablet with pen.
2. *Low cost*: PC-level processor with about 50 MIPS.
3. *Wide range of subjects*: varying by age, nationality, writing style.
4. *Wide range of symbol sets*: numerals, alphabetic characters, gestures.
5. *Usage*: adaptation to regular users; persistent response to occasional users.
6. *On-line recognition*: both response and adaptation in real time.
7. *Self-criticism*: reliable measure of likelihood of correct response.
8. *Context-free classification*: symbol by symbol recognition.

SCL successfully recognizes handwritten characters from writers on which it has trained (digits, lowercase, uppercase, and others) at least as well as known current systems (96.5% - 99.2%, depending on character sets). It adapts to its user in real time with a 50 MIPS processor without loss of response to occasional users. Finally, its estimates of its correctness are strongly correlated with actual likelihood of correctness.

WASHINGTON UNIVERSITY

SEVER INSTITUTE OF TECHNOLOGY

SUPERVISED COMPETITIVE LEARNING:

A TECHNOLOGY FOR PEN-BASED ADAPTATION IN REAL TIME

by

Thomas H. Fuller, Jr.

Prepared under the direction of Professor Takayuki Dan Kimura

---

A dissertation presented to the Sever Institute of
Washington University in partial fulfillment
of the requirements for the degree of

DOCTOR OF SCIENCE

December, 1994

Saint Louis, Missouri

copyright by

Thomas H. Fuller, Jr.

December, 1994

Table of Contents

Page

List of Figures ..... **ix**

List of Tables ..... **xi**

Chapter

1. Introduction **1**

Integrating Adaptation with an Interface Model *3*

Modeling Human-Computer Interfaces 3

The Need for Interface Models 4

Adaptable Pen-Based Systems *5*

Why the Pen? 5

One Critical Barrier — Recognizing Handwriting 6

The Role of Gestures in a Pen-Based User Interface 8

Why Adapt? 9

The Learning System *12*

Background: Learning Systems 12

Mission of the Learning System 14

2. Definition of the Problem 15

Representative Problem Environment — Learning Mathematics 15

Formulating the Problem 16

General Problem Abstraction 16

Problem Specification 18

Justification of the Specification 19

Completing the Paradigm 22

3. Review of Related Research 23

Research Related to Adaptation 23

Learning Theory and Interfaces for the Learning Computer 24

Gestures and Pen-Based User Interfaces 26

Review of Learning System Architectures Related to SCL 27

Carpenter and Grossberg: ART 29

Other Adaptive Learning Models 30

Review of Other Learning System Elements 34

Feature Abstraction 34

Preprocessing and Postprocessing 37

Some Proven Theoretical Results 38

Review of Handwriting Recognition 40

4. Solution Methodology 47

SCL Model 47

Overview of the Model 48

Interface 49

Preprocessor 51

Prototype Units 59

Selector 61

Training Scheduler 64

Implementation 68

Data Elements 68

Interface 75

Preprocessor 78

Prototype Units 87

Selector 87

Training Scheduler 87

Experimental Methods 88

Scribble Database 88

Testing on Sample Data 92

Cross-Validation 93

Observational Methods 94

5. Results 95

Organization and Reporting Conventions 95

Organization of the Chapter 95

Reporting Timing Results 97
Experiments that Guided the Design and Evolution of SCL *97*

Backpropagation versus Fuzzy Logic 98

System Architecture versus Complexity of Individual Units 100
Experiments Testing Adaptation to Single Users *109*

Training a Recognizer from Scratch 109

Retraining an Existing Recognizer 112

Dependence of Recognizers on Digitizers 115
Other Specific Performance Results *119*

Behavior of Prototypes in Gathering Exemplars 119

Confidence 121

Segmenting a Stream of Strokes into Individual Scribbles 124

Complexity Analysis and Performance 126

Variations in Parameters 131

Comparison of Feature Sets 134

Comparing Smoothing Techniques and Ratios 138
Tests with Children and Instructors *140*

Testing Kumon Students 141

Testing Kumon Instructors 143
Results from Extensions to SCL *145*

SCL and Gestures 145

Spoke-Based Features 146

Hierarchies of SCL Recognizers 146

Segmenting Digits 147

Segmenting and Recognizing Fractions 147
SCL on Digital Signal Processors 147
Other Platforms 148
6. Evaluation 149
Evaluation of General Experiments.....149
Backpropagation versus Fuzzy Logic 149
Complexity, Learning System Architecture and Performance.....150
Digits, Letters, and Combined Recognizers.....153
Comparison of Feature Sets 154
Comparison of Filters 154
Evaluation of Application Tests.....154
Children's Tests 155
Instructors' Tests 159
Evaluation Relative to the Problem Statement.....163
Formal Specification Elements 163
General Problem Abstraction.....166
Comparison of SCL to Other Models.....169
Evaluation Relative to Related Learning Architectures 170
Recognition Rates Compared to Other Systems.....174
Relation to Other Research.....175
Relation of this Work to Pursuant Research 175
Environment to Support Pen-Based Research 175
Learning Systems and Human Learners.....176

Pen-Based Adaptable Human-Computer Interfaces and Teaching 176

New Windows on How Children Learn.....177

7. Conclusions.....179

General Conclusions.....179

Specific Performance.....179

Confidence 181

Speed.....181

Extensions to SCL.....182

Unsatisfied Application Goals.....182

Contributions of this Research.....183

Summary.....183

8. Future Directions 185

Further Research in Pen-Based Adaptable Human-Computer Interfaces.....185

Further SCL Research.....185

Appendix

A. Pen-Based Adaptable Human-Computer Interface Model.....190

The Hierarchical Model.....190

Related Research.....192

Aristotle's Model of HCI.....192

Learning Theory and Interfaces for the Learning Computer.....194

Elaborating the Layers of PAHCI.....195

The PAHCI Model of the Adaptation Environment.....195

Relations between Layers.....200

Examples of Adaptation at the Different Levels.....202

Examples of Adaptation in Three Application Settings.....203

A Primitive Example Traced through the Levels.....205

Potential Advantages for the Hierarchical Design Model.....209

Software Design and the PAHCI Model.....209

Better Management of Candidate Representations.....210

PAHCI — A New Window on How Children Learn.....210

Better Interfaces Call for Better Interface Models.....211

Acknowledgments.....213

Bibliography.....215

Vita.....237

Figures

Figure **Page**

1. Classifiers **13**

2. Boundary between Two Nestor Category Regions **31**

3. The Five Modules of SCL **48**

4. Velocity Features **57**

5. The Numeral *5* in the Feature Grid **58**

6. Selector Algorithm **62**

7. Raw Scribble Data for 5 ..... **69**

8. SCL Test in PenPoint **78**

9. Magnified Scribble **80**

10. Simple Smoothing: e = 0, 1/2, 3/4, 7/8 ..... **81**

11. Bi-directional Smoothing: e = 0, 1/2, 3/4, 7/8 ..... **81**

12. Symmetric Smoothing: e = 0, 1/2, 3/4, 7/8 ..... **82**

13. Shear Transformation **82**

14. Velocity and Acceleration Viewed in HCR **86**

15. Virtual Keyboard to Correct a Misrecognition **89**

16. An English Word Composed of Isolated Characters **90**

17. Digit Recognizer Training with Three Shuffles **107**

18. Letter Recognizer Training with Two Shuffles **108**

19. Scribble Sequence for Training from Scratch **110**

20. Incremental On-Line Training **112**

21. More Incremental Training **113**

22. Exemplars Gathered by Digit Prototypes **120**

23. Exemplars Gathered by Lowercase Prototypes.....121

24. Section of Automatic Training Log.....134

25. Uppercase Writing of Instructor T6.....161

26. Uppercase Writing of Instructor T5.....161

27. Lowercase Writing of Instructor T5.....162

28. Adaptation by the User.....167

Tables

Table **Page**

1. Scribble Collection 92

2. Training and Testing with Lombart Data: Backpropagation 99

3. Training and Testing with Lombart Data: Fuzzy Logic 100

4. Comparison of Backpropagation and Fuzzy Logic 100

5. Training and Testing with Wacom Data 101

6. Digit Recognizer Performance (3,200 Digits) 103

7. Lowercase Letter Recognizer Performance (3,900 letters) 104

8. Mixed Recognizer Performance (8,660 Scribbles) 105

9. Performance with 8, 4, 2, or 0 Hidden Units 109

10. Training Times (seconds) 111

11. Recognition Stability after Incremental Training 115

12. Recognizer Dependence on Digitizer 116

13. Incremental Training with Mixed Coordinate Systems 118

14. Confidence Factor versus Correctness — Lowercase 122

15. Confidence Factor versus Correctness — Mixed 123

16. Profile of SCL Functions 128

17. Preprocessor Time 130

18. Backpropagations Necessary to "Fully" Train 131

19. Varying Two Parameters for Lowercase 132

20. Varying Three Parameters for 1500 Digits 133

21. Average Prototype Weights Associated with Feature Sets 136

22. Comparison of Feature Sets: Lowercase 137

23. Comparison of Feature Sets: 3,200 Digits 138

24. Comparing Smoothing Techniques and Ratios 139

25. Number of Points in a Scribble 140

26. Children's Summary Results 142

27. Training and Recognition during Children's Testing 143

28. Recognition and Training Time during Instructors' Testing 144

29. Instructor Test: Bimodal Recognition 160

30. PAHCI Hierarchical Model 191

31. Aristotle's Elements of Drama 193

1. Introduction

For decades, computers have inched toward "user-friendliness." Still, few human beings find them friendly enough to apply to many areas of potential benefit. In a seminal collection of interface writings, *The Art of User Interface Design*, editors Laurel and Mountford observe [106: xiii]:

> When the concept [of human-computer interface as a proper discipline] first began to emerge, it was commonly understood as the hardware and software through which a human and computer could communicate. . . . The noun, *interface*, is taken to be a discrete and tangible thing that we can map, draw, design, implement, and attach to an existing bundle of functionality. One of the goals of this book is to explode that notion and replace it with one that can guide our work in the right direction. That direction, the goal of all our efforts, is to empower the user. . . . As computer technology has become available to more and more people in a greater variety of devices and contexts, the need for accessibility, ease of use, and user engagement has grown more and more pronounced.

The gap between technological potential and general accessibility continues a disappointing, but seemingly inevitable, historical pattern. The increasing processing power of computers causes little change in "friendliness" — accessibility and user empowerment — until the paradigm of the user's relation to the processes shifts.

Such a shift occurred with the advent of the mouse and the graphical user-interface (GUI) first introduced in the late 60's, but not widely realized until the late 80's [36, 83]. Significantly, neither the mouse and display scheme nor the object-oriented GUI were *individually* sufficient to cause the shift — *both* were necessary.

This decade is bringing half of a new paradigm to the fore: the pen-based computer. Pen-based computers use a stylus (*pen*) as the input device. Its movements on the

computer's surface are tracked magnetically or electrostatically and are marked by a trail of pixels, *ink*, on the display surface (usually liquid crystal display). The pen may input characters, commands (through gestures), and selections (taps). If it is taken as a mere change of medium (as was the mouse with graphical display), it will have limited impact on the feel of the interface. What must accompany the pen to radically transform the user interface? We feel that some form of *pen-based adaptable human-computer interface* must emerge as the *rest* of the paradigm. Such a plastic interface must gain knowledge of its user's behaviors and apply this knowledge to interpret user inputs and to customize outputs.

The pen's very nature expands the user's expressivity by creating a *continuous input stream* from user to system — a continuous input medium. (For example, sound and video are continuous input and output media.) Though the mouse is potentially continuous — when dragging or drawing — it is used generally as a pointing and selecting device. In any medium, adaptation to the user increases the economy of effort by eliminating redundant elements of the user's input. Thus the goal of a pen-based adaptable user interface may be summarized as empowering the user in two specific dimensions: expressivity and efficiency.

Furthermore, the inherent flexibility of the pen naturally leads the user to expect adaptation. An input event from a keyboard is well-defined and discrete. A mouse click on a screen-depicted object is similarly well-defined. The twisting trail of pen input is different with every user — indeed with every stroke. This variability is compounded by the fact that a single pen stroke may simultaneously select an object, issue a command, and provide parameters — further demonstrating the efficiency of pen-based input.

1.1 Integrating Adaptation with an Interface Model

We explore an enabling technology to integrate adaptation with a coherent model of human-computer interfaces. By way of defining the scope of the problem, we define a simple framework for pen-based adaptable interfaces. We then prescribe a machine learning system, Supervised Competitive Learning (SCL), as a model for such a technology and the basis for a representative implementation. Adaptability through machine learning is an indispensable element of a comprehensive and practical model of pen-based interaction. It completes the pen paradigm to the end of empowering the user.

1.1.1 Modeling Human-Computer Interfaces

The communications between a computer and a user (or between any two agents) depend on models of self and other, and the establishment of a shared protocol for the modification of these models. This field has one foot in classical education and cognitive science theory (for example Piaget [75, 76, 130, 131], Bruner [13, 14, 15, 16, 18, 17], and Greeno [58]), and the other foot in information and communication theory (such as Papert [127, 128, 129]). Alan Kay, for example, argues that the study of interfaces cannot be divorced from the study of how people learn [83, 84].

Piaget, Bruner and others model human cognition in a hierarchy from trivial sensory input to deeply symbolic understanding. We observe that a user's interactions with a computer occupy comparable levels of complexity and commitment of cognitive resources. Some interaction events are as trivial as flipping the page. Others are as complex as composing a sonata. At the simplest level the computer models the pen user as an incoming sequence of *points* — xy-coordinate pairs. All the points written during one interval of pen contact with the writing surface are grouped into a *stroke*. A set of one or more strokes that convey a command, gesture, digit, letter, or symbol (such as a musical note) is

defined (after GO [55]) as a *scribble*. This hierarchy (not completed here) leads eventually to the sonata and beyond.

Adaptation at the lowest level involves the transformation of points and is achieved primarily in specialized hardware or firmware. For example, the device driver for the digitizing tablet might adjust the sampling rate. Adaptation at the highest layer relates the presumed user goals to the (adaptable) operating system. Our particular research is aimed at adaptations in the lower to middle layers of the hierarchy, the layers that receive the user's strokes and assemble them into characters and gestures. This adaptation relates to the user's psychological facilities respectively denominated as *enactive* and *iconic* by Bruner [14]. Thus adaptations at this layer must correspond to some combination of enactive (kinesthetic) and iconic (visual) features abstracted from the incoming data stream.

In Appendix A, we elaborate this hierarchical model by borrowing the scheme from the International Standards Organization Open Systems Interconnection (ISO OSI) reference model [35].

1.1.2 The Need for Interface Models

Chen and Leahy [27], among many others, voice the need for new interface devices including voice and specialized hand devices (pens of course, but also devices like thumb wheels) that take advantage of *muscle memory* — of the kinesthetic sense that accompanies the act of writing. After reviewing extant models such as the mouse-based GUI, they point to a central problem: “What is lacking, however, is a coherent, high-level model for application-input device interaction.”

The need for such a model (or more likely, several models relevant to different interface missions and modalities) is echoed in the effort to create a textbook of

human-computer interaction by Dix et al. [36]. In the foreword, Stuart K. Card of Xerox PARC dates the discipline of human-computer interaction from the 1982 Gaithersburg Conference on Human Factors in Computing. He feels that, "The development of the area has reached the point where the first steps are being taken to codify the field. This text-book is one of these first steps and part of that process." Regrettably though, this first codification gives no more than a passing glance to the emerging pen paradigm — under-scoring the need for models *and* technologies of adaptation applicable to the pen. If the pen is to empower the user through enlarged expressivity and increased efficiency, it must not be hampered by models indifferent or alien to its modus. As explained above, a viable model must take into account the pen-specific issues of kinesthetic sense and iconic repre-sentation. The purpose of our work is to propose a model of adaptation to support this development and then to instantiate this model.

1.2 Adaptable Pen-Based Systems

An adaptable pen-based interface needs to change as the user and the uses change. It must be tractable, responsive, and self-adjusting. It must constantly cycle through sta-bility, change, reinforcement, and re-stabilization. Its channels may eventually extend to expression (input and output) through audio, keyboard, scanning, and other media as well as the pen. It might embrace both discrete events (keystroke or message) and continuous events (pen tracing or video camera) of many kinds, but in our research we concentrate on the pen, and principally, we focus on SCL as a mechanism to manage pen-oriented adap-tation.

1.2.1 Why the Pen?

The concept of silicon paper — the use of pen-based computers to instantiate a pencil and paper paradigm — was introduced by Kimura in 1990 [90, 91]. Since then the

fortunes of pen technology have moved in fits and starts. Johnson et al. describe how readily users adapt to a paper-like interface [78]. Kempf [87] notes that fast typists can easily outperform fast writers (70 wpm to 30 wpm), but that they type 10 to 20 words before reaching that advantage. Besides, not all users are fast typists. For many people, and for many tasks that involve intermittent data input, the pen is faster than the keyboard. This is true for entering and changing data in spreadsheets, editing documents, creating presentations, and diagrams. Apte and Kimura [5] found that the pen was twice as fast as the mouse for creating diagrams and charts.

Another aspect of the pen's value stems from the particular needs of education. With the pen, the child is touching the face of the computer and not a detached appendage of it. She is using a motor skill deeply associated with the learning process [90]. Learning takes place as the pen moves. In contrast with keyboards and mice, learners comment that the pen feels more natural for "doing" math. This is reminiscent of Hall's [65] study of the critical role played by drawing — even doodling — to activate a successful strategy for the solution of algebraic word problems. As Newman observes, "The pen is the most common communications device in the world. . . . The pen is not a computer input device. It's a person output device." [124:62]

1.2.2 One Critical Barrier — Recognizing Handwriting

A stubborn shadow looms over the pen's promise. It was the central theme of the *Newsweek* article whose title bemoans the pen-based computer as "The Revolution that Wasn't" (Schwartz [146], but was also voiced in Allen [2], Baran [7], Baran et al. [8], Dix et al. [36], Ramirez [135]). Keller cited it in AT&T's downplay of EO [86] leading to its eventual closing [113]. This is the difficulty in recognizing handwritten input. Even Doonesbury cartoonist Garry Trudeau joked in this vein for a week about a thinly-disguised

Newton [168]. Schwartz and others opine that once this shadow is cleared, the pen *will* revolutionize our use of computers. Wagoner (editor of PenWorld's *Personal Electronic News*, and longtime pen stalwart) admits the disappointment of current recognizers and suggests that viable markets lie in applications that bypass recognition such as check lists and signature verification [172:4].

Moreover, a successful pen-based interface must not only recognize input, but must learn to *improve* its recognition of characters, digits, and gestures from specific, regular users. Since the pen is a continuous input device, the system must use sampling and time-out assumptions to transform the input into discrete events for recognition and action. From this perspective, Supervised Competitive Learning (or any other handwriting recognizer) operates as a statistical "transducer" — a real time mechanism to convert the continuous stream of data into discrete symbols.

In the late 1950's, scientists were reported to be twenty years away from harnessing fusion power to provide limitless electric power "too cheap to meter." In the 1990's, scientists are at least twenty years away from harnessing fusion power, and the electric companies have yet to reclaim their meters. Handwriting recognition seems to follow a similar cycle of bold claims and cold realism. Since Dimond's 1957 paper (Suen et al. [160]) and 1963 patent (Tappert et al. [163]), and the Sylvania Data Tablet in 1968 [166], researchers have studied the problem with many claims of success. Modest advances are published, percentages are bandied about, and handwriting recognition remains a stubborn obstacle to pen-based architecture. Attendees at Newton developer classes are warned about lending a PDA to another, less it not recognize its owner's writing when returned. This unduly imposes the *personal* on personal digital assistant. The *New York Times* article that announced AT&T's return "to the drawing board" on the EO cited the difficulties

of developing software to convert user's handwriting to machine intelligible form [86]. The replacement Newton released in March of 1994 features a tiny on-screen keyboard to alleviate the frustration of failed script [187].

1.2.3 The Role of Gestures in a Pen-Based User Interface

There is more to the pen paradigm than just writing characters instead of typing them. Gestures may prove even more important than handwriting in redefining the user's relation to the processes. Kurtenbach and Hulteen note [103] that humans use gestures to communicate more clearly, to maintain secrecy, or to add the dimension of feeling. "Imagine human communications without gestures — people saying good-bye without waving, giving directions without pointing, conducting a hundred musicians in a symphony by sending messages to each of their music stands. We sit down in front of our computers today and gestures stop." They define a gesture as a motion of the body that contains information. Thus, handwriting *per se* is not a gesture because the motion of the hand expresses nothing; it is only the resultant words that convey the information. The user could have typed the same words with no loss of semantic content.

They claim that gesture input provides two advantages. First, in certain interface tasks, it outperforms other input techniques; that is, a gesture, may be faster and/or more intuitive than selecting from a menu or clicking a button. It is possible to simultaneously select an object, send it a command, and include parameters for the command in a single gesture [142, 143]. Second, gesture input may provide the user with more functionality. Some tasks cannot be done without gesture input. The physicality of gesture input has a value in itself in such applications as computer music, sculpture, and simulation. Imagine a conductor trying to conduct a symphony by using a textual interface to specify tempo

changes and emphasis. This physicality is inseparable from the creativity, effectiveness, and satisfaction of the conductor. They write:

> In each of [such] tasks, gestures allow the many subtasks involved to be chunked or compounded into a single interaction or motion. This, coupled with the intuitive nature of gesture, is the reason why gestures as computer input are powerful and exciting. . . . Once a whole interface has been designed, implemented, and introduced, gestures will begin to compete effectively with other input techniques.

They conclude by observing that the difficulty of recognizing gestures has limited their use so far. Rubine [141, 143] explored the area of gesture utility and recognition in the effort to solve just this problem.

Buxton writes [20] "There is little dispute that the user interface is a bottleneck restricting the potential of today's computational and communications technologies. When we begin to look for solutions to this problem, however, consensus evaporates." Myers states that the user interface typically accounts for 70-80% of the product cost of modern applications [120]. Beyond esthetics and user empowerment, this gives a bottom-line incentive to software engineers to improve the efficiency and maintainability of interface models and technologies.

1.2.4 Why Adapt?

The recognition task is sufficiently onerous by itself. Why add the processing burden and potential errors inherent in adaptation? People expect adaptation from other people; and this remains our dominant interface model. If a single interface met every need of every user, adaptation would be superfluous. For example, if the only users of a computer (like an embedded system) were other computers, adaptation would be pointless — in fact, an obstacle to the interface. But pen-based computers are to be used by humans. The

very diversity of expectations, experience, visualizations, and motor skills in the user population demands adaptive systems to genuinely accommodate "the rest of us."

The designers of computer-assisted mathematics learning systems have tried to win over their young clientele for at least a quarter century. The successes have been modest and sparse (Papert [127, 128, 129], Turkle [169], Solomon [155], Kearsley [85], Cuban [31]). For any instructional approach to succeed, the student must in some measure be engaged in and take possession of the process. Adaptation to the regular user makes the system more attractive, and more likely to cultivate a response of engagement and possession.

IBM pen researcher Tetsu Fujisaki proposed (in a conversation) the following thought experiment between two hypothetical pen-based learning systems: The first recognizes the user's writing at the rate of 97% from the start but never improves. The second begins by recognizing the user's writing at the rate of 75%, but quickly improves and eventually reaches 95%. Which one will be better received by a learner? Which will evoke feelings of ownership and accomplishment? He suggests that the latter will. This hints at an aspect of user interface design (and learning theory) that seems largely to have slipped past both educational and computer science researchers.

Moreover, children are remarkably adaptive themselves. Our candidate application for SCL is the Kumon Machine — a system to embody in silicon the widespread Kumon supplemental mathematics learning method. The most likely Kumon Machine users are in elementary school (as are 75% of the million-plus Kumon students [47]). They are daily growing in motor skills, cognitive abilities, and interests. A static system is less likely to succeed with them during this process. A system that can offer new accommodations to their developing motor skills (at the lower layers of the interface model) and new methods

of getting the job done (at higher layers) is more likely to cultivate the sense of ownership vital to success in the educational enterprise.

The intent of the Kumon Machine is mathematical learning within the Kumon method (Shiba [150], Kumon [98], [99], [100], Finn [38], Murr [119], Fuller [47], and Walters [173]). Thus the system should maintain as low a profile as possible. It should stay out of the way of the encounter between learner and concepts. On the other hand, it should not let the learner flounder endlessly, nor should it require the user to repeat routine processes endlessly to accomplish the task at hand. The balance between the extremes of support and streamlining is not fixed across learners, across lessons for one learner, nor even across an extended session for a learner growing in confidence. It is very difficult to imagine a design that will succeed for so many situations other than through constant adaptation.

Suchman [159] distinguishes between the well-formed plans postulated by cognitive scientists to account for intelligible activity and what she calls "situated actions." She says that real-life interactions are a series of tentative responses to current situations, based on the model of the interlocutor. She also defines and illustrates the need for detection of rips in the fabric of communication, and subsequent repair. She writes as the principal of her five main points, " . . . mutual intelligibility is achieved on each occasion of interaction with reference to situation particulars, rather than being discharged once and for all by a stable body of shared meanings." But any non-adaptable system must depend on just such "a stable body of shared meanings." It cannot fathom the variations in context associated with the *situated* nature of general human interactions with machines. Situated action demands adaptable user interfaces.

The above justifications of adaptability share the psychological observation that people expect a communicative relationship to *grow*. They accept the difficulties that arise in the first communications with a newcomer, but anticipate that these will abate as shared protocols and contexts are established. They *don't* expect to start over with every subsequent conversation, and show little patience with interlocutors who cannot remember the protocols and contexts of former conversations. Moreover, there is a natural rhythm of this accommodation. A system that is sluggish in adaptation falls away from this rhythm, and imposes the burden of adaptation on the user. It fails to be transparent. Our research aims to bring some measure of such genuine "friendliness" to the learning computer. In particular, we seek to elaborate (through a specific instance) the pen-based adaptable interface.

1.3 The Learning System

We introduce a technology for managing adaptation in the lower to middle levels of the user interface, Supervised Competitive Learning (SCL). The particular problem environment that impelled our research is the need for a pen-based computer to teach children mathematics — a "learning computer" to help children learn. The principal application is the recognition of handwritten digits, letters, and gestures, which we now formally introduce.

1.3.1 Background: Learning Systems

Let $\mathcal{X}$ be the set of input patterns generated by some environment. Let $\mathcal{C}$ be a partition of these patterns. A *classifier* is an function $S:\mathcal{X}\rightarrow\mathcal{C}$ that maps a pattern X to its category C_X $(X \in \mathcal{X},\ C_X \in \mathcal{C})$. The set of categories is called the *target set* of the classifier (digits or lowercase letters for two examples). An *ideal classifier* S_{Ideal} perfectly realizes some desired mapping from patterns to categories. For a particular category C, a

single-category classifier S_C:$\mathcal{X} \rightarrow [0,1]$ is a classifier such that

$$S_C(X_1) \geq S_C(X_2) \Leftrightarrow P\ (\ S_{Ideal}(X_1) = C\ ) \geq P\ (\ S_{Ideal}(X_2) = C\ )$$

where *P* is the usual probability function. A single-category classifier for *C* gives the likelihood that a pattern belongs to *C*. Its target set need not be limited to the closed real-valued interval [0,1]. In practice, single-category classifiers also use target sets of (-1,1) and {0,1}. The former applies to *sigmoidal* functions, the latter to *discrete* functions.

A *learning system* S_L is a classifier with an internal state s_i which changes in response to a series of exemplar events (X_i, C_{x_i}) $1 \leq i \leq T$ where $S_{Ideal}(X_i) = C_{x_i}$ and *T* is the number of exemplars presented to S_L. Thus S_L is a sequence of functions $S_L(s_i)$:$\mathcal{X} \rightarrow \mathcal{C}$ with an updating function *f* which series approaches S_{Ideal}. That is

$$s_{i+1} = f\ (s_i, X_i, C_{x_i})$$

$$S_L(s_i)\ (X_i) = C_i$$

$$\lim_{i \rightarrow \infty} S_L(s_i) = S_{Ideal}$$

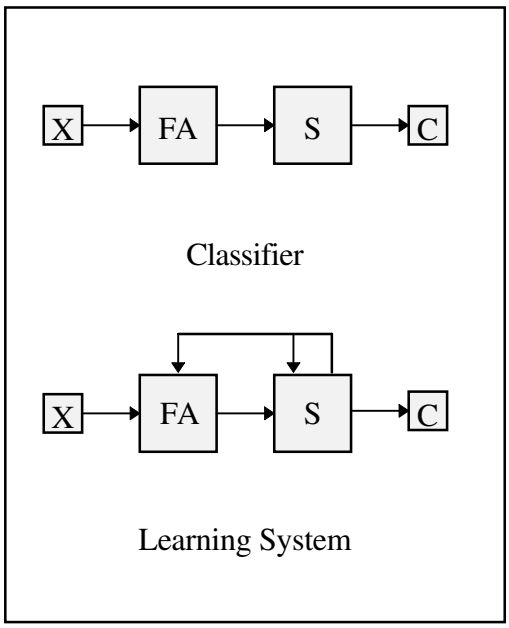

Figure 1. Classifiers

In practice a classifier is often realized by the model shown in Figure 1. The input is transformed into a more abstract representation as a vector of features by a *feature abstractor* (FA) and then this feature vector is classified. A learning system is depicted below. The back-directed arrow indicates the modification of the system state (of either the feature

abstractor or the classifier). A *handwriting recognizer* is a classifier (which may be also a learning system) whose patterns are handwritten scribbles (as defined before) and whose categories are gestures, digits, letters, or some related target set..

1.3.2 Mission of the Learning System

We can define an *interface* for a given computer system as a classifier S_F whose input pattern set $\mathcal{X}_F$ is all possible entries from input devices of the system and whose categories $\mathcal{C}_F$ are system responses. An *adaptable human-computer interface* is an interface whose classifier is a learning system.

As emphatically declared by the opening quotation, we must not let this useful formalism cloud the mission of *freeing the user*. The ideal classifier S_{Ideal} for a user is not known at first (if ever) and must co-evolve with that user. Thus a pen-based adaptable interface implies more than the replacement of mice with pens. It calls for a flexibility and naturalness that transfigures the "look and feel" of the computer, especially the feel. The computer must become a touchable, teachable partner in the building of a user interface that grows and emerges individually for each user.

In Chapter 2, we define and explain our specific problem and the problem environment that animates our research.

2. Definition of the Problem

The first chapter hinted at the vast field of adaptable human-computer interfaces. No single line of research encompasses its sweep. We narrow our focus to a representative problem environment. We refine this environment to a specific solution methodology for managing adaptation in the lower to middle tiers of the adaptation hierarchy. The following sections specify and justify the problem statement.

2.1 Representative Problem Environment — Learning Mathematics

The need for adaptable recognition is clearly far broader than any single application, but a natural application for this model is a handwriting recognition system for a pen computer — the *Kumon Machine* — to be used by Kumon mathematics students. Currently about a million students employ Kumon as a supplement to the mathematics taught in regular schools [47]. Though most users are of elementary school age, users vary from toddlers to adults. The distinctive feature of the Kumon approach is its emphasis on learning mathematics by *writing* mathematics. There are no multiple-choice tests and no calculator exercises. It builds math in "muscle memory" [47]. Input patterns are predominantly numbers, but alphanumeric characters are used by students when solving algebra problems, and are also used occasionally by instructors and parents for database access. Thus the system has to adapt in a complex environment that embraces different character sets and different handwriting styles — remaining in a continuous learning (training) mode. At any point, the user may enter a correction and the system should train itself until it recognizes the new pattern consistently.

2.2 Formulating the Problem

We first abstract three general problem elements from the above environment. Then we formulate the specific problem statement.

2.2.1 General Problem Abstraction

We recast the problem environment as three general problem areas discussed in the following sections.

2.2.1.1 ♦ Stability versus plasticity of adaptation.

- ♦ The two-body learning problem.
- ♦ Managing adaptation with confidence.

2.2.1.2 Stability Versus Plasticity of Adaptation

When an adaptive learning system is used to encode input patterns from an evolving environment, it suffers the stability-plasticity dilemma formulated by Grossberg [24, 59, 60, 61] for the competitive learning paradigm: How can a learning system remain plastic in response to significant events and yet remain stable in response to irrelevant or routine events? How can it maintain previous knowledge while continuing to gain new knowledge? Suppose that a system has been successfully trained to recognize the handwritten character *7* by a person who writes it consistently with two strokes (European style) as . Now, the same system is to be trained by another person who writes *7* with one stroke as . After adapting to the one-stroke , it may not be able to recognize the two-stroke as well as it did previously. This problem leads to Apple's warning about not lending a Newton to someone else less it forget the owner's writing.

2.2.1.3 The Two-Body Learning Problem

A computer and a user constitute a *two-body learning system*. Each forms an implicit or explicit model of the other — a model modified over time, hopefully toward some

unknown ideal. The output of one forms the input of the other. The cycle of modification is not guaranteed to converge to the ideal system. For example (from a conversation with Jayanti), if each of two opposing pedestrians in a hallway acts in a manner that mirrors the other, the system of two will block forever.

Similarly, the two inner models of the other in the case of computer and user may fail to converge to a useful basis for exchange. A child may write a *5* that is not recognized by the system. The system then trains on the misrecognized *5*, but the child changes her style of writing *5*'s. Another misrecognition occurs since the system is now expecting the newly-trained version of *5*, and the confusion continues. A graphic example of a child's adaptation (on a system that was not adapting) is shown in section 6.3.2.2 on page 167 of Chapter 6.

The two-body learning problem was mentioned by Rubine in his dissertation on gesture design and recognition [143]. Gibbs also mentions it briefly in his evaluation of commercial recognition systems [53]. An adapting system must adapt at a speed and with a predictability that does not perplex the user. The user and system must converge in their mutual adaptations and avoid chaotic oscillations. Our observations of writers on pen-based systems suggest that if either adapts at a rate significantly faster than the other, this oscillation may be avoided. For this reason, a successful adaptable system must complete each adaptation in about a second.

2.2.1.4 Managing Adaptation with Confidence

Some demands of the problem environment require us to manage the adaptability itself. For one example, an adaptable system must ensure that newly accepted scribble-category pairs remain unambiguous in their target sets. A user may write a *1* that is identical to her *7*'s (on which the system is already trained). The system should not permit itself

to be trained in a manner that forces ambiguity between the two categories. We would want it to reject the training request on the seven-like one.

A second example arises from children learning to write. We would naturally want to restrict adaptation that promotes an illegible substitute for the standard alphabet [47]. At the same time, we do not want to defeat or frustrate the child whose motor skills are still developing. This example and the one in the previous paragraph are inseparable from the need for credible confidence. Unlimited adaptation undermines the distinctions necessary to maintain confidence in the classifications. Alternatively, confidence can be used as a guidepost to set limits on adaptation. Any candidate solution to this problem must manage adaptation with a reliable measure of confidence.

2.2.2 Problem Specification

We seek to develop a technology for the recognition of handwritten symbols that satisfies the following requirements:

1. *Pen-based technology*: digitizing display tablet with pen.
2. *Low cost*: PC-level processor with 50 MIPS.
3. *Wide range of subjects*: varying by age, nationality, writing style.
4. *Wide range of symbol sets*: numerals, alphabetic characters, gestures.
5. *Usage*: adaptation to regular users; persistent response to occasional users.
6. *On-line recognition*: both response and adaptation in real time.
7. *Self-criticism*: reliable measure of likelihood of correct response.
8. *Context-free classification*: symbol by symbol recognition.

2.2.3 Justification of the Specification

If the solution system is to meet the demands of the representative application described above, it must meet each of the eight criteria for the reasons given in the following sections.

2.2.3.1 Pen-Based Technology

As explained in Chapter 1 and in our earlier article [47], a significant dimension of mathematics learning inheres in the *writing* of mathematics. The mouse or keyboard cannot expropriate this unique aspect of learning.

2.2.3.2 Low Cost

Prior attempts to negotiate the stability-plasticity dilemma are expensive in terms of specialized hardware or software complexity. Some of the systems described in the next chapter require (or are commonly realized in) specialized hardware; examples include Grossberg's ART [24, 59, 60] and the Nestor Adaptive Modules™ of Reilly et al. [28, 29, 136, 137]. Others require hours to converge even on powerful workstations (such as Laine's wavelet-based classifiers running on SPARC 10's [104, 105]). What is needed is a general-usage, learning system technology that can adapt (while meeting the other requirements of this arena) in real time. Given the educational and home use for such a technology, cost is a dominant issue [31], it is most likely of acceptance on a general-purpose processor such as Intel 80486. A supplemental processor equivalent to a digital signal processor (DSP) could provide the processing strength (which we estimate to be about 50 MIPS), but the rapidly escalating capabilities of processors almost certainly render a supplement unnecessary.

2.2.3.3 Wide Range of Subjects

Mathematics learners in general and Kumon students in particular vary by age (from three to adult), writing speed and skill (including some special needs writers) , na-

tionality (at least eighteen countries), and writing style (at least American, European, and Japanese character formation styles).

2.2.3.4 Wide Range of Symbol Sets

The system must abstract from the continuous input stream of the pen a sequence of discrete symbols. It must constantly recognize numbers and gestures, and occasionally recognize other symbol sets such as letters. It must do so as well as known current systems (95% to 99% depending on the symbol sets) while meeting the other criteria of this specification.

2.2.3.5 Usage

The system must adapt to regular users for the reasons given in the last chapter, but must not lose the prior or occasional recognition ability. In abstract terms, the system must gain knowledge of its user(s) and encode this knowledge in ways that permit it to adapt over time to patterns of usage. Thus it may be described both behaviorally (through the eyes of the user), and systematically. Behaviorally, the adaptable system alters the mapping from input stream to output stream over time. To the user, the system manifests itself in improved recognition of characters and gestures. Relating this to a specific system such as the Kumon Machine, it remains stable in recognition responses to a set of occasional users (instructors and parents) while dynamically adapting to high-frequency users (students).

2.2.3.6 On-Line Recognition and Adaptation

Both response and adaptation must occur in real time not to frustrate or confuse the user, and not to interfere with the learning task in which speed of completion is quite as important as the output. Furthermore, it must adapt at a speed and with a predictability that successfully negotiates the two-body learning dilemma described above. The user and system must converge in their mutual adaptations and avoid chaotic oscillations. This im-

plies responses and adaptations in about a second — in a time span below the user's distraction threshold.

2.2.3.7 Self-Criticism

The system must develop a reliable measure of the likelihood of correct recognition. Blind adaptation is worthless at best and dangerous at worst. Obviously, if the system is to adapt, it should adapt in the "right" direction. To do so, it must constantly evaluate its adaptation. Failing this assurance, the historical default has been at least to remain consistent and not risk adaptation. The system must therefore develop some reliable measure of its own likelihood of correctness. This value is usually quantified as the *confidence factor* associated with a given response. If sufficiently high, the system may presume that it has identified the incoming data successfully and may take appropriate steps such as (1) associating the particular input with a given symbol, (2) retaining the input for later reference, or (3) confirming the validity of the current candidate representation (as opposed to suggesting an alternate representation). If the confidence factor is below some parameter, the system may ask for clarification, or may offer the user choices. In any case the development of this measure is critical to successfully managing adaptation in realistic environments.

2.2.3.8 Context-Free Classification

Unlike English or music, the language of mathematics furnishes no context cues. The system has to recognize each digit, letter, or symbol on its own merits. This problem differs in this respect from many handwriting recognition tasks where Viterbi maps or dictionaries may be consulted to resolve ambiguity.

2.3 Completing the Paradigm

The pen has the latent power to shift the paradigm of the user's relation to the processes (as did GUI's). If it is to do so and not remain just a wireless mouse, a broadly adaptable system must complete the paradigm. Furthermore, the disappointing performance of current handwriting recognition continues to hamper the reception of pen-based technology. Success in solving the problem posed above would open doors to many home and educational uses of pen-based computers. It contributes to the progress toward more broadly adaptable pen-based computers.

The issues raised by this problem specification call for the design and construction of systems of non-trivial adaptability, and extensive observation of actual users at the fluctuating frontier of learners and systems — systems such as the Kumon Machine. Our particular research furthers this exploration. Chapter 3 reviews related research in human-computer interfaces and in learning systems. Further chapters development the details of our proposed solution and present results, evaluations, and conclusions.

3. Review of Related Research

Adaptation to computer users is nearly as old as computers themselves. The hierarchy mentioned in Chapter 1 and elaborated in Appendix A helps to categorize adaptation. Within the spectrum of possible adaptations, our specific technology, SCL, is proposed as a method to successfully manage the adaptation layers associated with individual characters and gestures. This chapter reviews the research relative to general adaptable interfaces, and learning system research specifically relevant to our work.

3.1 Research Related to Adaptation

Adaptation to users wears many guises. The history-based macros of Kurlander and Feiner [102], Halbert's programming by example [64], and Kimura's Hyperflow [92] adapt at higher layers. Trainable recognizers (like that of GO's PenPoint in Carr and Shafer [25]) adapt at middle layers (scribbles). Input filters such as exponential smoothing may be used to let the system adapt at the lower layer (points).

Dix et al. review human-computer interface (HCI) models [36]. Krasner and Pope [97] discuss the advantages and limits of the popular Model-View-Controller (MVC) user interface paradigm especially as it relates to Smalltalk80. Rubine [143] extends this basic model to let multiple controllers and views relate to a model, and illustrates the power of the gestural interface (discussed below). Chen and Leahy [27] write that a successful model of the user interface should permit event transmogrification (one physical device can model several others). Laurel [107] associates human computer interaction with the relation of dramatist to audience. She discovers a uncanny parallel between HCI and the

classical elements of drama in Aristotle's *Poetics*, written in the fourth century B.C. This parallel is discussed in considerable detail in Appendix A.

3.1.1 Learning Theory and Interfaces for the Learning Computer

While Laurel finds an insightful model in drama; many interface researchers have found useful models in education. A central tenet of our adaptation research is Alan Kay's claim [83] that human interface design is inextricably related to learning theory. Many learning models bear directly or indirectly on the relation of the user to the system.

Noted learning theorist Jean Piaget talks of the sequential conceptualization of the external world built over the first decade and a half of the child's development. He shows how children first engage their surroundings by direct action; later they draw analogies from concrete examples; finally they grow into the abstract or formal reasoning (Piaget [130], [131], Inhelder and Piaget [75], Inhelder et al. [76]). He describes the power of early kinesthetic memories, "unconscious regulations," as a protolanguage of later problem-solving. The learner gradually gains knowledge of how to understand and manipulate an environment. Bruner identified these learning stages as enactive, iconic, and symbolic. The GUI metaphor lets user manipulation retreat from linguistic constructs (as in command line interfaces) to more visual patterns — Bruner's iconic learning ([14]; cf. Bruner [13], [15], [16] and Bruner et al. [14], [18]).

Alan Kay comments on the powerful link between learning paradigms and the design of sound user interfaces [84]. He states that our interfaces must let each user trace the same path from doing to seeing to symbolically understanding that we did as young learners. He notes that the GUI interfaces that emerged from Xerox PARC in the late 70's [82] were designed to guide the user through just such a progression. *Doing* (with the mouse)

with *images* (literally the icons of the interface) makes *symbols* (leads to the abstraction of activity in the Smalltalk programming language). Many other computer science writers (Strohecker [158], Papert [127], Kay [83], [84], and Pree [133]) have related Piagetian learning stages to interface design.

Computer-assisted instruction (CAI) is a special case of human-computer interaction. It fails often simply because even the more clever interfaces are capable of very limited adaptation (Papert [127], [128], Turkle [169], Solomon [155], Kearsley [85], Cuban [31]). The transactions generally proceed according to standardized templates. Notable exceptions such as Rocky's Boots and LOGO succeed by introducing the broad expressivity of a new language — adapting in the broad sense in which programming might be considered adaptation. Human teachers are successful for many reasons (coaching, caring, encouraging), but one element of good teaching is that learning occurs constantly on both sides of the transactions. The interface between teacher and learner is not a wall, but a malleable membrane which stretches, reconfigures, and adjusts.

Most computerized teaching tools tend to overstay their welcome at one layer of the interface. In contrast, Snyder and Palmer find that computerized games are successful largely because they offer a number of skill levels to challenge the learner successively [154]. Young video adepts embody a certain rhythm of focus, intensity, familiarity and new challenge [110]. The game makers have plumbed it; we educators have not. LOGO was exceptionally successful because it thus allowed itself, its features, and its capabilities to grow with the learner [127, 169]. It remained "new" — in effect, adaptive — longer than its peers.

3.1.2 Gestures and Pen-Based User Interfaces

Gestures may provide a high degree of functionality in a very compact form, but the gestures must be memorized by the user. Taysi explored the relations between the design of gestures, their utility, and the ease with which the user masters them [164]. Buxton et al. developed SSSP, a gestural system for music transcription [19, 21]. Rubine's drawing gestures simultaneously convey the object, the operation, and as many as two parameters to the system with one gesture [142, 143]. His system introduced a type of swift adaptation in the concept of "eager recognition." Unique subgestures may be recognized as soon as the subset of entered points renders the gesture unambiguous. This is a compelling adaptation, but requires great care in selecting the suite of gestures. For example, the vertical stroke of a "4" might be eagerly recognized as a "1." Thus digits do not form a set of gestures capable of "eager recognition." Zhao illustrated the advantages of diagram editing with gestures [185].

Although they do not address adaptation directly, Baecker and Buxton provide a taxonomy of input devices including pens and digitizing tablets. [6:360]. They also stress the need for careful observations of actual users to evaluate interface design [6:216]. Buxton discusses the need to associate *kinesthetic* connectivity to *conceptual* connectivity in input system design [19]. With Hill and Rowley, he developed a primitive paint program to illustrate the possibilities of touch-based interfaces [22].

A plethora of commercial pen-based systems has flooded the market in the past four years. The results has been modest at best. The PenPoint system from GO was one of the first. It is not only a pen-based user interface, but a full object-oriented, pre-emptive multitasking operating system [54]. Its principal instantiation in the market is through the extensive marketing arm of AT&T. This product (EO) was drastically redefined earlier in

1994 into a smart cellular phone and later abandoned completely. The Microsoft offering, PenWindows, received much early attention. It is not an operating system, but an application interface on top of Windows which runs (currently) on top of DOS. Microsoft's President Bill Gates declared [52], "We think the pen is the best pointing device. It's direct, it's more natural than even the mouse. It's familiar, unobtrusive, small and very precise." This enthusiasm notwithstanding, Microsoft has downplayed its offering, PenWindows, to near invisibility. The Newton has fared somewhat better. Its handwriting recognizer adapts to the current user, but loses earlier learning in so doing. Furthermore, the first Newton iteration stirred so many complaints about the handwriting recognizer (besides the week of Doonesbury lampoons), that the replacement Newton announced in March of 1994 offers a tiny keyboard for on-screen typing of problematic input [187].

3.2 Review of Learning System Architectures Related to SCL

Many researchers are active in learning systems architecture. This review groups research by its relation to our own (adaptive learning, feature abstraction, and so forth). The earliest learning system forebears were Hebb in 1949 [66] and Rosenblatt in 1962 [140]. Their perceptrons had a single layer whose learning consisted of modifying the weights from input to output. This approach to pattern classification required that the patterns be linearly differentiable. Minsky and Papert demonstrated in 1969 [114] that many significant patterns were not linearly differentiable. Since no training algorithm existed for a multiple-layer learning system (that is, one having "hidden" layers mediating input and output), Rosenblatt's scheme languished for years in retreat.

During these years, advances in machine learning were made by Grossberg in 1976 [59, 60], Fukushima's Cognitron in 1975 and Neocognitron in 1980 [41, 42, 43], Widrow's bootstrap learning in 1966 and later Madalines (multiple-layer adaptive linear elements)

[181], and Kohonen's self-organizing maps in 1982 [93, 94, 95, 96]. Kohonen's unsupervised learning systems organized the input stream into maps that fitly characterized the topology of the feature space; patterns that share many features are near neighbors. The Neocognitron learns in either supervised or unsupervised modes. Unsupervised learning systems like those of Kohonen and Grossberg simply group the input stream into clusters of similarity in the feature space. Hsieh and Chen [74] combined many of these technologies. Their system used multiple Neocognitron-like layers each one of which performed self-organizing maps (*a la* Kohonen) with layers employing both supervised and unsupervised learning.

While searching for a better model of national assimilation and mobilization in bicultural societies, Werbos discovered in 1974 a simple and elegant multiple-layer training algorithm — backpropagation by gradient descent — for non-linear systems like multilayer perceptrons [179]. Thus trained, a neural net is theoretically a universal function approximator [74]. Werbos' methods remained unknown to the general world of machine learning for a decade; in fact several modern references including the compendious *Encyclopedia of Computer Science* incorrectly place his dissertation in 1984 [134:933]. It was "rediscovered" by Parker in 1982, but not until the watershed text of Rumelhart et al. in 1986 [144] was the backpropagation training method given the systematic and clear framework that impelled it to widespread attention. Since then, research on neural networks has exploded. Within three years of Rumelhart et al. more than 4000 articles had appeared on the subject [186]. Not a month passes without workshops, conferences, and new books appearing on myriad phases of the field. A slightly partisan, but useful historical summary may be found in Widrow and Lehr [181].

Other significant contributors to the field of machine learning during the last decade include Carpenter and Grossberg ART I, II, and III (introduced from 1976 to 1988, and discussed in more detail below), Hopfield's 1982 signal feedback nets, Hinton, Sejnowski and Ackley's Boltzmann Machine in 1984 with Hopfield nets and simulated annealing [69], and the 1982 Nextor learning system of Cooper, Reilly, Elbaum and Scofield, also discussed below.

3.2.1 Carpenter and Grossberg: ART

Adaptive Resonance Theory (ART) was proposed originally by Grossberg in 1976 [59, 60] and refined by Carpenter and Grossberg [24] as a solution to the stability-plasticity dilemma in the competitive learning paradigm. It consists of two sets of processing nodes: the *attention* subsystem and the *orienting* subsystem. The nodes in the attention subsystem compete with each other when activated by an input pattern. Grossberg called this *short term memory* reflecting the dominant biological inspiration of the model. The winning node represents the learned category of the input pattern and also retains a representation of the prototype pattern associated with the category. The orienting subsystem compares the prototype pattern to subsequent input; if the two are significantly different, it resets (disables) the winning node, triggering a new round of competition. The resetting response (which we incorporated into SCL) assumes that the input pattern does not truly belong to the category represented by the current winner. This corresponds to *long term memory* in the biological counterpart. If all prototype patterns in the attention subsystem are sufficiently different from the input, the input pattern itself becomes the prototype of a new node representing a new category.

The ART model assumes no teaching input and performs unsupervised learning. It organizes itself to group "similar" input patterns into the same category. The degree of

similarity and consequently the degree of category proliferation is controlled by the vigilance parameter (also incorporated into SCL). An ART system with low vigilance will permit grouping of patterns that are only grossly similar; a system with high vigilance will try to form separate categories for patterns that have only minor differences. The phrase "adaptive resonance" is a holdover from the original implementation through analog electronics which produced a literal electrical resonance between the short term memory traces and long term memory traces when the two reached agreement. In 1994 a team at the National Microelectronics Center of Seville, Spain, built ART1 into a VLSI chip capable of clustering 100 binary pixel input patterns into up to 18 different categories [147].

In the ART2/BP network [157], Sørheim uses the ART2 [23] network to control a collection of supervised backpropagation subnetworks. The ART2 network controls the creation and training of the small BP networks (one hidden layer with two nodes). When an incoming pattern matches an existing prototype closely enough (according to the Long Term Memory traces of ART2), the pattern trains those LTM traces and then trains the BP network as well. When the incoming pattern does not match (distance > vigilance) any of the existing ART2 prototypes, a new one is formed, and an associated BP network is formed and trained. He claims that this system (1) helps to resolve of the stability-plasticity dilemma associated with neural network training, (2) permits supervised training of an ART network, and (3) remains stable in old responses while learning new responses [157]. This model has been used to simulate liquid flow into storage tanks.

3.2.2 Other Adaptive Learning Models

Our work follows most directly that of Grossberg and Carpenter, but several other learning architectures are sufficiently akin to SCL to merit review. These are presented in rough order of their kinship to our own architecture.

3.2.2.1 Reilly, Cooper, Elbaum and Scofield: Nestor

Reilly, Cooper, Elbaum and Scofield [28, 29, 136, 137] recognized the value of multiple prototype formation as well as the retention of training examples for these prototypes. They employ control and learning mechanisms quite different from those of Grossberg and Carpenter and our own; their model predates the backpropagation revival of the mid-1980's. They use "radius-limited perceptrons" or cells which are organized into supervised learning collections which are in turn compounded into a learning system architecture; the latter two are commercially known as Nestor Adaptive Modules™ and the Nestor Learning System™ .

Each cell captures a single input exemplar, adopts its category label, and begins with a default value for λ — the radius of its influence. Every subsequent input within this radius is categorized with its label. If this labeling proves correct (since this is supervised, the correct label is given), λ is increased. If the input item has a different label, λ is reduced. Thus the feature space is gradually claimed by the cells.

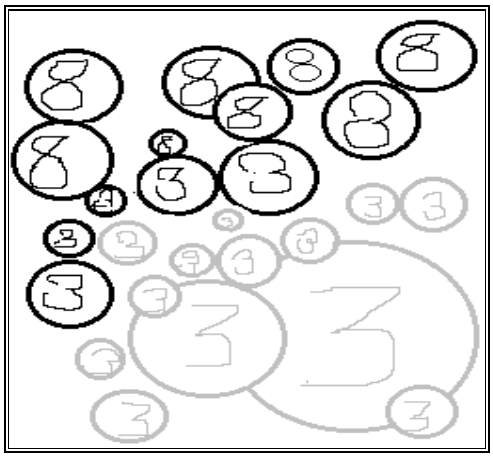

Figure 2. Nestor Boundary

Figure 2 shows the boundary between two categories (*3* and *8*) in a hypothetical two-dimensional feature space. The 3-cells and 8-cells nearest the boundary have prototypes which are similar to each other and have smaller radii (smaller λ) since they are gradually pushed back by contrary categories in the neighborhood. An input within a cell labeled *3* is marked as *identified.* A new input that falls outside of all current cells is marked as

unidentified and may form a new cell, claiming a new region of the feature space for its category. Finally an input that lies in an overlap between cells of differing category names is marked *confused* and leads to reduction of the cells' λ's and possibly a new cell at the boundary.

A potentially unlimited number of prototypes may be formed from unclaimed regions of the feature space. Each completely defines the associated category response; that is, the Nestor cells do not (as ART and SCL) average patterns in the formation of hopefully more general prototypes. This leads to a large number of prototypes to categorize patterns and improve generalization.

A collection of prototype cells are collected together into a Nestor Adaptive Module™. Further, a number of such units are compounded into the overall Nestor Learning System™. Each unit may receive different features, hence a different view of the environment. The units compete among each other for *priority*. Those which are most consistent in identifying inputs have increased priority. When new inputs are received, the units are polled in priority order; as soon as one returns an unambiguous response, no more are polled. Thus inputs will tend to be identified by units best suited (by feature set and training) to distinguish them. If no unit is able to unambiguously identify the input, a simple vote-counting combination of the responses determines the system response.

The principle application area involves the processing of aerial images and industrial assembly tasks. The various feature sets in this image processing use pixel templates to detect edge contours, and raw bit maps. The most recent instantiation of Nestor learning units uses neural networks to manage the community of Nestor elements. This learning system runs in transputers built by SGS Microelectronics [182]. Applying Nestor Learning Systems™ to the task of recognizing handwritten digits, it recognizes characters at the

rate of 10 characters per second per transputer unit (TRAM), so 6 TRAMs yields 60 characters per second in off-line character recognition. This system was able to recognize 98% of handwritten digits in a test set.

3.2.2.2 Kohonen: Self-Organizing Maps

Kohonen introduced topology-preserving maps in 1982 [93, 94], using Learning Vector Quantization (LVQ) as an element of such maps in the special case where the maps are to be used as classifiers [95, 96]. These maps preserve topology in the sense that neighbors in the input feature space are mapped onto neighboring nodes of the network — a one- or two-dimensional lattice. The weight updating is first done across large neighborhoods and progressively refined as the map is organized into finer detail. These nodes act in league with their neighbors to establish pairwise boundaries, and consequently require quite a few processing units (as many as ten or more per category) to build suitable borders in the feature space. These maps have been used in speech recognition, robotics, classification, and numerical integration [30] [96].

3.2.2.3 Fukushima: Neocognitron

Fukushima's Cognitron and Neocognitron [41, 42, 43] are designed to closely simulate hypothesized brain functions. The forward signals manage the function of pattern recognition; the backward signals manage the function of selective attention, pattern segmentation, and associative recall [41:66]. The Neocognitron uses a hierarchy of progressively more abstract feature detectors (dozens of "cell planes" in a each of nine layers for a digit recognizer). Pixel data is entered at the lowest level. At an intermediate level, such features as horizontal, vertical, and diagonal strokes and intersections are detected. The classification categories themselves form the highest level. Thus, errors in local features are gradually tolerated through the cell planes. This tolerance, plus the ability of the cell

planes to modify connection weights ("gain") (within themselves and their weights to others) make the model more robust in the face of deformations and juxtapositions.

3.2.2.4 Smieja: MINOS Pandemonium

The original Pandemonium architecture was proposed by Selfridge in 1958. Each category is represented by an expert "demon" with possibly differing feature filters. The one who "shouts" the loudest is recognized as the winner. Smieja's MINOS demons [153] are neural networks each devoted to a category and mediated by a confidence-calculating monitor that attempts to keeps the demon network confined to areas where it is competent. His results were stronger in learning real-valued functions than corresponding single neural networks that had four times the complexity. His system was not significantly better though for classifying uppercase characters (88%).

3.3 Review of Other Learning System Elements

Our own learning architecture rests on other significant system elements such as feature abstraction and input stream filters. In this section we briefly review literature relevant to these elements.

3.3.1 Feature Abstraction

Much research has been published on feature abstraction. We focus this review on systems akin to our own, namely those incorporating energy-related or graphical features, that is, those operating on enactive and iconic layers of the interface.

3.3.1.1 Energy-Based Feature Abstraction

The germ of the notion of using energy-based features to decode handwriting was in the method used by document examiners (and is explained further in the next chapter). Such "ballistic" muscular action (that is, unmediated by visual feedback) was first

proposed by Lashley in 1917, and was applied to signature verification by von der Gon and Thuring in 1965. Herbst and Liu [68] used this idea to construct a signature verifier that tracked the hand's acceleration on the writing surface. Their quite sophisticated hardware included x and y accelerometers, and a resistance foil-strain gauge to capture "downward" (z) pressure as well. The data was then passed through a linear phase recursive digital filter. Rumelhart uses a variant of this in recognizing cursive handwriting [145] as do Morasso et al. [116, 117]. Rodin, Wu, and Amin [139] described a more straightforward form of this idea in tracking the successive deflections of the strokes that form characters. This is similar to the "spring" distortions and relaxations of Webster et al.[177] and the spline deformations of Hinton et al. [70, 138].

3.3.1.2 Grid-Based Feature Abstraction

For all the utility of energy-based features, several researchers have found that a stronger recognizer could be constructed from a combination of orthogonal feature sets than from any single feature set alone. We therefore review several relevant static feature abstractors.

As mentioned, Fukushima's Neocognitron [42, 43] includes several layers of neural discriminators to detect features such as the orientation of pattern subsets. Particular discriminators may detect the upper right corners of *7*'s; others may detect the straight bar across the top of 7's and 5's. Hsieh and Chen [74] use a similar scheme with four directional lines (horizontal, vertical, two diagonals — like Grothe [62]) as feature data. Nellis and Stonham [123] employ random access memory units combined into associators as their learning engine; their feature abstraction uses positional features "masked" from four directions. Their work also illustrates the effectiveness of flexible prototypes and the processing economy gained by disabling infrequently-used classifiers. Several researchers

have shown that recognizers may be combined hierarchically to improve the total recognition performance (cf. Fontaine and Shastri [39], Kohonen [96], Cooper et al. [28], [29], and Shmatikov [151]). Grothe [62] suggested a simple three by three grid overlaying the bounding rectangle of the (smoothed) input pattern (cf. Fontaine and Shastri [39]). Lin [109] used a similar two by two grid. Kawatani [81] uses a four by four grid and eight ranges of inclination within the sixteen cells. One OCR version of the Nestor model uses 30 grids (each including 12 edge templates) for pixel template matching (Wright and Scofield [182]); this results in 360 input features.

Rumelhart [145] uses a clever recurrent neural net to produce a strong cursive recognizer. It uses Hollerbach's "coupled oscillator" to model handwriting as sinusoidal x and y components. He also uses the x velocity at the y extrema to detect forward loops (*m*'s and *n*'s), abrupt peaks (*i*'s, *t*'s and *j*'s), and backward loops (*q*'s). He defines a stroke as the distance from one y extremum to the next. A cursive *m* has the highest stroke count of any cursive letter, namely six. His network detects the letters or subletters at any of 60 stroke positions into which it divides the input stream. (Typical words require 20 strokes thus defined.) This is similar to the feature scheme used by Pachikov in CalliGraph developed by the Russian firm ParaGraph [165].

3.3.1.3 Feature Trimming

The difficulty of this wide collection of potential feature sets is that it may overwhelm the processing available. Some systems are capable of trimming irrelevant or redundant features. Gramss' Feature-Finding Neural Network (FFNN) progressively prunes a large feature space (or alternately grows a feature set from a large set of possible features) to discover a minimal feature set for recognizing spoken words. The features are calculated from a set of 50,000 possible combinations of time and frequency data, but after

several hundred substitutions, his algorithm is able to select a subset of a hundred features that meet the recognition criterion of 98% success [56, 57]. Gramss adds that linear networks (networks of one layer) achieve better generalizations precisely because they have fewer degrees of freedom. In linear neural nets, the relative importance of each feature is measurable from the absolute value of the weight.

3.3.2 Preprocessing and Postprocessing

Many input streams may be usefully processed by more than one kind of classifier (such as neural nets). Some inputs may be easily detected algorithmically (taps, drags, shapes, for example.). In such cases, a neural net would add computational expense, if not confusion, to the recognition task. Such algorithmic filters may be monitored and adjusted by specialized adaptation that responds to the success of the algorithms. This is one type of preprocessing. Statistical filters and shear corrections are other examples. Morasso et al. use a fourth order high-pass Butterworth filter to eliminate natural tremors in the writer and surface roughness effects [117].

Input to a recognizer may be constrained to "pre-segmented" boxes; this practice guarantees that only one member of the category set is present. In other cases, the input stream must be segmented before or during recognition. Morasso et al. analyze full words both through bottom-up pattern matching (with dictionaries of a few hundred to a couple thousand words) and top-down lexical constraint [116, 117]. Rumelhart uses recurrent neural networks to detect individual characters within the stream of incoming strokes [145]. Both Rumelhart and Morasso et al. use the points of minimum speed (y, and both x and y respectively) to demarcate "stroke" boundaries within the stream of cursive input.

Postprocessing may prove useful after neural nets have made specialized gross categorizations (for example in detecting attributes such as a high crossing stroke, a period, or multiple crossing strokes). Some systems use decision trees; these may be used in conjunction with a neural network to produce more robust systems for detecting and correcting errors.

3.4 Some Proven Theoretical Results

Despite four decades of learning system research, proofs of convergence, classification performance, robustness, and complexity are often elusive. Many theoretical results of computational learning theory are borrowed from the sister theories of stochastic systems, dynamic programming, and Markov models. Yamanishi [183] and Sompolinsky [156] offer summaries of these results. Widrow and Lehr [181] give a good summary of early results concerning linear elements and statistical pattern capacity. (Typical of neural network literature, the same article gives much empirical advice about weight initialization, learning rates, number of training patterns, and such.) They also argue that Widrow's multiple-layer adaptive linear element (Madaline III) is equivalent to backpropagation:

> ...to the extent that the sample derivative $\Delta\varepsilon / \Delta w$ [Here ε is the error term and w is the weight term to be adjusted. This ratio is calculated empirically by perturbation measurements.] is equal to the analytical derivative $\partial\varepsilon / \partial w$, the two rules follow identical instantaneous gradients, and thus perform identical weight updates. The backpropagation algorithm requires fewer operations than [the Madaline rule].

The only thorough analysis of Kohonen's topology-preserving maps has been for the one-dimensional case. The proof of the self-organizing property and of convergence was presented by Cottrell and Fort in 1987 [30]. Some partial results have been produced for higher dimensions.

The validity of backpropagation was demonstrated by Werbos in 1974 [179], later by Rumelhart et al. [144] and algebraically for acyclic networks by Kimura [89]. Blum and Rivest proved that training even a very simple neural net (two layers with a total of three nodes and n binary inputs) is NP-complete [12]. Furthermore they observe:

> Our results, like Judd's, are described in terms of "batch"-style learning algorithms that are given all the training examples at once. It is worth noting that training with an "incremental" algorithm that sees the examples one at a time such as back-propagation is at least as hard. The NP-completeness result given here also implies that incremental training algorithms are likely to run slowly.

Within the past year, DasGupta, Siegelmann, and Sontag [32] extended this result to real-valued inputs; Maass and Turin [111] showed that training a multilayer linear threshold network is NP-complete.

In 1987, Hecht-Nielsen employed a clever extrapolation of Kolmogorov's general representation theorem (proven thirty years earlier) to prove that $2n+1$ hidden units are sufficient to implement any continuous real-valued function defined on the n-dimensional unit cube [67]. Kurkova [101] extended this in 1992 to a neural network of *two* hidden layers acting as a *general* function approximator (differing only in weights from the second to the output layer to realize a particular function). He proved the upper bound for the number of weights in the two hidden layers to be $\boldsymbol{O}(n^n)$which is hardly surprising given the universality of the network.

Maass [111] gives a clear theoretical summary of current results and open problems in complexity bounds for networks that learn from mistakes and for probably approximately correct (PAC) learning of nets. On the application side, MacKay [112] notes:

> There are many knobs of the black box of "backprop." Generally these knobs are set by rules of thumb, trial and error, and the use of reserved test data to assess generalization ability (or more sophisticated cross-validation). The knobs fall into

two classes: (1) parameters that change the effective learning model, for example, number of hidden units, and weight decay terms; and (2) parameters concerned with function optimization technique, for example, "momentum" terms.

His paper gives a Bayesian framework for understanding and selecting among the trade-offs implicit in these knobs on the black box of backprop. His analysis focuses on interpolation networks (as opposed to classification networks), but he claims that the techniques are useful in either setting.

3.5 Review of Handwriting Recognition

Limits in the theoretical results of general learning architectures shift emphasis to practical demonstration. Due to the sponsorship and standardization of the postal systems of the United States and Great Britain, much attention has been given to the off-line recognition of handwritten letters and digits [10]. The rapidly growing interest in pen-based systems during this decade has sparked an interest in on-line, real-time recognition.

3.5.1 Off-line Handwritten Character Recognition

Off-line handwritten character recognition is often called optical character recognition (OCR) (thus confounding it with the same process applied to machine written text). Despite nearly four decades of research, results have remained modest. Some of the best results to date on USPS ZIP code digit images are reported by Fontaine and Shastri [39] using a system that converts the static representation of a character into a time-varying signal (cf. Lee and Pan [108]). This greatly simplifies the dimensionality of the problem, and led to impressive results: 96% recognition of test digits with no rejections. If 15% of the digits are rejected, the recognition rate rises to 99%. Hinton et al. used spline deformations to achieve recognition rates of 95.0% to 97.5% [138]. The latter rate represents digits which were well segmented before presentation to the recognizer. Kawatani used

learning by discrimination analysis (emphasizing the features that disambiguate) to achieve recognition rates as high as 99.4% [81] on data written by Japanese subjects.

Laine et al. use wavelets at several resolutions as features into single-layer neural net [104, 105]. They report typical number recognition rates (95-96%), but surprisingly higher performance for the recognition of letters and symbols (97-98%).Their input sources were engineering drawings which exhibited better legibility than could be expected from on-line recognition systems in general use. The time required for training convergence ranged from 9 hours on a Sun 10/30 workstation to 43 hours on a Sun 4/380 workstation.

The computational complexity of the recognition task has naturally led to the investigation of massive parallelism. In the largest character recognition project known to the author, the National Institute of Standards and Technology developed a massively parallel (Digital Array Processor) recognition system for digits (off-line) [170]. Its 22,500 lines of code represent 12 man-years of effort. It segments scanned data into individual digits and then recognizes them. The segmentation algorithm is 89% accurate. If segmentation errors are *excluded*, the recognizer itself is 96% accurate. Constrained by feature extraction, it handles 806 characters per second.

Nellis and Stonham [123] experimented with handwritten addresses made available from the British post. Their work used “discriminators” compounded from random access memories (RAMs) (cf. Aleksander and Norton [1]). Their system uses a masking technique for balancing the pixels between white and dark to accommodate the RAMs, and turns off certain discriminators if not recently used. They report recognition of 73% of test letters and digits (a substantially more difficult task than just digits) in context-free OCR recognition, and 94% with the addition of context-related data (dictionaries, Viterbi

maps and word frequency tables). Two authoritative surveys of handwritten character recognition are given in Suen et al. [160] and Tappert et al. [163].

Other research has been done using a stream of points, but these have often been collected from a mouse (Rubine [142], [143] and others) or from awkwardly mounted tablets as appendages to regular computers. The close-in pencil-on-paper paradigm unquestionably shifts the character of the interaction to call for renewed investigation, and demands a more tightly-knit, seamless, and flexible user interface. A novelist and software designer, Rob Swigart [162] makes this point by analogy with an imagined desktop of Charles Dickens:

> Contrary to current mythology, writing is not an entirely 'organic' process. The product is shaped by its tools in ways unconscious and unsuspected. The movement of the hand manipulating the quill pen, the play of muscle and tendon as the mind directs words to flow from the tip, the pauses to dip for more ink, all impact the outcome. . . . The relatively slow movement of the hand (compared, for example, to the speed at which one can type) leaves mental time for longer, more intricately constructed and elaborately rhetorical sentences than are encouraged by a faster medium.

Educators echo this sentiment in the value of the paper and pencil paradigm to the process of solving mathematical problems (cf. Hall [65], Davis [34], Baecker and Buxton [6:210]). Pen-based researchers Nakagawa et al. [121] note that the pen works well for creative work since it freely expresses all manner of ideas without regard to specialized methods (such as draw programs) — "modeless" creativity in effect. To just this end, we now turn our attention to the various efforts to recognize characters as they are created on-line.

3.5.2 On-Line Handwritten Character Recognition

Real-time character recognition imposes strong constraints on complexity. The advent of pen-based computing is accelerating the already heightened interest in this field. However, this process dates at least back to the mid-sixties with the work at Rand [33]. In a broad survey of the state of the art, Tappert et al. note [163] that the recognition task is simplified by having actual temporal data. For example, it is much simpler than the superimposed temporal structure by Fontaine and Shastri [39]. The on-line recognizer may observe the number, order, and speed of the individual strokes composing the input pattern. This may be a mixed blessing since characters that appear identical (once written) may be formed by very different processes. For example, a nine may be formed from the bottom to the top, or vice versa, or by separate strokes for the stem and the circle. Tappert et al. also note that an on-line recognizer can always request clarification of ambiguous data, and that users tend to adapt (even unconsciously) to forms that are more palatable to the recognizer.

Much of the significant research in this area is regrettably obscured by the real and hoped-for commercial potential of pen-based computers. GO Corporation has permitted the publication of its work in handwriting recognition within the PenPoint Operating System (Carr and Shafer [25]). Their recognition engine is typical of commercially crafted recognizers (as opposed to research vehicles). It includes hundreds of prototypes for its alphabet of interaction: 82 symbols (52 uppercase and lowercase letters, 10 digits, and 20 punctuation marks) and 50 gestures. The system allows for training sessions by a particular user in any of these symbols and gestures. These training sessions are distinct; that is, they are not integrated into the normal recognition activity. In the recognition of what Carr and Shafer describe as “neatly printed” characters (after some training of the users),

the PenPoint system recognizes an average of 94% of the 62-symbol set that omits punctuation.

GO's PenPoint system makes rich provision for context support such as Viterbi mappings, dictionary support, and constrained fields, but the performance reported in our research focuses on the hardest task: context-free recognition. After all, this is required for the mathematical tasks of the Kumon Machine where the austere symbolism of arithmetic and algebra precludes help from context clues.

When the input is constrained to 10 digits, GO's recognition barely improves, to 92-95%. When constrained to 26 lowercase characters, the recognition rate is 89-93%. Since these results were published, GO has released an improved version of the recognition engine for which comparable results have not been published. The PenPoint recognition engine requires about 64 KB of code and about 128 KB of static data. On a 16 MHz 80286-based system, it can perform about 3 recognitions per second — nearly the fastest sustainable writing speed with segmented (non-continuous) characters.

3.5.3 Reported Results of On-Line Handwritten Character Recognition

Comparison with human recognition helps to give the adventurous claims of commercial recognizers a bit of perspective. Most purveyors of handwriting recognizers would be delighted to achieve results equal to humans. Surprisingly, humans fare little better than the best recognizers in context-free recognition. The most frequently-cited study is that of Neisser and Weene in 1960 [122]. They used 36 target categories of uppercase block letters and digits specifically to determine an "appropriate performance standard for a simple [machine] character recognizer." Two hundred subjects contributed handwritten characters. Ten subjects participated in the identification of the pool of over 5000 characters.

Each of the ten viewed twenty examples of each of the 36 targets. One subject was myopic enough to render his results worthless. The other nine subjects successfully identified from 94.9% to 96.5% of the input characters (with mean of 95.9%).

Furthermore, the above-reported success rates reflect an adjustment for two high frequency misrecognition pairs. The subjects were not able to distinguish the letter *O* and the digit *0* nor were they able to distinguish the letter *I* and the digit *1* consistently. Therefore these misrecognitions were not counted as errors in the above data. Subsequent studies indicate that this rate has not changed significantly in the intervening years. Webb reported human recognition rate of 95% of handprinted lowercase characters [176]. (Interestingly, his fusion-reflection algorithm recognized 96% of the same data set.)

Tappert et al. [163] indicate the difficulty of making useful comparisons given the wide diversity of experimental environments (e.g., writer dependency, size of alphabet, hardware capabilities). The best results for Western alphabets in Tappert's survey of 1990 were in the range of 95% to 97% with recognition requiring from .1 to 1.0 seconds per character. This is a bit better than the results reported above by GO with PenPoint.

Gibbs reported ([53]; cf. Wagoner [171]) recognition of Western alphabets in the range of 90% to 98%, but notes that this is based on the vendors' own claims and obscures considerable ambiguity in the quality of writing, degree of adaptation to individual users, and range of sampling among writers. Hsieh and Chen reported 93% with handwritten digits [74].

To carve out a clearing of rationality in this jungle of claims, a number of researchers have formed Unipen to standardize data and testing procedures for handwriting recognition. Consortium participants include more than 170 hardware corporations, software vendors, and academic institutions: for example, AT&T, IBM, Microsoft, NCR, and the Kumon Machine Project at Washington University.

This consortium is first standardizing the formats of data and hardware devices. Preliminary data standards were reviewed by consortium members in October, 1993. Unipen members, in conjunction with the National Institute of Standards and Technology (NIST), are currently reviewing standards for recognition testing and reporting. Then Unipen will conduct impartial tests on broadly-based sample data. A practice test to calibrate the process should occur in fall in 1994. General testing should shortly follow.

This concludes the review of related research. In Chapter 4, we present SCL — both as a model and a system.

This concludes the review of relevant research. In the next chapter, we elaborate SCL more fully.

4. Solution Methodology

This chapter presents the Supervised Competitive Learning (SCL) *model* with the algorithms that define it (section 4.1). Then we describe its implementation as the SCL *system,* and explain the elements that realize the algorithm for practical recognition tasks (section 4.2). The *SCL System Manual* [49] may be consulted for fuller implementation details. Finally, we outline our experimental methodology (section 4.3).

SCL is our test bed for interface adaptation. It encounters the problems confronting general pen-based adaptable interfaces: adaptation, stability versus plasticity, and the evaluation of candidate representations through internally-generated confidence factors. Candidate representations arise from segmenting successive inputs from one domain such as digits, or in domain shifts — whole numbers to fractions or numbers to letters.

SCL is designed to be flexible enough to be applied to several layers of the interface hierarchy. It contains controls to ensure convergence in reasonable time and balancing mechanisms to bring the pragmatic governance to cope with real user environments. SCL has been presented in Fuller and Kimura [50, 51], Fuller [48], and Wang, Kimura and Fuller [174].

4.1 SCL Model

SCL gathers a set of *learning units* into a supervised *learning system*. In this section we present the technology of both the individual learning units and the architecture which binds them into a learning system. We begin with the overall architecture.

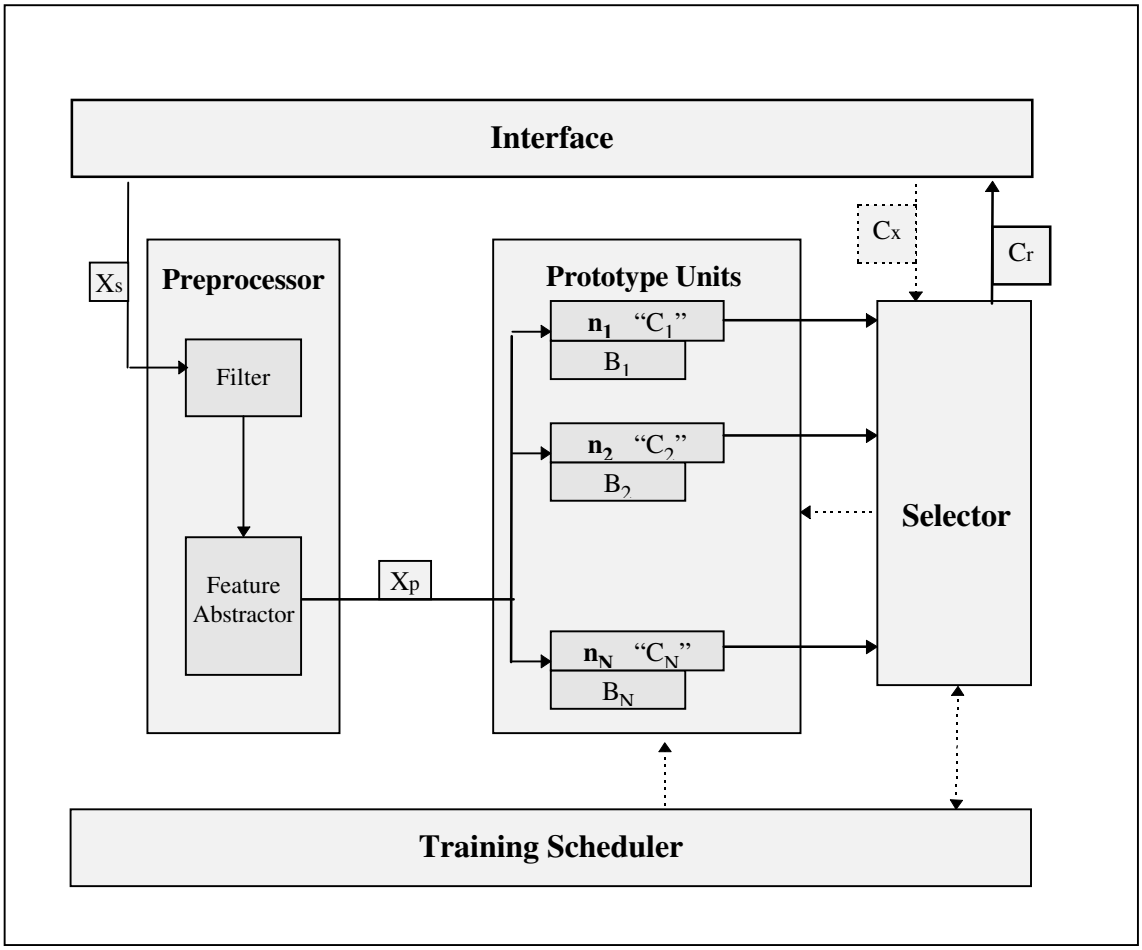

Figure 3. The Five Modules of SCL

4.1.1 Overview of the Model

SCL is comprised of five basic modules shown above in Figure 3. Solid arrows show the required flow of data; dotted arrows show optional flows. The Interface sends user input in scribble form X_s to the Preprocessor and receives back the category as recognized by SCL C_r. If C_r is incorrect, the user responds with the correct category C_x to initiate a round of training to associate X_s with C_x. The Preprocessor receives the input scribble X_s from the Interface, filters it, and translates it into a feature vector. It includes

several filters and several feature abstractors. The real-valued feature vector X_p is sent to the prototype unit module.

The heart of SCL is the collection of prototype learning units (*prototypes* for short). Each prototype is a one-category learning system with real-valued output normalized to some common range. Each quantifies similarity for a set of exemplar patterns considered sufficiently alike, and maintains a *buffer* (B_i) of these exemplar patterns for later training. Every prototype n_i receives the feature vector X_p and every prototype sends its own response $n_i(X)$ to the Selector. The Selector sends the winning category name C_r to the Interface. It also includes a prototype resetting mechanism (akin to that of ART [23, 24, 61]) to manage the creation and deletion of new prototypes. It oversees the competitive training regimen and calculates confidence — the estimate of the reliability of C_r. The Training Scheduler runs the system through combinations of exemplar gathering and reinforcement to balance the training.

The next five sections of the chapter define the algorithms for the Interface, Preprocessor, Prototype Units, Selector and Training Scheduler in that order.

4.1.2 Interface

The Interface's algorithmic responsibility is only collection and segmentation of scribbles from the user. The system aspect of the Interface module is presented in the implementation section.

4.1.2.1 Scribble Input

We adopt the convention (introduced by PenPoint [55] and by Morita and Mori [118]) of defining incoming patterns (digits, letters, or gestures) as *scribbles*. A scribble *S*

is composed of one or more *strokes* K_s, which in turn are each composed of one or more *points*. Thus

$$S = (K_1\ K_2\ ...\ K_k)$$

$$K_s = (P_{s_1}\ P_{s_2}\ ...\ P_{s_{ps}})$$

where k is the number of strokes in S, and p_s is the number of points in stroke K_s. A point is a xy-coordinate pair. A scribble is assumed to represent one member of the *target set*, the set of characters or gestures corresponding to the category names C_i. A *multi-scribble* is an array of strokes as defined above that represents one or more members of the target set.

4.1.2.2 Multi-Scribble Segmentation

The Interface may employ either of two algorithms for segmenting multi-scribbles into scribbles. The first algorithm demands that the system be able to recognize any member of the target set by just its first stroke. It is therefore limited to target sets where this is possible (*not* uppercase letters, for example). For this algorithm, the Interface takes the first stroke K_1 from the scribble and applies the rest of the algorithm to this single stroke for classification and learning. The model must therefore learn to recognize the first stroke of any two-stroke scribble as just that. For an example with digits, an *L*-shaped stroke must be recognized as the first stroke of a *4*. This recognition causes the Selector to notify the Interface module to discard the second stroke during training or recognition.

The second segmentation filter is nearly as simple, but is limited to multi-scribbles that contain no more than two scribbles, and scribbles that contain no more than two strokes each. If the incoming multi-scribble has only one stroke, then it is necessarily one scribble and is sent forward to be recognized. If the incoming multi-scribble has 4 strokes,

then it must consist of two 2-stroke scribbles. These are ordered by relative x-coordinates and sent forward. If two or three strokes are received, various stroke combinations are formed, sent forward through the Selector, and the combination generating the highest average confidence is returned by the Selector to the Interface.

4.1.3 Preprocessor

The Preprocessor transforms the incoming scribble into a feature vector X_p for input into the prototype units. This module is divided into *filters* (smoothing and shearing) and *feature abstractors* (based on energy and grid).

4.1.3.1 Smoothing Filters

The scribble points are normalized to an array and passed through filters to smooth and shear the points (both defined later in this section). Three types of exponential smoothing are available to the Preprocessor. (The choice is made via a parameter that is usually constant during the life of a recognition system.) These are simple, bi-directional and symmetric. Simple first and second order exponential smoothing is widely used by statisticians to smooth data in quest of significant trends. For an example, see [44:163-5]. We are not aware of any description of the bi-directional and symmetric smoothing variants in the literature.

Let e be the smoothing constant — the ratio of the weight of past points to the weight given to the current point. Thus $0 < e < 1$. For simple smoothing, the incoming points $p_i = (x_i, y_i)$ of each stroke are adjusted by the formulae:

$$x_i = e\, x_{i-1} + (1 - e)\, x_i \qquad y_i = e\, y_{i-1} + (1 - e)\, y_i$$

for $i > 0$. The first point p_0 in the stroke is not adjusted.

For bi-directional smoothing the same formulae are used to create a forward set x_f of smoothed points:

$$x_{f_i} = e\, x_{f_{i-1}} + (1 - e)\, x_i \qquad y_{f_i} = e\, y_{f_{i-1}} + (1 - e)\, y_i$$

for i > 0. A backward version is applied to create a backward set x_b of smoothed points:

$$x_{b_i} = e\, x_{b_{i+1}} + (1 - e)\, x_i \qquad y_{b_i} = e\, y_{b_{i+1}} + (1 - e)\, y_i$$

Finally, the bi-directional smoothed point is the average of the two for all i:

$$x_i = (x_{f_i} + x_{b_i}) / 2 \qquad y_i = (y_{f_i} + y_{b_i}) / 2$$

Simple smoothing uses the first point p_0 as an anchor against change whose influence is decreasingly distributed over subsequent points. Bi-directional smoothing uses both the first and last points as anchors. Symmetric smoothing uses each point as its own anchor and weights the points on either side (a local neighborhood) with symmetrically decreasing weights. First the size of the neighborhood is computed. Define the number of points to be included on either side as

$$N = \left\lfloor \frac{1}{(1-e)} \right\rfloor$$

Thus the neighborhood, 2N + 1 points, increases as the smoothing ratio *e* increases. The weights for the neighboring points are defined as:

$$w_k = \frac{(1-e)e^k}{2} \qquad \text{for } 1 \le k \le N$$

$$w_0 = 1 - \sum_{j=0}^{N} 2 w_k$$

Thus w_0, the weight applied to the incoming point, inherits the remainder of the weight not apportioned to its neighbors. The symmetrically smoothed point (for all i) is:

$$x_i = w_0 x_i + \sum_{k=1}^{N} w_k \left(x_{i-k} + x_{i+k} \right)$$

$$y_i = w_0 y_i + \sum_{k=1}^{N} w_k \left(y_{i-k} + y_{i+k} \right)$$

Where the algorithm would encounter non-existent points (near the beginning and end of the stroke), it uses the nearest actual point. This completes the definition of smoothing filters. We now turn to the shear filter.

4.1.3.2 Shear Transformations

Many writers exhibit a natural slant to their characters with the top inclined toward the hand used for writing. We employed a shear filter to correct for this. We determine a likely degree of slant from the right edge of the scribble and then shear [4:122-24] the entire scribble to compensate for this presumed slant. The slant is measured by determining the rightmost point (Rx_{high}, Ry_{high}) in the top fourth of the bounding rectangle and the rightmost point in the bottom half (Rx_{low}, Ry_{low}). These two points determine the degree of shear necessary to reconstruct a vertical right edge. Although this approach could equally well be applied to the left hand edge of characters, we chose to only apply it to the right edge. The slope of a line between these points is the putative slant of the scribble:

$$m_s = (Ry_{high} - Ry_{low}) / (Rx_{high} - Rx_{low}).$$

This slant is presumed constant over the scribble, so if Δy_i is the height above the base of the scribble, and the adjustment to x_i is Δx_i, then $\Delta y_i / \Delta x_i$ must also equal m_s so

$$\Delta x_i = \Delta y_i / m_s \qquad x_i = x_i + \Delta x_i.$$

y_i is unchanged for all i. Thus all the points of the scribble are sheared proportionally to their height above the minimum y-coordinate for the scribble.

4.1.3.3 Energy-Based Feature Abstraction

We use a combination of feature abstractors to sample the user's intent at two layers. Energy-based feature abstraction is associated with the enactive psychological facility of the user [48]. Grid-based features are associated with the user's iconic psychological facility [62].

Much of the research about handwriting recognition (as noted in Chapter 3) has centered on off-line recognition and consequently has rarely considered the dynamic features associated with the actual creation of the characters. Our research [48, 50, 51] suggests that some dynamic features of handwriting, especially the velocity and acceleration of the pen as it creates the strokes, might be usefully consistent across the general population, and strongly stable in individuals through time.

A professional document examiner can detect a forged signature by microscopically examining its sharp turns (such as the top of an *A* or the bottom of a *v*). At normal writing speed, the centrifugal acceleration of the pen causes tiny droplets of ink to be cast off as the pen moves around a cusp. At the slower speed of most forgeries, these droplets are absent or land much closer to the inked line. The forger is able to mimic the *graphical* (static) features of the original, but not the *dynamic*. Conversely, the authentic signer may vary the pattern somewhat from signature to signature, but as this work and that of others suggests, the velocity and acceleration that create the strokes tend to remain constant.

The Codebook-Enhanced Linear Prediction (CELP) algorithm for compressing voice data derives its advantage from the consistent use of musculature and vocal

apparatus by human speakers. Early in our research, it occurred to us (my advisor and me) that this might also be true of writers. For example, most writers write a *9* as successfully with eyes closed as opened. The graphical feedback, while useful for *positioning* the digit, is unnecessary for *writing* it. The signal is apparently encoded in what is popularly called "muscle memory" — possibly as energy quanta to be expended by particular muscle groups to achieve the task. Herbst and Liu observe [68:85]:

> Feedback control in response to error signals from the eyes or the hand requires on the order of 200 ms while the time it takes for the individual muscle forces used to generate the strokes is 30 to 100 ms. These rapid motions are characterized by the fact that their durations are predetermined by the brain, and that the agonist and antagonist muscles for each degree of freedom are alternately energized.

This timing precludes visual feedback. The movement is purely *ballistic* in Lashley's phrase. It should be possible to recover this energy encoding in the velocity and acceleration of the pen (simple derivatives as in [167:289-293]) to a degree sufficient to determine the character or gesture being created. The expected benefit of these features lies in the hoped-for correspondence to the ballistic energy encoding in the user. This encoding in turn relates to the user's enactive psychological interface level. These features thus attempt to abstract the signal at this cognitive layer — bypassing the intermediate representation on the glass in favor of the presumed intent that lies behind it.

We represent the average velocity and acceleration in the x and y directions for each of the 4 to 9 sampling intervals of the scribble. The algorithm assumes that the point sampling period of the input digitizer is constant over time. Slight variations do not materially affect the algorithm's success due to the averaging explained below.

The points of *all* strokes of the scribble are gathered into unified arrays

$$X = \left(x_{P1_1} x_{P1_2} \ldots x_{P1_{p1}} x_{P2_1} \ldots x_{Pk_{pi}} \right)$$

$$Y = (y_{P1_1} y_{P1_2} \cdots y_{P1_{p1}} y_{P2_1} \cdots y_{Pk_{pi}})$$

$$0 \le j \le p_{total} - 1,\ p_{total} = \sum_{k=1}^{s} p_k$$

where p_{total} is the total number of points in all the strokes of the scribble. The differences between successive points are loaded into velocity arrays. The differences between successive velocities are stored in acceleration arrays.

$$V_{X_j} = X_{j+1} - X_j$$

$$V_{Y_j} = Y_{j+1} - Y_j \qquad 0 \le j \le p_{total} - 2$$

$$A_{X_j} = V_{X_{j+1}} - V_{X_j}$$

$$A_{Y_j} = V_{Y_{j+1}} - V_{Y_j} \qquad 0 \le j \le p_{total} - 3$$

The number of sample intervals (V_{samp}, typically 7) determines the *boundary points*, b_m, $1 \le m \le V_{samp} - 1$, of the sampling intervals. For each velocity interval, we calculate the average velocity, and set the corresponding velocity feature equal to this average divided by the maximum velocity for the scribble.

$$V_{Xmax} = \max (V_{X_j}) \qquad 0 \le j < p_{total} - 1$$

Then for the m^{th} sampling interval:

$$V_{Xavg_m} = \frac{\sum_{j=b_{m-1}}^{b_m} v_{x_j}}{v_{x_{max}} \cdot (b_m - b_{m-1} + 1)}$$

Only this final operation requires float values. Similar normalized velocities are calculated for the y velocity vector, V_Y, and for the x and y acceleration vectors, A_X and A_Y. These normalized ratios serve as real-valued feature inputs to the SCL recognizer.

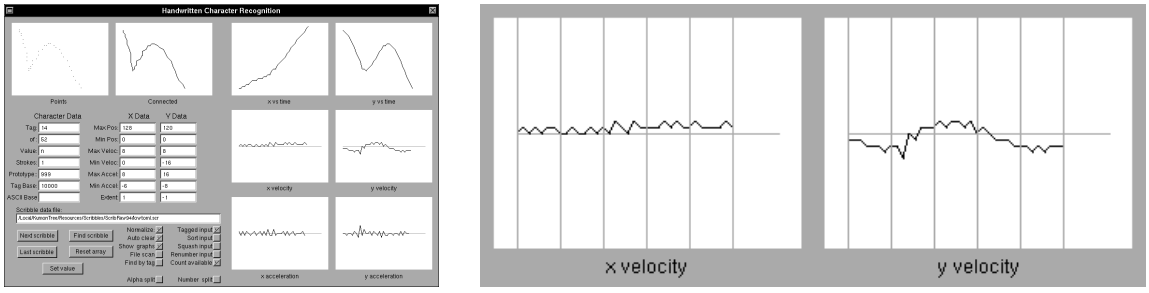

Figure 4. Velocity Features

We offer an example calculation of the velocity features for the letter *n* scribble shown above in Figure 4. (The figure is taken from HCR, one of several graphical user interfaces to the SCL system. Section 4.2.2.4 gives a larger view of the HCR panel and a description of its fields.) On the right, we have magnified the x velocity and y velocity graphs. The horizontal gray line in each graph marks zero velocity. The vertical lines divide the scribble into five sections (V_{samp} = 5). This scribble has 37 points. Therefore the four boundary points (b_m) are points 7, 14, 22, and 29 of the scribble. The greatest horizontal displacement between any two points in this scribble was two digitizer units (V_{Xmax} = 2). The point by point horizontal displacements for the seven points in the first sample (the leftmost division in the magnified x velocity graph above) is 0, 0, 1, 0, 1, 0, 1 so the average x velocity for this fifth of the scribble is 3/7 or .429. Therefore the first x velocity feature is this average divided by 2 (V_{Xmax}) = 0.214. The other four x velocity features are 0.214, 0.438, 0.643, and 0.625. These values correspond to the x velocity graph shown above; the x velocity is always positive and slowly increasing. The five y velocity features are -0.321, -0.393, 0.375, -0.143, and -0.562. Again, the y velocity graph

confirms this pattern. The middle sample is positive; the others are negative. The acceleration features are computed in the same manner. This completes our discussion of energy-based features. Further details may be found in [48].

4.1.3.4 Grid-Based Feature Abstraction

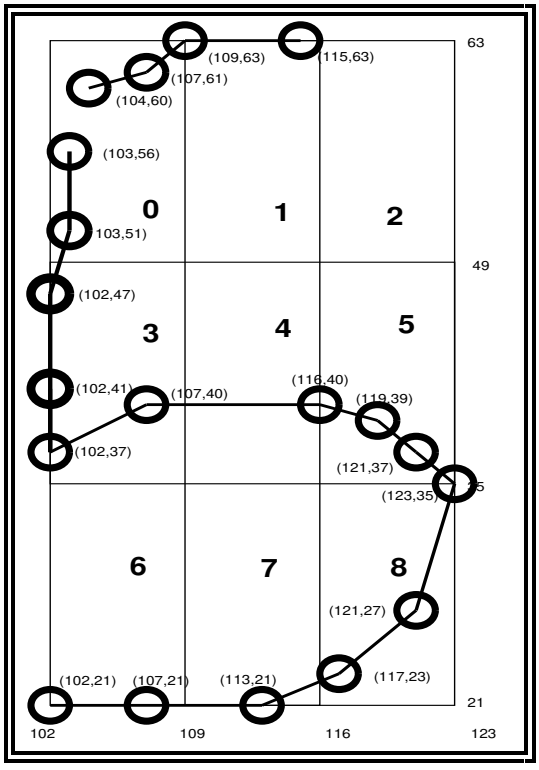

Figure 5. The Numeral *5* in the Feature Grid

For each input scribble, the bounding rectangle is divided into thirds in each dimension, creating nine zones (numbered 0 to 8 horizontally from the top left as seen in Figure 5). For each consecutive pair of points in each stroke (but not crossing stroke boundaries), we calculate the zone into which the majority of the segment falls. Within each zone are four *orientations*: horizontal, right diagonal, vertical, left diagonal. These are determined from the angle β that each segment makes with the horizontal. We determine the arctangent of β (making the y difference positive to limit the angle to the first two quadrants). Then we add the length of the segment to the appropriate orientation sum of this zone:

horizontal: $0 \le \arctan(\beta) < \frac{\pi}{8}$ or $\frac{7\pi}{8} \le \arctan(\beta) \le \pi$

right diagonal: $\frac{\pi}{8} \le \arctan(\beta) < \frac{3\pi}{8}$

vertical: $\frac{3\pi}{8} \leq \arctan(\beta) < \frac{5\pi}{8}$

left diagonal: $\frac{5\pi}{8} \leq \arctan(\beta) < \frac{7\pi}{8}$

When this has been completed for all point pairs, the values are normalized by dividing each of the 36 length sums by the total length of all 36 values.

This completes the algorithms for feature abstraction and for the Preprocessor module. The output of this module, the feature vector X_p is sent to each of the prototypes. We now turn to the algorithms for those prototypes.

4.1.4 Prototype Units

The prototype module consists of a set of N (> 0) *prototype learning units* (n_i). Each prototype unit, n_i ($1 \leq i \leq N$) is a single-category classifier. Indeed, since it limits itself to patterns strongly similar in the feature space, it is more accurate to call each unit a single-*subcategory* classifier. It seeks to identify input patterns that belong to its particular subcategory. When the input pattern X is given, the output value, $n_i(X)$, from unit n_i represents the likelihood of X belonging to the subcategory of the particular unit. Or equivalently, it represents the similarity between the input pattern and the prototype pattern represented by that unit's subcategory. (We use just X to represent the input when its form — scribble or feature vector — is immaterial.) We further assume that the output of each prototype unit is normalized to [-1,1], that is, $-1 \leq n_i(X) \leq 1$.

We have tested five different learning algorithms for the individual prototype units. Four are variants of backpropagation:

(1) Prototypes with no hidden units

(2) Prototypes with independent hidden units for each prototype

(3) Prototypes which share a pool of hidden units

(4) Prototypes which share a recurrent neural network

The fifth learning algorithm is a fuzzy-logic classifier built by Wang [174, 175]. Her results are briefly reported in Chapter 5. The most successful of the five algorithms is the first. It is based on a simple backpropagation neural network (Rumelhart and McClelland [144]) with connections directly from input to output [89]. The definitions that follow relate to either of the first two algorithms. The recurrent version (the fourth of the above algorithms) closely follows Servan-Schreiber, Cleeremans, and McClelland [148]. In this case we omit the grid features and group the velocities and accelerations in time sequence. The third case may be considered a special case of the fourth in which the depth of recurrence is one.

Each learning unit is an acyclic network consisting of 3 layers: typically, 30 to 70 input units (equal to the number of features), 0 to 8 hidden units, and 1 output unit. The input layer is fully connected to both the hidden layer and the output layer. The hidden layer is fully connected to the output layer. There are no lateral connections, that is, no connections within the same layer.

We use the activation and error functions of Kalman and Kwasny [79, 80], namely,

$$a = \lambda(x) = (1 - e^{-x}) / (1 + e^{-x}) \qquad \text{(activation function)}$$

$$\varepsilon = \Sigma (c^2 / (\lambda'(x)) = \Sigma (c^2 / (1 - a^2) \qquad \text{(error function)}$$

where *c* is the difference between the *training value* (1 if correct, -1 if incorrect) and the actual *activation value* of the unit. Since $\lambda(x)$ is precisely *tanh (x/2)*, the output activation of prototype units ranges from -1.0 to 1.0. Finally, if Δw_{ij} is the current change to the individual weight w_{ij} and $\Delta w_{ij}'$ is the last update of this weight, then

$$\Delta w_{ij} = \eta\varepsilon + \alpha\Delta w_{ij}' \qquad \text{(weight update function)}$$

where ε is the *learning rate* [144] and α is the *momentum* [37, 152]. As seen in Figure 3 (The Five Modules of SCL), at the beginning of this section, the prototypes n_i all send their activations $n_i(X_p)$ to the Selector which must arbitrate among them to select a winner and to reset them when necessary. This leads us to the algorithm of the Selector.

4.1.5 Selector

The Selector receives the input of all prototypes and ultimately returns the recognized category C_r to the Interface. It works closely with the Training Scheduler to manage the training episodes of the prototypes. It also generates the confidence associated with the recognition and may offer alternate choices to C_r.

4.1.5.1 Selector Algorithm

The algorithm for the Selector is given in pseudo-code in Figure 6 on the following page. The parameters and variables are explained in the narrative description that follows.

When the input pattern *X* is received, each unit responds with $n_i(X)$ the likelihood that *X* belongs to that unit's subcategory. The unit n_j with the largest output value greater than the *vigilance* ρ, is selected as the winner, and its category name C_j is given to Interface as the recognition output C_r. If the output is correct (Λ never being correct), no prototypes are changed. Otherwise, the system receives the correct category name C_x.

Parameters:	$-1.0 < \sigma_L < \rho < \sigma_H < 1.0$.
Initialization:	$C_i = \Lambda$ $f_i = 0$, $B_i := \varnothing$ (the empty set), for $1 \le i \le N$.
Recognize:	1. The input pattern X_p is sent to the prototypes.
	2. $K := \{ i \mid C_i \ne \Lambda$ and $n_i(X) > \rho \}$.
	3. If $K = \varnothing$ then Produce Λ; Goto 7.
	4. Find j such that $n_j(X) = \max\{ n_i(X) \mid i \in K \}$.
	5. Return $C_r := C_j$ to Interface.
	6. If accepted by Interface, then Goto 1.
Train:	7. Get the correct category name C_x.
	8. $K := \{ i \mid C_i = C_x$ and $n_i(X) > \rho \}$.
	9. If $K = \varnothing$ then Goto 14.
	10. Find j such that $n_j(X) = \max\{n_i(X) \mid i \in K\}$.
Add exemplar:	11. $B_j := B_j \cup \{X\}$; $f_j := f_j + 1$.
	12. $R := \cup B_i \mid C_i \ne C_j$ and $\big[(n_i(X) \ge \sigma_L)$ or $(v \in B_i$ and $n_j(v) \ge \sigma_L)$ or $(v \in B_i$ and $n_i(v) \le \sigma_L)\big]$
	13. Train n_i $(C_i \ne C_j)$ with $\{(u,-1) \mid u \in B_j\}$ until $n_i(u) < \sigma_L$; train n_i $(C_i \ne C_j)$ with $\{(v,1) \mid v \in B_i \cup R\}$ until $n_i(v) > \sigma_H$; train n_j with $\{(v,-1) \mid v \in R\}$ until $n_j(v) < \sigma_L$; train n_j with $\{(u,1) \mid u \in B_j\}$ until $n_j(u) > \sigma_H$; Goto 1.
Add prototype:	14. $K := \{i \mid C_i = \Lambda\}$.
	15. If $K \ne \varnothing$ then select $j \in K$; Goto 17.
	16. Find j such that $f_j = \min\{ f_i \mid 1 \le i \le N \}$.
Initialize prototype:	17. $f_i = 0$; $B_i := \varnothing$; $C_i := C_x$; Goto 11.

Figure 6. Selector Algorithm

If there is no winner, but a unit remains with the empty name Λ, it becomes the winner, thus forming a new prototype, with C_x as its category. If there are no more null units, then the unit with the lowest *frequency count* f_i (the prototype that has won least

often) will become the winner after re-initializing its settings. The winning unit n_j concatenates X to its buffer of exemplars B_j and increments its frequency count f_j.

A set R is formed of exemplars of prototypes from those losing units n_i whose category name is not C_j and which exemplars meet one of three conditions.

- Does n_i respond to the new pattern X too eagerly ($\geq\sigma_L$)?
- Does n_i respond to its own exemplars ($v \in B_i$) too weakly ($\leq \sigma_H$)?
- Does n_j respond too eagerly ($\geq \sigma_L$) to the exemplars of B_i?

The n_i are negatively trained on the patterns in B_j until $n_i(u)$ becomes less than σ_L. If their response to their own exemplars $v \in B_i$ is less than the *high activation value* σ_H , they are positively trained until $n_i(v) > \sigma_H$. Similarly, the winning unit is positively trained on all the patterns in B_j (including X) until the average output value for those patterns becomes greater than σ_H. It is negatively trained on exemplars with other category names until its response is below σ_L. Units sharing the category name C_j are not trained even if they lose the competition.

4.1.5.2 Confidence and Alternate Responses

The Selector algorithm leads to a natural definition of confidence. If the model responded "perfectly" to an input, the appropriate prototype unit would show the highest possible activation, namely 1, and all other units would respond with the lowest activation, -1. We define the *confidence* of a given recognizer's response as the difference between the highest activation for a category and the highest activation for the second-place category divided by this limit of 2. Let r be the recognition response C_r to pattern X. Recall that $n_i(X)$ is the response of prototype unit i to input X.

The confidence *C(X,r)* associated with a recognition *r* to input *X* is defined as:

$$C(X,r) = (n_i(X) - n_j(X)) / 2 \qquad n_i(X) \geq n_j(X)\ \forall^j \mid C_j \neq C_i$$

$$\text{and}\ \ n_j(X) \geq n_k(X)\ \ \forall^k \mid C_k \neq C_j \text{ and } C_k \neq C_i$$

Moreover, we define the first *alternate response* to the input *X* as the C_j found by this algorithm. The second alternate and subsequent alternate responses are defined analogously. These may be returned to the Interface upon request.

This completes the definition of the Selector. The Training Scheduler, the last module of our five, works with the Selector and the Prototype Module to oversee training episodes.

4.1.6 Training Scheduler

The basics of the SCL training scheme were shown in steps 12 and 13 of the Selector algorithm in Figure 6. This process is called *new training* since it occurs when new exemplars are introduced to the model. The Selector calls upon the Training Scheduler to perform it. The Training Scheduler then exercises the prototypes appropriately to achieve the training goals (activations above σ_H and below σ_L). The SCL model involves two variations on this training scheme. When the number of new exemplars reaches a preset limit T_n, the system pauses for *old training* of all prototypes: positively on their own exemplars and negatively on exemplars from prototypes with different category names. This continues for (another limit) T_o training events. Prior to the next round of new training, the buffers of all prototypes are gathered into a common pool *P* and then reset.

$$P = \cup B_i; \qquad B_i = \varnothing \qquad \text{for } 1 \leq i \leq N$$

The prototypes must compete anew following the same Selector algorithm shown in Figure 6 to gather buffers of exemplars B_i from the common pool *P*. This is called the *shuffle*. When the shuffle is complete, new exemplars are again added from the Interface and the process continues as before. These three modes of training are discussed in more detail in the following sections. First we explain the process common to all modes of training — building a pool of trainworthy exemplar-prototype pairs *(v, n_i)*.

4.1.6.1 Building the Pool of Exemplars and Prototypes for Training

This algorithm precedes any training activity. Suppose that a new prototype n_w is being added. Exemplars from the buffers B_i of some number of existing prototypes n_i ($C_i \neq C_w$) will be added to the pool if the prototype meets one of four criteria as mentioned in the Selector algorithm.

- Does n_i respond to the new pattern *X* too eagerly ($\geq\sigma_L$)?
- Does n_i respond to its own exemplars ($v \in B_i$) too weakly ($\leq \sigma_H$)?
- Does n_w respond too eagerly ($\geq \sigma_L$) to the exemplars of B_i?
- Does n_w respond too weakly ($\leq \sigma_H$) to *X*?

If a prototype meets one of these criteria, *all* of its exemplars are included in the training pool. The pool is organized such that half the training — (half the backpropagations in the neural net case) will be applied to the new prototype and half to existing prototypes. Within each half, half again will train positively on a prototype's own exemplar(s) and the other half will train negatively on another prototype's exemplar. Thus the training cycle is divided into fourths.

- n_w trains *up* on its new pattern *X*.
- n_w trains *down* on a randomly chosen exemplar from another category $v \in B_i$.

- A randomly chosen existing prototype n_i trains *up* on its own randomly chosen exemplar $v \in B_i$.
- The same existing prototype n_i trains *down* on X.

If the prototypes do not show sufficient progress towards responding above σ_H on their exemplars and below σ_L on exemplars from other categories, the training round will be summarily terminated at some maximum number of backpropagations (typically 1,000 to 10,000).

The proportion of prototypes that meet one of the four criteria for inclusion in the training pool depends on the "maturity" of the recognizer. In new recognizers, 90% or more of the prototypes may be included at first, but after several rounds of training, many no longer need to be included in the pool. The training pool is typically refilled every 2,000 backpropagations, getting smaller as training progresses. If a new prototype is added to an existing well-trained recognizer, the proportion of inclusion may be much smaller, perhaps 10% or so. Some experiments have allowed the pool to continue declining until virtually no candidates for training (meeting the above criteria) are available, but this tends to result in *overtraining* — specialization to the training data to the detriment of performance on general inputs.

There is another algorithm available for filling the training pool. Earlier in this chapter (page 63), we describe the confidence associated with a recognition. In a variation on the above-described Selector algorithm, prototypes may be included based on confidence. If a prototype's exemplars are recognized above this threshold level of confidence (on average), this prototype's exemplars are excluded from the training pool (unless used negatively by other prototypes, of course); otherwise, its exemplars are included.

4.1.6.2 New Training — Creating New Prototypes

The *new training* function follows the selector algorithm delineated above, but some specifics should be noted. The number of exemplars in a buffer is typically limited to 10 to 40. The number of prototypes is limited according to the recognition domain: 20 to 100 for digits, 50 to 150 for lowercase letters, 100 for gestures, 100 to 300 for combined recognizers (digits, lower and uppercase letters). If this limit has been reached, the least frequently used prototype unit is replaced by this new one. When incoming scribbles are coming from a file, they may be presented randomly or in file order as dictated by a parameter.

4.1.6.3 Old Training — Reinforcing Existing Prototypes

After a round of new training, it is useful to conduct a round of *old training* to reinforce all prototypes' response to their newly acquired buffer of exemplars. Although this training may use the same selection mechanisms described above (selection by σ_H, σ_L, or confidence), it usually is conducted across all units of the current recognizer. The cycle of training alternates between positive and negative training as described above.

4.1.6.4 Shuffling Exemplars

The shuffle, defined above, is introduced to overcome the difficulties of training a recognizer from scratch — from a state of no prototypes and no exemplars. Its earliest choices may result in rather odd groupings of exemplars. The shuffle removes the exemplars (all of them, or some fraction of them in descending order of the strength of the prototypes' response to them). If the exemplars are fully removed, the prototypes now compete in a new round of exemplar-gathering (always in random sequence through the set of training data). Prototypes which gather exemplars fewer than some minimum (such as 1 or 2) are dismissed; these usually were over-specialized to an exemplar which is statistically an *outlier* — a pattern several standard deviations removed from the norm. The

remaining prototypes generally gather a better coordinated set of exemplars after a shuffle, but this is examined more deeply in the next chapter.

This completes the definition of the Training Scheduler. It also completes the algorithmic definition of the SCL *model*. We now discuss its realization as the SCL *system* following the same order through the five modules.

4.2 Implementation

The basic implementation of the SCL *system* is realized in about 12,000 lines of ANSI C code (about 215 functions), but we have tested SCL in a wide variety of research settings, and on different platforms. Nearly as many lines of code have been created to implement these related activities. SCL has been implemented in formal application program interfaces (API's) for Sun Solaris, NeXT Interface Builder, and GO PenPoint systems. It has been informally implemented with command-line menu interfaces. Other members of the Kumon team have built versions of SCL in the Niklaus Wirth-derived ET++ and in X-Windows. Some versions use a mouse, and many naturally use a pen. About a dozen other researchers have worked with us in these broadening explorations. More will said about their work in Chapter 5. First we present the fundamental data structures that underlie the SCL system.

4.2.1 Data Elements

We present the principal data representations employed by the SCL system to clarify the implementation of SCL. Details of these and other implementation methods may be found in the *SCL System Manual* [49].

4.2.1.1 Scribble

Along with stroke and point data, it is convenient to record with each scribble its bounding rectangle, an identifying tag number (unique across our scribble inventory), an assignment to a recognizer (defined below) and prototype, and an assigned value (the scribble's category name, usually the ASCII value for the given or presumed character or gesture). Further, we store within each stroke the point data and the bounding rectangle for that stroke. The data is captured by the x and y movement of the pen on a digitizing tablet. For example, the raw data for the five scribble shown in Figure 5 above on page 58 is sampled by the digitizing tablet as shown here in Figure 7.

```
108 53 2
15   103,56 103,51 102,47 102,41 102,37 107,40 116,40 119,39
     121,37 123,35 121,27 117,23 113,21 107,21 102,21
4    104,60 107,61 109,63 115,63
```

Figure 7. Raw Scribble Data for 5

This scribble has the tag 108, value 53 (ASCII '5'), and consists of 2 strokes. The first stroke consists of 15 points (the first of which is `(103,56)`) and the second stroke of 4 points (the first of which is `(104,60)`). The coordinates may be collected in local window reference frames or in full screen reference frames; thus no significance can be ascribed to the absolute magnitudes of the coordinates. Indeed even the coordinate density per unit length is not fixed between digitizers, so we cannot associate a particular physical size with a scribble. This makes the discrimination between some uppercase and lowercase letters (*viz. C, O, P, S, V, W, X*, and *Z*) problematic.

4.2.1.2 Features and Patterns

The *feature* data structure collects features of scribbles. It depends on the *feature environment* for data necessary to define the current filter and feature abstraction

processes. Every *recognizer* (a collection of prototype learning units defined below) thus knows its own feature environment. The *pattern* is a data structure to represent the features once they have been transformed into real-valued inputs; it transmits X_p to the prototypes.

4.2.1.3 Feature Environment

The *feature environment* defines parameters for training and recognizing. It defines filters and feature abstractors to be applied to the input stream. It defines the recognition domain (numbers, two-stroke numbers, lower-case letters, and so forth).

4.2.1.4 Prototype Learning Unit

Each prototype learning unit n_i (usually just called prototype) allocates sufficient memory to retain the real-valued weights for its own learning algorithm. (In the usual case, artificial neural networks are the learning elements.) The *trainable* attribute identifies prototypes that will receive further training. The *value* is the category name C_i — usually the ASCII value of the category. The prototype records f_i the number of times during its history that it has won the competition. This is used when it is necessary to delete the least-used prototype. Each prototype retains a buffer B_i of exemplars used while training. Each prototype not only retains its own weights w_{ij}, but also the most recent changes to those weights Δw_{ij} to be used during backpropagation in conjunction with the *momentum* term (defined in section 4.1.4).

When fuzzy logic is employed as the learning unit, the prototype is simpler. No weight data nor exemplar buffer is retained. The prototype only maintains a rolling computation of the average membership attributes. Further details on the fuzzy logic version of SCL prototypes are in [174, 175].

4.2.1.5 Recognizer

The *recognizer* is the most complex data structure of the SCL system. The prototype described in the last section instantiates the Prototype Units' algorithm; the recognizer instantiates the Selector's algorithm. Since it is at the crossroads of the SCL system, it contains data that connects to all five modules. The recognizer data structure has pointers to the collection of prototypes, and to the filters and feature abstractors of the Preprocessor. It contains the training parameters: ε, α, σ_H, σ_L, and ρ which are passed to the Training Scheduler during *new training* and *old training*. The recognizer is loaded from a parameter file that is described below in section 4 (Recognizer Parameters).

When the recognizer is to be trained from a file of scribbles, a single file of input scribbles is opened. Two arrays of integers store indices to this unified set of patterns for the *training set* and *testing set*. This simplifies cross-validation (explained below). Each recognizer may optionally retain information regarding its historical performance. Some is included directly in the recognizer such as the historic number of backpropagations. Specific performance statistics are retained in the *statistics* structure and may be saved to and retrieved from disk.

Some support modules (notably Viewer) require the summarization of statistical data over a number of scribbles. The *outside statistics* structure furnishes a record into which such external data as average and standard deviation of scribble properties are accumulated. The data are retrieved and updated by the Viewer module.

4.2.1.6 Recognition Event

The recognition event stores information about the last recognition performed by the system. This data structure is used by the Interface during on-line recognition and training. If the user requests training, the recognition event contains the name of the

recognizer to train and which scribble to use for the training. It also receives alternate category names from the Selector. The Interface may present these choices to the user to facilitate the use of an alternate character if the first choice is wrong.

4.2.1.7 Recognizer Parameters

The following parameters are used by the recognizer to modify its operations. This data is stored in an external text file to permit easy manipulation. These parameters are gathered below into natural groups (just as they are stored) to elucidate their purpose.

```
Recognizer domain:        SCL_NUM
Network architecture:       noHid
Number of inputs:              70
Hidden units per prot:          0
Maximum prototypes:           100
Target categories:             10
Lowest ASCII value:            48
```

The choices for recognizer domain are: `SCL_NUM, SCL_LET, SCL_GES, SCL_NUMLET, SCL_NUMLET_ADJ, SCL_TWOSEG, SCL_LOWER, SCL_UPPER, SCL_UP_LOW,` and `SCL_SPECIAL`. Only four are routinely used:

SCL_NUM : digits

SCL_LET : lowercase alphabetic characters

SCL_NUMLET : both upper and lowercase letters and numbers

SCL_NUMLET_ADJ : upper and lowercase letters and numbers,
adjusted for ambiguous letters.

The choices for network architecture (explained in 4.1.4 above on page 59) are: `noHid, protHid, sharedHid`, and `recurHid`. The number of inputs must agree with the feature abstractor used. `Hidden units per prot` is just that. If more than

the `maximum prototypes` are called for, the system will begin replacing prototypes. The maximum number of replacements is governed by the parameter with just that name lower in the parameter file. `Target categories` is the number of category labels (10 for digits, 62 for uppercase, lowercase, and digits). The lowest ASCII value for digits is 48, for uppercase letters 65, and for lowercase letters 97.

```
Output to console:           yes
Output to file:               no
Output scribble data:         no
Output training data:         no
Reset trainable protos:       no
Output file prefix:            N
```

The first four of these parameters control the output of SCL. If `output to file` is chosen, SCL composes a unique file name from the prefix and the current day, hour, and minute. The flag to enable the resetting of prototypes allows a recognizer to change the weights of existing prototypes. If set to `no`, the prototype simply adds new prototypes and does not retrain the old ones.

```
Name used in reports:    Number_Recognizer
Pattern file name:       ppn_samp.bff
Prototype file name:     ppn.bpf
Statistics file name:    ppn.st
Scribble file suffix (bff, bsf, chk, pch, scr): bff
Weight file suffix (bpf, chk, txt):             bpf
```

These parameters give the names of needed files and what kind of files to request. The report name is used in error messages, warning messages, and in some statistical reports. There are a number of formats in which SCL data may be stored (for example, a text file (.txt) of weights, or a binary feature file (.bff) of scribbles). The suffixes indicate the file type.

```
Incremental train interval:    2
Incremental train amount:  10000
Maximum prototypes replaced: 100
Max bprops for new prot:    1000
Bprops for oldTrain:      200000
Bprops for post-shuffle: 1000000
Minimum exemplars to save:     2
```

This collection of parameters controls the training functions described in section 4.1.6 above. The first two determine how often a recognizer training from scratch pauses to refresh all prototypes with a round of old training and how many backpropagations of old training will be performed. `Maximum prototypes replaced` was explained above. `Max bprops for new prot` limits the training when a prot is first formed. `Bprops for oldTrain` is used when the old training is called after new training. `Bprops for post-shuffle` refers to the training done after a shuffle of exemplars. Finally, the last number of this series controls how many of the prototypes will be saved to disk. In this case, if the prototype has not garnered at least 2 exemplars, it is considered too specialized to deserve saving; only prototypes with 2 or more exemplars in their buffer will be saved to disk.

```
Sigma low and high:   -0.300  0.700
Vigilance:                 0.000
Learning rate:             0.00100
Momentum:                  0.900
```

These parameters control the Selector. They are explained in the Selector algorithm section (4.1.5.1) above on page 61.

```
Feature environment: Smoothing?:  yes
   Smoothing ratio:               3/4
   Smallest stroke:                 2
   Smoothing function:            twoSmooth
      Shear?:                       yes
      Shear before smooth?:          no
   Load scrib to pat function:    loadScribToPatternX
   Straightness threshold:        0.900
   Horizontalness threshold:      0.500
   Crosser stroke tapering:       0.300
   Average scribble height:        85
```

These parameters control the feature environment. They indicate whether smoothing will be performed in incoming scribbles (explained in the Preprocessor section). Then appear the `smoothing ratio`, the `smallest stroke` (strokes with 2 or fewer points are discarded), and the particular smoothing function called. `Shear` indicates whether shear is called, and if so whether before or after smoothing. There may be any number of functions which specifically load the incoming scribbles into the pattern structure shown above. The parameter `Load scrib to pat function` indicates which one this feature environment expects. The last four parameters are used to determine the presence of crossing strokes. The last parameter is not currently used. It was intended to help discriminate between upper and lowercase letters (especially *C, O, P, S, V, W, X,* and *Z*) but we have not yet collected appropriate sample data to use it.

This concludes the key data structures of the SCL system, and leads to the first of the five modules, the Interface.

4.2.2 Interface

We have created several different Interfaces to the SCL system during the past three years. Each of these interfaces calls a common pool of functions, but presents a quite different view of SCL to the user. In the next five sections (4.2.2.1 through 4.2.2.5), we briefly describe five separate, independent interfaces to the SCL system.

4.2.2.1 Menu-Based Interface

The simplest interface to SCL (for the programmer) simply presents a command-line menu and permits SCL operations on *existing* scribble databases. This interface precludes direct scribble entry; separate programs exist for batch entry of scribbles. This interface does permit loading and saving recognizers, training, testing, and displaying data

structures. This interface runs on UNIX workstations like the HP 720, NeXTstation, and SPARCstation 4, and on DOS systems.

4.2.2.2 Automated Training and Testing Interface

This interface permits the basic commands of the SCL system (file loading, training, testing, and so forth) to be automated. Pre-recorded training scripts may be conducted in an unattended mode. This is especially valuable for reproducing exact sequences of training and testing for cross-validation analysis (explained below). It also permits the testing of particular parameters by fixing other variables while allowing the test variables to range over the pre-determined suite of values. This too runs in both UNIX and DOS environments.

4.2.2.3 Viewer Interface — Visualizing and Gathering Statistics on Scribbles

Scribbles are surprisingly diverse. We occasionally need a detailed examination of the data. For example, what is the average number of points in a scribble? What is the average gap between a lowercase *i* and its dot? How many points are sampled in a dot? What are the minimum and maximum point counts for dots? What is the standard deviation? These questions are answered by the Viewer, a GUI interface to the SCL system written in NeXT Interface Builder [125] (about 1,000 lines of Objective C). As its name suggests, it also allows the viewing of specific scribbles or collections of scribbles in Display PostScript from which they may be readily transferred to a PostScript-capable printer. The scribble illustrations in this document were rendered in PostScript either by Viewer or by HCR described in the next section.

4.2.2.4 HCR Interface — Managing the Data

Scribbles may exist in a variety of formats over their research lifetimes: raw sampled data, normalized, filtered, loaded into pattern arrays, sorted in various ways, and in the new standard formulated by the Unipen consortium mentioned in Chapter 3. The HCR

(Handwritten Character Recognition) interface principally serves to manage our inventory of scribbles although it too may serve as a GUI interface to SCL. It allows hand-correction of scribble category names, and can convert among the above-named formats. It also presents an illustration of the scribble as a collection of points, a series of connected line segments, and in the velocity and acceleration domains. Figure 14 on page 86 shows the HCR interface panel (also built through NeXT Interface Builder: about 2,500 lines of Objective C). Details of HCR are in [45, 46].

HCR has extensions to display immediate recognition, velocities, accelerations, points, and training statistics on scribbles entered from an on-line digitizer. A Lombart pen-based notebook with digitizing tablet, monochrome display, and Intel 80286 CPU may be connected to the serial port of a NeXTstation to run this modified version of HCR.

4.2.2.5 PenPoint Interface

The most extensive interface to SCL is in the PenPoint API from GO Corporation. SCL is implemented in PenPoint as a new subclass of handwriting recognition engines: *clsSCL_Xlate*. About 2,500 lines of PenPoint C were devoted to this interface. The second and third prototype Kumon Machines both use this API for their handwriting recognizer. A test shell (shown in Figure 8 below) was also constructed within PenPoint for on-line interactive testing of the various recognizers. In this figure, the input *7* has not yet been recognized since the pen is still down. The last digit recognized was *6*. The top row of buttons shows the alternates to the recognition of *6* (namely *5, 4, 0, 8*).

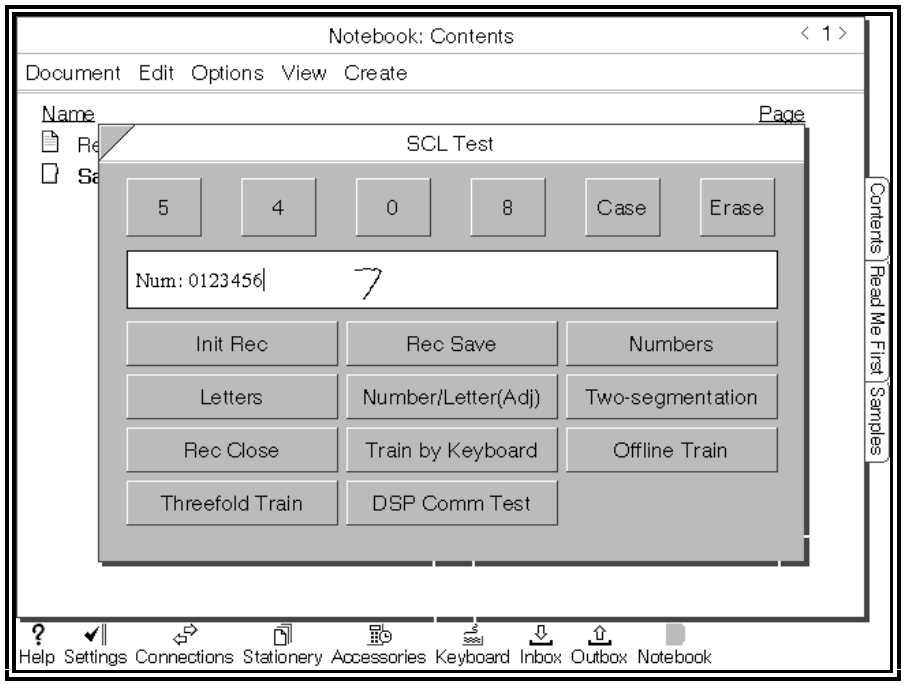

Figure 8. SCL Test in PenPoint

This concludes our discussion of the implementation of the five independent user interfaces to the SCL system. Proceeding in the rough order of data flow (as we did in the SCL model above), we now consider the implementation of the Preprocessor module.

4.2.3 Preprocessor

The Preprocessor transforms the incoming scribble into a feature vector X_p for input into the prototype units. As defined above, this module is divided into filters (smoothing and shearing) and feature abstraction (based on energy and grid). In this section we discuss the implementation of both.

4.2.3.1 Segmenting a Stream of Strokes into Individual Scribbles

The issue of segmentation of the input stream arises for pedagogical reasons. The two general approaches to segmentation are spatial and temporal. The writing surface may be broken into input boxes or zones. If a collection of strokes are within a box or

separated by some distance, they are assumed to be part of the same scribble. Similarly, the strokes may be separated by a time-out (the PenPoint default is 600 ms). Spatial separation tends to fail in the Kumon setting for two reasons: (1) young children overlap strokes from adjacent digits and leave unwieldy gaps within a digit, (2) if boxes are used to compartmentalize the answer, they either give information to the learner if a box is provided for each digit of the answer, or they mislead and confuse the learner if some standard box count is provided. The temporal delay interferes with the fluid flow of pen through the exercise which is at the heart of the Kumon method.

The problem is simplified for digits since nearly all writers use no more than two strokes per digit. Some writers use only one stroke for all 10 digits. Many writers write *4* and *5* with two strokes. Some writers in our database of scribbles write *1*, *7*, *8*, and *9* with two strokes. As explained earlier, our first effort at segmenting digit scribbles involves recognizing the first stroke of each two-stroke scribble as such and then discarding the second stroke. Our second solution addresses a subset of the problem by addressing input that consists of no more than two digits. This function accepts as input a multi-scribble that may include one or two scribbles. It returns its best estimate as to the first and second actual scribble based on the algorithm described in section 4.1.2.2.

4.2.3.2 Normalization and Smoothing of Scribbles

Since the on-line system is vulnerable to the vagaries of the user, some safeguards are required. Input is limited to 1000 points per stroke and 10 strokes per scribble. These are comfortably above the largest inputs we have observed in the collection of more than 24,000 scribbles. Any points or strokes beyond these bounds would be discarded. Furthermore, the feature computation would be quite unwieldy for 10,000 points. Besides, our experiments indicate that no significant information is gained beyond 60 points of

data. The system performs a stroke-wise normalization of points to a maximum of 60 for the whole scribble by selecting points at a constant interval throughout the strokes.

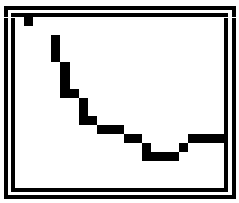

Figure 9. Magnified Scribble

It was also found helpful to use exponential smoothing to smooth out anomalies in the digitized input. Many handwritten digits display a surprising irregularity when examined closely. This is caused to some degree by the nature of the digital sampling. Envision the drawing of a zero under a microscope. Though moving, the pen may be sampled at the same coordinate for two samples, then move just enough upward to change to the next (integer) y coordinate, but not the next x coordinate, and so on. Figure 9 shows the actual curve of a zero scribble magnified several times. The stairstep nature of the individual segments introduces "noise" into any feature abstraction. This is especially detrimental to the grid-based features. All the segments in this figure would be horizontal and vertical.

This may be overcome by applying some variant of weighted averaging to the stream of sampled points. Three methods for smoothing were defined in section 4.1.3.1 on page 51 above. Figure 10 shows the effect of simple exponential smoothing with progressively larger smoothing constants on typical scribbles. The top row shows the unprocessed scribbles. The second shows the effect of smoothing with *e* of one half; the third and fourth rows show progressively larger *e*.

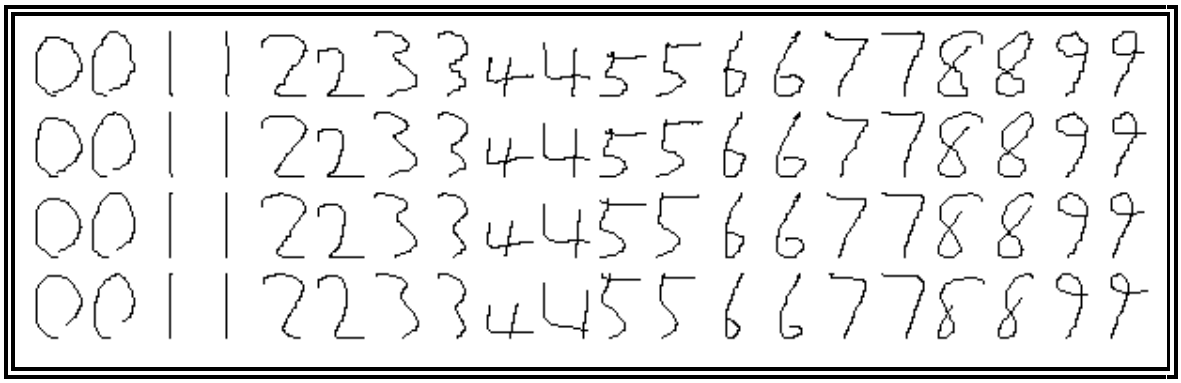

Figure 10. Simple Smoothing: *e* = 0, 1/2, 3/4, 7/8

Although simple exponential smoothing does reduce the stairstep noise introduced by the digitizer, it has the undesirable side-effect of causing some distortion of its own. The larger values of *e* result in smoother strokes, but tend to pull the stroke toward the initial point as is evident in the bottom line of Figure 10 — illustrating a smoothing constant of 7/8. One way to mitigate this effect is bi-directional smoothing (defined in section 4.1.3.1): smooth the points as above and then smooth them again backwards and average the two results for each point. This is illustrated in Figure 11.

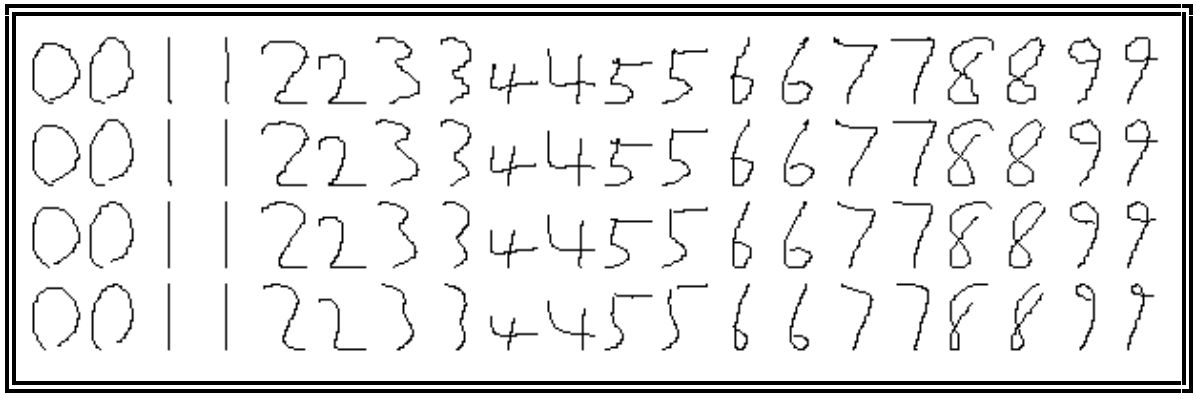

Figure 11. Bi-directional Smoothing: *e* = 0, 1/2, 3/4, 7/8

Figure 11 illustrates the balanced effects of bi-directional smoothing, but again, this improvement comes at a price: the system now tends to pull the points toward the two

ends of the stroke thus flattening scribbles like the *6* and *9* in the above figure. Symmetric smoothing was introduced to correct this distortion. As explained above, bi-directional smoothing takes the endpoints as *anchor* points in the smoothing algorithm; symmetric smoothing take each point as its own anchor, and averages its coordinates with those of points chosen symmetrically before and after it. It is shown in Figure 12

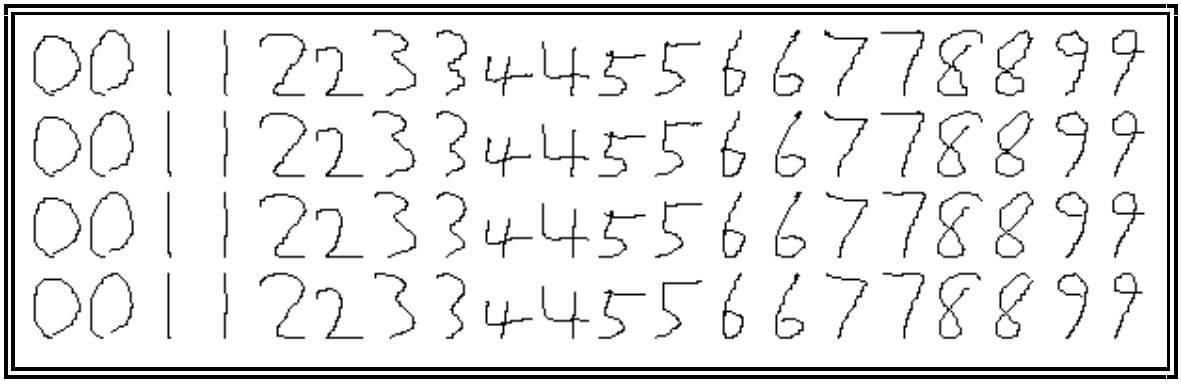

Figure 12. Symmetric Smoothing: *e* = 0, 1/2, 3/4, 7/8

Having finished with smoothing, we consider the next class of scribble filters, the shear transformation.

4.2.3.3 Shear Transformations

The algorithm is implemented as described in section 4.1.3.2. The top line of Figure 13 shows a series of digits as written; the bottom line illustrates the application of the right-sided shear transformation to the digits.

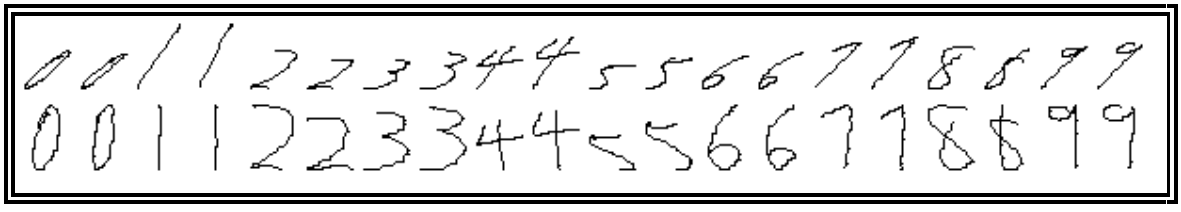

Figure 13. Shear Transformation

4.2.3.4 Stroke Filters

Special-purpose Preprocessor filters may eliminate strokes smaller than a particular threshold (such as 3 points). Other filters may replace strokes that serve a narrowly-defined purpose (dots for *i*'s and *j*'s and crossing strokes for *5*'s, *7*'s, *t*'s, and *f*'s) with some direct representation of that purpose. The use of such filters is controlled by feature environment parameter files. In some cases, the filter makes a final "decision" regarding the presence of, for example, a dot stroke. It then passes this feature to the feature abstractor. In other cases, the filter uses fuzzy membership attributes (Zadeh [184]) to add an additional feature to the feature set.

An example of the latter is the family of crossing strokes (high, middle, low). We use linear regression [40:337,355] to detect the presence of horizontal crossing strokes. The three categories of high, middle, and low overlap somewhat as they do among human writers. We permit a (typical) zone of membership inclusion that spans 30% of the height of the scribble. This gives rise to small overlapping zones from 70% to 80% of the height of the scribble where a crossing stroke may be regarded as both a high crossing stroke and a middle crossing stroke. For example, if a horizontal stroke of a *t* crosses its central stroke at the very top, the scribble has 1.00 high-crosser attribute and none of the others. If however it crosses the central stroke at 78% of its height, the scribble has a high-crosser attribute of .27 and a middle-crosser attribute of .07. Thus the ambiguity is left to be resolved by the prototype units. Currently only the three crosser attributes use this fuzzy filter algorithm. This filter simplifies the shear transformation by reducing the number of strokes to be sheared and by recording and eliminating crossing strokes and dot strokes before calculating the slant of the remaining strokes.

In the realm of gesture recognition, some members of the target set would be better recognized algorithmically than by a neural net. In such cases, the Preprocessor would filter out gestures like single and multiple taps and bypass the prototypes and Selector completely. This capability could easily be added to SCL, but is outside the scope of the current research.

4.2.3.5 Energy-Based Feature Abstraction

The average velocity and acceleration algorithm defined above on page 54 assumes that the point sampling period of the input digitizer is constant over time. The speed of sampling and the constancy of the sampling interval depend on the digitizer and the firmware that drives it. We used either Lombart, Wacom or Scriptel digitizer in most of our experiments. We typically used device drivers written by other members of the Kumon team. In practice, the sampling rates remain more or less constant (192/sec for Wacom, 200/sec for the Scriptel). The slight variations do not materially affect the algorithm's success due to the averaging.

The selection of the velocity and acceleration sampling intervals will not necessarily coincide with scribble's stroke boundaries. For example, suppose we divide the points of a snowman eight into seven velocity sampling intervals. It may happen that the first three and a half intervals fall into the upper stroke, and the last three and a half intervals fall into the lower stroke. This means that the fourth of the seven intervals spans both strokes. This could possibly even be useful in making the "virtual" stroke (between the end of one stroke and the beginning of the next) susceptible to the usual measures of velocity and acceleration. Indeed, as shown in Chapter 5, this assumption yielded satisfactory results. However, the happenstance nature of the placement of the break between strokes (for example, in the snowman eight) introduces an unnecessary oddity in the input feature

vector. Therefore we adjust the sampling interval to stretch to the nearest stroke boundary. We would let the first four intervals fall into the upper stroke and the last three fall cleanly into the lower stroke. This results in both improved performance of the recognizers and led to reduced complexity (fewer hidden units) in the prototype units.

Figure 14 illustrates the velocity and acceleration associated with a *7* scribble as seen in the HCR interface. The upper left display shows the actual *points* contained in the scribble. The next display shows these same points *connected*. The two graphical displays at the top of the right side show plots of the *x coordinate versus time* and the *y coordinate versus time*. In all six graphs on the right side, time is the horizontal axis. The middle two graphs show the *x* and *y velocities* and the pair below shows the *x* and *y accelerations*. In the velocity and acceleration graphs, the zero line is shown at the center of the graph in light gray. This is the raw (unfiltered) scribble data.

The *Character Data* column shows that this scribble has a *Tag* number of 21076 (its serial number in our data base). It is from a file (shown in a separate field below) containing 3200 scribbles. It has a *Value* of 7 and is assigned to *Prototype* number 999 (which means that it has not been claimed by any prototype). The *Tag Base* and *ASCII Base* are used when loading scribbles without tags and values.

The *X Data* column gives the *Max*imum and *Min*imum *Pos*itions of x coordinates, the *Max*imum and *Min*imum x *Veloc*ities, the *Max*imum and *Min*imum x *Accel*erations, and the *Extent* of x (a feature no longer used by SCL). The *Y Data* column give corresponding information for the set of y coordinates.

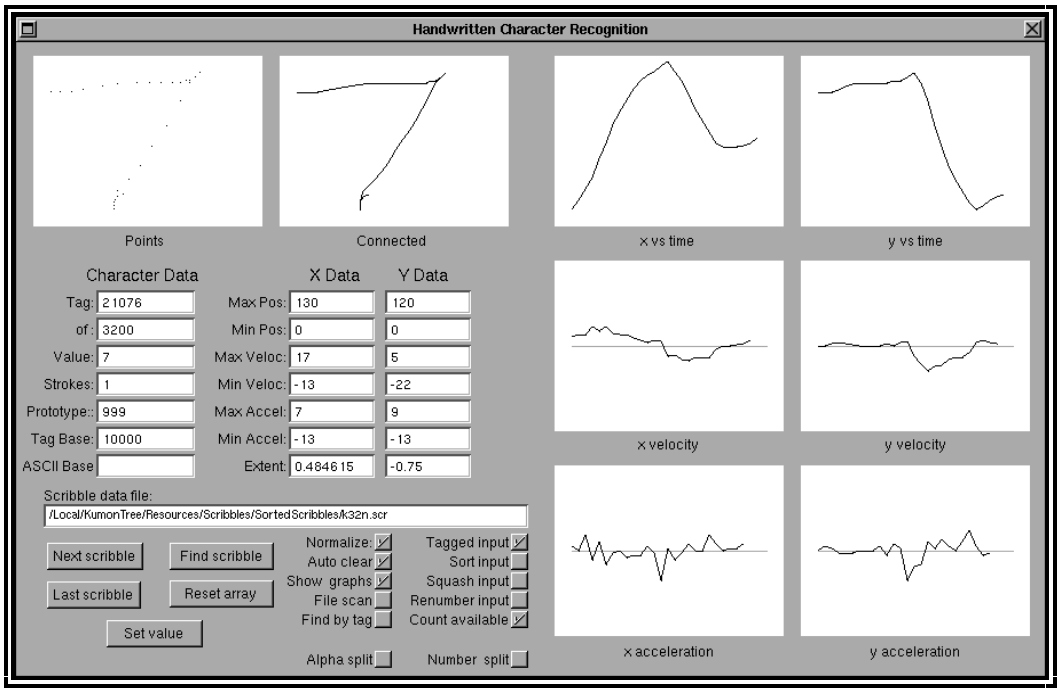

Figure 14. Velocity and Acceleration Viewed in HCR

4.2.3.6 Grid-Based Feature Abstraction

Energy-based features give respectable recognition performance in many settings such as digits, but are not fully satisfactory for all recognition domains. The diversity of handwritten upper and lowercase letters probably surpasses that of numerals. (The section on data quality illustrates several ambiguous examples.) In view of these difficulties, we also include graphical features. Several families of features were tried including grid--based orientation features and angular position, velocity, and acceleration. After numerous experiments, we settled on a simple grid-based scheme suggested by Grothe [62] as defined earlier.

This completes the description of the Preprocessor. We now turn to the recipient of the Preprocessor's feature vector, the prototype learning units.

4.2.4 Prototype Units

As mentioned in the model section on page 59, SCL supports four different neural network architectures. One of these four is selected by parameters in the externally-maintained parameter file described on page 72. The learning rate η was empirically determined to range from 0.0005 to 0.001. Similarly, the momentum α ranges from 0.3 to 0.9. Momentum tends to let neural networks escape from local minima by emphasizing the general direction of weight change and thus overriding the effect of smaller features in the error function terrain. (Cf. [9], [37], [77:299], and [152].)

We worked with Shmatikov [151] to develop a hierarchy of trainable recognizers, thus combining the best of each in the recognition of incoming characters. This platform also permits direct comparisons of the effectiveness of various families of features. More is said in Chapter 5 about this extension to SCL. This concludes our discussion of the implementation of prototypes and leads to the consideration of the Selector.

4.2.5 Selector

In support of the Selector, the system must maintain data necessary to track the responses from the prototypes and to oversee their competition. The Selector is instantiated in the recognizer data structure described above on page 71. The parameters that govern the variations in Selector implementation are explained in section 4.2.1.7 on page 72.

4.2.6 Training Scheduler

The implementation of the SCL training regimen follows precisely the model described above on page 64. It is implemented through functions to build the training pool, perform new training, old training, and the shuffle of exemplars. The Training Scheduler is the last of the five modules of the SCL system. Our final section discusses the experimental methodologies in preparation for the presentation of experimental results in Chapter 5.

4.3 Experimental Methods

The two prior sections elaborated the SCL model and its implementation as the SCL system. This final section reviews several elements of our experimental methodology. Then in Chapter 5, we report the results of these experiments.

4.3.1 Scribble Database

Our experimental methodology begins with gathering and cataloguing scribbles. These are the raw materials for the creation, training, and testing of SCL recognizers. In the following sections, we address the issues of collection, quality, and scope of the scribbles used in our experiments.

4.3.1.1 Collection

Scribbles were collected in three modes: *unrecognized batch*, *recognized batch*, and *passively* during a task. The first mode made no effort to recognize the scribbles, but merely stored them for later processing. The subjects typically wrote pre-determined sentences or strings of numerals, letters, or gestures either sequentially through the target set (0-9, 0-9, 0-9, etc.), or by target element (0,0,0,0..., 1,1,1,1,..., and so on).

In the recognized batch mode, the scribbles were recognized as entered, and training might or might not have been conducted at the option of the researcher or subject. When SCL misrecognized a character, the user could have requested a virtual keyboard as shown in Figure 15 below, and entered the intended character to initiate a brief round of training. SCL presented *2, 5, 4* and *0* as alternatives (on the top row of buttons) to the *7* just recognized. The user may have tapped one of these buttons instead of requesting the virtual keyboard. Scribbles so collected are susceptible of unconscious (or conscious) adaptation on the part of the subject to improve the rate of recognition. Unrecognized batch collections are not vulnerable to this hidden and unwanted influence.

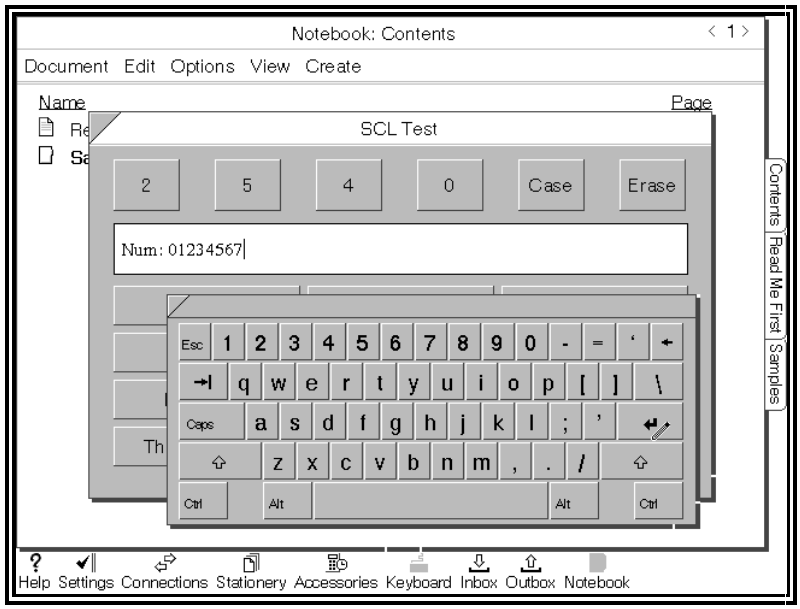

Figure 15. Virtual Keyboard to Correct a Misrecognition

Thirdly, scribbles were collected passively as subjects completed a Kumon mathematics worksheet. These scribbles are available for later training and/or testing. During such tasks, mistakes on the part of the recognizer were usually corrected by the subject (sometimes by the researcher), but the recognizer was not allowed to train on the scribbles incorrectly identified.

4.3.1.2 Quality

The collected scribbles were all verified by examination through Viewer and HCR. Nonetheless, human writers exhibit ambiguities that limit even human context-free recognition to about 95% [122]. The following characters (each identified by the unique serial numbers or *tags* assigned by the HCR program when collected) illustrate the natural ambiguity of human writers.

Character 2128 is actually a *c*, but could be mistaken for *l* or *e*. Compare it with character 5147 which is an *e*. In the pair of characters, the first, 1238,

which in an *n* strongly resembles the second, 3179, which is an *h*. Figure 16 portrays an English word assembled of four lowercase letters.

Figure 16. An English Word Composed of Isolated Characters

These are the characters 5168 (*g*), 5108 (*a*), 5170 (*g*), and 1289 (*s*). They spell "gags." Although many letters are of much better quality, there are some letter pairs that are consistently cross-identified. An extended *a* such as character 5103 may be mistaken for a *q*. An open style of *u* such as character 4308 resembles an *n*. Finally, an open style of *r* such as character 6280 is easily misidentified by people or processors as a *v*.

Although digit recognition has a smaller target set than letters, digit scribbles are also susceptible of ambiguities. This *4* is quite similar to these *9*'s . This collection of *6*'s is close to some of these *0*'s . This zero is not far from *6*. On the other hand, these zeros are not far from these eights . Note that half of the four snowman eights are made with one stroke and the other half with two strokes although all are from the same writer during the same session. These ones from three writers could pass for *2*, *6* and *7* respectively. These *4*'s are not unlike this *6* .

When the recognition domain spans digits and letters, additional ambiguities arise. This *2* could be interpreted as the letter *z*, and this *7* could pass for *2* or *z*. These *3*'s resemble lowercase *f* and this collection of *4*'s resemble *k*'s. Either of these *9*'s could pass for *g*'s.

Other experimental circumstances can reduce the quality of the scribbles sent to the system for recognition and training. Sometimes younger writers try to "touch up" scribbles as in and . Some writers redo the digit: . Sometimes the scribble is incompletely received by the digitizer (perhaps written partially outside the collection zone) such as this *5* , or this *2* which could pass for an uppercase *L*.

Humans instinctively compensate for such ambiguity by contextual clues. Commercial recognition software does so as well by the use of Viterbi maps and dictionary searches, but these aids are outside the scope of our problem. In mathematical tasks, each number or letter must be identified strictly on its own merits. Context support could easily be added to SCL later for use in other settings.

4.3.1.3 Scope

During our research, we retained about 24,000 scribbles from more than a hundred subjects. Only the earliest 2,800 were collected on the Lombart digitizer; the rest were collected on Scriptel, Wacom, and Mitsubishi Amity digitizers. Table 1 shows the scope of our scribble inventory. The gestures were collected from children by Taysi as part of his research in gesture design and evaluation [164]. The domain category of *mixed* indicates that a combination of uppercase, lowercase and digits were collected (usually for use with a single recognizer that handles all three domains). The 1,700 digit scribbles (Scriptel) and the 4,470 digits scribbles (Scriptel, Amity) were collected primarily from young Kumon students from 5 to 13 years old. The rest were collected from high school students, college students, and those beyond school age.

Table 1. Scribble Collection

Domain	Digitizer	Mode	Scribbles	Subjects
digit	Lombart	unrecognized	2,800	40
digit	Wacom	unrecognized	1,500	15
digit	Scriptel	unrecognized	500	5
digit	Scriptel	unrecognized	1,200	12
digit	Scriptel,Amity	passive	4,470	22
lowercase	Wacom	unrecognized	3,900	15
uppercase	Scriptel	unrecognized	1,560	6
mixed	Scriptel,Amity	unrecognized	1,648	8
mixed	Scriptel	recognized	1,200	10
gestures	Wacom	unrecognized	5,460	16
Totals			24,238	149

4.3.2 Testing on Sample Data

In the earliest experiments, we divided the data into two files. About two-thirds of the collected scribbles within a recognition domain (lowercase letter, for example) would be stored in one file to be used exclusively for training, and the remainder of the data from that domain would be stored in another file for testing. The desirability of cross-validation (explained in the next section) led us to abandon this practice in favor of keeping all data for a recognition domain in a single file. Thus for a given training and testing run, the data could be split arbitrarily into separate arrays.

4.3.2.1 Single Response — Simple Tests

Two types of testing are employed during a round of off-line training and testing. The simple method is simply to run through the set of test inputs and record the number of

correct responses. This is usually expressed as the ratio of correct answers to total test inputs, and is called the *recognition rate*.

4.3.2.2 Alternate Responses — Deep Tests

Alternately, the system can be asked to not only respond with the "winning" prototype, but with alternate responses. This requires somewhat more time, but furnishes useful information. The system may measure its performance on the first response, and on first response plus first alternate response, and so on.

The process of in-depth recognition also permits the system to return alternate choices to the user, perhaps eliminating the re-entry of a character. The in-depth testing generates the *confidence factor* (defined above) and can track the number of responses in selected ranges of confidence (0.00 - 0.05, 0.05 - 0.10, and so forth). This data is used to measure the reliability of the confidence factor in predicting the likelihood of correct classification.

4.3.3 Cross-Validation

The question of generalization naturally arises in such testing. To what degree do particular test results reflect the real ability of the system to generalize its learning to representative input patterns from the real world? Cross-validation is frequently used to measure the validity of statistical pattern associators. Weiss and Kulikowski [178] give a good introduction to this method.

We divide a given scribble file into fifths, then run five full training sessions where one-fifth is held in reserve for testing and the other four-fifths are used for training. Thus every scribble in the file is used once for testing and four times for training. Plutowski [132] has shown that cross-validation estimates Integrated Mean Squared Error (IMSE)

within a constant due to noise. IMSE is a version of the mean squared error that accounts for the finite size of the training set. He describes it as an idealized measure of generalization that embodies the statistical bias and variance tradeoff; that is, it can be decomposed into the sum of two terms which directly quantify the bias and variance.

4.3.4 Observational Methods

Many of the successes and pitfalls of any user interface can only be judged by watching representative users in action. Children present special challenges and rare opportunities in designing user interfaces. Kay [83] notes to what degree his researches on adult interfaces depended on of his careful observation of children. We had the opportunity to observe children (elementary through college age) and adults using various elements of SCL.

For the formal testing, we recorded (under the auspices of our sponsor's representative) the use of demonstration interfaces. We captured sampling data (point by point) to disk files, and taped with two video cassette recorders. One recorder continuously captured the VGA output to the tablet screen and the other recorded the participants.

This concludes our presentation of the solution methodology of SCL and the introduction of our experimental methods. The next chapter reports the results of these experiments.

5. Results

The chapter reports the results of our experiments with SCL. Many of the results are reported in figures and tables to which Chapters 6 and 7 refer for evaluations and conclusions.

5.1 Organization and Reporting Conventions

We first explain the chapter's organization and the conventions used in reporting our results. These help provide a context for the diverse data reported in this chapter.

5.1.1 Organization of the Chapter

During the past three years, we have conducted hundreds of tests involving billions of backpropagations and dozens of SCL design issues. (The *summary* logs of our experiments are hundreds of pages.) We report a sampling of those results in the hope of conveying the scope and import of the explorations — especially as they point back to the goals of the problem statement.

The results are arranged into five divisions.

Division 5.2 (Experiments that Guided the Design and Evolution of SCL) reports experiments that influenced the design decisions of SCL. Although the basic SCL algorithm has not changed in three years, implementation refinements have strengthened its performance. This section gives a chronological overview of this evolution emphasizing the choice of backpropagation over fuzzy logic, the shuffle, and the gradual reduction of hidden units (ultimately to zero). Readers who want to quickly see the current best results

of SCL may turn to Table 6, Table 7, and Table 8 on pages 103 - 105. The other divisions of the chapter reflect this current version of SCL.

Division 5.3 (Experiments Testing Adaptation to Single Users) reports tests of adaptation to individual users, and retention of prior learning after adaptation to new users.

Division 5.4 (Other Specific Performance Results) reports experiments that test narrower methodological elements of SCL — speed, confidence, complexity, filters, feature sets, sensitivity of parameters, and others.

Division 5.5 (Tests with Children and Instructors) reports two tests of SCL (each lasting about a week) with Kumon students and with Kumon instructors.

Division 5.6 (Results from Extensions to SCL) reports the work of the author and others in extending the basic SCL model to other recognition environments (such as the recognition of fractions), and useful additions to the modules of SCL (such as stronger segmentation filters and different feature sets).

The central theme of all five divisions — indeed of every experiment — is adaptation. SCL starts with no knowledge of the training task. When the first scribble is presented, since SCL has no prototypes, it "guesses" the first category name, forms the first prototype, and begins adapting. It cannot distinguish "good" or "poor" exemplars, just similar exemplars. The only distinction then among adaptation tasks lies in the sequencing of exemplars and training regimens.

Most experiments are summarized in a table or figure. Generally, the discussion of the experiment *precedes* the table or figure. The next section explains timekeeping conventions.

5.1.2 Reporting Timing Results

We performed experiments on systems spanning a decade of CPU technology: from an Intel 80286 to a SPARC 10. The experiments in this chapter suggest that a rate of 12,000 backpropagations per second fully achieves the real-time goals of our research. The Analog Devices DSP 21020 [3] realizes this performance (50 MIPS, 12,000 back-propagations, or bprops, per second), as do the SPARC 10 (11,300 bprops/sec) and the HP 720 (57 MIPS, 12,000 bprops/sec). Several other processors achieve a significant fraction of the necessary speed. If we relax the one second for *any* training interval criterion to a few seconds for the *longest* training interval, and one second for the *average*, the NeXTstation (Motorola MC68040 25 Mhz, 2,200 bprops/sec) and the PC (Intel i486 DX2 66 Mhz, 3,100 bprops/sec) qualify. All five processors were used (and others), but the majority of our experiments were run on the HP 720 at Principia College, so it is used as a standard timing index. For some historical results, the times are reported as they occurred on the particular system for that experiment; these exceptions are so indicated in the text. Otherwise, times are translated to the HP 720 equivalent.

5.2 Experiments that Guided the Design and Evolution of SCL

This section shows several critical experiments that molded SCL into its final form. These results are generally presented in chronological order. Important design questions arose in the subtle relation between individual learning units and the system that manages them. Elements of preprocessing (such as feature abstraction) proved critical in designing a system to achieve our goals. In fact, divorced from the chronology of experiment and result, some features of the final design (such as the shuffling of exemplars and the filtering of incoming stroke data) seem arbitrary, but in the context of the demands that produced them, the overall design and the later results are better understood.

5.2.1 Backpropagation versus Fuzzy Logic

The earliest results showed the need to balance positive and negative training. The first SCL recognizer built a new prototype for every new pattern; it exhibited *no* generalization. We had to limit the initial training sequence to encourage the prototypes to gather kindred exemplars. Secondly, negative training was clearly necessary for a new prototype; otherwise, the new prototype is too eager to vote for every subsequent pattern. A working balance between the countervailing forces of over-specialization and over-generalization laid the foundation for the following results. Our earliest comparison was between backpropagation-based learning units and learning units based on fuzzy logic [174, 175, 184].

5.2.1.1 Recognition Performance with Backpropagation

This first task for SCL was to recognize handwritten digits, 0 - 9, collected on a pen-based Lombart computer. The velocity and acceleration were divided into 7 and 6 intervals respectively to generate 26 floating point features; the stroke count is used as the 27^{th} feature. Early experiments used only energy-based features. SCL was limited to 40 prototype units (N = 40). The 1,400 handwritten digits collected from 20 subjects on the Lombart computer were divided into 800 for training and 600 for testing. The column "Bprops" gives the backpropagations; a *bprop* is one backpropagation for one prototype learning unit. In the experiment reported in Table 2 below, for example, since 19 prototypes were generated, each unit was trained for an average of 11,000 backpropagations. After the training, the system correctly recognized 92.7% of the 600 digit patterns in the test group. The column "Time" is training time on a NeXTstation (which is based on the Motorola 25 MHz MC68040 rated at 15 MIPS). All the results in this section (comparing backpropagation and fuzzy logic) are actual times on this NeXTstation.

Later experiments (reported on the third row of the table) used a combination of 57 to 63 real-valued features (stroke count, 20 to 26 energy-based features, and 36

grid-based features). The 63 features, training set, and testing set are identical to those used for SCL fuzzy logic digit training reported in Table 3 below.

Table 2. Training and Testing with Lombart Data: Backpropagation

SCL Parameters				Hidden Prototypes			Time	
η	σ_H	ρ	σ_L	units	created	Bprops	(min)	Recognition
Energy-based features only								
.0005	0.5	-0.10	-0.5	4	19	211,761	<20	92.7%
.0005	0.5	-0.10	-0.5	4	26	2,791,989	<250	94.8%
Both energy-based and graphical features								
.0010	0.3	0.05	-0.5	4	35	648,287	32	97.5%

5.2.1.2 Recognition Performance with Fuzzy Logic

In Wang, Kimura, and Fuller [175], we reported the results of using prototype classifiers based on fuzzy logic. Each such prototype unit responded with a number between zero and one indicating the degree of membership in the set; this degree is defined by the exemplars historically associated with that unit. Each new exemplar requires only a recalculation of the weights (akin to a rolling average). This greatly reduces the CPU load, but fuzzy logic has no counterpart to the negative training that was found necessary to prevent "greedy" prototype detectors from seizing neighboring turf. For this reason, the test results were not quite as high as those for backpropagation. The fuzzy logic results are reported below in Table 3. The membership value ρ in this table refers to the threshold (as defined above) applied to the fuzzy membership function. See Wang [174] for a fuller description of SCL with fuzzy logic.

Table 3. Training and Testing with Lombart Data: Fuzzy Logic

Membership value ρ	Number of prototypes	Training time (min.)	Recognition rate train:(800)	Recognition rate: test(600)
0.80	88	4	96.62%	95.67%
0.85	88	4	96.88%	96.83%
0.90	157	11	99.50%	97.17%

Finally, we group these results with comparable results using backpropagation-based learning units in Table 4. The features, training set and testing set are identical between the tests of the two algorithms.

Table 4. Comparison of Backpropagation and Fuzzy Logic

Learning unit algorithm	Number of prototypes	Training time (min.)	Recognition rate (testing data)
backprop	37	4	98.1%
backprop	35	19	99.7%
fuzzy	88	4	96.8%
fuzzy	157	11	97.2%

5.2.2 System Architecture versus Complexity of Individual Units

In earliest experiments, individual prototype learning units required 4 to 8 hidden units. Experimentation revealed that the flexibility of the SCL model permits the substitution of more units for depth (hence complexity) of each unit. This led to SCL configurations roughly an order of magnitude faster with no loss — in fact, improvement — in recognition performance.

5.2.2.1 Toward Simpler Learning Units

Later experiments used data sets of 1,500 digits and 3,900 lowercase letters abbreviated below as *digits* and *lowercase* respectively. The data were collected on a Wacom digitizer. In each experiment, we usually employed four fifths of the set for training and one fifth for testing. Improvements to filtering and feature abstraction led to the reduction from four to two hidden units which in turn significantly sped up learning convergence as illustrated in Table 5.

Table 5. Training and Testing with Wacom Data

SCL Parameters				Hidden Units	Prototypes Created	Bprops	Time (min)	Recognition
η	σ_H	ρ	σ_L					
Energy-based features (27 features) — digits								
.0010	0.9	0.10	-0.5	2	39	712,466	9	98.4%
Both energy-based and graphical features (63 features) — digits								
.0010	0.8	0.10	-0.5	2	29	65,921	2	97.5%
.0010	0.8	0.10	-0.5	2	35	906,315	19	99.7%
Energy-based features (27 features) — lowercase								
.0010	0.9	0.10	-0.5	2	144	3,383,493	<200	91.1%
Both energy-based and graphical features (63 features) — lowercase								
.0010	0.9	-0.10	-0.5	2	139	2,079,554	139	95.2%
.0010	0.9	-0.05	-0.5	2	143	3,271,052	181	95.7%

Continuing improvements in preprocessing (smoothing and shearing) and the periodic shuffling of exemplars led to even simpler learning units (no hidden units at all) with slightly better performance on equal or more difficult data sets. The details of these improvements are explored in later sections. Our most recent experiments used a collection of 3,200 digits from children to adults, the 3,900 lowercase letters described above, and a file combining the 3,200 digits, the 3,900 lowercase letters and 1,560 uppercase letters. These three domains we abbreviate as digits, lowercase, and mixed respectively.

In comparing the following results with the former, it should be noted that the 3,200-digit scribble set included younger writers. The original 1,500 scribbles were from one high school student, twelve college students, and two adults. Twelve hundred of the additional scribbles were collected from twelve children of ages 5 to 13. The performance on the 3,200 digits for similar recognizers was always poorer than the performance on the 1,500 digits used in the earlier experiments.

Table 6 illustrates a digit recognizer with *no* hidden units. It employs smoothing (with a ratio of 3/4), shear transformation, and two full exemplar shuffles. The learning rate ε is 0.001 and the momentum α is 0.9 for all the experiments reported for the rest of this section. The recognizer is using 70 input features (including both grid and energy features). The table shows the performance of the recognizer on the test data at three intervals during the training: after 49 seconds, after 1 minute: 32 seconds, and after 2 minutes: 17 seconds. This snapshot look at the recognizer during the training is continued in the other tables in this section. As part of the shuffling regimen, the recognizer may raise or lower ρ. If the number of prototypes generated is close to the limit, it lowers ρ; otherwise it raises ρ (as happened here) to best use available learning unit space. The parameters σ_H and σ_L are not changed during the training.

This test was one part of a series of five using the five-way cross validation explained in Chapter 4. Since the data were collected in cycles of 10 samples from each subject, it is natural to refer to the five runs of training and testing by their relative position in the collection cycle. Thus for the first test, we withheld samples 0 and 1 (of each digit) for testing and trained on samples 2 through 9. In the fifth test of the five (shown in the table below), we withheld samples 8 and 9 of the ten for each digit, and trained on samples 0 through 7. The average recognition performance for all five cycles was 99.2%; the

recognizer created an average of 49.2 prototypes. The training times and number of back-propagations for the other four cycles were virtually identical to the cycle reported here. Comparable results for the subset of 1,500 digits used in the last section may be found in Table 20 (Varying Three Parameters for 1,500 Digits) on page 132 in this chapter.

Table 6. Digit Recognizer Performance (3,200 Digits)

SCL Parameters σ_H	ρ	σ_L	Prototypes created	Bprops	Time (m:s)	Recognition
0.70	-0.05	-0.20	41	609,073	0:49	97.5%
0.70	0.00	-0.20	43	1,148,049	1:32	99.2%
0.70	0.05	-0.20	44	1,792,263	2:17	99.5%

Table 7 reports current results for lowercase recognition. The training schedule included three shuffles and more backpropagations, since more prototypes are needed, but was otherwise the same as the last experiment.

This was the fourth of the five cycles. For the full five-way cross validation, the average recognition rate was 98.0% with the recognizer creating an average of 121.4 prototypes. The second column indicates that the recognizer increased the value of ρ from -0.10 to 0.05 during shuffles, hence creating more prototypes, but improving its recognition performance. During this suite of training and testing, the system averaged 10,800 bprops per second.

Table 7. Lowercase Letter Recognizer Performance (3,900 letters)

SCL Parameters σ_H	ρ	σ_L	Prototypes created	Bprops	Time (m:s)	Recognition
0.60	-0.10	-0.50	120	2,394,465	3:41	97.8%
0.60	0.05	-0.50	121	5,157,149	7:58	98.6%

Table 8 presents similar testing for the mixed (uppercase, lowercase, and digit) recognizer. Following Neisser and Weene [122], we treated several ambiguous characters as equivalent. We treated single-stroke digit *1*, lower case *l*, and (one or two-stroke) uppercase *I* as equivalent. We treated digit *0*, lowercase *o*, and uppercase *O* as equivalent. We treated the uppercase of *C*, *O*, *P*, *S*, *V*, *W*, *X*, and *Z* as respectively equivalent to their lowercase forms. This reduces the number of categories from 62 to 51. Not surprisingly, the recognizer generated about twice as many prototypes and settled after twice as many backpropagations as the lowercase recognizer with 26 categories.

The average for all five cycles was 96.5% with 271.0 prototypes. The first half of the training called for testing only at the end of a particular mode of training (such as *newTrain()* or *oldTrain()*) which permitted 12,300 bprops/sec. The last half of the training called the *test()* function every 100,000 bprops. Testing demands much computationally; each of the 1,732 scribbles in the test set must be presented to each of the 292 prototypes for one test. During the last half, testing consumed almost a third of the processor cycles leading to 8,300 bprops/sec.

During the shuffles, the recognizer gradually raised the value of ρ (second column — from -0.10 to 0.05) to make better use of available memory. This led to an increased number of prototypes and to better recognition performance. The finally trained recognizer missed 55 of the 1732 test scribbles. The most frequent error was the classification

of a single-stroke *5* as an *s*, accounting for 5 of the 55 misrecognitions. If the alternate guesses are considered, this recognizer achieved recognition rates of 96.8%, 98.7%, 99.1%, 99.4%, 99.5% for the first guess, first two guesses, first three, four and five guesses respectively (of the 51 possible categories for this domain).

Table 8. Mixed Recognizer Performance (8,660 Scribbles)

SCL Parameters σ_H	ρ	σ_L	Prototypes created	Bprops	Time (m:s)	Recognition
0.60	-0.10	-0.50	243	2,626,063	3:44	93.1%
0.60	0.00	-0.50	272	9,257,131	12:09	95.7%
0.60	0.05	-0.50	278	13,567,829	20:33	96.9%

Even without the adjustments for ambiguous letters noted above, one test reached 93.1%. Perhaps subjects contributed idiosyncrasies of scribble formation that allowed the recognizer to discern between letters that *appear* identical. Generally, mixed recognizers failed to do better than 91% to 92% without the ambiguity adjustment.

5.2.2.2 Shuffle Training

Extensive experimentation led to a number of refinements in the training regimen. In early experiments, SCL tended to be overly sensitive to small differences in the patterns, resulting in nearly as many prototypes as exemplars. This could partly be compensated by low values of ρ, but was further strengthened by dividing the training into phases with different rules. The new training (while gathering exemplars) was limited (usually into 1000 or 2000) for each new prototype, divided equally between the newest prototype, and old prototypes that either responded too weakly to their own patterns ($< \sigma_H$) or too strongly to other patterns ($> \sigma_L$). The prototypes could then undergo old training without competing for new exemplars.

Still, performance reached a plateau caused often by unfortunate collection of exemplars, especially during the very earliest interval of the exemplar gathering phase. Since the system starts from scratch with no prototypes, the first prototype for a category is quite indiscriminate. As the prototype settles down, the added wisdom of more mature weights may cause the first few exemplars to be rather embarrassing oddities compared to its current profile.

For example, one digit prototype, $n_{s,}$ happened to gather a few eights made with two circles ("snowman eights"), but since it was the only eight collector around, it eventually gathered more eights made by starting at the central intersection ("inside-out eights"). By then another prototype, n_i, had started gathering the snowman eights which were by now not within ρ similarity of the original prototype. Obviously, it would be desirable to have n_s give up its snowman eights to n_i.

This useful result was achieved by permitting one, two, or three shuffles during the training where all prototypes start afresh with no exemplars, but retain their weight values. They compete in a new round of exemplar-gathering. Those which gather no exemplars are dismissed; such prototypes usually were over-specialized to an outlying exemplar. The others generally gather a more coordinated set of exemplars after a shuffle. The improvement afforded by the introduction of shuffles is suggested by Figure 17 below. This shows the recognition rate of a digit recognizer during training. The entire sequence is 6,710,153 bprops (or 6.7 Mbprops) during 501 seconds. After rising quickly for the first 20 seconds, the rate bumped into a ceiling at about 94.0% prior to the first shuffle. After the shuffle it rose quickly to about 98.0%. After the second shuffle the rate rose again to about 98.3% and again leveled off. After the last shuffle (at 4.7 Mbprops, 350 seconds into the training), the rate rose to a ceiling of about 98.6%

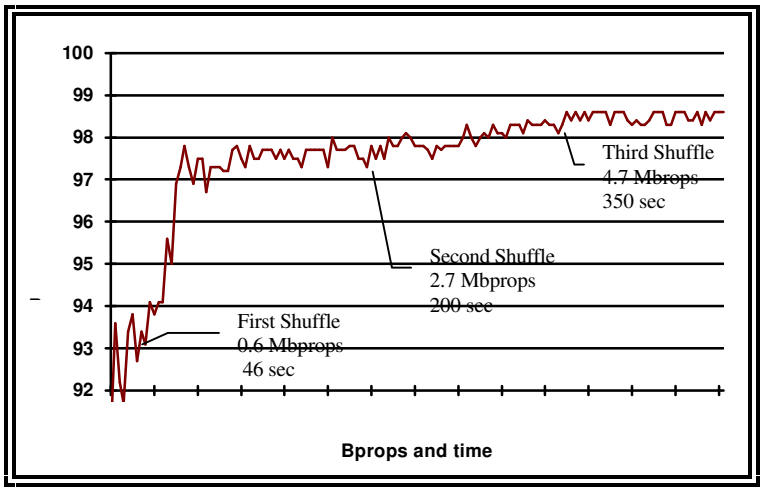

Figure 17. Digit Recognizer Training with Three Shuffles

A second illustration is shown below in Figure 18 using a lowercase letter recognizer. The full graph represents a training sequence of 10.7 Mbprops and 19 minutes: 10 seconds. Two shuffles are included. The recognizer rose to bump between the limits of about 95.0% and 95.6%. After the first shuffle, it rose to an upper bound of about 96.6%. After the second shuffle, it rose again to about 97.6%. A third shuffle at this point (10.7 Mbprops, 19:10) did not significantly raise the recognition rate further. Our tests indicate that a first shuffle is always helpful, but that subsequent shuffles tend to have diminishing returns. Three is the largest number of shuffles seen to produce significant improvement. We presume that the exemplars are sufficiently rationalized after three shuffles to preclude further benefit.

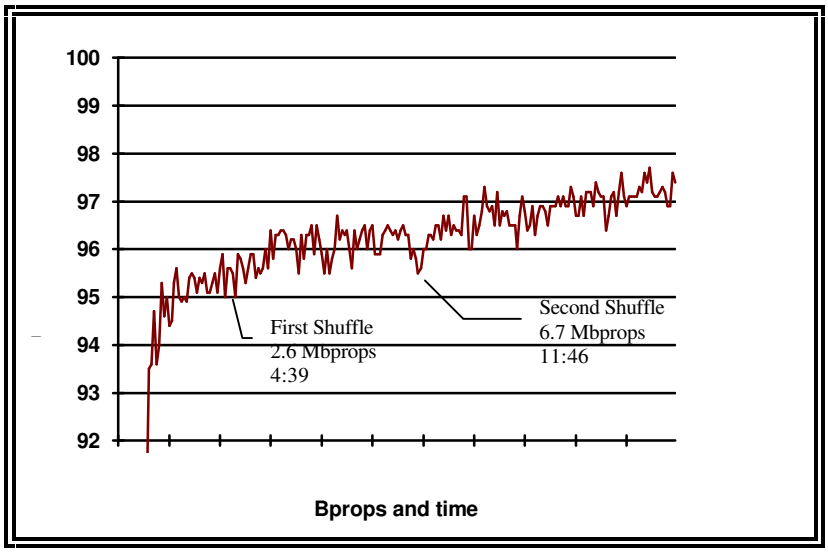

Figure 18. Letter Recognizer Training with Two Shuffles

5.2.2.3 Current Architecture versus Hidden Units

To provide a retrospective comparison, we tested the *current* system architecture, training regimens, preprocessing, and feature abstraction with (digit) prototype units of 8, 4, 2, and 0 hidden units each. We allowed the number of prototypes to increase as the number of hidden units decreased. The results are shown below in Table 9. The unexpected conclusion is that once the above-listed refinements to SCL were in place, an increase in hidden units did not raise performance; in fact, it slightly reduced it. Even when the number of prototypes is held constant, the performance is generally stronger with fewer hidden units.

Table 9. Performance with 8, 4, 2, or 0 Hidden Units

Hidden units	Prototypes Limit	Actual	Rec'n rate	Bprops/ sec	Bprops ratio	Weights/ prototype
8	20	19.0	97.5%	1,300	10.0	638
4	20	19.5	97.4%	2,600	5.1	354
4	30	29.5	98.4%	2,600	5.1	354
4	40	34.0	98.7%	2,600	5.1	354
2	20	19.0	96.8%	3,700	3.6	212
2	30	29.0	98.4%	3,700	3.6	212
2	40	36.5	98.5%	3,700	3.6	212
0	20	19.5	97.0%	13,200	1.0	70
0	30	29.5	98.3%	13,200	1.0	70
0	40	38.0	98.8%	13,200	1.0	70

5.3 Experiments Testing Adaptation to Single Users

The above experiments illustrate adaptation to a collection of users. This section investigates the adaptation of SCL to a single user. First we consider an empty recognizer in its first encounter with a user — training from scratch. Then we consider a recognizer that has been trained on a collection of subjects, but is now required to adapt to a new user.

5.3.1 Training a Recognizer from Scratch

As noted before, SCL training is always from scratch; that is, the recognizer begins with no prototypes. As the stream of patterns is passed to SCL, it forms prototypes as necessary until an adequate set in constructed. At all times, unless the prototypes are specifically frozen, SCL is ready to use its current set to recognize the next pattern, and is equally ready to form a new prototype (or replace one) if necessary.

Given this design, it is natural to train a recognizer by simply presenting a series of scribbles on-line as it learns. When no prototypes are present, it responds with the lowest ASCII value in its category set. In six experiments with this method, SCL learned digits, Roman numerals, Japanese numerals, and Sanskrit. It was generally given 2 or 3 exemplars for each category, and then expected to recognize later scribbles with no further training. In fact, in four of the tests, the recognizer never missed a character after the first exemplar. (The first exemplar of each category is always missed since no prototypes for that category exist yet.) In the other two tests, the recognizer missed only one scribble throughout the training and testing. In each case, 15 to 50 scribbles were presented.

We present one sequence below in Figure 19 in the same order in which the scribbles were presented to the recognizer — in this case representing 7 Japanese numerals. Three training exemplars were used to train on each numeral. These 21 scribbles form the first row of the figure. The second row is the testing sequence. The shaded scribble is the only one missed in the sequence of 44 (except of course for the 7 first exemplars).

Figure 19. Scribble Sequence for Training from Scratch

Two parameters of the parameter file are particularly relevant to this training:

```
Incremental train interval:    5
Incremental train amount:   5000
```

Whenever a new prototype is created, the system goes through a cycle of new training. This will eventually be limited (another parameter, 2,000 bprops in these tests),

but the limit is not relevant since the limit was never reached in this example. After 5 exemplars have passed through new training, the system is instructed to train all existing prototypes for 5,000 backpropagations (old training). The choices of 5 and 5,000 are rather arbitrary; values from 1 to 10 and 1,000 to 100,000 respectively have been tried without greatly different results. The following chart shows the training times in seconds for each of the 21 scribbles shown on the top row of Figure 19. The last time represents the training after the shaded kanji was missed.

Table 10. Training Times (seconds)

First exemplar training	0.96			
	0.02	2.69	0.02	0.03
	0.02	0.02	1.85	0.03
	4.20	0.02	0.02	2.01
	0.02	3.14	0.02	0.02
Reinforcement training	20.65	20.09	25.65	19.47
Corrective training				5.39

The average training (including the reinforcement) is 106.34 seconds / 22 exemplars or 4.83 sec each. If the reinforcement time is omitted, the average training time is 1.13 sec. If the corrective training is also omitted, the average is 0.72 sec. per exemplar. The system went through 42,000 backpropagations during this time — about 400 backpropagations per second. This test was conducted on an Intel 80386 CPU. At the target speed for SCL technology (about 12,000 bprops/sec), the longest training interval of this sequence (10,000 bprops, 25.65 seconds) would be under a second. Even if the processor performed only 2,000 bprops/sec, all but the reinforcement training would require less than a second.

5.3.2 Retraining an Existing Recognizer

When training from scratch is impractical or unnecessary, an existing recognizer may be trained to accommodate a new user. We present a few brief examples of such *incremental* training in this section. Incremental training was also employed in the testing with Kumon users, both children and adults. Detailed results are shown in that section (page 140 of this chapter).

The sequence of training in Figure 20 illustrates six events of incremental training with a user who makes *l*'s that are rather like *z*'s. The shaded characters were misrecognized and triggered incremental training. The first two cursive *l*'s were correctly recognized. The first *z*-like *l* is recognized as a *z*. When this occurred, the user indicated the error which in turn initiated the brief interval of incremental training (5,000 bprops maximum). After three missed *l*'s and subsequent incremental training, the recognizer gradually recognized the user's *l*'s as *l*'s but now recognizes *z*'s also as *l*'s as shown by the three misrecognized *z*'s. After training on these three examples of *z*'s, the recognizer was always successful in recognizing *l*'s and *z*'s (with or without the middle crossing stroke through the *z*) for this user.

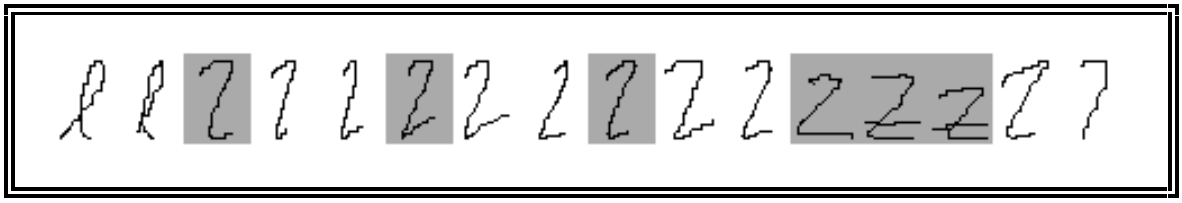

Figure 20. Incremental On-Line Training

A longer sequence of incremental training is shown in Figure 21. Here too the misrecognitions are indicated with shading. The user was chosen for this test due to her rather unusual writing of several characters. She writes *p* from the bottom, as one might write

the Greek letter ρ. Her *y*'s are cursive, and her lowercase *o*'s resemble the Greek letter θ. This was deemed a useful test of incremental training on previously unseen writing styles. Moreover, this training sequence illustrates a secondary point that will be amplified later — the tight dependence of a trained recognizer on the digitizing hardware and software. The recognizer had been trained on scribbles created under one digitizing scheme, but was currently receiving scribbles created by another scheme. This hampered the incremental training, and tended to undermine the original training. Thus this example might be considered an extreme case of incremental training.

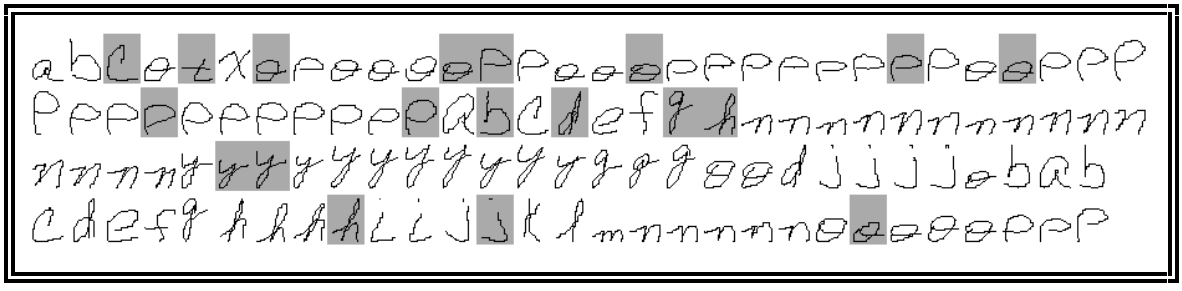

Figure 21. More Incremental Training

This scribble sequence was captured and later submitted to the incremental training function with a variety of parameters (shown in Table 13 on page 118). The recognizer misrecognized from 18 to 34 of the 119 scribbles presented, depending on the amount of training permitted in response to misrecognitions. Figure 21 was taken from a later run of the same 119 scribbles in which 19 scribbles were misrecognized. The pattern of misses in the original case (when the user was on-line) and during later tests generally resembled the pattern shown. The exception is the rho-like *p* which was blocked during the on-line case by a stubborn *o* made in virtually the same manner. After three or four examples, the new *p* always showed as the first alternate, but the *o* pushed in front as the recognizer's response. Otherwise, the first several instances of the unusual character were missed; then

the character was correctly recognized. In this example, none of the new letter forms caused a loss of recognition in other of the user's letters (as opposed to what happened with *l* and *z* in the previous case shown in Figure 20).

When a recognizer is incrementally trained, to what degree does it lose its ability to respond to its original training domain? A malicious user could always hamstring a recognizer by training it on the wrong scribbles (telling it that an *a* is a *z*, a *k* is a *p* and so on). In the absence of malice, a recognizer is obviously susceptible to confusion if new scribbles are similar to past scribbles of another category. This is what happened with the *l*'s and *z*'s of Figure 20. The following section will discuss the dependence of a trained recognizer on the digitizing environment. In the absence of changes in the digitizing environment, SCL shows a measurable stability when adapting. In the testing of Kumon students described below, the recognizer learned the writing styles of several students. We use this data to test the stability of the recognizers in the following experiment. For each of five students, we test the recognizer on the set of 3,200 digits before and after it has incrementally trained on the student's input. These results are shown in Table 11.

For each of these incremental training experiments, the recognizer was set to train if necessary up to a maximum of 10,000 bprops after every other input (training interval of 2). The table is divided into three time zones: before, during, and after the incremental training. The recognition rates for the original 3,200 digits and the individual student's 100 digits are shown both before and after the training. Before the incremental training, the recognizer correctly recognized 3,166 of the 3,200 digit scribbles (98.9%). After each of the five experiments, it recognized either 3,165 or 3,166 of the same 3,200, indicating no significant loss of prior training. While the training proceeded, the recognizer may form as many or few new prototypes as the vigilance demands. In two cases (cs and sa), it formed

more prototypes than scribbles misrecognized; this happens when the category is correct, but the activation is lass than ρ. Training time depends on the number of prototypes added.

Table 11. Recognition Stability after Incremental Training

	Before		During			After	
Stu-dent	**Rate for 3200**	**Rate for 100**	**Missed**	**Protos added**	**Time (sec)**	**Rate for 3200**	**Rate for 100**
br	98.9%	97%	3	3	12	98.9%	99%
cs	98.9%	96%	3	4	25	98.9%	100%
gn	98.9%	99%	1	1	5	98.9%	99%
ka	98.9%	100%	0	0	1	98.9%	100%
sa	98.9%	98%	2	3	14	98.9%	99%

5.3.3 Dependence of Recognizers on Digitizers

As mentioned earlier, a recognizer depends strongly on the digitizing hardware and software that created its exemplars. In this section we furnish two illustrations of this dependence. Table 12 shows three recognizers being tested on three sets of scribbles. The first two recognizers were trained on Wacom-collected data, and Lombart-collected data respectively. The third was trained on a combination of data collected on the Wacom, Scriptel and Mitsubishi digitizers. (Most of the Scriptel and Mitsubishi data was collected from five- to thirteen-year olds.) Each trained recognizer was then tested on scribbles from each of the three digitizing environments. Recognizers in "native" environments (shaded in the table) achieved recognition rates of 98.1% to 99.7% as expected. In completely "non-native" environments, the rate fell to as low as 82.8%. The results suggest that the Lombart digitizer is furthest removed from the other three. All four digitizers have a nominal

sampling rate of 5 ms (200 point samples per second). However, scribbles collected on the Lombart typically had from 6 to 30 points. (The digitizer driver apparently dropped some of the points.) Similar scribbles collected on the other three digitizers had from 30 to 100 points. The Wacom, Scriptel, and Mitsubishi differed little in point count per scribble. The number of prototypes created is shown in parentheses under each digitizer.

Table 12. Recognizer Dependence on Digitizer

Digitizers (Training right) **(Testing below)**	**Wacom only (27)**	**Lombart only (43)**	**Scriptel Wacom/Mit (42)**
Wacom (1500 scribbles from 15 subjects)	99.2%	91.3%	99.7%
Lombart (2800 scribbles from 40 subjects)	84.5%	98.8%	82.8%
Scriptel/Mit (1400 scribbles from 14 subjects)	90.1%	86.1%	98.1%

Another series of experiments built a recognizer based on the *combined* data from all four digitizing environments. It required a large number of prototypes to accommodate the divergent environments — 90 to 100 — and achieved recognition rates in the range of 97% to 98%.

This dependence is not limited to hardware. The device driver and the collection software can prove just as critical. In the next case, the same digitizer was used in both the original training and incremental training, but the latter employed different software that transformed the coordinates of the collected points into logical window coordinates. For example, here are the *same* points of the same scribble under both coordinate systems.

```
Digitizer coordinates:
178,43  178,44  177,45  176,45  176,45  175,45  175,44  174,43  173,42  172,41  171,40
171,39  170,38  169,36  167,34  166,33  166,31  166,30  165,28  165,25  165,23

Logical window coordinates
 66,16   66,16   65,17   65,17   65,17   65,17   65,16   64,16   64,15   63,15   63,15
 63,14   63,14   62,13   62,13   61,12   61,11   61,11   61,10   61, 9   61,8
```

The range of x in digitizer coordinates is from 165 to 178 = 13; the range of y is 23 to 43 = 20. In logical window coordinates, the x range is 61 to 66 = 6; y is 8 to 16 = 8. The number of points does not change, but the dynamic range of the points is reduced by more than half. The set of scribbles shown in Figure 21 on page 113 was collected under this logical window coordinate transformation. Table 13 shows five sessions of incremental training (with this data) of a lowercase recognizer trained on scribbles collected with digitizer coordinates. The difference in the coordinates evidently weakened existing prototypes and did little to improve the recognizer's performance on the training set.

Although it succeeded in recognizing many of the scribbles, it was weaker on the *training set itself* after training than before. As expected, it was significantly weaker on the "native" 3900 lowercase scribbles than before the training; this rate fell from 97.9% to 78.9% to 95.8%. Paradoxically, the more the recognizer trained on the new scribbles, the poorer its recognition rate on those same new scribbles after the training. The third and fifth test had the longest interval between training, hence the fewest bprops. These had the highest performance on the 119 scribbles after training. The mixing of coordinate systems is thus seen to be just as detrimental to performance as the mixing of other elements of the digitizing environment.

Table 13. Incremental Training with Mixed Coordinate Systems

	Before		During				After	
Train interval	Rate: 3900	Rate: 119	Missed	Protos added	Time (sec)	Bprops	Rate: 119	Rate: 3900
1	97.9%	77.3%	19	12	122	1,908,934	62.2%	78.9%
1	97.9%	77.3%	19	12	122	1,909,316	62.2%	80.0%
10	97.9%	77.3%	34	11	19	248,756	65.5%	86.7%
1	97.9%	77.3%	18	3	121	1,911,400	64.7%	92.4%
10	97.9%	77.3%	31	3	13	206,792	68.1%	95.8%

The dependence of recognizers on digitizers extends also to the filters used in preprocessing the scribbles. Our experiments found quite small differences between various types of smoothing (simple, bi-directional, and symmetric) and various smoothing ratios (1/2, 3/4, and 7/8). Contrariwise, the following experiment (just reported in this paragraph, not in a table) shows how very dependent the recognizer is on the filter that was used at the time of its training. We trained a digit recognizer with 1,000 of the 1,500 digit scribbles filtered with simple smoothing of 7/8. Then we presented a test set of 500 scribbles (those not used during training) filtered with bi-directional smoothing of 3/4. The recognizer correctly recognized only 415 of 500 (83.0%). When the same set of 500 was presented with simple smoothing of 7/8 (same as the original training), it recognized 497 of 500 (99.4%).

Thus we conclude from the experiments reported in this section that SCL recognizers are critically dependent on digitizing hardware and software, and on smoothing filters. If these remain constant, SCL shows strong retention of earlier training even after accommodation to a new user.

5.4 Other Specific Performance Results

This section reports results that illustrate SCL's exemplar gathering, recognition speed, confidence calculations, complexity, and sensitivity to parameters.

5.4.1 Behavior of Prototypes in Gathering Exemplars

Figure 22 and Figure 23 show exemplars gathered by the prototypes of two recognizers (digits and lowercase). If SCL is gathering prototypes effectively, the exemplars gathered into one prototype should appear similar. (The word *appear* is misleading; the exemplars should be similar in the grid and energy feature space.) Both recognizers trained from scratch on data files which were presented in the order collected (*sequential* new training). This sequential training possibly biases the system in favor of grouping the scribbles of individual writers. However, the experiments reported above show that post-shuffle new training — in which exemplars are always presented in random order — is more successful than sequential training. Thus, the sequential presentation might be a liability more than a benefit; the two illustrations therefore present a worst case gathering. Exemplars are usually gathered from many writers into a single prototype unless the writing is unusually distinctive.

Prototype 0 gathered zeros formed from the bottom. Prototypes 1 through 3 gathered zeros formed in the usual manner from the top. Prototype 5 gathered left-leaning ones. Prototype 19 gathered the European-style sevens (with an exception of one American-style seven that shuffle training would likely cast off). Prototype 22 also gathered exemplars that are not obviously close — some eights that begin in the middle intersection and others that begin between the middle and the upper right. However, these might be closer than they appear in the energy-based feature space. Prototype 24 gathered the

"curly" nines. Prototype 26 gathered the "lollipop" nines — nines formed by a circle stroke above a vertical stroke.

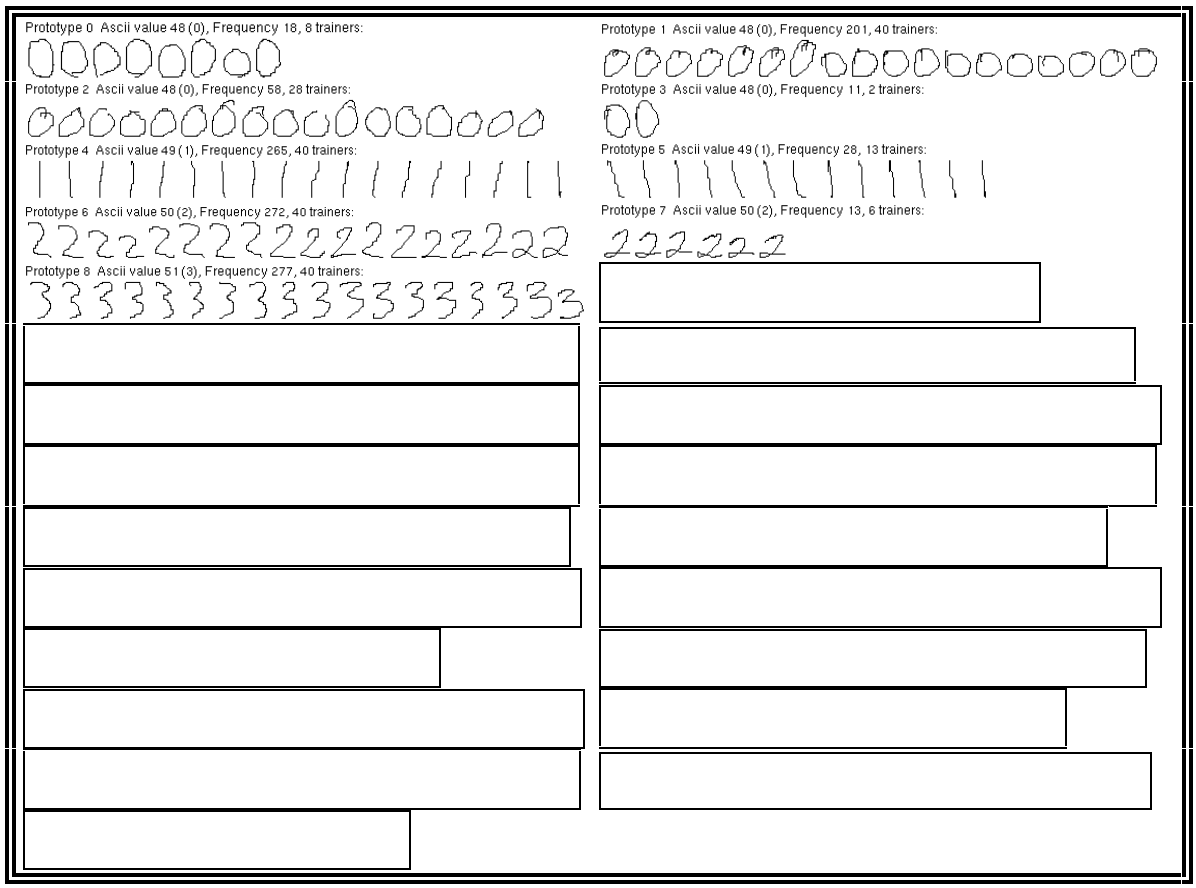

Figure 22. Exemplars Gathered by Digit Prototypes

Figure 22 showed the full set of 27 prototypes created during training. Figure 23 on the following page shows a subset of prototypes for the letters *a*, *k*, *p*, *q*, and *r*. The five styles of *a* are clearly differentiated by the five *a* prototypes. Likewise, the prototypes gathered exemplars of the various styles of the other letters. Note the last *r* prototype (prototype 79) gathers exemplars that closely resemble *v*.

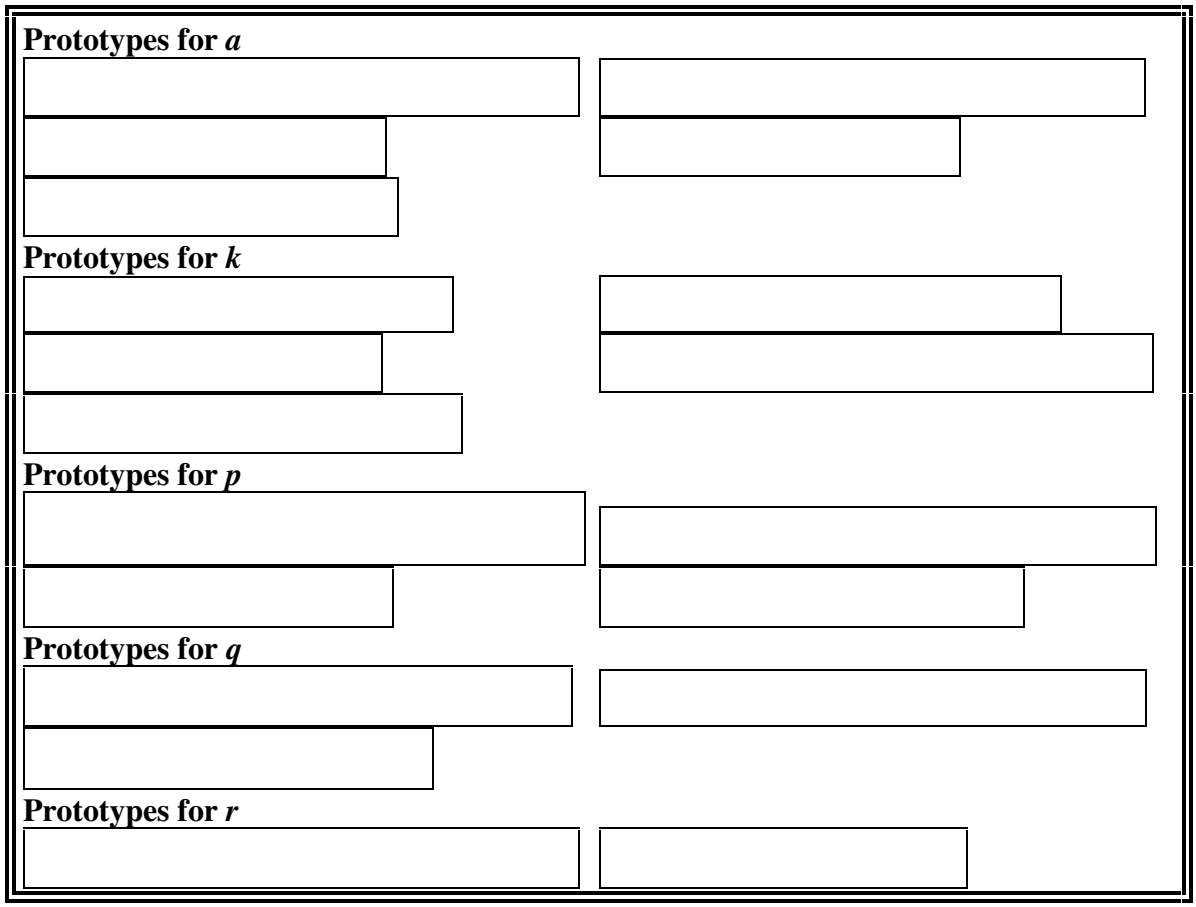

Figure 23. Exemplars Gathered by Lowercase Prototypes

5.4.2 Confidence

Table 14 illustrates the correlation between confidence (as defined in Chapter 4) and actual correctness. Results with digits (99% recognition rate) limit the relevance of any confidence measure, but it is meaningful in the recognition of letters. The following table illustrates the correlation between confidence and correctness. In this particular test, SCL identified 97.4% of the test set (lowercase letters) correctly. It correctly identified 760 of the 780 test scribbles. In 13 of the 20 cases of misclassification, the second highest guess is correct. The following comments refer to the three shaded rows in Table 14. For

373 of the 780 recognitions (47.8% of the test set), the confidence was above 0.45; for this group — nearly half of the test set — the recognizer was 100% correct. For 596 of the 780 recognitions (76.4% of the test set), the confidence was above 0.25; the recognizer was 99.5% correct for these. For 735 of the 780 (94.2% of the test set), the confidence was above 0.10 and the recognition rate was 98.5%.

Table 14. Confidence Factor versus Correctness — Lowercase

Confidence Factor	***For this CF*** Correct	Ratio	***Cumulative*** Accumulated	***for all CF≥ this CF*** Correct	Ratio
1.00	0 of 0	100.0%	0.0%	0 of 0	100.0%
0.95	4 of 4	100.0%	0.5%	4 of 4	100.0%
0.90	2 of 2	100.0%	0.8%	6 of 6	100.0%
0.85	18 of 18	100.0%	3.1%	24 of 24	100.0%
0.80	17 of 17	100.0%	5.3%	41 of 41	100.0%
0.75	20 of 20	100.0%	7.8%	61 of 61	100.0%
0.70	30 of 30	100.0%	11.7%	91 of 91	100.0%
0.65	39 of 39	100.0%	16.7%	130 of 130	100.0%
0.60	63 of 63	100.0%	24.7%	193 of 193	100.0%
0.55	56 of 56	100.0%	31.9%	249 of 249	100.0%
0.50	54 of 54	100.0%	38.8%	303 of 303	100.0%
0.45	70 of 70	100.0%	47.8%	373 of 373	100.0%
0.40	54 of 56	96.4%	55.0%	427 of 429	99.5%
0.35	66 of 66	100.0%	63.5%	493 of 495	99.6%
0.30	51 of 51	100.0%	70.0%	544 of 546	99.6%
0.25	49 of 50	98.0%	76.4%	593 of 596	99.5%
0.20	47 of 50	94.0%	82.8%	640 of 646	99.1%
0.15	34 of 35	97.1%	87.3%	674 of 681	99.0%
0.10	50 of 54	92.6%	94.2%	724 of 735	98.5%
0.05	36 of 45	80.0%	100.0%	760 of 780	97.4%

Table 15 shows similar results for a recognizer trained on digits, lowercase, and uppercase letters. For 988 of 1732 test scribbles (shaded below — 57.0% of the test set), the response had a confidence of 0.30 or higher. The response was correct for 985 of

these 988 (99.7% correct). The last line of the table indicates that the confidence was less than 0.05 for 154 of the test scribbles; the recognizer was correct for 117 (76%) of these.

This adjusted mixed set includes 51 categories. On this test, the recognizer missed 58 of 1732 test scribbles for a recognition rate of 96.7%. If the second response is added, the recognition rate increases to 99.2%. If the third response (of the 51 possible) is added, the rate rises to 99.5%. Of the 58 misrecognitions, the most frequent error (4 cases) was mistaking lowercase *k* for an uppercase *K*.

Table 15. Confidence Factor versus Correctness — Mixed

Confidence Factor	***For this CF*** Correct	Ratio	***Cumulative*** Accumulated	***for all CF≥ this CF*** Correct	Ratio
1.00	0 of 0	100.0%	0.0%	0 of 0	100.0%
0.95	0 of 0	100.0%	0.0%	0 of 0	100.0%
0.90	6 of 6	100.0%	0.3%	6 of 6	100.0%
0.85	7 of 7	100.0%	0.8%	13 of 13	100.0%
0.80	13 of 13	100.0%	1.5%	26 of 26	100.0%
0.75	35 of 35	100.0%	3.5%	61 of 61	100.0%
0.70	53 of 53	100.0%	6.6%	114 of 114	100.0%
0.65	64 of 64	100.0%	10.3%	178 of 178	100.0%
0.60	75 of 75	100.0%	14.6%	253 of 253	100.0%
0.55	83 of 83	100.0%	19.4%	336 of 336	100.0%
0.50	123 of 124	99.2%	26.6%	459 of 460	99.8%
0.45	111 of 111	100.0%	33.0%	570 of 571	99.8%
0.40	123 of 123	100.0%	40.1%	693 of 694	99.9%
0.35	150 of 151	99.3%	48.8%	843 of 845	99.8%
0.30	142 of 143	99.3%	57.0%	985 of 988	99.7%
0.25	154 of 157	98.1%	66.1%	1139 of 1145	99.5%
0.20	151 of 155	97.4%	75.1%	1290 of 1300	99.2%
0.15	129 of 136	94.9%	82.9%	1419 of 1436	98.8%
0.10	138 of 142	97.2%	91.1%	1557 of 1578	98.7%
0.05	117 of 154	76.0%	100.0%	1674 of 1732	96.7%

The rank-correlation coefficient [40:365] between the confidence factor and cumulative correctness is .9932 and .9925 for the two tables — nearly perfect. We created confidence data for four examples in the mixed domain. The average percentage correct for first guess, first two guesses, and first three guesses is 96.1%, 99.1%, and 99.4% respectively. The average rank correlation coefficient between the confidence factor and cumulative correctness for these four tests is .988. We feel that these results fully satisfy the need for a reliable measure of the system's likelihood of correctness.

The strong correlation between confidence and success could be used to establish a threshold of acceptance in the recognition process. If the confidence falls below 0.25, it may be appropriate to show a question mark and ask for re-entry. Given SCL's high success with the second guess, it would be easy to show both the first and second guesses in such cases (perhaps each half size, beside each other), and accept the letter tapped by the user. In special settings where context information is available (such as database searches), the strong correlation between confidence and success could be used to establish a highly-refined template search.

5.4.3 Segmenting a Stream of Strokes into Individual Scribbles

In Chapter 4, we explained the pedagogical importance of segmentation methods that are independent of spatial and temporal assumptions. Here we report the results of the two segmentation algorithms described there.

5.4.3.1 First Stroke-Based Algorithm

For this algorithm, an SCL recognizer was allowed to train only on the first stroke of each scribble and was required to build different categories for one-stroke and two-stroke exemplars. This was intended to enable it to learn from just the first stroke *both* (1) which digit is intended and (2) whether or not another stroke will follow as a part of

the same scribble. (When there is one, the following stroke is merely discarded.) If the requirement to recognize the number of strokes is dropped, this recognizer succeeded for about 96% of the digits. However, this requirement cannot be dropped. Even if SCL identified the scribble's category correctly, but missed on the number of strokes, it would be "out of sync" to receive the next scribble correctly.

Under this more stringent but necessary requirement, SCL was still able to recognize both the digit value and the number of strokes for 94% of the test scribbles. This affords one quick path to an unsegmented recognizer of tolerable performance. More experimentation in this area would likely yield improvements to this performance, since these results came from early versions of SCL achieving only 98% recognition with boxed (pre-segmented) input.

5.4.3.2 Confidence-Based Segmentation

This algorithm, which is restricted to multi-scribbles containing no more than two digits, was also introduced in Chapter 4. It searches for the combination of strokes that produces the highest average likelihood from the recognizer. The likelihood is a combination of the horizontal overlap and the confidence of the recognition combinations.

In an early experiment, we created 100 digit scribbles (of one or two strokes) and 100 scribble pairs (100 multi-scribbles of two to four strokes that represented two digits). All but 3 of the 300 digits (99%) were correctly segmented. Overall, the system correctly recognized 277 of the 300 digits (92.3%).

This algorithm was further tested during August, 1993, with actual Kumon students, where the SCL digit recognizer achieved an overall recognition of 95.8% of the

scribbles. This percentage reflects both segmentation and recognition performance. See the Kumon children's testing on page 141 for details.

5.4.3.3 Algorithm-Based Segmentation

There are various algorithmic approaches to solving the segmentation problem. We investigated two as just reported, but have deferred this research to others on the Kumon team. Khalid Sherdil's extension to SCL illustrates one method. His work is described later in this chapter on page 147.

5.4.4 Complexity Analysis and Performance

Our real-time goals mandate the reporting and analysis of SCL performance, which we do in this section. There are several computationally demanding tasks associated with the SCL model. Obviously these are interdependent; some call upon others to achieve their goals. The following analysis treats the general case where backpropagation is used and inputs are fully connected to hidden nodes and to output. It assumes no recurrent architecture and no shared weights; that is, every prototype includes its own weights.

5.4.4.1 Theoretical Complexity

An approximation of computational complexity can be made by considering the number of floating point additions (A) and floating point multiplications (M) as a function of the relevant variables:

I	length of real-valued input vector
P	number of active prototypes
N	maximum number of prototypes
H	number of hidden units for each active prototype
T	number of test patterns

Propagate input pattern values through a single prototype (*forwardProp()*)

$$F = (I + IH + H) * (M + A)$$

Backpropagate weight changes through a single prototype (*backProp()*)

$$B = F + (4 + 2 * F) = 3F + 4$$

Recognize an input pattern with no alternates (*recognize()*)

$$R = F * P$$

Recognize an input pattern with alternates (*recognizeDeeply()*)

$$R = F * P$$

Test the current performance of the recognizer on test data (*test()*)

$$Test = R * T$$

The handling of floating point addition and floating point multiplication varies greatly over the various processors tested. In the 80386 CPU without a 80387 co-processor, all floating point arithmetic is handled in software on the integer-based CPU, consuming dozens of cycles. In the Analog Devices ADSP 21020, both are handled in one cycle; in fact, in one cycle, a multiplication, an addition and a difference (all floating point) may be performed. If we code SCL to avail itself of these features, the ADSP 21020 performs about 12,000 bprops/sec.

Let the floating point multiply and add time of the first (forward propagation) equation be abbreviated *MA*. In the HP 720, the *MA* cycle is about 120 nsec. If we assume 73 floating point features (the largest ever used was 75) and no hidden units, this gives a forward propagation time of about 12 μsec. A single backpropagation is about 85 μsec. This too equates to about 12,000 bprops/sec. As seen in the preceding sections, the training cycle usually requires from 2,000 to 10,000 bprops or 0.2 to 0.8 seconds on the DSP or HP. On the NeXTstation, the *MA* time is about 650 nsec. This equates to about 2,000 bprops/sec, or about 1 to 5 seconds to train on a new prototype. On the 40 MHz 80386 (without floating point co-processor and without DSP), the *MA* time is about 3500

nsec leading to 400 bprops/sec or 5 to 25 seconds for incremental training as shown in Table 10 on page 111.

5.4.4.2 Actual Computational Performance

Table 16 shows the distribution of run time over the principal functions of SCL. We removed all initial file loading and preprocessing functions (which are shown separately below in Table 17). We grouped the internal functions called by *tanh()* with "Activation function." These four functions account for 90% of the total execution time. The graph profile adds the time spent within the designated function to that spent by all functions that it calls. This column adds to more than 100% since *forwardProp()* is called by both *backProp()* and *buildTrainingPool()*.

Table 16. Profile of SCL Functions

Function of SCL	Number of calls	Flat profile Run time percent	Graph profile Run time percent
Backpropagation	261,337	61.3%	72.4%
Forward propagation	606,746	17.9%	25.8%
Activation (tanh/exp)	606,746	10.3%	10.3%
Build training pool	165	0.3%	4.2%

We caution about performance assumptions. If we use the actual profile times to calculate the *MA* times (for 73 features), we calculate *MA* for the HP 720 of 7.30 µsec / 73 = 0.10 µsec. A similar calculation from a profile on the NeXT yields 0.67 µsec. The actual time of *MA* on the NeXT and HP may be found by the simple test program:

```
for (i=0; i < 100000000; ++i) a += m * m;
```

which runs in 16.4 seconds on the NeXT and 18.4 seconds on the HP (both are user time). This implies *MA* of 0.16 µsec and 0.18 µsec respectively; that is, that the NeXT is

slightly faster than the HP for floating point multiplication and addition. In fact, the HP is about 7 times faster in SCL functions. Both are fully optimized for speed on their respective processors. This demonstrates how heavily other factors contribute to computational performance: compiler design, parallel and pipelined floating point co-processing, and others. For this reason, we emphasize actual over theoretical in meeting the performance criteria.

Similarly, the space complexity trade-off does not lend itself to an algorithmic optimum. The total weight space for a recognizer is

$$F = (I + IH + H) * N.$$

In Table 9 (Performance with 8, 4, 2, or 0 Hidden Units) on page 109, $I = 70$ inputs (features). A recognizer with 20 prototypes of 2 hidden units uses the same memory as a recognizer with 60 prototypes of no hidden weights. In considering the memory trade-off between complexity of individual prototypes and the number of prototypes, the experiments reported in Table 9 decidedly favor the trade-off point of zero — no hidden units.

5.4.4.3 Preprocessor Performance

Our final performance table, Table 17, shows the Preprocessor time during the above training on the HP 720. The several filters and feature abstractors are applied once to each scribble presented to the system. Moreover, all filters and feature abstractors operate sequentially on the array of points in the scribble, so the time and space complexity of the Preprocessor is $O(p_{total})$ where p_{total} is the total number of points in the scribble.

The percentages in the flat profile are of the total Preprocessor time — 4.61 seconds for 8,660 scribbles. This is 532 μsec — just over half a msec — for each scribble. This figure would not vary for different domains, but it does depend (as a multiplied

constant) on the particular suite of Preprocessor functions applied to the incoming scribbles; this suite is typical.

Table 17. Preprocessor Time

Function of SCL	Number of calls	Run time (seconds)	Percent of Preprocessor
twoSmooth	8,660	1.45	31.5%
gridFeatures	8,660	1.33	28.9%
energyFeatures	8,660	1.29	28.0%
findStraight	9,759	0.35	7.6%
loadScribToPattern	8,660	0.14	3.0%
normalizeScribble	8,660	0.05	1.1%
Total		**4.61**	**100.0%**

5.4.4.4 Backpropagations Necessary to "Fully" Train a Recognizer

The final quantity necessary to complete our consideration of SCL performance is the time necessary to "fully" train a collection of prototypes. The word *fully* is the key. SCL (as any neural network-based system) may train *ad infinitum*, but the returns diminish and the risk of *overtraining* increases over time. Overtraining is training to the particularities of the training set to the detriment of generalization. In practice, SCL is highly resistant to overtraining. We (mistakenly) trained one mixed domain recognizer for 120 million backpropagations. It achieved the highest performance of any of our recognizers for that domain. Still, practical considerations impel a more stringent answer to the question above. We assume a 40-exemplar buffer with an average of 20 exemplars per prototype. The time to reach "good" performance depends on the size of the domain, the number of subjects (both of which tend to more prototypes), and the desired recognition rate as shown in Table 18.

Table 18. Backpropagations Necessary to "Fully" Train

Domain	Categories	Subjects	Recognition rate	Number of prototypes	Backprop-agations
Numbers	10	15	99%	30	600,000
Numbers	10	32	99%	50	1,000,000
Numbers	10	40	99%	40	2,000,000
Lowercase	26	15	97%	140	5,000,000
Mixed (not adjusted)	62	53	92%	250	15,000,000

5.4.4.5 Recognition Speed

As shown above, each recognition requires a forward propagation through all prototypes in the search for the maximal activation. Thus, the recognition time is $O(p+n)$ where p is the time for preprocessing and n is the number of prototype units. With 122 prototypes, lowercase alphabetic recognition on the 25 MHz MC68040-based NeXTstation takes about 15 ms of which 2 ms are required for the Preprocessor. This corresponds to about 65 characters per second. A numeric recognizer (with a third the number of prototypes) requires the same 2 ms for preprocessing and about 4 ms for the forward propagations, or 6 ms total. Letter recognition requires about 120 ms on a 80386 without co-processor. On the HP 720, as shown in section 5.4.4.3 above, preprocessing requires about 0.5 ms and 122 forward propagations (for a lowercase letter recognizer of 122 prototypes) require about 2 ms leading to about 400 recognitions per second. The rate for digits is about 650 recognitions per second.

5.4.5 Variations in Parameters

The results reported so far in this chapter only show a subset of the parameters available to SCL recognizers. The question naturally arises: How sensitive are recognizers

to variations in the training parameters? We have conducted tests on nearly every constraint known to SCL: learning rate, momentum, amount of initial training, and many others. Many neural network researchers have also done so. SCL introduces three parameters that are unique to its algorithm: σ_H, ρ, and σ_L. (Actually, the middle of these, ρ, is taken from the ART model.) The following two tables illustrate the effect of varying these unique parameters.

In Table 19 we vary the vigilance (ρ) and the lower training limit (σ_L) for a lowercase letter recognizer. The vigilance directly controls the number of prototypes formed as is evident from the increasing number of prototypes in lower rows. The recognition rates varied only from 96.0% to 97.4%. The first of the five values for the vigilance ρ was entered mistakenly. It is noteworthy that even this extreme value had only a slight effect on the recognition rate, despite its clear effect on the count of prototypes generated.

Table 19. Varying Two Parameters for Lowercase

	Four values of σ_L							
Five values of ρ	**-0.8**		**-0.6**		**-0.4**		**-0.3**	
-0.8	96.3%	(61.0)	96.4%	(62.0)	96.0%	(59.0)	96.0%	(58.0)
-0.2	96.5%	(116.0)	96.8%	(123.0)	96.7%	(112.0)	96.8%	(115.0)
-0.1	97.2%	(131.0)	96.2%	(135.0)	97.2%	(130.0)	96.7%	(140.0)
0.0	96.2%	(140.0)	97.4%	(144.0)	97.1%	(142.0)	96.8%	(147.0)
0.1	97.1%	(146.0)	97.1%	(146.0)	97.1%	(142.0)	96.7%	(146.0)

The next example, shown in Table 20 (1,500 digits), involved variations in three parameters. These are five-way cross validation averages after two shuffles (2.6 Mbprops, about three minutes). Although the number of prototypes increased from 25.8 to 39.0 as ρ

increased, the average performance only varied from 99.3% to 99.7% over the three-dimensional range of parameters.

Table 20. Varying Three Parameters for 1,500 Digits

	Three values of σ_L					
Three values of ρ	**-0.6**		**-0.5**		**-0.3**	
	σ_H = 0.5					
-0.2	99.5%	(28.6)	99.4%	(28.2)	99.5%	(26.8)
0.0	99.7%	(35.2)	99.6%	(35.6)	99.7%	(34.8)
0.1	99.6%	(39.0)	99.5%	(40.6)	99.7%	(39.0)
	σ_H = 0.9					
-0.2	99.3%	(25.8)	99.4%	(25.8)	99.3%	(25.4)
0.0	99.5%	(32.4)	99.5%	(30.6)	99.5%	(31.2)
0.1	99.5%	(35.0)	99.5%	(33.8)	99.5%	(32.6)

Although the test reported in Table 20 was run to show the stability of parameter variations, it also serves to briefly illustrate the experimental setting. By many counts, it is an attractive experiment for so doing. To create the 18 combinations of parameters reported in Table 20, 18 suites of 5 training and testing sequences were conducted. Seven of these 90 sequences attained recognition rates of 100.0% (300 correct of the 300 test scribbles). To increase the speed, we suspended all testing during the preliminary training. Tests were performed at changes of training mode (*newTrain()* to *oldTrain()* for example), and after the final shuffle. We conscript one especially precocious recognizer to illustrate the log entries created by the automatic training facility in Figure 24.

This recognizer was using the last two scribbles of each cycle of ten (cycle members 8 and 9) for testing; cycle members 0 through 7 were used for training. It posted a 99.0% recognition (297 correct of the 300 test scribbles) after 6 seconds of training (first pair of shaded bold lines in Figure 24, and 100.0% after 38 seconds (second pair of shaded bold lines). Ironically, since we only retain the recognition rate attained subsequent to the final shuffle, this sequence was recorded as 99.7% — one scribble refused to yield after the initial training. This serendipitous run also shows the value of averaging the five-way cross validation; it tends to level out unusually good or unusually bad experimental conditions. We belabor this illustration for a final comment about the speed. The absence of testing and the relatively small count of prototypes brought the system up to 15,000 bprops/sec — well above its norm of 12,000 bprops/sec.

```
CYCLE of testers 8-9 (in cycle of 10)
Hid:0, SEQ, a:0.90, feat:0-69, sd:1, sm:0.750: Sun Jul 24 23:36:56 1994
Time: Elapsed:  0:00:04  Now: Sun Jul 24 23:37:00 1994
  48179 b_props; eta: 0.0010;  36 prots Par:0.50,0.00,-0.50; 286 of 300: 95.3%.
Relearning, hid:0, alpha:0.90, seed:1, at Sun Jul 24 23:37:00 1994
Time: Elapsed:  0:00:06  Now: Sun Jul 24 23:37:02 1994
  90011 b_props; eta: 0.0010;  36 prots Par:0.50,0.00,-0.50; 297 of 300: 99.0%.
Time: Elapsed:  0:00:09  Now: Sun Jul 24 23:37:05 1994
 131843 b_props; eta: 0.0010;  36 prots Par:0.50,0.00,-0.50; 298 of 300: 99.3%.
Time: Elapsed:  0:00:37  Now: Sun Jul 24 23:37:33 1994
 550163 b_props; eta: 0.0010;  36 prots Par:0.50,0.00,-0.50; 298 of 300: 99.3%.
Shuffle off 100% of trainSets: Hid:0 alf:0.90 sd:1 at Sun Jul 24 23:37:33 1994
Hid:0, RAND, a:0.90, feat:0-69, sd:0, sm:0.750: Sun Jul 24 23:37:33 1994
Time: Elapsed:  0:00:38  Now: Sun Jul 24 23:37:34 1994
 553691 b_props; eta: 0.0010;  36 prots Par:0.50,0.05,-0.50; 300 of 300: 100.0%.
Relearning, hid:0, alpha:0.90, seed:0, at Sun Jul 24 23:37:34 1994
Time: Elapsed:  0:01:43  Now: Sun Jul 24 23:38:39 1994
1575131 b_props; eta: 0.0010;  36 prots Par:0.50,0.05,-0.50; 299 of 300: 99.7%.
Shuffle off 100% of trainSets: Hid:0 alf:0.90 sd:0 at Sun Jul 24 23:38:39 1994


          . . . and so on . . .
```

Figure 24. Section of Automatic Training Log

5.4.6 Comparison of Feature Sets

Almost all the test results shown so far in this chapter reflect a combined feature set that includes four subsets of features for a total of 70 features with real values between

-1.0 and 1.0. (The exceptions are the 1991 and 1992 tests in section 5.2.1 (Backpropagation versus Fuzzy Logic) which used 63 features.) The four feature subsets are:

Scribble count: (1, 2, 3-or-more coded as a 1.0 in one of the first 3 features).

Energy: (14 x and y velocity, 12 x and y acceleration — 26 features)

Grid: (4 orientations in 3 x 3 grid — 36 features).

Shear support: (high, middle, low crossers; high dot; aspect ratio — 5 features).

One straightforward measure of the relative contribution of features is the magnitude of the weights associated with them by a trained neural network. Larger weights (absolute value) mark features that contribute more to the prototypes' activations. Of course every prototype generates individual weights for the feature vector. Also these weights for the same category will vary from one recognition domain to the next. For example, in the digit recognition domain, the third feature (stroke count of three or more) is ignored. To get a general measure of the relative weights, we trained a mixed domain recognizer; it formed 278 prototypes for the 51 categories. Then we averaged the absolute values each of the 70 features for the 278 prototypes. Table 21 shows these averages grouped by feature set. The average weight for all 70 features was 1.047. The four single features with the largest average weights are single stroke (4.712), double stroke (3.888), aspect ratio (3.166), and middle crosser (2.089). The set of stroke count features have the highest average weights among the five feature sets. (For this table we separated the velocity and acceleration features.)

Table 21. Average Prototype Weights Associated with Feature Sets

4.712	3.888	1.642						
Average for 3 stroke count features								**2.836**
0.797	0.640	0.857	0.632	1.049	0.607	0.340	0.244	0.856
1.002	1.242	0.616	0.474	0.725	1.012	0.459	0.874	0.844
0.526	0.668	0.696	0.717	1.185	0.466	0.877	0.870	0.944
0.470	0.950	0.469	0.439	0.412	0.710	0.636	1.013	0.768
Average for 36 grid features								**0.725**
1.395	1.444	1.864	1.558	1.684	1.654	1.531		
1.768	1.631	1.583	1.692	1.693	1.680	1.632		
Average for 14 velocity features								**1.629**
0.427	0.508	0.542	0.535	0.548	0.531			
0.556	0.517	0.527	0.477	0.472	0.434			
Average for 12 acceleration features								**0.506**
1.290	2.089	0.364	1.143	3.166				
Average for 5 shear support features								**1.610**

The above table gives a helpful view of the relative contributions of the different features. The general pattern is consistent with our observations in other experiments and in other recognition domains. However, there is an opportunistic character to a neural network that masks the redundancy of information. If needed information is available in two places, neural networks tend to grow weights in both places. Therefore, we isolated the features to assess their unique contributions to the overall recognition. The scribble count and shear support features are included in all three of the following cases. To this set of 8 base features is added the energy feature set to give 34 features for the first case. We add to the 8 base features the grid feature set to give 44 features for the second case. Finally, we include the full set of 70 features for the last case.

Each cell of Table 22 gives average recognition (upper) and average number of prototypes generated (lower) during a five-way cross validation. This data pair was created for each of five typical values of ρ. The five average recognition rates are themselves averaged in the penultimate column. These results show that the energy features with 34 features perform nearly as well as the full set of 70 features (97.1% versus 97.9%).

Table 22. Comparison of Feature Sets: Lowercase

	Five values of ρ					**Average**	**Bprops / sec**
	-0.2	**-0.1**	**0.0**	**0.1**	**0.2**	**recog'n**	
Energy	96.9%	97.2%	97.4%	97.2%	96.8%	97.1%	17,100
Protos	113.8	123.2	134.2	142.8	141.6		
Grid	89.1%	89.7%	89.5%	89.6%	88.7%	89.3%	15,300
Protos	86.8	102.8	119.2	130.6	124.8		
Combined	97.7%	98.0%	97.9%	98.0%	97.8%	97.9%	11,100
Protos	109.2	121.4	132.0	140.4	142.6		

Table 23 shows a similar comparison using 3,200 digits as input. Five-way cross validation averages are reported. The values for σ_H and σ_L were 0.7 and -0.2 respectively. Only the *initial* value of ρ is given since the recognizer may raise or lower it during shuffles. Again, the system performed nearly as well with the 34 energy features as with the 70 features (98.8% versus 99.1%). The backpropagation rates shown in the last column of both tables are roughly proportional (except for testing) to the feature counts of 34, 44, and 70.

Table 23. Comparison of Feature Sets: 3,200 Digits

	Five values of ρ					Average	Bprops /
	-0.2	**-0.1**	**0.0**	**0.1**	**0.2**	**recog'n**	**sec**
Energy	98.6%	98.7%	98.8%	98.8%	98.9%	98.8%	24,300
Protos	50.2	54.0	57.6	61.2	66.6		
Grid	96.7%	97.1%	96.5%	96.4%	96.2%	96.6%	19,300
Protos	47.4	52.0	57.0	74.2	55.6		
Combined	99.2%	99.1%	98.9%	99.1%	99.0%	99.1%	14,100
Protos	49.2	52.2	52.4	58.0	64.2		

5.4.7 Comparing Smoothing Techniques and Ratios

Table 24 compares the three methods of exponential smoothing described in Chapter 4. We chose to test them on the (presumably typical) task of recognizing lower-case letters. For each method, we tried three smoothing ratios: 1/2, 3/4, and 7/8. For each ratio, we ran seven tests with increasing values of ρ (hence more prototypes). The right-most column shows the average recognition rate for the seven values of ρ. It gives a comparison of the relative effectiveness of the smoothing methods and ratios in improving recognition. For all 77 tests, σ_H and σ_L were set to 0.7 and -0.5 respectively; only one shuffle was performed.

The second row illustrates another approach to filtering noise out of the data. We run the same seven experiments with the scribble point limit set to 30. This limit for all other rows (and all other experiments reported in this work) is 60. This has the effect of selecting an evenly spaced subset of points from the original scribble (every other point, or every third point, for example). No smoothing is performed on this data.

Table 24. Comparing Smoothing Techniques and Ratios

	Seven values of ρ							Average
Ratio	**-0.15,**	**-0.10,**	**-0.05,**	**0.00,**	**0.05,**	**0.10,**	**0.15**	**rate**
No smoothing								
60 points	95.7%	95.9%	96.0%	95.3%	95.4%	95.6%	95.6%	**95.6%**
30 points	96.1%	96.2%	96.2%	96.2%	96.2%	96.2%	95.6%	**96.1%**
Simple exponential smoothing								
1/2	96.3%	96.8%	96.1%	96.3%	96.0%	96.1%	96.3%	**96.3%**
3/4	96.7%	96.8%	96.6%	96.4%	96.6%	96.4%	96.2%	**96.5%**
7/8	96.1%	96.1%	96.1%	96.2%	96.2%	96.3%	96.5%	**96.2%**
Bi-directional smoothing								
1/2	96.2%	96.5%	96.4%	96.5%	96.4%	96.2%	96.7%	**96.4%**
3/4	96.6%	96.9%	96.6%	96.8%	96.9%	96.8%	96.3%	**96.7%**
7/8	96.8%	96.6%	96.8%	96.5%	97.0%	96.6%	96.4%	**96.7%**
Symmetric smoothing								
1/2	95.6%	95.9%	95.9%	95.8%	95.9%	95.8%	95.6%	**95.8%**
3/4	96.2%	96.1%	96.4%	96.2%	96.2%	96.2%	96.1%	**96.2%**
7/8	96.7%	96.8%	96.7%	96.4%	96.5%	96.6%	96.6%	**96.6%**

We observed that the effects of smoothing are in proportion to the number of points in a scribble. Thus a high smoothing ratio of 7/8 might cause significant distortion of a scribble with only a dozen points. It might be better to increase the smoothing ratio for scribbles with more points and decrease the smoothing ratio for less populous scribbles. First, we used the Viewer application (described in Chapter 4) to examine the distribution of point count in the lowercase scribble population. The distribution is reported below in Table 25.

Table 25. Number of Points in a Scribble

Range of point count	Number of scribbles in that range	Proposed smoothing ratio
0 - 10	0	0
11 - 20	9	1/2
21 - 30	151	3/4
31 - 40	346	3/4
41 - 50	702	7/8
51 or more	2,692	7/8

Few scribbles are sparse enough to cause problems, but those that are represent a possible point of improvement. We implemented a smoothing function that adjusted the smoothing ratio depending on the number of points in the incoming scribble. The adjusted ratios are shown in the right column of the above table. We tested this function in fifteen tests similar to those performed for the other three smoothing methods; the average recognition rate was slightly higher than that of any of them, 97.0%. However, when combined with shearing, slightly better performance (98.0% average versus 97.2%) comes with bi-directional smoothing of 3/4 than from the adjustable smoothing.

This concludes our presentation of the specific elements of SCL recognizers. We turn now to tests with actual SCL users.

5.5 Tests with Children and Instructors

Under the auspices of one of our corporate sponsors, we conducted two week-long test series with members of the end user audience — young Kumon students and Kumon instructors. This section reports the results of this testing.

5.5.1 Testing Kumon Students

The following children's testing was conducted from August 9 to 13, 1993. Children were invited from Kumon Centers in the St. Louis area to complete Kumon worksheets on our pen-based computers. Two machines were used for the testing. "Chagall" is based on a 40 MHz 80386 processor with 16 MB of memory and a VGA adapter connected to a Scriptel digitizing tablet. The Mitsubishi Amity notebook computer is based on a 20 MHz 80386 processor. Neither system has a floating point co-processor. All worksheets had *unsegmented* input, that is, input that is *not* partitioned into one box per expected digit. Thus the recognizer had to segment the data as well as recognize it. This task was simplified by the fact that none of the problems had an answer of more than two digits. (Of course this doesn't mean that children never *wrote* more the two digits, but such gross errors were very rare.)

5.5.1.1 General Performance

Overall, the two machines correctly recognized 4,282 of the 4,470 digits entered on worksheets (95.8%). The recognition rate for individual children ran from 91.8% to 99.5%. For the 11 participants in the incremental training, the average recognition during training was 96.5% and the training rate was 1.39 seconds per character. This latter statistic counts *all* the characters processed during the training phase. Since most of the characters were correctly recognized, only a handful triggered incremental training. Thus the reported average per character is artificially low relative to this slow processor.

5.5.1.2 Summary Results on the Two Systems

Results from the two systems are compared below in Table 26. The mean time between misrecognitions is an important measure of the subjective perception of how frequently the computer blocks the progress of the user. It is not derivable from the recognition rate unless failures are evenly distributed over the test interval. If several failures

occur near each other, this reduces the mean time between misrecognitions without changing the recognition rate. The differences in response time are directly related to the two processor clock rates.

Table 26. Children's Summary Results

Performance Criteria	Chagall	Mitsubishi
Overall recognition rate	95.1%	96.1%
Mean time between misrecognitions	20	25
Average response time	169 ms	332 ms
Longest response time	327 ms	569 ms

Table 27 shows the results of the training for the nine Kumon students and two adults who took part in the unrecognized batch training. These eleven entered 100 digits each; the system then trained for the time indicated. Eight individuals completed some number of Kumon worksheets on Chagall resulting in the recognition rate shown in the rightmost column. In some cases, the worksheet recognition was actually lower than the recognition during the training sequence; one would expect it to rise after the training. However, the worksheet recognition also includes segmenting some portion of the data, rendering the recognition task somewhat more difficult. Thus the overall recognition rate on the worksheets (95.1%) was lower than that of the training sequence (96.5%).

Table 27. Training and Recognition during Children's Testing

	During training			During worksheets		
Student	**Missed**	**Protos added**	**Time (sec)**	**Right**	**Total**	**Rate**
la	2	2	42			
tm	4	3	191			
br	1	1	28			
gn	0	2	44	199	200	99.5%
cr	2	5	100	183	191	95.8%
ka	1	2	45	169	184	91.8%
kn	1	1	27	48	49	98.0%
tm	10	10	345	176	190	92.6%
pa	8	7	214	86	91	94.5%
js	3	4	85	168	180	93.3%
sa	7	9	408	196	203	96.6%
Averages	3.5	4	139	1225	1288	95.1%

5.5.2 Testing Kumon Instructors

From October 4 to 6, 1993, we conducted tests on a small group of Kumon instructors from the St. Louis area. This section reports the results. Recognition performance was less than expected for several reasons elaborated in Chapter 6.

5.5.2.1 General Results

The two systems used for the children's testing were also used here. In a slight procedural variation, the data were collected on the Mitsubishi by our corporate sponsor's representative and later trained and tested on Chagall. Collection problems (incomplete files) invalidated the data of one instructor. The other seven instructors entered 5 samples of each digit, and 3 samples of each uppercase and lowercase letter. The instructors were

asked to enter cursive characters, although not all did so. These scribbles (206 for each instructor) were used for incremental training in a manner similar to that of the children shown in Table 27 above.

5.5.2.2 Specific Results

Table 28 gives the specific results of incremental training on the data. The results for the numeric and lowercase recognizers are shown first. The original results for the mixed (uppercase, lowercase and numeric) recognizer are shown in the second (shaded) band of rows. The results for particular subsets of the domain are shown below the overall performance. These results are based on a recognizer that attempted to recognize all 62 possible categories of these three sets. Adjusted mixed recognizers (developed after the testing) make no effort to distinguish between uppercase and lowercase *c*'s, number *0*'s and letter *O*'s, and so forth. (Since no size data is retained in the training set, such distinctions are virtually impossible.) The last row (unshaded) shows the modestly better results with this adjusted mixed recognizer.

Table 28. Recognition and Training Time during Instructors' Testing

Recognizer domain		Time (min)	Seconds/character
Numeric	98.0%	2.0	1.7
Lowercase	78.0%	14.0	10.7
Mixed (overall)	65.3%	50.2	14.6
Mixed (uppercase)	60.4%		
Mixed (lowercase)	52.7%		
Mixed (number)	92.3%		
Mixed (overall, adjusted)	75.2%		

We observed a strong bimodality in the instructors' data which may be associated with the degree to which they complied with the request to write in cursive. (Few of the

above recognizers have many cursive exemplars.) We discuss this interpretation and its implications further in Chapter 6.

5.5.2.3 Extended Incremental Training

We experimented with different training parameters to permit about twice the training on missed characters. These and similar tests with training up to 50,000 bprops per missed scribble did not significantly improve the performance, but commensurably increased training time. We turn from user testing of SCL to the final section of this chapter, results developed from extensions to the basic SCL model.

5.6 Results from Extensions to SCL

A number of researchers have extended aspects of SCL in promising directions. These are divided about equally between NSF-sponsored Summer Undergraduate Research Associates (SURA), and Master degree students. The work of Wang with fuzzy logic in SCL was reported earlier in section 5.2.1 of this chapter. This section briefly reports some findings of others.

5.6.1 SCL and Gestures

Taysi [164] used an early version of SCL to build a gesture-based graphical editor for a simplified version of the visual programming language Hyperflow [92]. His program, Box and Arrow Editor (BAE) used 34 gestures for all aspects of the editor (opening and saving files, editing, aligning, asking for help, superimposing a grid on the editing space, and so forth). He collected 5,460 gestures from 23 children, using 3,120 to train the SCL recognizer and the remaining 2,340 to test it. SCL showed 85% accuracy on the test data after training. While training, the recognizers formed from 80 to 147 prototypes.

As Taysi notes, the system's recognition performance fell into two clear categories. For several very closely related gestures (especially the "line," "dashed line," and "thick line" gestures which were very close in appearance and drawing execution), SCL was unsuccessful in consistently disambiguating the patterns (60% or worse recognition rates). For the other gestures, SCL was able to achieve nearly 100% recognition. His work also showed the difficulties associated with gathering consistent data from children (including some as young as eight).

5.6.2 Spoke-Based Features

Ben Whaley proposed a set of features based on the distance of the stroke from the center of the scribble along a set of radial spokes. He used the SCL architecture with these features to recognize digits with a rate of 96.7%. For more information, see his SURA report [180].

5.6.3 Hierarchies of SCL Recognizers

We worked with Vitaly Shmatikov [151] to develop a hierarchy of trainable recognizers, which would hopefully combine the best of each in the recognition task. This also permits direct comparisons of the effectiveness of various feature abstractors. Each recognizer within a meta-recognizer (using its own feature set) is given a meta-weight corresponding to its effective contribution to the recognition. This gives a quantified measure of the relative value of feature vectors. He examined three families of features: (1) the Grothe grid, (2) energy-based features, and (3) meshing of points into 2x2, 3x3, 4x4,. . ., 8x8 grids. In the last case, the features were proportions of total point count lying in each cell. His results were not conclusive on the relative value of combining features through such a hierarchy versus simple concatenation of the feature vectors. He found the

energy-based feature set to be significantly stronger than the grid-based feature set. Among the mesh features, the strongest were the 4x4 and 5x5 grids.

5.6.4 Segmenting Digits

For his undergraduate thesis , Khalid Sherdil investigated the segmentation of multi-scribbles that were ten-digit telephone numbers. His algorithm used both specific characteristics of digits (for example, horizontal strokes in *5* and *7*), and general functions (such as intersection detection and slope analysis) to break the multi-scribble into individual scribbles containing one digit each. His work showed "how several of our segmenting techniques, such as relative stroke lengths, relative stroke positions, order of stroke entry, stroke direction, stroke intersection, etc. can be combined to yield results of about 95%" in successfully segmenting the input scribbles. For details, see his thesis [149].

5.6.5 Segmenting and Recognizing Fractions

SURA student Ki-Wing Ho built a recognizer that could recognize a fraction or mixed number [71]. It resolved the scribble into its whole number, numerator, and denominator. All 280 whole numbers of her test set were recognized as non-fractions and thus were not segmented. All but 5 of the 280 test fractions were recognized as fractions. Thus, her program achieved a 100% success rate in the recognition of non-fractions, and a 98.2% rate in the recognition and segmentation of fractions into parts.

5.6.6 SCL on Digital Signal Processors

A distinguished group of graduate and undergraduate students labored long but unsuccessfully to implement SCL on digital signal processors. Two DSPs were used during the course of the investigation: the Motorola 56001 rated at 13.5 MIPS and the Analog Devices ADSP 21020 rated at 50 MIPS. The principal researcher was Ed Hutchins,

and secondarily, the author. Neither DSP had sufficient on-board memory to handle SCL. Therefore Dale Frye designed a supplemental board of memory. Ed, Han Ko, Asmat Tanauli, and the author attempted to get the memory working successfully. Often success was tantalizingly close. The DSP did actually run SCL; it recognized many digits, and even successfully trained on new styles of digits (such as "snowman eights"). With custom optimization of the assembler code (by Ed), it could run 12,000 backpropagations per second (with 70 features). Regrettably, we were never able to eliminate the intermittent address bus errors that caused random miswrites to memory. Given the remarkable strides in CPU technology, it is likely that more and more of the tasks conceived for the DSP will move into the CPU.

5.6.7 Other Platforms

SCL has been ported to a wide variety of different platforms. Weilan Wu ported it to Sun OS on the Sun 4. With direction from Wolfgang Pree, Tim Brown and he ported it to Niklaus Wirth's Oberon Environment with an interface in ET++. Mahesh Thermal ported SCL to Solaris on a SPARCstation 10 using the pen-oriented application programming interface described by Kempf [87, 88]. The author has ported it to DOS and PenPoint environments using Intel 8080-based processors in such pen-based systems as Mitsubishi Amity, NCR 3125, Fujitsu FM PenNote (FMPenG5), and others. We also ported SCL to OS-9 on custom hardware fabricated around the Motorola 68030 CPU with a PC bus.

This completes the reporting of our results. In Chapter 6, we evaluate and interpret these results and in Chapter 7, we state our conclusions drawn from these results.

6. Evaluation

This chapter evaluates and interprets the results reported in Chapter 5. We begin with tests of SCL recognizers — first the general tests (section 6.1 — Evaluation of General Experiments), then the focused tests with end users from the Kumon community (section 6.2 — Evaluation of Application Tests). This prepares us to address the principal question in 6.3 (Evaluation Relative to the Problem Statement) — the degree to which the methodology of SCL satisfies our research goals. Section 6.4 (Comparison of SCL to Other Models) compares SCL to similar systems. We briefly consider extensions to SCL in section 6.5 (Relation to Other Research). Finally, we comment on the relation of this work to teaching and learning (section 6.6 — Learning Systems and Human Learners).

6.1 Evaluation of General Experiments

In this section, we evaluate SCL as a *general* recognition engine. In the next section, 6.2 (Evaluation of Application Tests), we evaluate it in a specific application setting.

The design of SCL grew from the extensive experimentation reported in the last chapter. We trace milestones along the path of SCL's empirical evolution: broad design elements (learning unit algorithm) and lesser elements (filters and feature abstractors).

6.1.1 Backpropagation versus Fuzzy Logic

This section reviews the comparison of these two potential learning algorithms as presented in Tables 2 through 4 of Chapter 5. The advantage of fuzzy logic is fast training. It only entails updating attribute values for the prototypes. However it lacks a *negative* training mechanism to teach an eager prototype to reduce its response to exemplars from

other categories. Not surprisingly, backpropagation prototypes (98.1% to 99.7% for 1,500 digits) performed somewhat better than those based on fuzzy logic (95.7% to 97.2%) for just this reason. It was surprising that the training times were virtually identical for the two approaches on the same hardware. This is likely due to the larger number of prototypes generated by the fuzzy logic learning units (88 to 157 prototypes for fuzzy logic, 35 to 37 for backpropagation).

The increase in prototypes might be related to the absence of negative training. Since the units cannot reduce their response to patterns of other categories, the only avenue to successful discrimination is higher vigilance (ρ), which in turn begets more prototypes. These results led us to choose backpropagation-based prototype learning units.

6.1.2 Complexity, Learning System Architecture and Performance

After experiments (not reported in Chapter 5) with recurrent neural networks (following Servan-Schreiber, Cleeremans, and McClelland [148]), we found that we could maintain comparable or better performance with simpler prototype units. In fact, the individual prototype units could be reduced to no hidden units, or linear classifiers. This is emblematic of the trend found throughout this investigation and depicted in Tables 5 through 8. As the current minimalist SCL evolved, the shared hidden units, and later all hidden units, were removed. As preprocessing and scheduling of training became more sophisticated, the complexity of the prototype recognizers was reduced without loss. The reduced complexity fully compensated the additional time required for preprocessing and testing for inclusion in the training schedule; in fact, the overall training and recognition times fell significantly.

Earliest experiments with traditional neural nets (for example, 70 inputs, 10 outputs for digits) suggested the value of hidden layers. Some input patterns are simply not linearly separable, so hidden layers permitted the learning of these non-linear functions. As our technology matured, the hidden layers became superfluous. Indeed, our most recent experiments show that SCL performs better without hidden units — and is an order of magnitude simpler, hence faster — as is shown in Table 9.

We conjecture that additional prototypes take the place of hidden units. Consider a traditional neural net learning to recognize a standard eight and a snowman eight. Presumably the two are linked to each other via an intermediate representation in the hidden layer. In SCL, the two settle into different prototype learning units, but both bear the category name "eight." We further conjecture that not all prototypes require the complexity of hidden units to achieve their classification task. A common architecture among the prototypes (with hidden units) unduly penalizes those that do not need them. By allowing more prototypes, but no hidden units, the classes that require more complexity may build it by proliferating prototypes; those that do not, get by with fewer prototypes.

No single design element accounts for all the improvement, but we mention several milestones. Earliest SCL models used the scribble's aspect ratio to distinguish between strongly vertical characters (the digit *1* and the lowercase letters *l* and *i*) and others with a more conventional aspect. After testing several ratios (generally about 3.0), we found that half of the target characters were slipping through the aspect ratio filter (for example, crooked *1*'s and curly *l*'s), and half the strongly vertical characters caught by the filter were not target characters (such as tall thin *9*'s, *7*'s or *g*'s). Our recognition performance improved when we abandoned direct algorithmic efforts to differentiate based on aspect

ratio, and left this to the network itself. The aspect ratio became an additional real-valued feature.

Energy-based features divide the input scribble into some number of sections for averaging. If the input scribble has two (or more) strokes, it is possible that one of the sections contains pieces of both strokes. Since many characters have only one stroke, this practice was less than disastrous (and simplified the preprocessing). Later versions of SCL adjusted the velocity and acceleration sampling to coincide with stroke boundaries, and thus to disregard the implicit acceleration associated with the movement from the end of one stroke to the beginning of another. (This movement may be considered a "virtual" stroke.) This too added to the preprocessing burden, but improved training and recognition performance.

We conducted many experiments involving various schedules of exemplar competition and training (including one, two and three shuffles with appropriate training in between). No significant difference emerges among different numbers of shuffles, but at least one is highly beneficial as shown in Figures 17 and 18. (This is also evidenced in Tables 6 through 8 and Tables 19 and 20 which all involve shuffling.) Although shuffling and subsequent training adds to the overhead of the training task, shuffling has been a major contributor to simplified prototype units and rapid weight convergence. It seems to allow a more refined response to useful outlier exemplars without penalizing the successful generalization of the more "mainstream" prototypes. The outliers are shaken loose from the "mainstream" prototypes and congregate in their own prototypes. Since these outliers by definition contain fewer exemplars, they demand very little training time. It affords another mechanism for managing complexity selectively without penalizing the whole system.

Although considerable experimentation was conducted with varying values of the learning rate ε and the momentum α, our main interest is in the learning system architecture. Other researchers have done much more work in the areas of decreasing learning rates and subtle selections of momentum. We leave that field to them.

6.1.3 Digits, Letters, and Combined Recognizers

The average recognition rates (over five-way cross validation) of the recognizers were 99.2% for digits, 98.0% for lowercase, and 96.5% for mixed domain. (These averages were shown in each of the paragraphs following Tables 6 through 8.) This meets the recognition performance demanded of this research. These rates *for writers on which SCL has trained* are comparable to or slightly better than the performance of humans for context-free recognition [122] and of course much faster. The most frequent error made by the mixed recognizer, five single-stroke *5*'s classified as *s*, is also likely to be made by humans. The inclusion of energy-based features permit SCL recognizers to "see" the data in a spectrum partially different from human observers. For this reason, SCL seems uncanny in some discriminations and dumb in others.

A harder task, and one not directly demanded by our research goals, is the recognition of writing from unknown writers. Our research principally addresses adaptation to users. Nonetheless, SCL recognizers perform quite well on *digits* from writers for which it has not been trained (such as the Kumon students who took part in the children's tests). SCL recognizers do not perform nearly as well as humans do with unfamiliar writers in the alphabetic domains. This is at least partly due to the many variations in the formation of lower and uppercase letters. A recognizer with few or no cursive exemplars is utterly nonplused by cursive input as is seen in the Instructors' Tests (Table 28).

6.1.4 Comparison of Feature Sets

Earlier tests (not reported) showed a considerable gap between the combined feature set and the energy-based and grid-based subsets. For example, with lowercase letters, the combined feature set achieved just over 95%; the energy-based feature subset reached 91%, and grid-based features never reached 90%. The average prototype weight analysis in Table 21 indicates that the energy-related feature subset is the more influential of the two. The tests of the two subsets in Tables 22 and 23 confirm this. Energy-based features proved unexpectedly close to the combined feature set in tests with lowercase letters (97.1% versus 97.9%) and digits (98.8% versus 99.1%). Since an energy-based recognizer runs nearly twice as fast as a combined one, such a recognizer might be a viable option where space or time constraints preclude the larger model. For the general case, the combined features remain the strongest choice.

6.1.5 Comparison of Filters

Our experiments (both those reported and others) confirm the value of both the shear transformation and the smoothing of scribbles. The two seem to work best in combination. We ran tests with smoothing preceding shearing and vice versa and other tests with one or the other but not both. (These tests were not reported.) The best results came from smoothing first and then shearing. Table 24 showed small differences among smoothing ratios and methods, but the edge was to bi-directional smoothing with a ratio of 3/4.

6.2 Evaluation of Application Tests

This section evaluates and interprets the results of the two tests involving end users from the Kumon mathematics learning environment. Children serve as prospective regular users, and Kumon instructors represent intermittent users such as teachers, parents, or administrators.

6.2.1 Children's Tests

As reported in Chapter 5, the two machines correctly recognized 4,282 of the 4,470 digits entered on worksheets (95.8%). All worksheets had unsegmented input that included either one or two digits in the single answer box.

6.2.1.1 Individuality versus Overall Recognition

Statistics were retained for individual writers on Chagall (Table 27), but not on the Mitsubishi Amity computer. The overall recognition rate of 95.8% is potentially misleading; the recognition rate is highly individual. Some students (such as gn) were able to move very quickly and achieve high recognition (100% in the training set and 99.5% during the worksheets). Others (such as tm) had recognition rates near 90%. The difference often was caused by a single character. For example, tm makes *0*'s by starting at the top and drawing the circle counterclockwise about 450 degrees. This precisely mimics the velocity and acceleration of the numeral *6*. Virtually all his errors (indeed, a third of the errors for that whole day of testing!) were caused by this substitution of *6* for *0*.

This suggests the effectiveness of custom recognizers for young users. The author has watched with interest a variety of numeral creation wider than he imagined possible — "snowman eights," "lollipop nines," and "inside-out eights" that grow from the center. New writers may create nines and fours from the bottom up. They may add a number of strokes to "polish" the numeral on the writing surface. We saw one first-grader, obviously concerned about the linearity of his writing, using his left forefinger as a straightedge while drawing 4's with his right hand (on paper worksheets). Again, the only sure path to accommodate such wide diversity appears to be custom recognizers for everyman. This also accommodates the dependence on digitizers, device drivers, and scribble collection software.

6.2.1.2 Incremental Training

As we noticed that these idiosyncratic characters caused misrecognitions Tuesday and Wednesday of the week of testing (snowman eights, 450-degree *0*'s, down-then-up pointed *4*'s, and others), we changed the training function Thursday (August 12) to permit twice the training (20,000 bprops instead of 10,000) on such unusual characters. This might have improved the recognition of these characters during these earlier tests. It doubled the training time. Still, we stayed very close to the stated goal of incremental training. (The goal was one second per character. The average for August 9 through 11 on Chagall was 1.1 seconds per character.) Of course, the use of a DSP or comparable processor will reduce the time thirty-fold.

6.2.1.3 Parallax

During Monday's testing, no correction was made for the extreme parallax error of Chagall: the pen "wrote" 8 mm below its apparent position. This caused the child to dynamically alter her writing to make it conform to the expected appearance. Starting Tuesday midmorning, the children began each session by aligning the pen before entering any characters. Starting Wednesday, the children began by just doodling on the screen. It takes a few seconds to get the "feel" of the surface; they all commented on the difference between writing on glass and writing on paper. These procedures were not followed on the Mitsubishi Amity system (whose parallax is only 2 mm).

6.2.1.4 Ergonomics

The chairs and tables that we used were designed for adults. This left the smaller child at a very low angle to the glass surface of the digitizer. This low angle reduced the clarity of the screen (to a surprising degree) and worsened the parallax problem. One child wanted to stand and another chose to kneel in the chair to get a better view of the

worksheets. In further testing and in actual usage, these accommodations of the child should be taken into account.

One student (sa) chose to kneel in the chair, but after a while she slumped down. Most of the 7 errors (of 203 entries) occurred when her chin was actually resting on the computer, resulting in rather extreme parallax! In spite of this, her unusual fives (the top stroke is drawn first) were all correctly recognized, as was a single two-stroke nine. Actually, most of the errors occurred when three numbers were entered instead of two. For example a tall *1*, a high short *1,* and another tall *1* were segmented and recognized as 14. Since the segmenter was constrained to one or two digits, 14 is a good guess, but of course not what she wrote.

6.2.1.5 Pen Concerns

We noticed that the tilt of the pen relative to the plane of the writing surface had an unexpected effect on the recognition. When the pen was angled at lower than about 35 degrees from the surface, the tip occasionally lost contact with the surface. This caused a single stroke to appear as multiple strokes. For example, a nine might be recognized as a 0 followed by a 1. This is more problematic in unsegmented input (as was done in this test), but would cause problems in segmented recognition as well.

This problem is exacerbated by the tendency for young children to doodle in the drawing space, to "scratch out" numbers by writing several zigzag strokes over the erroneous entry, and to "touch up" characters after writing them. In earlier tests, we noticed that when a child's writing was not recognized, she tended to slow down and write more carefully. This is based on the natural assumption that the computer will be better able to read her writing if it is drawn more carefully. In fact, the velocity and acceleration features

are stymied by this slowdown and the recognition often got worse as the subject slowed down. This caused even slower entry.

During this test (August, 1993), this problem was not noticeable, perhaps due to somewhat better recognition in the first place. Perhaps the fact that the children were working on familiar-looking worksheets caused them to work them at accustomed speed instead of paying undue attention to the computer.

6.2.1.6 Time-Out and Recognition Speed

The speed of recognition shown in the statistics differs from the speed seen by the subject or an observer. SCL measures the time from receiving the scribble (from the Pen-Point input subsystem) until returning the recognized digit to the display subsystem. The input subsystem includes a "time-out" parameter that was set to 600 ms for this testing. Thus the scribble (or scribbles in the case of two digits) were not sent to the recognizer until 600 ms *after* the pen moved out of proximity. Some children held the pen very close to the screen after the scribble was complete, further delaying the delivery of the scribble. Interestingly, a few children intuited that this slowed things down, and learned (without prompting) to move the pen away to hasten the recognition. Of course, many children did not wait for digit conversion and just ran ahead at their natural speed irrespective of the recognition of past answers.

In actual use, it would probably be best to neglect proximity completely, and to adjust the time-out to individual writers. Small children need about a second or more to construct two-stroke characters (like 4) without having the system intrude by recognizing the initial stroke as one. For twelve-year olds, perhaps 200 ms suffices. This would significantly increase the user's *perception* of recognition speed. One solution is to measure the time between the strokes of two-stroke characters and adjust the time-out dynamically

to three or four standard deviations above the average; this would enable the system to operate quickly and predictably for nearly 100% of the writer's characters.

Recognition speed is dependent on the number of points sent into the feature extraction functions. Normalization currently reduces the point count to 60. This could be reduced even further with little penalty to recognition performance if more recognition speed is needed. The smoothing test reported in Table 24 showed that recognition was only reduced half a percent by the reduction from 60 points to 30 points, despite the fact that most scribbles have 50 or more points (as shown in Table 25). However, this is not likely to be necessary; 400 recognitions per second on the target processor hardware should be more than ample.

6.2.2 Instructors' Tests

The results in the instructor testing fell quite short of our expectations. There were several reasons for this, but it also indicates an area where SCL methodology is not strong (and for which it is not designed) — recognition of scribbles by writers for whom it has not been trained.

The data suffer from several defects besides poor performance from SCL. The writers (instructors) were told by the test administrator to use *cursive* writing. Few cursive prototypes have been used in training; we have always asked our subjects to print, but a few have a rather cursive style for one or more letters. The instructor files were collected on a different machine (Mitsubishi Amity) than the training set. The data for one of the eight instructors was incomplete, so testing could only be done for seven instructors.

SCL performed poorly on combined recognition (shown in Table 28) for several reasons. First, the target set of categories is larger (upper, lower, and numeric = 62

categories) so the task is intrinsically harder. Second, many character pairs of this set are not differentiable even by humans (numeric *0* versus uppercase *O*, numeric *1* versus lowercase *l*). Third, several characters are differentiable only by relative size (such as *C/c*. *O/o*, *P/p*, *S/s*, *V/v*, *W/w*, *X/x*, *Z/z*) and SCL has no provision for comparing received size with expected size. Some characters were "misrecognized" only by case. The later version of SCL which adjusts for these ambiguous characters caused a modest improvement in the performance (seen in the last line of Table 28) on the mixed (uppercase, lowercase and digit) data for all but one of the instructors.

Table 29 illustrates the strong *bimodality* of the performance data. As stated, all instructors were told to use cursive writing, but not all did. At least some of the difference in the two modal groups is caused by the fact that some instructors complied with the cursive request and others did not.

Table 29. Instructor Test: Bimodal Recognition

Instructors	Digits	Lowercase	Mixed Unadjusted	Mixed Adjusted
T1	100.0	89.7	74.3	84.5
T2	99.0	83.3	67.5	80.1
T4	99.0	82.1	67.5	81.1
T6	100.0	91.0	75.2	89.0
Average (4 instructors)	**99.5**	**86.5**	**71.1**	**83.7**
T3	99.0	74.4	55.8	68.4
T5	97.0	48.7	52.9	61.2
T7	99.0	75.6	63.5	61.7
Average (3 instructors)	**98.3**	**66.2**	**57.4**	**63.8**
Average (7 instructors)	**99.0**	**77.8**	**65.2**	**75.1**

Figure 25 is the uppercase data from instructor T6 who obviously did not try to write uppercase cursive letters. (The first 'U' was omitted.)

Figure 25. Uppercase Writing of Instructor T6

Figure 26 is the uppercase writing of instructor T5 who tried to write cursive uppercase letters. The interview revealed that she does not ordinarily write uppercase cursive. The inconsistency of letter formation is especially evident in the *A*'s, *F*'s, *H*'s, *P*'s and *Q*'s. The second *Q* has only one stroke; the other two have two strokes. The third *T* was partially lost.

Figure 26. Uppercase Writing of Instructor T5

Figure 27 shows her lowercase writing. Again, the inconsistencies of an unaccustomed style are evident in the *b*'s, *c*'s, *g*'s, *k*'s, *v*'s and *w*'s.

Figure 27. Lowercase Writing of Instructor T5

The fourth of the reasons for the weaker performance is the narrowness of the subject data base for the mixed domain. The digit recognizer grew from a data base of numerals collected from 32 subjects. It probably has examples of most of the ways in which numerals are drawn (although with children, one could never make such a statement absolutely). The lowercase recognizer is based on only fifteen subjects, and uppercase on only six subjects. Furthermore, letters appear to be more varied in execution than numerals. Thus, we would need a *larger* base of subjects for letters than for numbers to get equally strong results; instead, we have a much smaller subject base.

Also, it must be remembered that SCL recognizers are designed to adapt to users. This is not identical to high performance in "walk-up" recognition. It is likely that the adaptation requires a few (reasonably consistent) examples to acquaint the recognizer with the new character style.

These difficulties notwithstanding, this remains an area where SCL could stand some improvement. Energy-related features should be more reliable when users are writing in their accustomed mode, than when writing in an unaccustomed mode. Thus SCL would perform better in the general case where the user is doing just that. Another approach in such cases might be to abandon the pre-trained recognizer altogether, and to build a new recognizer from scratch.

6.3 Evaluation Relative to the Problem Statement

We evaluate the results in light of two formulations of the problem: the formal problem specification and the general problem abstraction.

6.3.1 Formal Specification Elements

First we consider in turn the eight elements of the formal problem specification in Chapter 2.

6.3.1.1 Pen-Based Technology

This pen-based technology — the paradigm of silicon paper — encourages us that the graphical and computational power of the computer can be employed without sacrificing the learning that inheres in the *writing* of mathematics. After a few minutes of practice (and when the digits are correctly recognized), the students appear to work on the computer as quickly and confidently as they do on paper. We offer this opinion cautiously, based on our observations; specific testing would be required to verify this. When digits were misrecognized, the students slowed down significantly.

6.3.1.2 Low Cost

A processor capable of 20 to 50 MIPS (roughly 4,000 to 12,000 bprops/sec) keeps training in the one second range. A training interval of 10,000 bprops was the longest of Table 10 or Table 27 (excepting the experimental increase on the last day). The training intervals are generally much less. The lower end of the scale (20 MIPS) is acceptable if occasional (one of five) episodes may go to three seconds. This is the performance of i486 DX2 CPUs (3,100 bprops/sec). Several 150 MIPS processors were released this year; it seems most likely that low cost processor technology in the 50 MIPS range will soon be available.

6.3.1.3 Wide Range of Subjects

Our 100-plus subjects included students from kindergarten through college and a number of adults with a balanced representation of all groups. Our writers hail from more than two dozen countries on four continents. Although we did not specifically test this hypothesis, differences *within* a national group seemed broader than differences *between* national groups. With both off-line and on-line testing of Kumon students, SCL achieved digit recognition of 96%.

6.3.1.4 Wide Range of Symbol Sets

For many tests, SCL was trained from multi-subject batches of character sets including numbers, lowercase letters, mixed digits/lower/uppercase letters, and a set of 34 gestures. It reached recognition rates of 99.2%, 98.0% 96.5% and 85% (nearly 100% if ambiguous gestures are omitted) for these four recognition domains respectively. This performance meets our research goals.

In training from scratch, SCL learned digits, Roman numerals, Japanese and Sanskrit with nearly perfect performance. In none of the tests did it miss more than one character (after the first exemplar for each category). We find this to be additional, though anecdotal, evidence that SCL performs satisfactorily in a wide range of symbol sets.

6.3.1.5 Usage

The batch tests (Tables 6 through 8, 19 and 20) involved from 15 to 53 subjects. In these cases, SCL could only reach acceptable levels of recognition on test data by retaining its ability to respond to nearly all the subjects since the test data always included the same number of scribbles from every subject. In cases where an existing recognizer was trained on a new writer's scribbles (Figure 20 and Table 11), the final recognizer retained its high response to scribbles from the original subjects.

An important exception to this retention occurs if the new data is collected with differing digitizing hardware and software (Figure 21 and Tables 12 and 13). SCL recognizers appear to be strongly dependent on the digitizing system.

6.3.1.6 On-Line Recognition and Adaptation

The goal was responses and adaptations in about a second — in a time span generally below the user's threshold of distraction. Responses are well below half a second even on i386 hardware (Table 26). As explained in section 6.3.1.2 above, the training times are comfortably within this range on 50 MIPS processors and almost always within this range on 20 MIPS processors.

One problem that remains no more than partially solved is segmentation of the incoming stream of scribbles. As mentioned in section 6.2.1.6 above, more work is needed on the time-out delay (segmenting by time) that causes the recognition to appear unacceptably slow to the subject. Better segmentation algorithms are needed.

6.3.1.7 Self-Criticism

The distribution of confidence reported in Tables 14 and 15 in Chapter 5, the strength of second (99.1%) and third guesses (99.4%) in the mixed domains (the most difficult), and the rank correlation coefficient in particular (0.988 average) provide strong evidence that SCL recognizers give a reliable measure of the likelihood of correct responses.

6.3.1.8 Context-Free Classification

Since every recognition event is independent, the system has to recognize each digit, letter, or symbol on its own merits. In none of the testing was the recognizer furnished any context data other than the recognition domain.

6.3.2 General Problem Abstraction

We now evaluate the results in terms of the three abstractions of the problem developed in Chapter 2.

6.3.2.1 Stability versus Plasticity of Adaptation

As explained in several elements of the preceding section, SCL recognizers demonstrated adaptation with retention in many settings. The exceptions indicate that they cannot adapt well to other digitizers, device drivers, filters, and preprocessors. They gather prototypical exemplars with evident similarity (Figures 22 and 23).

The speed and success of training from scratch (Figure 19 and Table 10) have led us to an unexpected approach. We have long assumed that the best route to customization is to provide a well-trained recognizer with prototypes generated from a wide range of subjects; then the recognizer would be incrementally trained on specific examples from the new user as necessary. It is assumed that relatively few new prototypes would be necessary. For small target sets (such as numbers), and especially in cases where the user's writing is not likely to be "standard," (such as small children), it would be better to create a new recognizer from scratch for each new user. The recognizers would be much smaller. In the "train from scratch" experiments, the recognizer usually generated exactly as many prototypes as categories, and at most, only one additional prototype beyond the category count was created. There also may be a certain appeal to our young audience in having a completely custom recognizer for every user — in effect, a special edition for each.

To accommodate the interests of occasional users as well as young writers, a compromise solution is to build a standard letter-form base recognizer with relatively few prototypes for each category. Then both occasional users and regular users could add to the recognizer just what is necessary of individual non-standard forms.

6.3.2.2 Two-Body Learning Problem

Figure 28 below furnishes a thumbnail illustration of the two-body learning problem. This is a series of *5*'s collected from one of the Kumon students. The rather large parallax error (8 mm) caused her to write the second stroke of each *5* (the top stroke) well above the top of the first stroke. Over the next few fives, the second stroke settled down on the first as she adapted to the parallax of the system faster (since this was off-line) than the system adapted to the user. There was about a second between each *5*; thus in a few seconds, she adapted sufficiently to produce what she perceived as correct *5*'s. The fourth 5 illustrate a rather novel approach to getting around the parallax problem. She omitted the second stroke completely, but this was probably only coincidental since she applied it to the next five *5*'s.

Figure 28. Adaptation by the User

Obviously, we cannot solve the two-body learning problem by hoping or insisting that humans will not adapt. If the recognition is fast enough and correct, the user has little time to dwell on the formation of the scribble, but is quickly presented with the finished product: the character in type form. When the system does fail to recognize the character, it is important to avoid asking the user to re-enter the errant character; this just invites adaptation on her part. This is the significance of the alternate characters presented by the interface shown in Chapter 4. The mixed set of digits, lower and uppercase letters represents the most difficult domain extensively studied. Since the system will present the correct character 99.1% of the time as the first guess or one of the first two, (and 99.6%+ in

the first five guesses), the user can choose the correction by tapping the alternate. Thus, the system trains on the scribble originally input and sidesteps the invitation to adapt.

If the training imposes too great a burden on the user, the desire to adapt (to avoid the delays of training) increases. The very rapid training cycle (on the order of a second) is the principal method to let the system adapt to the user and not vice versa. Even if the user does adapt (modify) her next scribble of the training-demanding category, the training round now includes the first (unadulterated) scribble and the second (possibly user-adapted). The second will likely be accepted as the intended category and the user will presumably gravitate back in the direction of the natural formation.

The detection of chaotic oscillations in a two-body learning system is not hard. The measurement of the degree to which a user is adapting to the computer is very difficult. The user has already made unmeasurable accommodations just to the different kinesthetics of writing on glass instead of paper. Certainly fast, accurate recognition and fast training at speeds below the distraction threshold of the user are a start. This is an area where cognitive scientists and computer scientists could fruitfully conduct further research.

6.3.2.3 Managing Adaptation with Confidence

As explained in section 6.3.1.7, the distribution of confidence reported, the strength of second and third guesses, and the rank correlation coefficient in particular (0.988 average) provide strong evidence that SCL recognizers give reliable measure of the likelihood of correct responses.

This confidence can be used to manage plasticity in several ways. It can be used as a limit on the acceptability of incoming scribbles. If the user's entry is not recognizable

above some threshold confidence, it might be rejected to avoid promoting non-standard character forms. This confidence threshold might be adjusted appropriately for writers with special difficulties. Reliable confidence also contributes to powerful algorithms for multi-scribble segmentation, and well-refined search templates to scan databases.

We provide an example where confidence might be used as a check on the acceptability of a new character. In one experiment (not reported in Chapter 5), we simply asked a training recognizer to learn a clear d as the category b. After several thousand backpropagations, the system never quite agreed to accept the d as a b. The earlier d prototypes responded with about 0.99 while the interloper generated an activation of about 0.96. In this case though the confidence was only (0.99 - 0.96) / 2 or 0.015. In such a case, the recognizer might respond that the new scribble is unacceptable because it is too close to scribbles in other categories. This keeps the performance of the recognizer stronger by screening out potential sources of ambiguity.

6.4 Comparison of SCL to Other Models

The distinctive aspect of Supervised Competitive Learning is precisely its "competitiveness." In all competitive learning systems, prototype units compete to recognize the input pattern. SCL units also compete to gain a collection of exemplars, and they compete for training time. Units with more exemplars (larger $|B_j|$) receive proportionately more training (until reaching the σ_H and σ_L criteria of course). The system is self-initializing. It builds classes from the stream of incoming patterns, and does not depend on "exemplary" exemplars to perform robustly as do many pattern recognizers.

6.4.1 Evaluation Relative to Related Learning Architectures

Here we compare the SCL model and the results of the SCL system to other learning architectures. We begin by admitting that our comparisons are based only on several published articles for each (and in the first two cases, two patent documents for each). Deeper comparisons would require implementation details and tests on identical data that are beyond the scope of this research. This is an area when the Unipen consortium will be very helpful.

6.4.1.1 The ART Model of Carpenter and Grossberg

ART [24] performs unsupervised, competitive learning (like Kohonen's feature-map forming model). It does not use the backpropagation training rule of Werbos and others, although Sørheim's variation uses both ART and backpropagation elements for real-valued function simulation. The usual ART (without attached backpropagation nets) uses top-down expectations (Long Term Memory adaptive weights) to multiply ("gate") the bottom-up short term inputs. The training adjustment is a form of steepest descent adjustment of the winning weights. When an output node is disabled, its top-down traces are turned off, and other output nodes form expectations. The hypothesis testing happens much faster than the resonant state which permits learning to take place [24:80].

The top-down expectations modify the attentional gain control. When a set of short term traces is activated each attention unit (prototype) compares the activation to the expected LTM values; if the match is poor, this unit is turned off. If an uncommitted unit remains, it is strengthened to correspond to the new pattern. Thus it learns only if the input is sufficiently similar to previous input (if it resonates during this cycle of top-down and bottom-up "hypothesis testing").

The system includes no negative training; only the winners get trained. This would appear to require many more exemplars to get the averaging effect produced by the balanced cycle of positive and negative backpropagation training used in SCL. In this regard, it might be akin to the learning units based on fuzzy logic.

At least 2 of 3 possible inputs must fire in order to allow modification (the 2/3 rule). This gives something distantly related to confidence in the merit of the incoming patterns and may prevent rogue patterns from corrupting the system. Although, it should certainly be possible to generate a confidence number from the differences among the prototypes, we are unfamiliar with its use in ART systems. This whole system may be incorporated into a single integrated circuit (24:86) and recent manufactures have encapsulated ART on very sophisticated chips [147]. ART may be extended into a hierarchical network [24:87].

In general, the analog roots of ART, evidenced in the many gain controls, gating, filtering, and so forth seem to present a much more complex system than SCL. The development of specialized hardware to run ART tends to confirm this.

6.4.1.2 Reilly, Cooper, Elbaum and Scofield

In some respects our work is closer to that of Reilly, Cooper, Elbaum and Scofield [28, 29, 136, 137]. Their work illustrated the value of multiple prototype formation. It uses different control and learning mechanisms, since it predates the backpropagation revival of 1986. Their "radius-limited perceptrons" were explained in Chapter 3. These perceptrons define influence regions of radius λ which then shrinks as necessary to avoid misclassifications.

The most recent instantiation of their work uses neural networks running in transputers built by SGS Microelectronics [182]. It recognizes characters at the rate of 10 characters per second per transputer module, TRAM, so 6 TRAMs yields 60 characters per second for off-line OCR. A potentially unlimited number of prototypes may be formed (a new one from any new pattern in an unclaimed region of the feature space). Each completely defines the associated category response. SCL (as ART) combines "exemplar" patterns in the formation of more general prototypes; this would be expected to reduce the number of prototypes necessary to categorize incoming patterns and improve generalization. Our experiments tend to confirm this. SCL performs about 400 recognitions per second (lowercase letters) on 50 MIPS standard processors (Tables 16 and 17).

In their system, every new pattern is handled as a new prototype. There is no reinforcement or negative training of existing prototypes — just adjustments to their sphere of influence. As explained above, backpropagation of neural nets brings a balance to the learning units that tends to reduce their proliferation. One traumatic Nestor prototype can sabotage the training of the system, if it lands in the middle of the wrong territory. Eventually its λ_i influence diminishes to zero because many other correct prototypes crowd around it. Prototypes with thresholds below a minimum are purged from the system. This approach misses the generalizations that might be made if somewhat roguish exemplars were combined to mediate each other's influence. In SCL, the rogue's failure to compete for exemplars would sharply limit its training, and likely lead to its dismissal (certainly if it only gathered one exemplar). In this way their system seems somewhat dependent on "good" prototypes and rather vulnerable to "bad" prototypes. We conjecture that SCL is less dependent on both, but this would require specific testing to be confirmed.

Reilly et al. offer mechanisms comparable to the confidence generated by SCL. There are three types of responses: *identified* (only one category responded above the threshold), *confused* (more than one category responded), *unidentified* (none responded above threshold). Further, the series of modules are polled in order of *priority* which allows more confidence (of a sort) to be associated with module sets that classify better than others. SCL provides a quantified confidence with a high correlation with likelihood of correctness.

Their systems can include multi-category prototypes to identify patterns that are not differentiable in any single feature space. This has been tried with SCL recognizers by Shmatikov with marginal results, although we see no reason that the techniques would apply differently to SCL than to Nestor-based systems.

Their hierarchical recognizer is used for machine-vision applications with aerial photography and industrial assembly; but has occasionally been used as a handwriting classifier. Reilly et al. were able to recognize 98% of handwritten digits in a test group (distinct from the pool of training digits) generating about 10 prototypes per category. Our own recognition rates were slightly higher (99%) on data that included a number of children and generated about half as many prototypes.

One OCR version of the Nestor model uses 30 grids (times 12 edge templates) for pixel template matching [182]; this yields 360 features. No SCL recognizer uses more than 75 features, and energy-based recognizers perform nearly as well (98% with digits) using only 34 features, an order of magnitude fewer than 360.

6.4.1.3 Kohonen

Kohonen's topology-preserving maps [93], and Learning Vector Quantization in particular [95, 96] present another learning system architecture. It differs significantly from SCL in that LVQ processors are critically dependent on their neighbors to establish pairwise boundaries, and consequently require more processing units (ten per category instead of the two to six prototypes typical of SCL). Also SCL prototypes are topologically independent; a classification category may be represented by disconnected prototypes scattered widely over the feature space.

6.4.1.4 Others

As mentioned, Fukushima's Neocognitron [41, 43] includes several layers of neural discriminators to detect features such as the orientation of pattern subsets. Each of the several layers of the Neocognitron appears to be nearly as complex as the single layer of an SCL recognizer, but this is hard to determine without implementation details.

As noted in Chapter 3, Nellis and Stonham [123] employ a different learning engine (random access memory units combined into associators), but exploit the power of clever static feature abstraction and the effectiveness of varying prototypes. It is otherwise rather difficult to relate their work to our own. While our recognition rates were a bit higher (96.5% versus 94%), this is not a valid comparison, since their task involved optical character recognition. (Furthermore, the 94% was achieved with contextual support: dictionaries, Viterbi maps, and word likelihood. Their context-free recognition peaked at 73%. All of SCL's recognition is context-free.)

6.4.2 Recognition Rates Compared to Other Systems

Chapter 3 explained the difficulties of making a persuasive claim about the recognition performance of SCL versus other research or commercial systems. Certainly the

percentages are high for the case of "known" writers. Where training does not precede use, the “walk-up” recognition for upper and lowercase letters as evidenced by the instructor tests is not as high as that claimed by the boldest commercial vendors (90%). The walk-up rates for digits (96.5% for children, 98% for instructors) are in the neighborhood of high claimants. Chapter 4 illustrated the many difficulties associated with student’s handwriting. As far as we can discover, commercial vendors have not made a claim about this challenging recognition subset.

6.5 Relation to Other Research

We briefly evaluate the relation of SCL research to that of others in this field.

6.5.1 Relation of this Work to Pursuant Research

SCL technology has furnished an open laboratory within which a number of undergraduate researchers (Whaley, Shmatikov, Ho, Sherdil) and graduate researchers (Wang, Taysi, Hutchins, Wu, Ko, Thermal) have labored. We expect this to continue.

SCL has proven highly portable to other hardware platforms: NeXT, KM 68030, NCR, Mitsubishi, Oki, HP 720, SPARCstation 4, SPARCstation 10, and to several operating systems: DOS, OS9, Unix, ET++, PenPoint. When (or if) pen-based systems proliferate, this flexibility may add to the ongoing usefulness of SCL technology.

6.5.2 Environment to Support Pen-Based Research

We expect that some of the tools developed in conjunction with SCL will continue to support research in this field. The scribble analysis and management software (viewer, printer, translator/converter, and so forth) has already been adapted to multi-scribbles and gestures, and should prove adaptable to nearly any form of pen-based input which represents discrete signs. The SCL_Xlate interface has been used and will continue to find use

in PenPoint applications that require adaptable recognition. The SCL Test interface has been adapted for (1) learning a recognition domain from scratch, and (2) segmenting adjacent scribbles. Research continues in both areas. Kumon team members are building better interfaces for training from scratch, and building better segmentation programs for multi-scribbles of 10 or more characters.

6.6 Learning Systems and Human Learners

We briefly consider the relation of our work to teaching and learning.

6.6.1 Pen-Based Adaptable Human-Computer Interfaces and Teaching

When a researcher is involved with computers to assist learning, the question inevitably arises: Will computer-based learning replace teachers? This is asking if a computer will replace the "sudden" leap of insight that follows hours of nurture and encouragement. One might as well ask if computers will replace mentoring, mothering, guiding, fathering, coaching, leading, or counseling. Obviously not.

The same egocentricity that leads us to expect children to learn like adults, also leads us to expect them to use computers like adults. This problem may be compounded by the introduction of a plastic system that evolves with the learner. This is discussed in Fuller [47]. This is also related to the need to bring teachers into the design of user-interfaces for children. As a teacher who is herself involved in such designs, Nicol, writes [126]:

> My observations have led me to believe that 'interfaces for learning' have special design requirements above and beyond the considerations we make in building good electronic tools for professionals and casual users. When the user is naive with respect to underlying concepts (not just the use of the program), the interface needs to reflect teaching principles and models that might be unnecessary in programs geared towards more knowledgeable users.

She describes three areas that need special attention:

- Cues and overviews (maps, guides, tutorials, menus) to act as advance organizers in unfamiliar territory.
- Careful modeling and facilitation of the problem-solving process.
- Movement and animation used accurately to represent changes in state capitalize on the user's intelligence.

She also notes that good teachers use students' mistakes as teachable moments. Good interfaces for children should do the same. Hakansson [63] observes that there is much literature about the psychological and physical development of kids, but few observations of kids interacting with electronic media. We do not know much about how an interface built for children should differ from one built for adults. "Kids are able to think in novel and unsuspected ways, but they need tools to facilitate and support them."

6.6.2 New Windows on How Children Learn

The combination of feature sets and adaptability of SCL open new windows on the process of education. Do we offer standardized prototypes to guide children toward standard forms for digits and letters? Since the system may recognize directly at the level of velocity and acceleration, children may invent terse (and unreadable) substitutes for the standard symbols for English, the language of our shared discourse. On the other hand, there may be times where the very directness of the system's response to primitive energy profiles helps us probe below the surface of learning. It catches the *process* of drawing more than the *product* — the intent more than the outcome.

We may have favored prototypes that represent standard forms, and may measure the feature distance from these norms, but keep "user-friendly prototypes" available as well. Since we are partly judging by the static features and partly judging by the energy-based features, we can fine tune the training of novice writers. The latter features are

more enactive, the former are more iconic. We can grow with the learner, and support her by encouraging constructive behaviors even before the final product is evident — we are touching both processes and products. This can possibly be kept in correspondence to kindred developing layers in the learner.

The pen should do more than just change the medium. The first CRT terminals were used as glass teletypes. Similarly, the first pens were used as thin wireless mice. The adaptable systems based on SCL and kindred technologies offer a possibility for starting fresh with the paradigm.

This concludes our evaluation of SCL. In Chapter 7, we draw conclusions about the SCL model and system. In Chapter 8, we explore opportunities for future work in this area.

7. Conclusions

This chapter summarizes our conclusions for the SCL model and system. We draw upon the detailed evaluations of section 6.3.1 to conclude that SCL has achieved the eight formal goals of the problem specification. We enumerate several application goals that remained unsolved. We also list what we feel to be unique contributions to the field.

7.1 General Conclusions

We conclude that the SCL learning system architecture is a suitable methodology for its goals. The experiments reported in Chapter 5 indicate that SCL recognizers adapt to a wide range of writing styles in individual users without significant loss of recognition to earlier (or occasional) users. It has demonstrated this capability in several recognition domains: digits, lowercase letters, and a mixture of digits, uppercase and lowercase letters. Other researchers (reported in section 5.6) have shown that it may be used to recognize gestures and fractions, and that it may be compounded into a hierarchical recognizer to evaluate the relative effectiveness of several feature sets. Others have also shown its versatility in using totally unrelated feature sets (spoked wheels) and unrelated learning modules (fuzzy logic) within the SCL model.

7.1.1 Specific Performance

We conclude that the combination of energy-based and grid-based features is sufficient to reach our general recognition goals. Indeed, recognizers of twice the speed could be built from the energy-based features alone to reach all recognition goals with the possible exception of the mixed domain. Shuffling exemplars one to three times improves the final performance of the recognizer. SCL methodology appears fairly insensitive to

variations in the training parameters. The exception is that the number of prototypes created varies directly and consistently with ρ.

The instructors' tests (section 5.5.2) indicate that training for an individual generally requires more than two samples. In other cases (such as training from scratch in section 5.3.1), three exemplars seem sufficient.

We draw the unexpected conclusion that SCL might best be implemented without pre-trained learning units for applications that have few targets and that involve somewhat inconsistent writers — especially young children writing numbers. The ease with which SCL learns such a target set makes this option highly appealing in such settings.

When pre-trained recognizers are required, it would probably be best to train on a small number of writers with relatively standard character formation. It is not possible to capture all the variations of character formation without adding considerable ambiguity. It is better to handle the general cases, and then let the SCL recognizer quickly adapt to the peculiarities of the new user or users.

The experimental recognition rates (average of five-way cross validation: 99.2% for digits, 98.0% for lowercase, 96.5% for mixed, and comparable results for many, but not all, gestures) demonstrate the success of SCL in negotiating the stability/plasticity dilemma. These are comparable to the highest that we have seen reported (which are self-reported by vendors), but comparison in this task is notoriously risky. SCL adapts to new writers without losing the recognition capability obtained in prior training (section 5.3.2). SCL exhibits observable success in identifying similar exemplars and accumulating them for training (section 5.4.1). While cautiously acknowledging its specialized charter

(recognizing scribbles in these four domains for "known" writers), we feel that SCL demonstrates a recognition performance that approaches that of humans.

7.1.2 Confidence

The very high correlation of confidence to success (0.988 rank correlation coefficient) permits the arbitrary assignment of an accuracy threshold (say 99.5%) for rejection of input or display of alternates. Even when misrecognizing the input, SCL recognizers are highly successful in the second or third guesses (99.1% and 99.4% for mixed domain). SCL also shows promise for working with unsegmented digits on the basis of this confidence, which will be especially helpful for mathematics worksheets as explained in Chapter 4.

System confidence is sufficiently reliable to guide the recognition process through various candidate representations, as it does during segmentation. It would presumably be suitable for dictionary and database search templates. Characters with sufficiently high confidence could be fixed, and a small number of likely candidates could be set for the remaining slots in the search template.

7.1.3 Speed

The timing goal of SCL, training at sufficient speed to not encumber the learner, is comfortably realized with processor performance of about 50 MIPS. We cannot offer a clear conclusion on the use of DSP technology for cheap neural network processing. The custom DSP processor board that we created is clearly not viable. Production DSP boards of sufficient memory ought to achieve the desired performance and cost benefits. As shown in Chapter 6, the speed to adapt in real time can be realized in any of several platforms. The remarkable progress in CPU performance in the last three years presents many

possibilities. The DSP option is still the least expensive route to the 12,000 backpropagations per second, but the new 100 MIPS RISC-based CPUs such as SPARC 10, Power PC 604, Pentium P54A (CISC), DEC Alpha AXP, and HP PA series all exceed our requirements. The "best" is probably a matter of economics and market reception.

7.1.4 Extensions to SCL

SCL incorporates a full complement of research facilities to support further investigation by other researchers; indeed, several undergraduate and graduate students have already done so. It may be used for gestures, fractions, and other symbol sets. It may be compounded into a meta-recognizer, both to measure the relative strengths of feature families, and possibly to lend increased robustness to the recognition task.

7.1.5 Unsatisfied Application Goals

Several specific application goals remain unsatisfied.

The use of DSP technology remains unsettled as explained above in section 7.1.3.

We cannot yet offer final conclusions about segmenting the stream of incoming strokes into scribbles. Segmenting methods that employ just SCL recognizers work to a degree (at least 95% on digits). Segmenting algorithms that rely purely upon positional features (intersection, proximity, and so forth) are roughly as successful. We expect that superior methods will emerge from a combination of the two methods. Members of the Kumon Team continue to explore just this possibility.

Fast adaptation, high confidence, and good alternates address some elements of the two-body learning problem, but much remains to be studied and resolved in this area.

SCL recognizers remain overly dependent on the digitizing hardware and software which created their exemplars (section 5.3.3). This dependence might limit their applicability in general applications that would span many digitizing platforms. Perhaps a stronger form of normalization would compensate for hardware and software differences and present more consistent patterns to the classifier.

We did not demonstrate the use of SCL technology as embedded in the Kumon application. Such testing must await solution of the segmentation problem, the related time-out problem, and the right combination of hardware and application software.

7.2 Contributions of this Research

This research led to several contributions to this field which are original to the best of our knowledge.

- SCL model and system (with shuffling, confidence, and so on).
- Energy-based features and their relation to the enactive facility in the user.
- Relation of cognitive schema to a hierarchy of adaptations in the interface.
- Bi-directional and symmetric exponential smoothing.

Although shuffling may be considered a form of perturbation (common to Boltzmann machines and similar systems), we are not familiar with a definition and usage comparable to our own.

7.3 Summary

We find SCL a useful adaptation methodology for the pen-based paradigm. It lends itself to managing adaptation among the lower to middle layers of the hierarchical

adaptation model. It may be used in the recognition of digits, characters, and gestures. It successfully negotiates the stability/plasticity dilemma; it responds consistently to learned patterns and adapts to new patterns. It operates in real-time in realistic recognition domains, and can adapt to new user samples in real time. SCL adapts rapidly enough to significantly bypass the two-body learning problem. Other mechanisms also help avoid this problem, such as good alternatives after misrecognition. SCL is robust in the face of poor exemplars and ambiguous data. Its confidence factor is a reliable critic of its adaptations.

The feature set accompanying SCL is a significant element in its success. The energy features proved to be surprisingly effective, perhaps due to their proximity to the enactive psychological facility of the user. This may be especially true with the youngest writers since the enactive intent (and energy!) are clearly more in evidence than the graphical result.

As Bill Gates said [52], "We think the pen is the best pointing device. It's direct, it's more natural than even the mouse. It's familiar, unobtrusive, small and very precise." We add that it is capable of a wide range of adaptations to the desires and habits of its user. But adaptation cannot succeed as a blind mechanical exercise. An adaptable pen-based interface must be informed by and it must collaborate with the schema and sensitivities of the user. The brave, if oft faltering, frontier of pen interaction cries out for the unification of adaptation and interface — an alliance of machine *learning* and machine *doing*. Our research brings a modest measure of definition and systemic illustration to help explore the promise of the pen paradigm.

8. Future Directions

This chapter suggests areas for further research in this field. We first consider adaptable pen-based interfaces in general and then SCL in particular.

8.1 Further Research in Pen-Based Adaptable Human-Computer Interfaces

The hierarchical model to relate cognitive schema to interface adaptations is briefly introduced in Chapter 1 and elaborated in Appendix A. The ISO OSI [35] model abstracts communication control into seven layers to the betterment of communication software design. We expect that adaptable interfaces can benefit from a similarly layered approach. It will be necessary to build systems based on this model to assess its potential contribution to pen-based interface design.

Human communication exhibits constant detections and repairs of errors, but this is one of the most difficult elements of interface design [107, 159]. Both detections and corrections grow more sophisticated and more difficult at higher layers, but this process is critical to the success of flexible user interfaces. Otherwise, the interface designer is forced to choose a modest compromise and make the system consistent (thus predictable), but not adaptive. Hofstadter sees the gulf between perception and conception as *the* critical issue of artificial intelligence [26, 72, 73]. The layered model of adaptation hints at potential bridges across that gulf. This is another promising area for future work.

8.2 Further SCL Research

SCL prototypes use unsophisticated network architectures. We hope to extend this to conjugate gradient methods, possibly combined with a more elaborate version of

recurrent neural networks (RNN). The time-sequential nature of the energy-based features suggests that RNN methods might yield good results.

Currently SCL builds prototypes which differ only by the weights applied to the shared feature vector. We would like to create prototype collections which differ by the input vectors as well. Reilly et al. [137] observe that each prototype module could have its own feature abstractor. Possibly a cluster of prototypes might be fed by one feature abstractor while another cluster is fed by a second feature abstractor. For example, one prototype unit or cluster might consider energy-based features and another grid features. One cluster might deal with single-stroke scribbles while another specializes in multiple-stroke scribbles. We expect that this would give SCL a broader look at the incoming data stream.

We could expand SCL to build prototype units which differ by network architecture as well. One prototype may include no hidden units, another five hidden units, and a third six layers of five hidden units used recurrently. The high correlation between confidence and success suggests experiments with a hierarchical tree of progressively more complex recognizers. The confidence could be used to assess the success of a suggested recognition early in the processing, that is, higher in the tree of recognizers. If the confidence exceeds a given threshold, the category is returned. Otherwise the recognition descends the tree, possibly using special discriminatory features to resolve the ambiguity. This should also exploit another largely unused feature of SCL: its strong success with second or third guesses.

The smoothing filters and feature abstractors could likely be improved by more sophisticated methods. Currently, SCL segments strokes at uniform intervals. This is convenient but rather arbitrary. The y extremum segmentation of strokes used by Rumelhart

[145] might work for SCL as well. We might also try segmentation based on cusp points of the scribble. A much simpler abstraction of acceleration sampling could be tested: the difference of successive velocity segment averages. The segmentation of multi-scribbles into scribbles already occupies the attention of other researchers on the Kumon team.

SCL will continue to be studied with diverse audiences, including further study with children where constant system training is performed. This will help to resolve the subtle interplay within the two-body learning problem. As mentioned in Chapter 6, this is an excellent area for collaboration between computer scientists, cognitive scientists, and educators. Indeed the whole question of relating the spectrum of adaptations to the human mentalium (Kay's phrase) unites these fields.

We briefly introduced the nesting of SCL learning systems in the research with Shmatikov reported in Chapter 5. It should be possible to use SCL recognizers as single category learning systems. These could be gathered to form the Prototype Units module of a "meta-SCL" recognizer. The categories of each SCL recognizer might be multiple prototypes of a particular category in the meta-SCL system.

The question of SCL performance admits of no final resolution. It would be valuable to test subjects on their ability to recognize our scribble collection (having seen a few examples from the same writers). This would give a better basis for comparing human and machine recognition. It is unlikely that the same mistakes would be made by human and machine; these differences themselves might be instructive.

The Unipen consortium is working with NIST to prepare the first broad test of recognizers with potentially more than 170 active researchers in this field, both commercial and academic. If enough momentum is generated, this test might become the “flight simulator” of pen-based systems; that is, it might become a necessary preliminary to any claims by purveyors of handwriting recognizers.

9. Acknowledgments

The Kumon Machine Project is jointly funded by the Kumon Institute of Education and Osaka Gas Information Systems through an agreement with Washington University of St. Louis. Their support made this research possible and is gratefully acknowledged.
I deeply appreciate the generous information given about the Kumon method, and the chance to observe students at various Kumon centers and at four regular schools which adopted Kumon. Thanks also go to the dozens of Kumon students, Kumon instructors, and Kumon parents who patiently endured the several rounds of experimentation.

The other members of the Kumon Machine Project provided significant tangible and intangible support to this research. Some of the scribble collection software was developed by Julie Chan, Dale Frye, Sam Sengupta, and Tonya Bartow. Julie supplied an endless stream of cheerful support as I murked about arcane operating systems (especially OS-9 and PenPoint). Ed Hutchins wrote several of the utility functions used by these programs, especially file compression and decompression routines, and the digital signal processor manager. He was also a good-natured de-mystifier of bizarre shell commands.

I first met my advisor literally as I walked the halls of the Computer Science Department. The first hour of our first conversation revolved around the real *meaning* of the number five. When I discovered that he too was interested in education, mathematics, and computer science, I intuited that I had found a research home. That intuition proved true over the years that followed. His faithful support and advocacy have strengthened this work in more ways than I can number here.

Dr. Peter Hodgson, Dean of Faculty at Principia College where I teach, tirelessly supported this work in his own way. He shielded me from several committee appointments, and provided steady support and great understanding throughout the long march. My colleagues in the Mathematics and Computer Science Department at Principia covered many meetings, crises, and student queries in my absence. The students themselves were most gracious about the frequent inconveniences of a professor who is himself learning. The only consolation to them is that I grew a bit more sympathetic toward their protests that every professor issues assignments as if there were no other claims on their time.

Finally and most importantly, I acknowledge that my family paid the highest price among all these supporters. The sacrificed vacations, unfixed appliances, uncut hedges, and midnight dinners were endured with more than stoic, genuinely loving, faithfulness. This work would never have been possible without their unselfish care, understanding, and encouragement.

10. Bibliography

[1] Aleksander, I. and H. Morton. *An Introduction to Neural Computing*. New York: Chapman and Hall, 1990.

[2] Allen, Dennis. Pen-Input Systems. *Byte*, Apr. 1992, 10.

[3] Analog Devices. *ADSP Family: C Tools Manual*. Norwood, Massachusetts, 1993.

[4] Anderson, Robert F. V. *Introduction to Linear Algebra*. New York: Holt, Rinehart and Winston, 1986.

[5] Apte, Ajay and Takayuki Dan Kimura. A Comparison Study of the Pen and the Mouse in Editing Graphic Diagrams. *Proceedings of Visual Languages 1993 Conference*, Bergen, Norway, 1993.

[6] Baecker, Ronald M. and William A. S. Buxton, eds. *Readings in Human-Computer Interaction: A Multidisciplinary Approach*. San Mateo, California: Morgan Kaufmann, 1987.

[7] Baran, N. The Outlook for Pen Computing. *Byte*. Sept. 1992. 159-164.

[8] Baran, N, D. Bricklin, J. Dao, N. Katz, S. Liffick, and D. Mentley. The Future of Pen Computing, Part 2. *Byte*, Apr. 1992. 99-102.

[9] Becker, S. and Y. le Cun. Improving the Convergence of Back-Propagation Learning with Second Order Methods. In *Proceedings of the 1988 Connectionist Models Summer School* (CMU, June 17-26, 1988), eds. D. Touretzky, G. Hinton, and T. Sejnowski. San Mateo, California: Morgan Kaufmann, 1988. 29-37.

[10] Belsie, Laurent. Big Challenge for a Postal Behemoth. *The Christian Science Monitor*, 10 Apr. 1991. 13.

[11] Bloom, Benjamin S., M. D. Engelhart, E. J. Furst, W. H. Hill, and D. R. Krathwohl. *Taxonomy of Educational Objectives: The Cognitive Domain, Handbook I*. New York: Longmans, 1956.

[12] Blum, Avrim and Ronald L. Rivest. Training a 3-Node Neural Network is NP-Complete. *Proceedings of the 1988 Conference on Computational Learning Theory*. 9-18.

[13] Bruner, Jerome S. *The Process of Education*. Cambridge, Massachusetts: Harvard University Press, 1964.

[14] ---. *Towards a Theory of Instruction*. Cambridge, Massachusetts: Harvard University Press, 1966.

[15] ---. *In Search of Mind: Essays in Autobiography*. New York: Harper and Row, 1983.

[16] ---. *Actual Minds, Possible Worlds*. Cambridge, Massachusetts: Harvard University Press, 1986.

[17] Bruner, Jerome S., R. R. Olver, and P. M. Greenfield. *Studies in Cognitive Growth*. New York: John Wiley and Sons, 1966.

[18] Bruner, Jerome S., et al. *Studies in Cognitive Growth: A Collaboration at the Center for Cognitive Studies*. New York: John Wiley and Sons, 1967.

[19] Buxton, William A. S. There's More to Interaction than Meets the Eye: Some Issues in Manual Input. In *Readings in Human-Computer Interaction: A Multidisciplinary Approach*, eds. Ronald M. Baecker and William A. S. Buxton, San Mateo, California: Morgan Kaufmann, 1987.

[20] ---. The "Natural" Language of Interaction: A Perspective on Nonverbal Dialogues. In *The Art of Human-Computer Interface Design*, ed. Brenda Laurel. Reading, Massachusetts: Addison-Wesley, 1990.

[21] Buxton, William A. S., R. Sniderman, W. Reeves, S. Patel, and R. Baecker. The Evolution of the SSSP Score Editing Tools. *Computer Music Journal* 3, no. 4 (1979): 14-25.

[22] Buxton, William A. S., Ralph Hill, and Peter Rowley. Issues and Techniques in Touch-Sensitive Tablet Input. *Proceedings of Siggraph '85*. San Francisco: ACM, 1985. 215-224.

[23] Carpenter, G. A. and S. Grossberg. ART2: Self-Organizing of Stable Category Recognition Codes for Analog Input Patterns. *Applied Optics* 26 (1987): 4919-4930.

[24] ---. The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network. *Computer* 27, no. 3 (1988): 77-88.

[25] Carr, R. and D. Shafer. *The Power of PenPoint*. Reading, Massachusetts: Addison-Wesley, 1991.

[26] Chalmers, David J., Robert M. French, and Douglas R. Hofstadter. High-Level Perception, Representation, and Analogy: A Critique of Artificial Intelligence Methodology. *Journal of Experimental and Theoretical Artificial Intelligence* 4 no. 3 (1992): 185-211.

[27] Chen, Michael and Frank Leahy. A Design for Supporting New Input Devices. In *The Art of Human-Computer Interface Design*, ed. Brenda Laurel. Reading, Massachusetts: Addison-Wesley, 1990.

[28] Cooper, Leon N., Charles Elbaum, Douglas L. Reilly. Self-Organizing General Pattern Class Separator and Identifier. U.S. Patent 4,326,259 (April 20, 1982).

[29] Cooper, Leon N., Charles Elbaum, Douglas L. Reilly, and Christopher L. Scofield. Parallel, Multi-Unit, Adaptive, Nonlinear Pattern Class Separator and Identifier. U.S. Patent 4,760,604 (July 26, 1988).

[30] Cottrell, M., J. C. Fort, and G. Pagès. Two or Three Things that We Know about the Kohonen Algorithm. *Proceedings of ESANN*, Brussels, 1994.

[31] Cuban, Larry. Neoprogressive Visions and Organizational Realities. *Harvard Educational Review* 59, no. 2 (May 1989): 217-222.

[32] DasGupta, Bhaskar, Hava T. Siegelmann, and Eduardo D. Sontag. On the Complexity of Training Neural Networks with Continuous Activation Functions. In *Theoretical Advances in Neural Computation and Learning,* eds. V. P. Roychowdhury, K. Y. Siu and A. Orlitsky. Norwell, Massachusetts: Kluwer Academic Publishers, 1994.

[33] Davis, M. R. and T. O. Ellis. The Rand Tablet: A Man-Machine Graphical Communication Device. *AFTIPS Conference Proceedings 1964 Fall Joint Computer Conference* no. 26. 325-331.

[34] Davis, Robert B. *Learning Mathematics: The Cognitive Science Approach to Mathematics Education*. Norwood, New Jersey: Ablex Publishing, 1984.

[35] Day, J. D. and H. Zimmerman. The OSI Reference Model. *Proceedings of the IEEE* 71 (Dec. 1983): 1334-1340.

[36] Dix, Alan, Janet Finlay, Gregory Abowd, and Russell Beale. *Human-Computer Interaction*. New York: Prentice Hall, 1993.

[37] Fahlman, S. E. Faster Learning Variations on Backpropagation: An Empirical Study. In *Proceedings of the 1988 Connectionist Models Summer School* (CMU, June 17-26, 1988), eds. D. Touretzky, G. Hinton, and T. Sejnowski. San Mateo, California: Morgan Kaufmann, 1988. 38-51.

[38] Finn, Jr., Chester E. Made in Japan: Low-Tech Method for Math Success. *The Wall Street Journal*, 12 July 1989, A16.

[39] Fontaine, Thomas and Lokendra Shastri. Character Recognition Using a Modular Spatiotemporal Connectionist Model. (File name: pub/neuroprose/fontaine. charrec.ps.Z; 05/22/92; 155,857 bytes; 15 pages). Available from ftp site: archive.cis.ohio-state.edu.

[40] Freund, John. *Modern Elementary Statistics*, 3rd ed. Englewood Cliffs, New Jersey: Prentice Hall, 1967.

[41] Fukushima, K. Neocognitron: A Neural Network for Visual Pattern Recognition. *Computer* 23, no. 3 (Mar. 1988). 65-75.

[42] ---. Neocognitron: A Hierarchical Neural Network Capable of Visual Pattern Recognition. *Neural Networks* 1. (1988): 119-130.

[43] ---. An Improved Neocognitron Architecture and Learning Algorithm. In *Computational Learning and Cognition: Proceedings of the Third NEC Research Symposium*, ed. E. B. Baum. Philadelphia: Society for Industrial and Applied Mathematics, 1992. 29-43

[44] Fuller, Jr., Thomas H. *Microcomputers in Production and Inventory Management*. Homewood, Illinois: Dow Jones-Irwin, 1987.

[45] ---. *Handwritten Character Recognition (HCR) Users' Manual*. Oct. 1991. Kumon Machine Project, Washington University in St. Louis.

[46] ---. *Handwritten Character Recognition (HCR) Final Project Report*. Dec. 1991. Kumon Machine Project, Washington University in St. Louis.

[47] ---. The Kumon Approach to Learning Mathematics: An Educator's Perspective. Technical Report WUCS 91-49. Washington University in St. Louis, Dec. 1991.

[48] ---. Energy-Related Feature Abstraction for Handwritten Digit Recognition. *Proceedings of the Fourth Midwest Artificial Intelligence and Cognitive Science Conference*, May 1992. 71-76.

[49] ---. *Supervised Competitive Learning (SCL 3.0) System Manual*. July 1994. Kumon Machine Project, Washington University in St. Louis.

[50] Fuller, Jr., Thomas H. and Takayuki Dan Kimura. Supervised Competitive Learning, Part I: SCL with Backpropagation Networks. In *Intelligent Engineering through Artificial Neural Networks*, vol. 2, eds. C. Dagli et al. New York: ASME Press, 1992. 185-190.

[51] ---. Supervised Competitive Learning. *Journal of Intelligent Material Systems and Structures* 5, (Mar. 1994): 232-246.

[52] Gates, III, William H. *Windows: Sources*. Premier Issue, February, 1993.

[53] Gibbs, M. Handwriting Recognition: A Comprehensive Comparison. *Pen*, Mar./Apr. 1993, 31-35.

[54] Glass, Brett. Another Look at PenPoint: It's More Than a Pen-Based "Shell" Interface. *Infoworld*, 11 Feb. 1991, 76.

[55] GO Corporation. *PenPoint API Reference*. Reading, Massachusetts: Addison-Wesley, 1992.

[56] Gramss, Tino. Word Recognition with the Feature-Finding Neural Network (FFNN). In *Neural Networks for Signal Processing: Proceedings of the 1991 IEEE Workshop,* eds. B. H. Juang et al. New York: IEEE, 1991. 289-298.

[57] ---. Fast Learn-Algorithms for a Self-Optimising Neural Network with an Application to Isolated Word Recognition. *IEE Proceedings-F* 139, no. 6 (Dec. 1992): 391-396.

[58] Greeno, James G. *Situations, Mental Models, and Generative Knowledge*. (Report No. IRL88-0005). Palo Alto: Institute for Research on Learning, 1988.

[59] Grossberg, S. Adaptive Pattern Classification and Universal Recoding: I. Parallel Development and Coding of Neural Feature Detectors. *Biological Cybernetics* 23 (1976): 121-134.

[60] ---. Adaptive Pattern Classification and Universal Recoding: II. Feedback, Expectation, Olfaction, Illusions. *Biological Cybernetics* 23 (1976): 187-202.

[61] ---. Competitive Learning: From Interactive Activation to Adaptive Resonance. *Cognitive Science* 11 (1986), 23-63.

[62] Grothe, Robert. Interview by author. October, 1991.

[63] Hakansson, Joyce. Lessons Learned from Kids: One Developer's Point of View. In *The Art of Human-Computer Interface Design*, ed. Brenda Laurel. Reading, Massachusetts: Addison-Wesley, 1990.

[64] Halbert, Daniel C. *Programming by Example*. Palo Alto: Xerox Office Systems Division, 1984.

[65] Hall, Rogers P. Making Mathematics on Paper: Constructing Representations of Stories about Related Linear Functions. Doctoral diss., University of California, Irvine, 1990.

[66] Hebb, D. O. *The Organization of Behavior: A Neuropsychological Theory*. New York: John Wiley and Sons, 1949.

[67] Hecht-Nielsen, Robert. *Neurocomputing*. Reading, Massachusetts: Addison-Wesley, 1990.

[68] Herbst, N. M. and C. N. Liu. Automatic Signature Verification. In *Computer Analysis and Perception, Volume 1: Visual Signals,* eds. C. Y. Suen and R. De Mori. Boca Raton, Florida: CRC Press, 1982. 83-105.

[69] Hinton, Geoffrey, Terrence J. Sejnowski, and David Ackley. Boltzmann Machines: Constraint Satisfaction Networks that Learn. Technical Report, CMU-CS-84-119, Carnegie Mellon University, May, 1984.

[70] Hinton, Geoffrey E., Christopher K. I. Williams and Michael D. Revow. Adaptive Elastic Models for Hand-Printed Character Recognition. Technical Report, University of Toronto, Canada, 1990.

[71] Ho, Ki-Wing. Fraction Segmentation. Summer Undergraduate Research Associate Report, Aug. 1993. Washington University in St. Louis.

[72] Hofstadter, Douglas R. *Gödel, Escher, Bach: An Eternal Golden Braid*. New York: Basic Books, 1979.

[73] Hofstadter, Douglas and Melanie Mitchell. The Copycat Project: A Model of Mental Fluidity and Analogy-Making. CRCC Report 58, Indiana University, Bloomington, Indiana, 1992.

[74] Hsieh, Keun-Rong and Wen-Tsuen Chen. A Neural Network Model which Combines Unsupervised and Supervised Learning. *IEEE Transactions on Neural Networks* 4, no. 2 (Mar. 1993): 357-360.

[75] Inhelder, Barbel, and Jean Piaget. *The Growth of Logical Thinking From Childhood to Adolescence: An Essay on the Construction of Formal Operational Structures*. Trans. Anne Parsons and Stanley Milgram. New York: Basic Books, 1956.

[76] Inhelder, Barbel, Hermine Sinclair, and Magali Bovet (Foreword by Jean Piaget). *Learning and the Development of Cognition*. Trans. Susan Wedgwood. Cambridge: Harvard University Press, 1974.

[77] Jacobs, Robert A. Increased Rates of Convergence Through Learning Rate Adaptation. *Neural Networks* 1 (1988): 295-307.

[78] Johnson, Walter, Herbert Jellinek, Leigh Klotz, Jr., Ramana Rao and Stuart Card. Bridging the Paper and Electronic Worlds: The Paper User Interface. *INTERCHI '93 Conference Proceedings: Bridges between Worlds* (ACM Special Interest Group on Computer and Human Interaction, Amsterdam, April 24-29, 1993). Reading, Massachusetts: Addison-Wesley, 1993. 507-512.

[79] Kalman, B. L. and Kwasny, S. C. A Superior Error Function for Training Neural Nets. *Proceedings of the International Joint Conference on Neural Networks*, Seattle, Washington, 1991, vol. 2. New York: IEEE, 1991. 49-52.

[80] ---. Why Tanh: Choosing a Sigmoidal Function. *Proceedings of the International Joint Conference on Neural Networks*, Baltimore, Maryland, June 1992. New York: IEEE, 1992. IV-578-583.

[81] Kawatani, Takahiko. Handprinted Numerical Recognition with the Learning Distance Function. *Pre-Proceedings of IWFHR-III* (Third International Workshop on the Frontier of Handwriting Recognition, Center of Excellence for Document Analysis and Recognition, Buffalo, New York, May 25-27, 1993). New York: Partners' Press, 1993. 324-330.

[82] Kay, Alan. Microelectronics and the Personal Computer. *Scientific American* 237, no. 3 (1977): 230-244.

[83] ---. Doing with Images Makes Symbols: Communicating with Computers Distinguished Lecture Series, University Video Communications, 1987. Videocassette.

[84] ---. User Interface: A Personal View. In *The Art of Human-Computer Interface Design*, ed. Brenda Laurel. Reading, Massachusetts: Addison-Wesley, 1990.

[85] Kearsley, Greg, ed. *Artificial Intelligence and Instruction*. Reading, Massachusetts: Addison-Wesley, 1987.

[86] Keller, John J. Small AT&T Firm to Shift Strategy on Data Devices. *New York Times*, 1 Feb. 1994, B3.

[87] Kempf, James. *Preliminary Handwriting Recognition Engine Application Program Interface for SPARC*. Mountain View: Sun Microsystems, April 5, 1993.

[88] ---. *Integrating Handwriting Recognition into Unix*. Mountain View: Sun Microsystems, 1993.

[89] Kimura, Takayuki Dan. An Algebraic Proof for Backpropagation in Acyclic Neural Networks. *Proceedings of International Joint Conference on Neural Networks*, San Diego, CA, June 1990, 554-557.

[90] ---. Silicon Paper and a Visual Interface for Neural Networks. *Proceedings of IEEE 1990 Workshop on Visual Languages*, Chicago. New York: IEEE, 1990. 241-246.

[91] ---. Kumon Machine: Learning Math with Silicon Paper. *Proceedings of the First Silicon Paper Seminar*, Osaka, Japan, 1991.

[92] ---. Hyperflow: A Visual Programming Language for Pen Computers. *Proceedings of the 1992 IEEE Workshop on Visual Languages*, September 15-18, 1992, Seattle, Washington. New York: IEEE, 1990. 125-132.

[93] Kohonen, Teuvo. Self-Organized Formation of Topologically Correct Maps. *Biological Cybernetics* 43 (1982): 59-69.

[94] ---. Analysis of a Simple Self-Organizing Process. *Biological Cybernetics* 44 (1982): 135-140.

[95] ---. An Introduction to Neural Computing. *Neural Networks* 1 (1988): 3-16.

[96] ---. The Self-Organizing Map. *Proceedings of the IEEE* 78 (1990): 1464-1478.

[97] Krasner, Glenn E. and Stephen T. Pope. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. *Journal of Object-Oriented Programming* 1 (Aug. 1988): 26-49.

[98] Kumon Institute of Education. *I'll take Kumon*. Pamphlet. Oct. 1987.

[99] ---. *Instruction Manual* 2 vols. Oct. 1987.

[100] ---. *Strengths of the Kumon method*. Undated pamphlet, circa 1987.

[101] Kurkova, Vera. Kolmogorov's Theorem and Multilayer Neural Networks. *Neural Networks* 5 (1992): 501-506.

[102] Kurlander, David and Steven Feiner. A History-Based Macro by Example System. *UIST Conference Proceedings*, Nov. 15-18, 1992.

[103] Kurtenbach, Gordon and Eric A. Hulteen. Gestures in Human-Computer Communication. In *The Art of Human-Computer Interface Design*, ed. Brenda Laurel. Reading, Massachusetts: Addison-Wesley, 1990.

[104] Laine, Andrew, Sergio Schuler, and V. Girish. Orthonormal Wavelet Representations for Recognizing Complex Annotations. *Machine Vision and Applications*. New York: Springer-Verlag, 1993. 110-123.

[105] Laine, Andrew and Sergio Schuler. Hexagonal Wavelet Representations for Recognizing Complex Annotations. Technical Report 93-033, Center for Computer Vision and Visualization, University of Florida, July, 1993.

[106] Laurel, Brenda, ed. *The Art of Human-Computer Interface Design*. Reading, Massachusetts: Addison-Wesley, 1990.

[107] Laurel, Brenda. *Computers as Theatre*. Reading, Massachusetts: Addison-Wesley, 1991.

[108] Lee, Sukhan and Jack C. Pan. Invariant Handwritten Numerical Recognition with Spatio-Temporal Feature Representation. *Pre-Proceedings of IWFHR-III,* Third International Workshop on the Frontier of Handwriting Recognition, Center of Excellence for Document Analysis and Recognition, Buffalo, New York, May 25-27, 1993). New York: Partners' Press, 1993. 331-336.

[109] Lin, W. C. Computer Processing of Hand-Drawn Sketches and Diagrams. In *Computer Analysis and Perception, Volume 1: Visual Signals*, eds. C. Y. Suen and R. De Mori. Boca Raton, Florida: CRC Press, 1982. 107-149.

[110] Loftus, Geoffrey R. and Elizabeth F. Loftus. *Mind at Play: The Psychology of Video Games*. New York: Basic Books, 1983.

[111] Maass, Wolfgang. Perspectives of Current Research about the Complexity of Learning on Neural Nets. In *Theoretical Advances in Neural Computation and Learning,* eds. V. P. Roychowdhury, K. Y. Siu and A. Orlitsky. Norwell, Massachusetts: Kluwer Academic Publishers, 1994.

[112] MacKay, David J. C. A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation* 4 (1992): 448-472.

[113] Markoff, John. AT&T Will Shut Down EO, a Wireless Pioneer. *New York Times*, 28 July 1994, C6.

[114] Minsky, M. and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, Massachusetts: MIT Press, 1969.

[115] Mitchell, Melanie and Douglas R. Hofstadter. The Emergence of Understanding in a Computer Model of Concepts and Analogy-Making. *Physica D* 42. North-Holland: Elsevier Science Publishers, 1990. 322-334.

[116] Morasso, P., L. Barberis, S. Pagliano, and D. Vergano. Recognition Experiments of Cursive Dynamic Handwriting with Self-Organizing Networks. *Pattern Recognition* 26, no. 3 (Mar. 1993): 451-460.

[117] Morasso, P., L Gismondi, E. Musante, and A. Pareto. SCRIPTOR: An On-Line Recognition Engine of Cursive Handwriting with Incremental Learning Capabilities. *Pre-Proceedings of IWFHR-III* (Third International Workshop on the Frontier of Handwriting Recognition, Center of Excellence for Document Analysis and Recognition, Buffalo, New York, May 25-27, 1993). New York: Partners' Press, 1993. 431-436.

[118] Morita, Toshihiro, and Shinji Mori. An Internal Model for Pen-Based Input Data and the Program Libraries Used with this Model (in Japanese). *Human Interface* 6 (1991): 261-268.

[119] Murr, Andrew. Samurai Math Gets a Tryout. *Newsweek*, 10 Apr. 1989, 60.

[120] Myers, Brad. User-Interface Tools: Introduction and Survey. *IEEE Software*, Jan. 1989, 15-23.

[121] Nakagawa, Masaki, Naoki Kato, Kimiyoshi Machii, and Toshio Souya. Principles of Pen Interface Design for Creative Work. *Proceedings of the Second International Conference on Document Analysis and Recognition*, Tsukuba, Japan, Oct. 20-22, 1993.

[122] Neisser, Ulric and Paul Weene.. A Note on Human Recognition of Hand-Printed Characters. *Information and Control*, vol. 3. New York: Academic Press, 1960. 191-196.

[123] Nellis, J. and T. J. Stonham. A Neural Network Character Recognition System that Dynamically Adapts to an Author's Writing Style. In *Intelligent Engineering through Artificial Neural Networks*, vol. 2, eds. C. Dagli et al. New York: ASME Press, 1992. 975-980.

[124] Newman, Jeremy et al. Position on Pens. *Personal Electronic News*. May/June, 1994, 44-62.

[125] NeXT Corporation. *NeXT Interface Builder Reference Manual*. Redwood City, California, 1991.

[126] Nicol, Anne. Interfaces for Learning: What Do Good Teachers Know that We Don't? In *The Art of Human-Computer Interface Design*, ed. Brenda Laurel. Reading, Massachusetts: Addison-Wesley, 1990.

[127] Papert, Seymour. *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Harper Collins, Basic Books, 1980.

[128] ---. *The Children's Machine: Rethinking School in the Age of the Computer*. New York: Harper Collins, Basic Books, 1993.

[129] ---. The Children's Machine. *Technology Review*, July, 1993, 29-36.

[130] Piaget, Jean. *The Origins of Intelligence in Children*. Trans. Margaret Cook. International Universities Press, 1952. Republished, New York: Norton, 1963.

[131] ---. *The Grasp of Consciousness: Action and Concept in the Young Child*. Trans. Susan Wedgwood. Cambridge: Harvard University Press, 1976.

[132] Plutowski, Mark E., Shinichi Sakata, and Halbert White. Cross-Validation Estimates IMSE. In *Advances in Neural Information Processing Systems 6*, eds. J. D. Cowan, G. Tesauro, and J. Alspector, J. San Mateo, California: Morgan Kaufmann, 1994.

[133] Pree, Wolfgang. Object-Oriented Versus Conventional Construction of User Interface Prototyping Tools. Doctoral diss., Johannes Kepler University, Linz, Austria, 1991.

[134] Ralston, Anthony and Edwin D. Reilly, eds. *Encyclopedia of Computer Science*. New York: Van Nostrand Reinhold, 1993.

[135] Ramirez, Anthony. New Software for Reading Handwriting. *New York Times*, 14 June 1993, C1, C4.

[136] Reilly, D. L., L. N. Cooper, and C. Elbaum. A Neural Model for Category Learning. *Biological Cybernetics* 45 (1982): 35-41.

[137] Reilly, D. L., C. Scofield, C. Elbaum, and L. N. Cooper. Learning System Architectures Composed of Multiple Learning Modules. *Proceedings, IEEE First International Conference on Neural Networks* , San Diego, California, 1987. II-495-503.

[138] Revow, Michael, Christopher K. I. Williams and Geoffrey E. Hinton. Using Mixtures of Deformable Models to Capture Variations in Hand Printed Digits. *Pre-Proceedings of IWFHR-III* (Third International Workshop on the Frontier of Handwriting Recognition, Center of Excellence for Document Analysis and Recognition, Buffalo, New York, May 25-27, 1993). New York: Partners' Press, 1993. 142-152.

[139] Rodin, Ervin Y., Yuanlan Wu, and S. Massoud Amin. Character Recognition: Qualitative Reasoning and Neural Networks. *Mathematical Computer Modeling* 16, no. 2 (1992): 95-102.

[140] Rosenblatt, R. *Principles of Neurodynamics*. New York: Spartan Books, 1962.

[141] Rubine, Dean. Integrating Gesture Recognition and Direct Manipulation. *USENIX Proceedings*, Summer, 1991.

[142] ---. Specifying Gestures by Example. *Computer Graphics* 25, no. 4 (July, 1991): 329-337.

[143] ---. The Automatic Recognition of Gestures. Doctoral diss., Carnegie Mellon University, 1991.

[144] Rumelhart, D. E., G. E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1, eds. David E. Rumelhart, and James L. McClelland. Cambridge, Massachusetts: MIT Press, 1986.

[145] Rumelhart, D. E. Theory to Practice: A Case Study — Recognizing Cursive Handwriting. In *Computational Learning and Cognition: Proceedings of the Third NEC Research Symposium*, ed. E. B. Baum. Philadelphia: Society for Industrial and Applied Mathematics, 1992. 177-196.

[146] Schwartz, John. The Revolution that Wasn't. *Newsweek*, 19 Oct. 1992, 51.

[147] Serrano, T., B. Linares-Barranco, and J. L. Huertas. A CMOS VLSI Analog Current-Mode High-Speed ART1 Chip. *Proceedings of ICNN '94*, Orlando, Florida, June, 1994.

[148] Servan-Schreiber, David, Axel Cleeremans, and James L. McClelland. Encoding Sequential Structure in Simple Recurrent Networks. Technical Report CMU-CS-88-183, Carnegie Mellon University, Nov. 1988.

[149] Sherdil, Khalid. Segmentation of Hand-Written Digits. Undergraduate thesis. Washington University in St. Louis, 1993.

[150] Shiba, Shoji. The Excellent Education System for One and a Half Million Children. *Programmed Learning and Educational Technology* 23, no. 4, (Nov. 1986): 306-330.

[151] Shmatikov, Vitaly. Supervised Competitive Learning with Multiple Feature Abstractors. Summer Undergraduate Research Associate Report, Aug. 1993. Washington University in St. Louis.

[152] Shynk, J. J. and S. Roy. The LMS Algorithm with Momentum Updating. *IEEE Symposium on Circuits and Systems*, Espoo, Finland. New York: IEEE, 1988. 2651-2654.

[153] Smieja, Frank. The Pandemonium System of Reflective Agents. (File name: pub/neuroprose/smieja.pandemonium.ps.Z; 10/06/93; 707,355 bytes; 17 pages). Available from ftp site: archive.cis.ohio-state.edu.

[154] Snyder, Tom, and Jane Palmer. *In Search of the Most Amazing Thing*. Reading, Massachusetts: Addison-Wesley, 1986.

[155] Solomon, Cynthia. *Computer Environments for Children: A Reflection on Theories of Learning and Education*. Cambridge, Massachusetts: MIT Press, 1986.

[156] Sompolinsky, H. Theoretical Issues in Learning from Examples. In *Computational Learning and Cognition: Proceedings of the Third NEC Research Symposium*, ed. E. B. Baum. Philadelphia: Society for Industrial and Applied Mathematics, 1992. 217-237.

[157] Sørheim, Einar. ART2/BP Architecture for Adaptive Estimation of Dynamic Processes. In *Advances in Neural Information Processing Systems 3*. (1991), ed. Richard Lippmann, John Moody, David Touretzky. San Mateo, California: Morgan Kaufmann, 1991. 169-175.

[158] Strohecker, Carol. Why Knot? Doctoral diss., MIT, Cambridge, Massachusetts, 1991.

[159] Suchman, Lucy A. *Plans and Situated Actions: The Problem of Human-Machine Communication*. Cambridge: Cambridge University Press, 1987.

[160] Suen, Ching Y., Marc Berthod, and Shinji Mori. Automatic Recognition of Handprinted Characters: The State of the Art. *Proceedings of the IEEE* 68, no. 4 (Apr. 1980): 469-487.

[161] Swaine, Michael. PowerPC: Going Beyond GUI. *MacUser*, July, 1994, 25.

[162] Swigart, Rob. A Writer's Desktop. In *The Art of Human-Computer Interface Design*, ed. Brenda Laurel. Reading, Massachusetts: Addison-Wesley, 1990.

[163] Tappert, Charles C., Ching Y. Suen and Toru Wakahara. The State of the Art in On-Line Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12, no. 8 (Aug. 1990): 787-808.

[164] Taysi, Burak M. Gesture System for a Graph Editor. Master's thesis, Washington University in St. Louis, August, 1992.

[165] Tazelaar, Jane M. Recognizing Script. *Byte*, April, 1991, 210.

[166] Teixeira, J. K. and R. P. Sallen. The Sylvania Data Tablet: A New Approach to Graphic Data Input. *Proceedings of the Spring Joint Computer Conference*, 1968. 315-321.

[167] Thomas, Jr., G. B. *Calculus and Analytic Geometry*. Reading, Massachusetts: Addison-Wesley, 1972.

[168] Trudeau, Garry B. Doonesbury, Universal Press Syndicate, *St. Louis Post-Dispatch*. 23-28 Aug. 1993.

[169] Turkle, Sherry. *The Second Self: Computers and the Human Spirit*. New York: Simon and Schuster, 1984.

[170] U.S. Department of Commerce, National Institute of Standards and Technology. *Massively Parallel Implementation of Character Recognition Systems*. Report NISTIR 4750. Prepared by M. D. Garris, S. L. Wilson, J. L. Blue, G. T. Candela, P. Grother, S. Janet, and R. A. Wilkinson. Gaithersburg, Maryland, 1992.

[171] Wagoner, John. Second Annual Reader's Choice Award Winners. *Personal Electronic News*, Mar./Apr. 1993, 11-21.

[172] ---. Ultimately Handwriting Recognition Will Work. *Personal Electronic News*. May/June, 1994, 4.

[173] Walters, Laurel Shaper. Math Mastery, The Japanese Way: The Kumon Method Stresses Building Speed and Accuracy via Drills, Not Conceptualization. *The Christian Science Monitor*, 11 Feb. 1991, 12.

[174] Wang, Ce. A Fuzzy Approach to Handwritten Digit Recognition for Pen-Based Computers. Master's thesis, Washington University in St. Louis, December, 1992.

[175] Wang, C., T. Kimura, and T. Fuller. Supervised Competitive Learning, Part II: SCL with Fuzzy Logic. In *Intelligent Engineering through Artificial Neural Networks*, vol. 2, eds. C. Dagli et al. New York: ASME Press, 1992. 351-356.

[176] Webb, Brandyn Jerad. Fusion-Reflection: Self-Supervised Learning (File name: pub/neuroprose/webb.furf.ps.Z; 08/19/93; 65,997 bytes; 20 pages). Available from ftp site: archive.cis.ohio-state.edu.

[177] Webster, Rodney G., and Masaki Nakagawa. The Feasibility of a Parallel Processing Oriented Character Recognition Method Based on a Dynamic Model. *Proceedings of the Second International Conference on Document Analysis and Recognition*, Tsukuba, Japan, Oct. 20-22, 1993.

[178] Weiss, S. M. and C. A. Kulikowski. *Computer Systems that Learn*. San Mateo, California: Morgan Kaufmann, 1991.

[179] Werbos, Paul John. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Doctoral diss., Harvard University, 1974.

[180] Whaley, Benjamin. The "Spoked-Square" Feature Abstractor for Handwritten Digit Recognition, Summer Undergraduate Research Associate (SURA) Report, Aug. 1992. Washington University in St. Louis.

[181] Widrow, Bernard, and Michael A. Lehr. 30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation. *Proceedings of the IEEE* 78, no. 9 (Sept. 1990): 1415-1441.

[182] Wright, D. P., and C. Scofield. Divide and Conquer: Neural Networks Take a Novel Approach. *Byte*, April, 1991, 207-210.

[183] Yamanishi, K. A Statistical Approach to Computational Learning Theory. In *Computational Learning and Cognition: Proceedings of the Third NEC Research Symposium*, ed. E. B. Baum. Philadelphia: Society for Industrial and Applied Mathematics, 1992. 238-276.

[184] Zadeh, L. A. Fuzzy Sets, *Information and Control* 8 (1965): 338-353.

[185] Zhao, Rui. Incremental Recognition in Gesture-Based and Syntax-Directed Diagram Editors. *INTERCHI '93 Conference Proceedings: Bridges between Worlds* (ACM Special Interest Group on Computer and Human Interaction, Amsterdam, April 24-29, 1993). Reading, Massachusetts: Addison-Wesley, 1993. 95-100.

[186] Zhu, Huaiyu. Neural Networks and Adaptive Computers: Theory and Methods of Stochastic Adaptive Computation. Doctoral diss., University of Liverpool, 1993.

[187] Zielenziger, David. Crowded Niche: Newton, Imitators Still Seeking Market. *St. Louis Post-Dispatch*, 9 Mar. 1994, 5C.

Appendix

A. Pen-Based Adaptable Human-Computer Interface Model

In the main body of this work, we described the recognition environment formally as a dynamic stream of patterns representing a set of categories. Adaptation may be abstractly represented as a dynamic remapping from stream to category. However, this notion of adaptation as transducer is too broad for many purposes. In this appendix, we develop this generic abstraction into a hierarchical taxonomy of adaptation.

A.1 The Hierarchical Model

This taxonomy borrows a scheme from the International Standards Organization Open Systems Interconnection reference model (ISO OSI model [35]) which describes the different layers of system activity necessary for data communication. The OSI model abstracts communication control into seven layers of activity. Thus, two communicating systems must agree within each of the seven layers. We draw upon this scheme to develop a more precise model of communications between a human user and a computer system — hence more precise definitions of adaptation throughout the spectrum of human-computer interactions. The following table outlines the eight layers of the Pen-Based Adaptable Human-Computer Interface (PAHCI) model.

Table 30. PAHCI Hierarchical Model

Level	Location of the adaptation	Adapted objects
Physical Physiological	hardware and firmware	points
Sensory Link	visual feedback and pen inking	strokes
Assembly	collection of points and strokes into taps, gestures, characters	characters gestures
Direction	connection from input stream to objects	current object of attention
Context	appropriate symbol sets and their interpretation	context structure and feature environment
Task	current work object	single task action
Structor	editor-like application	current task
Manager	operating system	current goal

Adaptation at the lowest level involves the transformation of points and is achieved primarily in dedicated hardware or firmware. For example, the device driver for the digitizing tablet might adjust the sampling rate. Adaptation at the highest layer relates the presumed user goals to the (adaptable) operating system. For a hypothetical example, consider Kay's goal cloning [84] — the commissioning of an army of intelligent agents to help the user achieve current goals. A *structor* (a word invented by the author in connection with PAHCI) is an editor-like system for creating, modifying, and displaying useful objects in the system. A word processor or a music editor are example structors.

SCL research is aimed at adaptations in the *sensory link* and *assembly layers* of the model — the layers that receive the user's strokes and assemble them into characters and gestures. These two layers relate to the user's psychological facilities respectively denominated as *enactive* and *iconic* by Bruner [14]. Thus the adaptations at these layers were designed to correspond to some combination of enactive (kinesthetic) and iconic (visual) features abstracted from the incoming data stream.

We next consider related research and then develop this hierarchy with broader illustrations.

A.2 Related Research

As explained in Chapter 3, adaptation to users comes in many forms. Now we are prepared to position these forms in our taxonomy. The history-based macros of Kurlander and Feiner [102] illustrate adaptation at the *task* layer. Halbert's programming by example [64] and Kimura's Hyperflow [92] adapt at the *structor* (editor) layer. Trainable recognizers (like that of GO's PenPoint [25]) adapt at the *assembly* layer. Exponential smoothing of input and other filtering techniques make the system adaptable at the *sensory link* layer.

Although they do not address adaptation directly, Baecker and Buxton provide a useful taxonomy of input devices [6:360]. Chen and Leahy [27] write that a successful model of the user interface should allow an extensible and semi-structured name space and should permit event transmogrification (one physical device can model several others). They conclude that, "Only through the adoption of a coherent high-level model of interaction between input devices and applications can both uses and programmers benefit from using new input devices, or from using two or more input devices simultaneously." PAHCI is an effort to provide such a high-level model, moreover, one which specifically accommodates adaptation.

A.2.1 Aristotle's Model of HCI

Laurel [107] associates human computer interaction with the relation of dramatist to audience. This analogy is arguably more apt in computer games than in general applications, but the insights could inform general HCI design. She discovers an uncanny

parallel between HCI and the classical elements of drama in Aristotle's *Poetics*, written in the fourth century B.C. Aristotle first defines the four *causal elements* of drama: *formal* (general form, such as tragedy or comedy), *material* (enactment, what unfolds in the story), *efficient* (tools and skills build by its creators — chunks of functionality), *end* (what has been finally achieved by this play or processing, catharsis). Then he outlines the six *elements* of drama. These (like the divisions of the PAHCI model) are best understood from the bottom up. As Laurel notes, each element is the *formal* cause of those below it and in turn is the *material* cause of those above it.

Table 31. Aristotle's Elements of Drama (from Laurel [107])

Element	**In Drama**	**In Human-Computer Activity**
Action	The whole action being represented. The action is theoretically the same in every performance.	The whole action, as it is collaboratively shaped by system and user. The action may vary in each interactive session.
Character	Bundles of predispositions and traits, inferred from agents' patterns of choice.	The same as in drama, but including agents of both human and computer origin.
Thought	Inferred internal processes leading to choice: cognition, emotion, and reason.	The same as in drama, including processes of both human and computer origin.
Language	The selection and arrangement of words; the use of language.	The selection and arrangement of signs, including verbal, visual, auditory, and other nonverbal phenomena when used semiotically.
Melody (Pattern)	Everything that is heard, especially the melody of speech.	The pleasurable perception of pattern in sensory [system-generated] phenomena.
Spectacle (Enactment)	Everything that is seen.	The sensory dimensions of the action being represented: visual, auditory, kinesthetic and tactile, and potentially all others.

A.2.2 Learning Theory and Interfaces for the Learning Computer

As explained in Chapter 3, a central tenet of our adaptation research is Alan Kay's claim [83] that human interface design is inextricably related to learning theory. We also explain there how Piaget's and Bruner's models of human cognition relate specifically to the layers at which adaptation takes place. Thus these learning models bear directly on the relation of the user to the system. The "unconscious regulations" of early childhood are the raw material of which the adolescent builds mastery. Similarly the enactive experimentation of the newcomer with mouse or pen can be a potent raw material out of which to formulate a mature understanding of a system.

> Alan Kay writes [84]:
>
> The work of Papert convinced me that whatever user interface design might be, it was solidly intertwined with learning. Bruner convinced me that learning takes place best environmentally and roughly in stage order — it is best to learn something kinesthetically, then iconically, and finally the intuitive knowledge will be in place that will allow the more powerful but less vivid symbolic processes to work at their strongest. This led me over the years to the pioneers of environmental learning: Montessori Method, Suzuki Violin, and Tim Gallwey's *Inner Game of Tennis*, to name just a few. . . .
>
> Now if we agree with the evidence that the human cognitive facilities are made up of a *doing* mentality, an *image* mentality, and a *symbolic* mentality, then any user interface we construct should at least cater to the mechanisms that seem to be there. But how? One approach is to realize that no single mentality offers a complete answer to the entire range of thinking and problem solving. User interface design should integrate them at least as well as Bruner did in his spiral curriculum ideas.

Kay notes that the GUI interfaces that emerged from Xerox PARC in the late 70's [82] were designed to guide the user through just such a progression. *Doing* (with the mouse) with *images* (the icons of the interface) makes *symbols* (leads to the abstraction of activity in the Smalltalk programming language).

Many computer science researchers (Strohecker [158], Papert [127], Kay [83], [84], and Pree [133]) have related Piagetian learning stages and interface design. The PAHCI layers have a natural analog in Bruner and Piaget, and to Bloom's taxonomy of learning [11]. As learning or adaptation at one layer is stabilized, it sinks into invisibility to become automatic — just as the blind man's cane becomes an extension of his arm (Wittgenstein and Merleau-Ponty in Suchman [159]).

An intelligent interactor must constantly be prepared to seek remedies at every layer of the interface. This may include detecting loss of input stream, misrecognition, awkwardness of interface, crowding of the tablet space, boredom, dullness, or wrong answers. For example, Hyperflow [92] acts at several layers, typically structor, task, and direction, but it may even control the device drivers. It also conveys sophisticated semantic intents from user to machine. It allows the user to elaborate the interface at its higher layers.

A.3 Elaborating the Layers of PAHCI

This model was introduced above. We extend that definition and then provide for each layer a description, cognitive correspondence, example activities and range of control, possible adaptations, and natural unit of representation.

A.3.1 The PAHCI Model of the Adaptation Environment

As mentioned above, the ISO OSI model abstracts communication control into seven layers of activity, and any two communicating systems must reach agreement within each. The specific agreements may take the form of compatible voltages, symbol sets, protocols, and so forth, but these are not significant to the abstract model. Every layer in the sending system talks to the corresponding layer in the receiving system. This is

analogous to communications between two businesses. I talk to you; my staff talks to your staff; my boss talks to your boss. In the PAHCI model the task of *adapting* communications between user and system is divided into the eight layers.

A given layer calls upon the methods of the layer below to succeed in its adaptations, and "conditions" the layer below with contextual parameters. It offers potential representation structures [26, 73] to the lower layer for the reception of the incoming data stream. Its methods, in turn, constitute a suite of services and information for the layer above it. The generation and evaluation of candidate representations by the superior layer enables the lower layer to respond robustly to varied input streams. It provides a mechanism for detecting and correcting errors at higher layers of the interface.

As mentioned in Chapter 8, human communication exhibits natural and immediate detections and repairs of errors, but this has proven to be one of the most difficult elements of interface design [107, 159]. The corrections are harder to build at higher layers of the interface model, but such building is critical to the success of flexible user interfaces. Otherwise, the interface designer must abandon adaptation for consistency.

Next we give a more developed version of the hierarchical chart with suggestive (but not definitive) connections between system hardware/software and the user's cognitive schema.

A.3.1.1 Physical/Physiological

System facility: digitizer driver.

User faculty: enactive.

Locus of adaptation: This layer adapts the pen resistance on the writing surface, the relative position, or rate of point samples per second.

Unit of measure: point.

A.3.1.2 Sensory Link

System facility: direct visual feedback.

User faculty: enactive and iconic.

Locus of adaptation: This layer adapts items like the inking, parallax adjustment, movement of objects, tap time-out, press time-out, multiple-stroke time-out. It might adapt by settings in the driver or in some low-level objects (like the screen, acetate, pen, or scribble). It might modify smoothing or shearing transformations as necessary.

Unit of measure: stroke.

A.3.1.3 Assembly

System facility: algorithmic.

User faculty: iconic.

Locus of adaptation: This layer assembles the points and strokes into taps, gestures, scribbles, data about extrema (bounding rectangle, hotpoint). It might adapt via neural network training and self-modifying algorithmic filters.

Unit of measure: scribble.

A.3.1.4 Direction

System facility: target for scribble.

User faculty: focus of attention.

Locus of adaptation: This layer directs the tap, movement, scribble, or word to an object. It might adapt by messages to objects and the establishment of streams to objects: broadcast, one-to-one, and so forth. It might also adapt by recording features of historical input patterns compared to the intended receiver and by noting shifts of attention such as "become first responder," or "call for first responder" [125].

Unit of measure: first responder.

A.3.1.5 Context

System facility: current task.

User faculty: syntactic.

Locus of adaptation: The layer deals with abstract features and appropriate signs. Its adaptation is done by modifying the context, creating hypothetical representations of input, evaluating representations. Adaptation may occur by one or more learning systems which train on the most successful representations. Adaptation may also take the form of creating new transactions or recalling old transactions.

Unit of measure: context structure and feature environment.

A.3.1.6 Task

System facility: work object.

User faculty: semantic, symbolic.

Locus of adaptation: This layer maps signs (in context) into symbols. In the pen setting, this might be one chord in a musical composition, one word or sentence in a document, or one figure, figure group, or animation sequence in a Hyperflow session. It might be one equation or the answer to one question. This level would be responsible for validation by parsing if appropriate or necessary.

Unit of measure: task action—akin to the transaction.

A.3.1.7 Structor

System facility: tool.

User faculty: capability.

Locus of adaptation: This level attempts to map symbol groups into data and actions. Such mappings are very specific to the particular application: for example, business presentation, phone message, video shopping trip. In a pen-based system, a task session (the unit of this layer) might be one drawing, one composition, one document, one musical playback, or one visual program.

Unit of measure: task session.

A.3.1.8 Manager

System facility: operating system.

User faculty: meta-language, goals.

Locus of adaptation: This level is the highest in the hierarchy of adaptive interfaces. It corresponds to the operating system in the computer. In the human, it corresponds to the current goals of the user. What is she trying to accomplish at the moment, and what are the patterns of situated action? Adaptation at this level implies the ability on

the part of the computer to map patterns of user intention (dare we say *meaning*?) across applications. To this end, it might compare requests from the user to the store of already existing data and potential actions. It might then select or recommend tools (applications) for the fulfillment of unsatisfied requests.

Unit of measure: goal structure.

A.3.2 Relations between Layers

Significantly, each layer continually engages a corresponding cognitive structure (akin to schema in Piaget's phrase) of the human user. The intent is for the layers to serve each other in a cooperative collaboration to discover and track the intent of the user, and thus to repair or correct actions that interfere with the reception of the user's message or execution of the user's purpose.

For example, the physical layer engages the user's tactile and kinesthetic sense. The sensory link engages the tight visual feedback loop of the eye-hand circuit. Both of these layers constitute the more primitive associations of stimulus-response mechanisms—one in the "muscle memory" and the other in visual memory. The assembly layer assembles the input stream of primitive subunits of signs — taps, drags, and strokes into the signs themselves.

The direction layer seeks or maintains a "taker" or "target" for this input. It corresponds to the user's basic sense of directing input toward some object. Every action is imperative in some sense; it commands the system to do something about it. The "doing" may be as simple as inking a doodle and ignoring it. Since several media may be active at a time (taps and voice for example), this element of the user interface may need to maintain several "first responders" — one for voice commands, one for pen movement, and

another for the keyboard. If the phone rings in the middle of an intense moment of musical composition, the voice command to "take a message" goes to one "first responder" while the taps go to a composition editor and the keyboard strokes go to a MIDI recorder (and thence to the composition editor). This level is the earliest, most primitive realization of what Kay calls the "semantic fovea." The above example also indicates the degree to which the direction layer depends on the context layer.

The task layer in turn supports that notion of context. It orients the focal point of attention in a past and hypothesized future. All human actions are situated (in Suchman's [159] phrase); they are part of a drawing, a composition, an editing session, a request for information, a purchase, a viewing of a TV show, and so forth. The coin of this realm is transactional syntax. In a given transactional context, certain signs and groupings of signs are syntactically valid and acceptable.

The word *structor* was coined by the author specifically to connote applications that construct, "destruct," structure, restructure, and instruct objects. It generalizes the notion of editor. The structor layer attempts to interpret the sign as a symbol; it is continuously interacting with the symbolic sense of the user. A given sign might be the next character of a paragraph, a command to turn attention elsewhere, or a mindless doodle. This is the first layer where the input stream meets candidate representations of semantic significance. It maps symbols into models of data and actions within the current application. Humans instantly make meaning of the input stream flowing inward second by second. They "see" analogies, models, patterns, puns, and surprises constantly. Systems to some degree must mimic this semantic infusion of meaning to support the users at this level. This is not a capability that currently extant or immediately forthcoming; it relates

to the deepest current researches in AI. For example, a musical expert system might adapt the input of a composer to helpfully anticipate her thematic intent.

The highest type of candidate representations may be thought of as mappings from symbol sequences to user intentions. The manager layer takes these maps from symbols to intents as input, and seeks patterns of intention across applications. This is most nearly related to cognitive science's efforts to model human learning and communications.

A.3.3 Examples of Adaptation at the Different Levels

Adaptation at the physical level occurs electronically — by either digital or analog means. An example of the former is an increase or decrease in the sampling rate, or a sampling rate that is constant over distance instead of time. This is most often implemented as a parameter to the digitizer device driver. The dynamic adaptation of in-line amplifiers or analog adaptive equalizers also furnish illustrations.

In the sensory link layer, the adaptation is illustrated in the dynamic time-out for multi-stroke scribbles. This may be calculated by examining the successfully-recognized scribbles to determine the average time (and standard deviation) between successive strokes of one scribble versus the average time between separate scribbles. The time-out could be dynamically adjusted to reflect the former average plus three or four standard deviations to ensure that nearly every input would be at near optimal speed without misinterpreting the input stream.

Another adaptation is the shift in alignment to correct for parallax. This almost certainly depends on direct manipulation from the user. Only the eye of the beholder can indicate when the ink is appearing where it "should" beneath the pen.

A speech recognizer may have to adapt at several of these layers. Continuous waveform transformation may go on at the physical layer. Adaptive sampling and filtering (such as CELP) may be considered as adaptations in the data link or the physical layer. The assembly layer would seek to identify phonemes and to gather them into words or phrases. Adaptation at this layer might be managed by neural networks.

The task layer might adapt by recalling old transactions (as do commercial applications like Quicken™, for example), or suggesting the creation of new transactions built from combinations of old transactions. The macro builder written for Macintosh computers uses the log of user actions to look for combinations that have been used in the past. It formulates a macro to combine these replicated patterns into a single action.

Adaptation at either the structor level or the manager level likely implies a degree of programming or reprogramming. The system might create new programs that correspond to intents discovered by the manager layer. If the viewer seems to be grazing heavily on home-shopper channels, the system might search out sources of such channels and make them available to the viewer. News filters and data base ferrets are in this class as are Kay's agents with cloned goals. The hypothetical fifth generation computer operating systems, driven by goal criteria instead of procedures, might have adapted at this layer.

A.3.4 Examples of Adaptation in Three Application Settings

Consider the user interfaces to three typical tasks regularly undertaken by novice computer users: word processing, drawing a diagram, and writing a musical composition. Myers [120] raises a subtle question that complicates a discussion of interfaces to even simple software products such as these: Where does the interface begin and the application end? Or equivalently, when the system adapts, are we witnessing adaptation of the interface

or of the application itself? The division is partly subjective, but Myers gives a hint when he notes that the interface of applications in the 60's probably consumed no more than 10% of the system code. Today's applications typically spend 75% of their code on the interface. If such a large share of a modern application is given to the interface, obviously much of the adaptive behavior will be in the interface.

A.3.4.1 Word Processing

The generic word processor is familiar enough to be a good first example. The application may be envisioned in terms of input and output. The actual written input is the minimal set (assuming no mistakes!). We further assume some minimal set of formatting requests. The stored document or the printed page serves as the product of this task session. This skeletal description stands for the task. The interface displays the partial document, offers help, spelling and grammatical corrections, and formatting prompts. We might assume a pen-based notebook environment.

The lower layers of adaptation might include increasingly successful recognition of the user's input. At the assembly layer, the system might improve in its ability to segment letters correctly. It might distinguish between cursive, unconnected cursive, and block printing. The task layer might include a dictionary or frequency tables to improve recognition. These might adapt to the vocabulary and frequencies accumulated from the user. Common spelling mistakes (such as "teh" for "the") might be corrected without questioning or interrupting the user (as does Microsoft's Word for Windows™ 6.0).

A.3.4.2 Composing Music

The act of composing music offers several intriguing possibilities. Lower layers could adapt to accept a scratch as the input of a quarter note at that location. Higher layers could adapt in ways supported by the representations given by the task layer.

Suppose that the current representation is the chord. Suppose further that the frequency of certain chords has been tracked for the composition in progress. After the first or second note, the rest of the chord might be presented (perhaps in halftone gray). If left, the chord is darkened to the usual color and the task action is assumed complete. At the structor layer, the representation might be the sonata, and the *progressions* of chords might be anticipated.

A.3.4.3 Drawing a Figure

Similar adaptations are possible for the drawing environment. Movements of a figure might be zipped to anticipated symmetries. Beginnings of figures might be completed in gray. If the user moves on to another figure, the presumptive completion would show as a change to black.

A.3.5 A Primitive Example Traced through the Levels

Suppose that a new student begins using a mathematics learning system. The system might require some knowledge of the current status of the user's mathematical maturity prior to selecting a diagnostic test or recommending some actions. Assume further that the system currently needs to know if the student understands (recognizing all the ambiguity of that word) that two plus three equals five.

We now trace this simple request through the layers of PAHCI to illustrate the responsibilities of each level, and to suggest the range and types of adaptation associated with each level.

A.3.5.1 Manager

We assume that the system has already "decided" to assess the user's mathematical knowledge. It may query the database to see if this knowledge is already present in one of several forms — past user actions, test results, instructor records, and so forth. If the

knowledge is not already at hand, it must decide how the notion of "2 + 3 = 5" should be presented to the user. At least theoretically, it may be possible to frame the query in many potential contexts. In fact, if absolutely no contextual or user-specific information is present, the system may try textual, graphical, or audio queries to establish contact with the user (who may after all be blind, deaf, or illiterate).

We must further assume that the user *desires* to communicate this tidbit of knowledge to the system. (In other words, a student's willful effort to frustrate the system will always succeed!) Various (natural) languages might be tried aurally and textually. It may represent the problem with dots instead of Arabic symbols. Adaptation at this layer is probably under the control of an expert system supported by a database that models historical information about the user. If a series of meaningful answers emerges, the system will remember the successful medium/language/approach and refer to it more quickly in future sessions.

A.3.5.2 Structor

Since this example is rather simplistic, namely, one task action long, little is contributed by the structor layer. In more general settings, it might regulate the frequency and timing of hints, retain records of specific capabilities associated with the user, and model probable or expected responses. Within the narrow confines of this trivial example, it merely generates the single-item "test" in the chosen format (dot pictures, English, Japanese, oral, or textual) and then evaluates the response.

In more elaborate examples, this layer might adapt by changing the sequence of user actions necessary to trigger the scoring of the answer. If the user desires, the scoring might follow the entry of the last answer on a page. Alternatively, the application might provide a "Score" button for the user. If the student always works on a daily ration of five

worksheets, it might suggest that the fifth "Score" request be made equivalent to "score the last worksheet, and quit the worksheet application if no mistakes are made on that worksheet."

This layer also manages the tasks associated with gross editing operations. It permits opening, saving, deleting, cutting, pasting, and such actions.

A.3.5.3 Task

This task action is simply to collect the answer to the single problem. This layer would receive the interpretation of the strokes (if Arabic numerals are assumed) or taps (if dot representation is assumed). If the answer is unintelligible or meaningless, it may propose remedies such as re-entering the answer. If it is unsuccessful, it may defer to the layer above it for a broader or more sophisticated remedy.

A.3.5.4 Context

The context is prepared and made available to the recognition engine at the lower levels. The context might be the expectation of an Arabic numeral, or a sequence of taps (to use the dot representation of numbers). Various corrections may be available. Adaptation may occur in any one of these areas. The feature set may be dynamically adapted to reach satisfactory confidence. The context might expect a single Arabic numeral, "boxed" Arabic numerals, or unsegmented sequences of Arabic numerals.

A.3.5.5 Direction

This layer maintains the current target of the input stream. In this example it is probably no more than twofold: the numeric scribble recognizer and (if redirected occasionally) the gesture recognizer to detect crossouts or copies.

A.3.5.6 Assembly

The recognizers operate at this layer. The input stream is separated into scribbles that are recognized as numerals (Arabic or tapped). In the case of SCL, this layer adapts by neural network learning and by the creation of new prototypes when necessary. In this example, the user may change from a one-stroke to a two-stroke *5*. The system creates the new prototype and is thereafter ready to receive answers in this form.

This layer also collects data about the average bounding rectangle and presumed baseline to "condition" the context layer above. In turn this layer uses this adaptive contextual data to improve its recognition. This layer also generates confidence data which performs many services for higher layers in evaluating questions: about direction (Do we have the right target for the scribbles?) and about context (Have we misaligned the baseline? Are we employing the proper feature set? Do we need to activate another recognizer? Is an alternative representation necessary to interpret the input stream?).

A.3.5.7 Sensory Link

This layer gathers the incoming point stream into strokes and governs the inking. It naturally adapts to minor variants in the stroke alignment or size. It includes algorithmic preprocessing of the stroke and point data. If the translated output is regularly enlarged by the user, this layer would receive the instructions to enlarge the translated character (and send a notification to the next higher layer that this has occurred). At the context layer, this might adaptively become the new default typesize for subsequent sessions.

A.3.5.8 Physical/Physiological

Adaptations at this layer are purely physical, including changes in the stroke timeout and alterations to the sampling rate. If we are using the dot representation of data, the sampling might be reduced. If the user regularly uses the pen at a low angle to the

digitizer glass, an automatic parallax correction might be passed to the visualization elements of the sensory link layer to better place the "ink."

Within a session, the system may often have to switch contexts, and therefore to restore earlier working sets of the transaction vocabulary. This process may be regarded as analogous to the way that the semantics of "2 + 3" grow very gradually out of the syntactic pattern matching of many hours' drill. Similarly, the interface emerges slowly from the pattern of user demands, responses and corrections.

A.4 Potential Advantages for the Hierarchical Design Model

We suggest two potential advantages for the PAHCI hierarchical design model. It supports better software design by its abstraction of the user interface. It also supports abstraction of the cycle of proposing and evaluating candidate representations in adaptation actions.

A.4.1 Software Design and the PAHCI Model

The first advantage has to do with the software engineering of PAHCIs. It allows well-structured, standardized designs that permit (even encourage) experimental improvements because the incremental changes are well-defined. They are confined either to one layer or to the relationship between two layers, leaving the rest of the structure intact. Each design increment and subsequent experimentation are small in scope. This scheme serves to encapsulate the fuzzy notion of adaptability into manageable segments. Moreover this structure is not only efficient in terms of software design, but is insightful and fruitful in terms of human interaction. It permits the relation of software capabilities to specific aspects of human cognition (enactive, iconic, symbolic, semantic foveal, and so

forth). Thus it leads to adaptations more likely of favorable receipt by the human partners in the interaction.

A.4.2 Better Management of Candidate Representations

Second, this model increases the likelihood of success in the cycle of proposing candidate representations, testing those representations, and evaluating them. Because the layers are smaller in scope, the amount of "knowledge" needed by the next higher layer (that is, the size of the domain of candidate representations) is minimized. This leads to a shorter cycle of proposing, testing, evaluation, and adaptation (when successful), hence one that is more likely of success.

We used the PAHCI hierarchical model to position SCL in the spectrum of adaptation. We sought a technology designed to manage adaptations in the *sensory link* and *assembly* layers of the model. These layers relate to the user's *enactive* and *iconic* faculties [14]. Thus the adaptations at these layers must correspond to some combination of enactive and iconic features derived from the incoming data stream. SCL employs precisely such features (energy-related and graphical) to manage its adaptation tasks. Even this brief analysis suggests the benefits of the PAHCI reference model to design methodology, and also shows its utility as a taxonomy for cataloging the adaptations between the user and the system.

A.4.3 PAHCI — A New Window on How Children Learn

PAHCI may even open a new window on the *processes* of the educational interaction. Do we encourage standardization or invention? Do we offer standardized prototypes to guide children toward standard forms for digits and letters? Since SCL operates at the level of velocity and acceleration, children may invent very terse substitutes for standard

symbols. The very directness of the system's response to primitive energy profiles may help us probe below the surface of learning. It catches the *process* of drawing more than the *product* — the intent more than the outcome. The PAHCI model offers a possibility for starting fresh with the pen paradigm and learning more about learning as we do so.

A.4.4 Better Interfaces Call for Better Interface Models

Kay [84] urgently voices the need for better interface models. First he assures the reader that 100 MIPS per user is just over the horizon. (He was writing in 1990 — the Power PC 604 offers 100 MIPS and the Pentium P54C offers 150 MIPS, both in 1994.) Then he writes:

> The only stumbling place for this onrushing braver new world is that all of its marvels will be very difficult to communicate with, because, as always, the user interface design that could make it all simple lags far, far behind. If communication is the watch word, then what do we communicate with and how do we do it? We communicate with our selves, our tools, our colleagues and others, and our agents. Until now, personal computing has concentrated mostly on the first two. Let us now extend everything we do to be part of a grand collaboration with one's self, one's tools, other humans, and increasingly, with agents: computer processes that act as guide, as coach, and as amanuensis. The user interface design will be the critical factor in the success of this new way to work and play on the computer....If the IBM 3270/PC way of doing things is "machine code," doesn't that make the Mac the COBOL of user interface design?

Alan Kay also gives a hint about the most significant future direction for research in adaptable user interfaces:

I read McLuhan's *Understanding Media* and understood that the most important thing about any communications medium is that message receipt is really message recovery; anyone who wishes to receive a message embedded in a medium must first have internalized the medium so it can be "subtracted" out to leave the message behind. When he said "the medium is the message" he meant that you have to become the medium if you

use it. That's pretty scary. It means that even though humans are the animals that shape tools, it is in the nature of tools and man that learning to use tools reshapes us.

Michael Swaine, columnist and futurist for *MacUser* writes [161]:

> I believe that the next decade will see radical innovations in software technology, implemented on fast processors, that will change our work habits as fundamentally as the graphical user interface did. . . . IBM is hard at work on technologies for what it calls human-centered computing, a new interface model that broadens the user/computer bandwidth, giving users choices in how they work with computers, making their interaction more natural and intuitive. Human-centered computing engages human senses (sight, sound, touch) more fully in the interaction and encompasses speech and handwriting recognition, intelligent agents. . . . Sure, there are some technical hurdles to be jumped in building such an interface. Right up until the day IBM or Apple delivers, skepticism is warranted. But on that day, GUI is history.

It is *not* our claim that pen-based adaptable human-computer interfaces systems will inherit the earth — will be the "object-oriented programming" of interfaces. We *do* claim that adaptable pen-based systems are a substantial thread in the future tapestry of interfaces. They are at least one legitimate successor to "the COBOL of user interface design."