Coevolution of Pattern Generators and Recognizers

Stewart W. Wilson
IlliGAL Report No. 2009006
May, 2009

Illinois Genetic Algorithms Laboratory
University of Illinois at Urbana-Champaign
117 Transportation Building
104 S. Mathews Avenue Urbana, IL 61801
Office: (217) 333-2346
Fax: (217) 244-5705

Coevolution of Pattern Generators and Recognizers

Stewart W. Wilson
Prediction Dynamics, Concord MA 01742 USA
Department of Industrial and Enterprise Systems Engineering
The University of Illinois at Urbana-Champaign IL 61801 USA
wilson@prediction-dynamics.com

May 25, 2009

Abstract

Proposed is an automatic system for creating pattern generators and recognizers that may provide new and human-independent insight into the pattern recognition problem. The system is based on a three-cornered coevolution of image-transformation programs.

1 Introduction

Pattern recognition is a very difficult problem for computer science. A major reason is that in many cases pattern classes are not well-specified, frustrating the design of algorithms (including learning algorithms) to identify or discriminate them. Intrinsic specification (via formal definition) is often impractical—consider the class consisting of hand-written letters A. Extrinsic specification (via finite sets of examples) has problems of generalization and over-fitting.

Many interesting pattern classes are hard to specify because they exist only in relation to human or animal brains. Humans employ mental processes such as scaling, point of view adjustment, contrast and texture interpretation, saccades, etc., permitting classes to be characterized very subtly. It is likely that truly powerful computer pattern recognition methods will need to employ all such techniques, which is not generally the case today. In this paper we are concerned mainly with human-related pattern classes[1].

A further challenge for pattern recognition research is to create problems with large sets of examples that can be learned from. An automatic pattern generator would be valuable, but it should be capable of producing examples of each class that are diverse and subtle as well as numerous.

This paper proposes an automatic pattern generation and recognition process, and speculates that it would shed light on both the formal characterization problem and recognition techniques. The process would permit unlimited generation of examples and very great flexibility of methods, by relying on competitive and cooperative coevolution of pattern generators and recognizers.

The paper is organized into a first part in which the pattern recognition problem is discussed in greater detail; a second part in which the competitive and cooperative method is explained in concept; and a third part containing suggestions for a specific implementation.

[1]Not patterns in data, although the methods proposed could be useful there too. In the terminology of Sec.2, patterns in data are usually PV, not RPRV, mappings.

2 Pattern Recognition Problem

The following is a viewpoint on the pattern recognition problem and what makes it difficult. Let us first see some examples of what are generally regarded as patterns.

Characters, such as letters and numerals. Members of a class can differ in numerous ways, including placement in the field of view, size, orientation, shape, thickness, contrast, constituent texture, distortion including angle of view, noise of construction, and masking noise, among others.

Patterns in time series, such as musical phrases, price data configurations, and event sequences. Members of a class can differ in time-scale, shape, intensity, texture, etc.

Natural patterns, such as trees, landscapes, terrestrial features, and cloud patterns. Members of a class can differ in size, shape, contrast, color, texture, etc.

Circumstantial patterns such as situations, moods, plots. Members of a class can differ along a host of dimensions themselves often hard to define.

This sampling illustrates the very high diversity within even ordinary pattern classes and suggests that identifying a class member while differentiating it from members of other classes should be very difficult indeed. Yet human beings learn to do it, and apparently quite easily. While that of course has been pointed out before, we note two processes which may play key roles, *transformation* and *context*.

Transformative processes would include among others centering an object of interest in the field of view via saccades, i.e., translation, and scaling it to a size appropriate for further steps. Contextual processes would include adjusting the effective brightness (of a visual object) relative to its background, and seeing a textured object as in fact a single object on a differently textured background. It is clear that contextual processes are also transformations and that viewpoint will be taken here.

A transformational approach to pattern recognition would imply a sequence in which the raw stimulus is successively transformed to a form that permits it to be matched against standard or iconic exemplars, or produces a signal that is associated with a class. Human pattern recognition is generally rapid and its steps are not usually conscious, except in difficult cases or in initial learning. However, people when asked for reasons for a particular recognition will often cite transformational steps like those above that allow the object to be interpreted to some standard form. For this admittedly informal reason, transformations are emphasized in the algorithms proposed here.

It is possible to provide a more formal framework. Pattern recognition can be viewed as a process in which examples are mapped to classes. But the mappings are complicated. They are unlike typical functions that map vectors of elements into, e.g., reals. In such a function, each element has a definite *position* in the vector (its index). Each position can be thought of as a place, and there is a value there. An ordinary function is thus a mapping of "values in places" into an outcome. Call it a *place/value* (PV) mapping. If you slide the values along the places—or expand them from a point—the outcome is generally completely different. The function depends on just which values are in which places.

Patterns, on the other hand, are *relative place/relative value* (RPRV) mappings. Often, a given instance can be transformed into another instance, but with the *same outcome*, by a transformation that maintains the relative places or values of the elements—for example, such transformations as scaling, translation, rotation, contrast, even texture. The RPRV property, however, makes pattern recognition very difficult for machine learning methods that attach absolute significance to input element positions and values.

For general pattern recognition, what seems to be required is a method that is intrinsically able to deal with RPRV mappings. This suggests that the method must be capable of transformations, both of its input and in subsequent stages. The remainder of the paper lays out one proposal for achieving this.

3 Let the Computer Do It

Studying pattern recognition involves choosing a domain, creating a source of exemplars, and trying learning algorithms that seem likely to work in that domain. Here we are looking broadly at human-related pattern recognition, or relative place/relative value mappings (Sec. 2). Such a large task calls for an extensive source of pattern examples. It also calls for experimentation with a very wide array of transformation operators. Normally, for practicality one would narrow the domain and the choice of operators. Instead, we want to leave both as wide as possible, in hopes of achieving significant generality. While it changes the problem somewhat, there fortunately appears to be a way of doing this by allowing the computer itself to pose and solve the problem.

Imagine a kind of communication game. A sender, or source, S, wants to send messages to a friend F. The messages are in English, and the letters are represented in binary by ASCII bytes. As long as F can decode bytes to ASCII (and knows English), F will understand S's messages. But there is also an enemy E that sees the messages and is not supposed to understand them.

S and F decide to encrypt the messages. But instead of encrypting prior to conversion to bits, or encrypting the resulting bit pattern, they decide to encrypt each bit. That is, E's problem is to tell which bits are 1s and which 0s. If E can do that, the messages will be understandable. Note that F also must decrypt the bits.

For this peculiar setup, S and F agree that when S intends to send a 0, S will send a variant of the letter A; for a 1, S will send a variant of B. S will produce these variants using a generation program. Each variant of A created will in general be different; similarly for B. F will know that 0 and 1 are represented by variants of A and B, respectively, and will use a recognition program to tell which is which. E, also using a recognition program, knows only that the messages are in a binary code but does not know anything about how 0s and 1s are represented.

In this setup, S's objective is to send variants of As and Bs that F will recognize but E will not recognize. The objectives of both F and E are to recognize the letters; for this F has some prior information that E does not have. All the agents will require programs: S for generation and F and E for recognition. The programs will be evolved using evolutionary computation. Each agent will maintain its own population of candidate programs. The overall system will carry out a *coevolution* [1] in which each agent attempts to evolve the best program consistent with its objectives.

Evolution requires a fitness measure, which we need to specifiy for each of the agents. For each bit transmitted by S, F either recognizes it or does not, and E either recognizes it or does not. S's aim is for F to recognize correctly but not E; call this a *success* for S. A simple fitness measure for an S program would be the number of its successes divided by a predetermined number of transmissions, T, assuming that S sends 0s and 1s with equal probability. A success for F as well as for E would be a correct recognition. A simple fitness measure for their programs would be the number of correct recognitions, again divided by T transmissions.

S's population would consist of individuals each of which consists of a generation program. To send a bit, S picks an individual, randomly[2] decides whether to send a 0 or a 1, then as noted above, generates a variant of A for 0, or of B for 1, the variant differing each time the program is called.

[2]For our purposes, the bits need not encode natural language.

The system determines whether the transmission was a success (for S). After a total of T transmissions using a given S individual, its fitness is updated. F and E each have populations of individual recognition programs. Like S, after T recognition attempts using a population individual, its fitness is updated based on its number of successes.

The testing of individuals could be arranged so that for each transmission, individuals from the S, F, and E populations would be selected at random. Or an individual from S could be used for T successive transmissions with F and E individuals still randomly picked on each transmission. Various testing schemes are possible. Selection, reproduction, and genetic operations would occur in a population at intervals long enough so that the average individual gets adequately evaluated.

Will the coevolution work? It seems there should be pressure for improvement in each of the populations. Some initial programs in S should be better than others; similarly for F and E. The three participants should improve, but the extent is unknown. It could be that all three success rates end up not much above 50%. The best result would be 100% for S and F and 0% for E. But that is unlikely since some degree of success by E would be necessary to push S and F toward higher performance.

4 Some Implementation Suggestions

Having described a communications game in which patterns are generated and recognized, and a scheme for coevolving the corresponding programs, it remains to suggest the form of these programs. For concreteness we consider generation and recognition of two-dimensional, gray-scale visual patterns and take the transformational viewpoint of Sec.2.

The programs would be compounds of operators that take an input image and transform it into an output image. The input of one of S's generating programs would be an image of an archetypical A or B and its output would be, via transforms, a variant of the input. A recognition program would take such a variant as input and, via transforms, output a further variant. F would match its program's output against the same archetypes of A and B, picking the better match, and deciding 0 or 1 accordingly. E would simply compute the average gray level of its program's output image and compare that to a threshold to decide between 0 and 1.

For a typical transformation we imagine in effect a function that takes an image—an array of real numbers—as input and produces an image as output. The value at a point x, y of the output may depend on the value at a point (not necessarily the same point) of the input, or on the values of a collection of input points. As a simple example, in a translation transformation, the value at each output point would equal the value at an input point that is displaced linearly from the output point. In general, we would like the value at an output point potentially to be a rather complicated function of the points of the input image.

Sims [2], partly with an artistic or visual design purpose, evolved images using fitnesses based on human judgements. In his system, a candidate image was generated by a Lisp-like tree of elementary functions taking as inputs x, y, and outputs of other elementary functions. The elementary functions included standard Lisp functions as well as various image-processing operators such as blurs, convolutions, or gradients that use neighboring pixel values to calculate their outputs. Noise generating functions were also included.

The input to the function tree was simply the coordinates x and y, so that the tree in effect performed a transformation of the "blank" x-y plane to yield the output image. The results of evolving such trees of functions could be surprising and beautiful. Sim's article gives a number of examples of the images, including one (Figure 9 in the article) having the following symbolic expression,

(round (log (+ y (color-grad (round (+ (abs (round
(log (+ y (color-grad (round (+ y (log (invert y) 15.5))
x) 3.1 1.86 #(0.95 0.7 0.59) 1.35)) 0.19) x)) (log (invert
y) 15.5)) x) 3.1 1.9 #(0.95 0.7 0.35) 1.35)) 0.19) x).

Such an image-generating program is a good starting point for us, except for two missing properties. First, the program does not transform an input image; its only inputs are x and y. Second, the program is deterministic: it is not able to produce different outputs for the same image input, a property required in order to produce image variants.

To transform an image, the program needs to take as input not only x and y, but also the input image values. A convenient way to do this appears to be to add the image to the function set. That is, add $Im(x, y)$ to the function set, where Im is a function that maps image points to image values of the *current* input. For example, consider the expression

(* k (Im (- x x_0) (- y y_0)).

The effect is to produce an output that translates the input by x_0 and y_0 in the x and y directions and alters its contrast by the factor k. It seems fairly clear that adding the current input image, as a kind of function, to the function set, is quite general and would permit a great variety of image transformations.

To allow different transformations from the same program is not difficult. One approach is to include a "switch" function, Sw, in the function set. Sw would have two inputs and would pass one or the other of them to its output depending on the setting of a random variable at evaluation time (i.e., set when a new image is to be processed and not reset until the next image). The random variable would be a component of a vector of random variables, one variable for each specific instance of Sw in the program. Then at evaluation time, the random vector would be re-sampled and the resulting component values would define a specific set of paths through the program tree. The number of distinct path sets is 2 raised to the number of instances of Sw, and equals the number of distinct input image variants that the program can create. If that number turns out to be too small, other techniques for creating variation will be required.

The transformation programs just described would be directly usable by S to generate variants of A and B starting with archetypes of each. F and E would also use such programs, but not alone. Recognition, in the present approach, reverses generation: it takes a received image and attempts to transform it back into an archetype. Since it does not know the identity of the received image, how does the recognizer know which transformations to apply?

We suggest that a recognition program be a kind of "Pittsburgh" classifier system [3] in which each classifier has a condition part intended to be matched against the input, and an action part that is a transformation program of the kind used by S (but without Sw). In the simplest case the classifier condition would be an image-like array of reals to be matched against the input image; the best-matching classifier's transformation program would then be applied to the image. The resulting output would then be matched (by F) against archetypes A and B and the better-matching character selected. E, as noted earlier, would compare the average of the output image with a threshold. It might be desirable for recognition to take more than one match-transform step; they could be chained up to a certain number, or until a sufficiently sharp A/B decision (or difference from threshold) occurred.

5 Discussion and Conclusion

A coevolutionary framework has been proposed that, if it works, may create interesting pattern generators and recognizers. We must ask, is it relevant to the kinds of natural patterns noted in Section 2?

Natural patterns are not ones created by generators to communicate with friends without informing enemies[3]. Instead, natural patterns seem to be clusters of variants that become as large as possible without confusing their natural receipients, and no intruder is involved. Perhaps that framework, which also may suggest a coevolution, ought to be explored. But the present framework should give insights, too.

A basic hypothesis here is that recognition is a process of transforming a pattern into a standard or archetypical instance. Success by the present scheme—since it uses transformations—would tend to support that hypothesis. More important, the kinds of operators that are useful will be revealed (though extracting such information from symbolic expressions can be a chore). It would also be interesting to observe what kinds of matching templates evolve in the condition parts of the recognizer classifiers. For instance, are large-area, relatively crude templates relied upon to get a rough idea of which transforms to apply? If so, it would be in contrast to recognition approaches that proceed from bottom up—e.g. finding edges—instead of top down.

Autonomously created processes would seem of great interest to more standard studies of pattern recognition. The reason is that standard studies involve choices of method that are largely arbitrary, and if they work there is still a question of generality. In contrast, information gained from a relatively unconstrained evolutionary approach might, by virtue of its human-independence, have a greater credibility and extensibility.

It is unclear whether the present framework will even work—for instance whether F's excess of *a priori* information over E's will be enough to drive the coevolution. It is also unclear, even if it works, whether the results will have wider relevance. But the proposal is offered in the hope that its difference from traditional approaches will inspire new experiments and thinking about a central problem in computer science.

References

[1] W. Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D*, 42:228–234, 1990.

[2] Karl Sims. Artificial evolution for computer graphics. *Computer Graphics*, 25(4):319–328, 1991.

[3] Stephen F. Smith. *A Learning System Based on Genetic Adaptive Algorithms.* PhD thesis, University of Pittsburgh, 1980.

[3] There may be special cases!