Towards collaborative agents for automatic on-line handwriting recognition.

Lambert Schomaker[1], Eduard Hoenkamp, Marshall Mayberry
Nijmegen Institute for Cognition and Information and University of Texas, Austin

Abstract

A important problem in many areas of complex information processing is to integrate heterogeneous information from different sources. For handwriting recognition we accomplished such an integration through an agent architecture of three co-operating agents. One agent represents the bottom-up knowledge as generated by a shape-based classifier of unistrokes. The second agent contains top-down (syntactic) knowledge about the language to be interpreted. The third agent is the user who produces written symbol shapes and who can be queried by the other agents. The chosen application paradigm is the real-time input and execution of Scheme code. Although this is not a practical domain, it contains all the essential elements of a full text-entry application. Notable improvements can be gained over a pure bottom-up approach to pen-based computing. [1]

Introduction

Complex systems often have to integrate information varying in kind and granularity, be it *nominal* (single bits), *ordinal* (ordered lists), *quantitative* (intervals and ratio scales), or *symbolic* (declarative knowledge such as logical propositions).

In the past, we have been successful in using neural networks for handwriting recognition [13, 14]. In spite of its success, the approach inherits the limitations of neural networks and statistical classifiers: (1) the fixed dimensionality of the feature vectors (all information, including the irrelevant, is used in any decision), (2) the massive amounts of training data, and (3) the difficulty of processing ordinal data such as word-hypothesis lists in an elegant way. Stated differently, the neural network approach assumes (as do many statistical approaches) that the integration problem can be reduced to identifying and parametrizing the intricate convex decision boundaries between classes, usually in a single processing pass. As the complexity of class boundaries increases, the amount of parameters and training data will increase steeply.

Recent years have shown promising developments in the area of so-called intelligent agents [17]. On the one hand this metaphor of autonomous, communicating, and sometimes even mobile entities, has generated a different way of looking at processes that share or negotiate information. On the other hand it has produced new applications based on this metaphor. For example, algorithms emerge for cooperative learning in multiple-agent systems [2, 6, 12], including genetic algorithms [5, 9, 4]. For multi-sensor fusion, algorithms overcome the problems of integrating measurements from multiple and unreliable sensors in real-world systems. Averaging, voting and negotiation algorithms have been proposed, like Byzantine agreement [3]. Many of these algorithms are known to converge [11].

This shift to multi-agent systems, in concept as well as in applications, may open a new avenue for pattern recognition, especially pertaining to the hard problem of information integration.

[1]Ref.: Schomaker, L., Hoenkamp, E. & Mayberry, M. (1998). Towards collaborative agents for automatic on-line handwriting recognition. Proceedings of the Third European Workshop on Handwriting Analysis and Recognition, 14-15 July, 1998, London: The Institution of Electrical Engineers, Digest Number 1998/440, (ISSN 0963-3308), pp. 13/1-13/6.

The benefits would be easy to assess: Will there be a higher classification performance than the traditional multiple-classifier algorithms and bottom-up/top-down integration approaches?

To explore this question, we decided to take an example where two different types of information can be easily discerned: handwriting recognition beyond the word level. One kind of information concerns the stroke shapes which the user produces. As the user is not randomly producing words, the other kind of information is the way the grammar constrains the interpretation of the data. A choice then would be to let subjects write in English, and evaluate how the system fares on recognition. This would call for a robust parser of English, and an intricate negotiation protocol. With it, the performance of the recognizer would very much be depend on the performance of the parser, which is undesirable. Instead, we decided to start experimenting with the much simpler problem of on-line recognition of program source code written by hand. The programming language we chose Scheme, a variant of Lisp with a clean, formally described syntax and semantics. Next we will describe the three agents in our system:

- Bottom-up shape classifier (SC agent)
- Top-down expression classifier (EC agent)
- the human user (US)

An agent for bottom-up advice: The shape classifier

A traditional shape classifier is wrapped into the agent framework, using a dedicated communication channel (socket I/O [7]) and protocol. The classifier is thus part of the three-agent consortium as the primary processor of the bottom-up information stream. A unistroke concept is used here: Only one single pen-down stroke is produced per character or symbol. The handwriting classifier processes the output (x_k, y_k) from the tablet. Each pen-down stroke is then spatially resampled to 60 samples, the running angle ϕ_k is added in the form of 59 $(cos(\phi_k), sin(\phi_k))$ pairs and the resulting 238-dimensional feature vector is matched with a number of templates which were trained on the system by using the Kohonen Learning Vector Quantization algorithm[8]. The templates consisted of shapes for all syntax elements (parentheses and letters) in Scheme. Typically 5-10 examples were provided by the user. Simple Euclidean distance was used for nearest-centroid matching.

An agent for top-down advice: The expression classifier

The agent that classifies expressions (EC) embeds (1) a shift-reduce parser and (2) the grammar for Scheme[1]. Below is the fraction of the grammar that pertains to the code in figure 1.

⟨program⟩ ⟶ ⟨command or definition⟩*
⟨command or definition⟩ ⟶ ⟨command⟩ | ⟨definition⟩
⟨definition⟩ ⟶ `(define` ⟨variable⟩ ⟨expression⟩`)`
| `(define (`⟨variable⟩ ⟨def formals⟩`)` ⟨body⟩`)`
| `(begin` ⟨definition⟩*`)`
⟨def formals⟩ ⟶ ⟨variable⟩*
| ⟨variable⟩$^+$. ⟨variable⟩
⟨body⟩ ⟶ ⟨definition⟩* ⟨sequence⟩
⟨sequence⟩ ⟶ ⟨command⟩* ⟨expression⟩
⟨command⟩ ⟶ ⟨expression⟩
⟨expression⟩ ⟶ ⟨variable⟩
| ⟨literal⟩
| ⟨lambda expression⟩
| ⟨conditional⟩
| ...
⟨conditional⟩ ⟶ `(if` ⟨test⟩ ⟨consequent⟩ ⟨alternate⟩`)`
⟨test⟩ ⟶ ⟨expression⟩
⟨consequent⟩ ⟶ ⟨expression⟩
⟨alternate⟩ ⟶ ⟨expression⟩ | ⟨empty⟩
...

Each time the expression classifier receives a token from the shape classifier, it changes state. The new state reflects the transition of the shift-reduce parser as sanctioned by the grammar. The expression classifier then informs the shape classifier (and the user interface) which tokens to expect on the next input of a unistroke or key press. The shape classifier uses this information to narrow down the possible interpretations of the next token it gets from the user interface. The expression classifier thus comes in lieu of a possibly contrived reduction of symbolic (i.e. syntactic) information to parameter settings.

User Interface

The user interface (written in Tcl/Tk) initiates both expression classifier and shape classifier and monitors their state. There are two windows which initially appear when the program is started: the primary window (Figure 1) and a small window which only echoes the last token recognized. When an ink chunk is classified by the shape classifier, the corresponding on-screen button flashes green, indicating that the expression classifier has consumed the token, and a new set of expected tokens are displayed. However, when an ink chunk is not recognized, the next most similar token is highlighted in red if it is among the expected tokens, otherwise, a separate window of buttons appears which show all the symbols the recognizer has in store, with the symbol most similar to the unrecognized script highlighted here in red. If the intended symbol is among those the recognizer already knows about, the user need only push the corresponding button (or the `Accept` button on the main window if the highlighted button is the intended token). If the script is a new symbol, however, then the user can enter it at the bottom of the pop-up window, so that it will be added to the list of symbols in the recognizer. Note that in this concept, free use is made of the available modalities, instead of focusing on the handwritten shape input only.

What makes a software module an agent?

A reasonable criticism would be: Why do you call these software modules 'agents'? Why not use the term 'module', 'daemon', 'expert' etc. as used in earlier studies [15]? Fortunately, the agent paradigm offers a number of useful definitions [17], as well as new forms of functionality which were not present - or only rudimentarily present - in older paradigms. Here, we follow the AI-style definition which requires: explicitly represented goals and beliefs, autonomy, and encapsulation. Traditional functions (like a classifier) can be wrapped up as a module within an agent, by adding communication layers, and adding the necessary knowledge base which describes the goals (what are you trying to achieve?), beliefs (how do you think you can achieve these goals?) and confidences (how certain are your statements?). It is this additional information which is the basis for the inter-agent negotiation. It is expected that application of the paradigm to pattern recognition problems may provide for the realization of complex decision boundaries.

Results

The overall performance of the system is the result of a complex interaction process in which bits of information are sometimes provided by the human user (by clicking on virtual keys in the interface), sometimes by the handwriting shape classifier, and sometimes by the expression classifier, with the result of 100% correct classification. The Scheme expressions can be evaluated immediately by pressing a button. Although the concept was implemented to explore the merits of the agent metaphor, the usability was much better than expected. However, it was also found that there is a catch. As long as the context is limited to a single procedure in Scheme, the number of tokens from lumped categories like **VAR, STRING, CHAR** or **NUMBER** is sufficiently limited such that the user will benefit from the information which the parser agent enters into the interaction. However, since the token category **VAR** will be expected by the parser in up to 94% of

the token input events, as measured in a 27310-token corpus of Scheme code, the benefits will drop steeply if more than a single Scheme function is assumed to belong to the current input context. In natural languages, the incidence of names is much lower than in programming languages.

Table 1: Information contained in a Scheme source-code corpus of 27310 tokens. "Lumped" means that instances of variable names, numbers, single characters and strings were reduced to the single, corresponding token category.

Symbols	$N_{alphabet}$	$^2log(N_{alphabet})$ (bits)	*Entropy* (bits)	*Redundance* (bits)
Raw token stream	2003	11.0	6.3	4.7
Lumped token stream	28	4.8	2.4	2.4

Table 2: Token expectancy by the expression classifier (parser): average number of token alternatives predicted by the parser, per input token, using a source-code corpus of 27310 tokens. "Lumped" means that instances of variable names, numbers, single characters and strings were reduced to the single, corresponding token category.

Symbols	Avg. $N_{alternatives}$
Raw token stream (scope=whole corpus)	1891.5
Raw token stream (scope=single function)	97.4
Lumped token stream	16.0

Tables 1. and 2. summarize some aspects of the information contained in Scheme source code and the information generated by application of the grammar. With a token alphabet of 2003 tokens, a shape classifier would need to generate about 11 bits of information. With unlimited scope of variable names, the expression classifier (parser) will only generate 0.6 bits (Table 2). Although the parser is quite successful in predicting the general (lumped) token category, the information generated by using the grammar [10] is clearly limited here. But by limiting the scope of variable names to single Scheme functions, the shape classifier will need to make a choice of 1 in 97.4 as opposed to 1 in 2003. In languages with less use of free names, numbers and strings, the advantage of an incremental parser aiding in user input will be evidently much more clear. Refinements can be easily envisaged, adding probabilistic information to the expression classifier. Such an approach, however, would be incompatible with the grammar-oriented and parameter-free design philosophy aiming at broad applicability (read: independence of the Scheme programming style of the user). More empirical results are needed to quantify the system performance. Summarizing, both the multiple-agent paradigm and the current paper are in a sense a revival of long-known concepts. Experiments with handwriting recognition of Fortran code originate in the early sixties. The results show, however, that a consistent integration of bottom-up and top-down information, and a proper and strongly interactive user-interface design may generate an application which is much closer to a usable and useful application than an isolated and sub-optimal shape classifier. On the basis of the success of this experiment, a number of new developments have been started to create a convenient platform for agent-based experiments [16]. Current experiments are based on the Java programming language (JatLite). The most important research topic, now, is the design of a *negotiation protocol* and a *language* which are optimized for the multiple-agents paradigm in pattern-recognition problems.

References

[1] H. Abelson, N. Adams, D. Bartley, G. Brooks, R. Dybvig, D. Friedman, R. Halstead, C. Hanson, C. Haynes, E. Kohlbecker, D. Oxley, K. Pitman, G. Rozas, G. Sussman, and M. Wand. Revised report on the algorithmic language scheme. Technical report, MIT, 1986.

[2] Albers, Wulf, and J.D. Laing. Prominence, competition, learning, and the generation of offers in computer-aided experimental spatial games. In R. Selten, editor, *Game Equilibrium Models*, volume III: Strategic Bargaining, pages 141–185. Springer Verlag, 1991.

[3] R.R. Brooks and S. Sitharama Iyengar. Robust distributed computing and sensing algorithm. *IEEE Computer*, 29(6):53–60, June 1996.

[4] G.O. Dworman, S.O. Kimbrough, and J.D. Laing. On automated discovery of models using genetic programming: Bargaining in a three agent coalition game. *Journal of Management Information Systems*, 1995.

[5] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA, 1989.

[6] I. Noda H. Matsubara and K. Hiraki. Learning of cooperative actions in multiagent systems: A case study of pass play in soccer. In *Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium*, page 63. AAAI Press, 1996. Technical Report SS-96-01.

[7] K. Hartung, S. Muench, Schomaker L., T. Guiard-Marigny, B. Le Goff, R. MacLaverty, J. Nijtmans, A. Camurri, I. Defee, and C. Benoit. Di3 - development of a system architecture for the acquisition, integration, and representation of multimodal information. report of esprit project 8579/miami, march 25, 1996. Technical report, NICI., Nijmegen, The Netherlands, 1996.

[8] Teuvo Kohonen. Learning Vector Quantization. *Neural Networks*, 1(Supplement 1):303, 1988.

[9] J.R. Koza. *Genetic Programming*, volume II. MIT Press, Cambridge, MA, 1994.

[10] W. Kuich. On the entropy of context-free languages. *Information and Control*, 16:173–200, 1970.

[11] A. Rubenstein. A perfect equilibrium in a bargaining model. *Econometrica*, 50:97–109, 1982.

[12] J. Schmidhuber. A general method for multi-agent reinforcement learning in unrestricted environments. In *Adaptation, Coevolution and Learning in Multiagent Systems: Papers from the 1996 AAAI Spring Symposium*, page 86. AAAI Press, 1996. Technical Report SS-96-01.

[13] L. Schomaker. Using stroke- or character-based self-organizing maps in the recognition of on-line, connected cursive script. *Pattern Recognition*, 26(3):443–450, 1993.

[14] L. Schomaker. From handwriting analysis to pen-computer applications. *Electronics Communication Engineering Journal*, 0(0), 1998. In Press.

[15] H.L. Teulings, L.R.B. Schomaker, J. Gerritsen, H. Drexler, and M. Albers. An on-line handwriting-recognition system based on unreliable modules. In *Computer Processing of Handwriting*, pages 167–185. World Scientific, 1990.

[16] L.G. Vuurpijl and L.R.B. Schomaker. Multiple-agent architectures for the classification of handwritten text. In *Submitted to IWFHR6, International Workshop on Frontiers of Handwriting Recognition*, 1998.

[17] M. Wooldridge and N.R. Jennings. Agent theories, architectures, and languages: a survey. In Wooldridge and Jennings, editors, *Intelligent Agents*, pages 1–22. Springer-Verlag, Berlin, 1995.

---

[1]The authors can be reached at:
schomaker@nici.kun.nl, hoenkamp@acm.org, martym@cs.utexas.edu

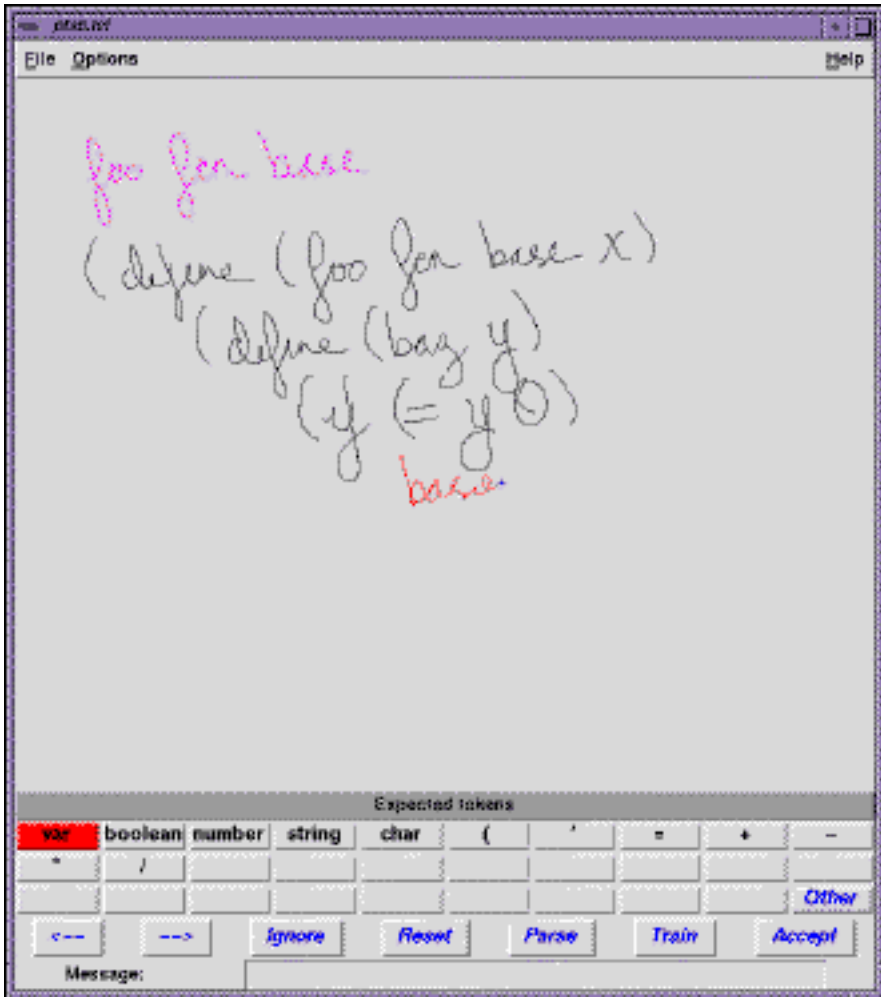

Figure 1. Main window of the user interface