Towards Sequential Structure-Processing Learning: Simulation of Forgetting/Retrieving Algorithm

Ivan Bruha and Frantisek Franek
McMaster University
Hamilton, Ont.
Canada

Abstract

Structure-processing learning systems are one of the useful means for automatic knowledge acquisition, what is one of the fundamental components of expert systems. However, most recently developed learning systems have several limitations, namely all training patterns have to be stored in the memory of the learning system; they are not able to process both structural and numerical data; they lack processing of uncertainty. To remove those limitations a sequential learning system capable of processing uncertainty has to be developed.

The paper discusses one possible combination of the processing of both structural and numerical information in one system, and embedding such reinforcement techniques that will form a genuine *sequential* learning system. The proposed system will be able

- to process structured patterns together with their numerical uncertainties;
- to form a structural description of classes (concepts) accompanied by their statistical attributes;
- to forget as well as retrieve elements of a class (concept) description (model).

The proposed similarity-based learning-from-examples system (whose working name is MILEARN) has two major stages: *batch* stage and *sequential* one. An initial class (concept) description (model) is formed in the batch stage, after that all training patterns are abandoned, and the learning system will subsequently read new training patterns. Only the latest modifications of the class (concept) description and one or a few training patterns will retain in the memory. Thus, the learning system will become genuine *sequential* one.

1. Introduction

Learning systems are one of the useful means for knowledge acquisition. However, most structure-processing learning systems developed so far exhibit several weak attributes (see e.g. [Ben87], [Da87]):

- majority of working learning systems are not *sequential*, i.e. all training patterns have to be stored in the memory of the learning system;
- they are not able to process both structural and numerical data;
- most systems lack processing of *uncertainty*.

Therefore, we have been looking into possible models that could process both structural and numerical information within one model. We are developing a learning system (with momentary name MILEARN) which would be able

- to process structured patterns together with their (numerical) likelihoods (uncertainties), see e.g. [Ve85];
- to form a structural description of classes (concepts) such that the models and their substructures will be accompanied by statistical (numerical) attributes;
- therefore, it will be able to forget already scanned patterns, i.e. it becomes a (genuine) sequential algorithm of learning.

We view learning as heuristic search through a space of possible class (concept) descriptions. We studied some machine learning algorithms and found out that the following models of learning were most promising vehicles for our project:

1. *Michalski et al.: AQ and INDUCE* [Ben87], [Mi80], [Mi83b], [Mi87]
   It is a well-known structure-processing similarity-based learning system, using deduction as the inference strategy. Entire training set has to be retained in the memory. There is no support of numerical (statistical) processing (and it cannot be embedded).
2. *Purswani: COHER & TRANSFORM* [Pur88]
   Besides structured representation of patterns and class (concept) description, there exists a little support for numerical (statistical) processing. However, it does not comprise uncertainty processing. The system uses three inference strategies: deduction, abduction, and association. Again, the entire training set must be stored in the memory of the learning system.
3. *Bratko et al.: ASSISTANT* [Ce87]
   It can process numerical data; acquired knowledge on classes (concepts) is represented as decision trees [Qui86]. Domain-specific knowledge or deduction, however, cannot be applied in principle.

Next three sections of our paper describe concisely both stages (i.e. batch and sequential one) of our learning system. We would like to emphasize here that the entire system has not been implemented and tested in full. The first (batch) stage is working (almost properly) at present, and it already includes representation of all objects involved in such a way as to allow an easy extension to the sequential stage, and allow to process statistical (numerical) data that will be embraced in both pattern and class descriptions. Details can be found in [Br89]. The Section 4 introduces the principal ideas of the sequential stage of our learning system, especially simulation of forgetting/retrieving algorithm. Notice that some ideas discussed in Section 4 display only rough conceptions required for genuine sequentiality of our learning system MILEARN.

2. Two stages of learning

Our learning system MILEARN, which uses the concept of learning-from-examples with a perfect teacher, has two major stages:

(1) *Batch stage* is a starting point using established techniques mentioned above. A relatively small training set of representative patterns is inserted into the internal memory of the learning system, and an initial class (concept) description (or a minor set of class descriptions) is formed. All training patterns have to be retained in the memory.
(2) *Sequential stage.* The learning system will forget the initial training set. New training patterns will be subsequently read and the existing class description(s) will be modified accordingly. The system will retain the latest modification(s) of the class description(s) and one or a few current training patterns only.

The principle ideas and attributes of representation of both stages of the proposed system are as follows:

- *likelihoods* (frequencies) of elements of class description and so-called *bond nets* above these elements,
- reinforcement (refinement) learning algorithm (partly based on the Stochastic approximation theory) for modification of the likelihoods and bonds, and one for sequential modification of existing structural class descriptions,
- the *forgetting and retrieving* algorithm, and two levels of class description bases.

Our learning system processes patterns as structures and class (concept) descriptions are depicted by relational structures, too. Following COHER & TRANSFORM, deduction and association is used for the inference in its batch stage. The entire algorithm is written in Prolog and both pattern and class (concept) descriptions are depicted as Prolog terms. As the consequence, we did not have to develop any special language for pattern and class descriptions. Moreover, the inference engine of our learning system utilizes all powerful Prolog utilities like pattern matching, backtracking, list processing.

More precisely, a *pattern* or *class description* is a *conjunction* of elementary descriptions. An *elementary description* is symbolized by a Prolog structure, i.e. a functor followed by one or more its components, or an (infix, prefix,

or postfix) operation. For the purposes of the learning algorithm, we distinguish two types of components of elementary descriptions:

- *individuals*, i.e. elementary objects involved in descriptions of both patterns and classes, e.g. `window1 , window2 , car1 ;`
- *properties* of functors such as `size, shape, colour .`

In the same way, we distinguish two types of functors:

- *relational functors*, usually with arity 2 or more, that express relationships between (among) their components (which are individuals), or unary functors that express truthvalue statements;
- *attribute functors* that express property values of individuals.

3. Batch stage

The learning system in its batch stage reads in a relatively small set of training patterns. These training examples have to be really representative ones since the learning system will form one or a few initial class (concept) descriptions (models) by generalizing the information involved in the training set. The request of really representative training set for initial stages of learning can be found in many other projects related to learning (see e.g. [Ber88a], [Ber88b]). After the initial class descriptions (models) are formed the entire initial set of training patterns is forgotten.

As for the representation, a training pattern is depicted by a set of elementary descriptions, each written as the following Prolog fact

```
pattern_descr(N,Z,D,F)
```

where `D` is an elementary description of the `N`-th training pattern of the class (concept) `Z` . Optionally, a training pattern description can be accompanied by the *likelihood* (*typicality*) `F` of the given pattern description.

As we have already indicated the batch stage of the learning system finds a few (more than one) descriptions (models) for the given classes (concepts). The `I`-th model of the class (concept) `Z` consists of a set of elementary descriptions represented by Prolog facts

```
class_descr(Z,I,D,F)
```

where `D` is an elementary description of the `I`-th model having the likelihood (typicality) `F` .

The likelihood `F` of an elementary description depicts the relative frequency of its occurrence in the entire initial training set. One possible way of determining likelihoods is to modify them sequentially according to a suitable recurrent formula.[1] If descriptions of training patterns comprise their likelihoods (typicalities), then they will be embedded to the above frequencies of class description elements, as well.

Although the combination of structural descriptions and numerical likelihoods is more powerful than a pure structural approach, the likelihoods, however, do not provide sufficient means for the forgetting/retrieving algorithm of the sequential stage. Therefore, we incorporate another type of numerical data that are involved in the class descriptions: bonds. Two elementary class descriptions e_1 and e_2 have the *bond* $b \in <0; 1>$, if they have been inserted into the class model (description) together in b cases out of 100 . All bonds form a so-called *bond net*. The initial bond net is created during the batch stage by means of suitable statistical recurrent formulas. A fragment of a class description (model) with a bond net is drawn as a relational structure on Fig. 1. Here *i1* to *i4* are individuals, *r1* to *r4* depict relational functors (*r2* is a symmetrical one, the others are unsymmetrical), *a1*, *a2*

[1] Another way is to compute likelihoods a posteriori using the number of training patterns that strictly match the given elementary description, divided by the total number of training patterns for the given class (concept).

are attribute functors, *p1* is a property of *a1* . The numbers in brackets attached to relational and attribute functors are their likelihoods. The dashed arches symbolize a bond net, and the numbers affixed to them are corresponding bonds.

Consequently, the result of the batch stage is a certain number of class models for each class (concept) of the given problem, each model consisting of elementary descriptions `class_descr` , and the bonds between them.

The batch stage is characterized by a production system with the following parts:

1. *Database* (a set of facts) is formed by training set of positive[2] and negative[3] training examples.

2. *Domain-specific knowledge* characterizes the assumptions and constrains related to both pattern and class descriptions. Such specifications cannot be comprised directly in the database since they do not have character of training examples. Nor can they be involved in the knowledge base because they are specific for a given set of problems. It currently involves the following pieces of knowledge:

- *Domain of linear functors*. If an attribute functor has an ordered set of its property values then it is called a linear functor.
- *Hierarchical tree* (or *set of hierarchies*) *of structural functors*. If an attribute functor has a hierarchical set of its property values then it is called a structural functor.
- *Relationship among functors*. We can capture some relations of functors used in pattern and class descriptions.
- *Additional knowledge* related to the given problem comprises any facts and rules that could by used in the inference process. The operator `->` is defined for such domain-knowledge expressions.

3. *Knowledge base* (a set of production rules) represents general knowledge which is used for finding a required class (concept) description. At present, it consists of two groups of production rules [Mi83a], [Pur88]:

Group I: Description-handling rules

- The *Dropping Condition Rule*: a class description can be generalized by removing an elementary class description.
- The *Deduction Rule*: if a left-hand side of a rule of type `A -> B` in the domain-specific knowledge is satisfied then its right-hand side `B` is added to the description.
- The *Association Rule*: if an elementary description involves an individual `X` , e.g. `f(X,p)` , and the domain-specific knowledge comprises a fact say `g(w,X)` involving the individual `X` then the latter fact is added to the description.

Group II: Component-modifying rules

- The *Domain Extension Rule*: if two elementary class descriptions are formed by the same linear functor whose property value is `A` and `B` respectively, then they can be replaced by a single elementary class description whose property value is specified by the interval `A .. B` .
- The *Hierarchy Climbing Rule*: if two elementary class descriptions are formed by the same structural functor whose property value is `A` and `B` respectively, then they can be replaced by a single elementary description whose property value is a more general value of both `A` and `B` .
- The *Turning Constants Into Variables Rule*: if two (or more) elementary class descriptions with the same functor involve various 'types' of individuals then they can be replaced by a single elementary description with a single individual of a 'general' (i.e. not specified) type.

---

[2] belonging to the given trained class

[3] belonging to other classes (concepts)

4. *Inference engine* finds out which production rules are applicable, chooses one among these rules and applies it. It produces a *discriminant* (i.e. consistent and complete) class description. Its top level flow chart is as follows:

1. Read in the first positive training pattern `x1`, and add all applicable elementary descriptions to `x1`.

2. By decomposing the description of the training pattern `x1` create all possible class descriptions each consisting of one elementary description of `x1`, and insert them to the list `Plausible` of plausible class descriptions.

3. Read in the rest of the training set, and add all applicable elementary descriptions to them.

4. Do the following routines *4A* and *4B* simultaneously:

4A. Form a *consistent* class description from the list `Plausible`. If it is not *complete*, *generalize* it by using the component-modifying rules (group II). Thus, the result is a consistent and complete (i.e., discriminant) description of the given class.

4B. Form a *complete* class description from the list `Plausible` in a similar way. If it is not *consistent*, *specialize* it by using description-handling rules (group I). Thus, the result is a consistent and complete (i.e. discriminant) description of the given class (concept).

5. Generate `Ncd` (a given number) consistent and complete (i.e. discriminant) class descriptions.

4. Sequential Stage

At the moment when the learning system completes its batch stage, all initial training patterns are forgotten, and only the discriminant class models retain in its memory. Now, the *sequential* stage begins: a new set of training patterns is subsequently inserted by the teacher, and the learning system reads one pattern at a time, processes it, modifies the current class model (or models), and forgets the pattern currently processed. However, if there arises a larger discrepancy between current model(s) and the current training pattern on its input, then the system retains a few (more than one) additional training examples in its memory for the purpose of renewing the model consistency.

The primary motto of the sequential stage is to refine the existing class models. It can be done in two ways:

- by *structural refinement*, i.e. by modifying the existing structure, or
- by *numerical refinement*, i.e. by refining the likelihoods (frequencies) of elementary descriptions and bonds.

The first type of the class model modification (structural refinement) will be invoked only if there is a larger discrepancy between existing class models and the current training pattern on the input, i.e. if a so-called *match accordance* is below a certain level δ. The match accordance can be defined as a percentage of elementary descriptions that match the given pattern (training example); the formula for the match accordance can comprise the likelihoods (frequencies) of the elementary class descriptions and the likelihood (typicality) of the input training pattern, too. To solve the given discrepancy, the learning system will retain the unfittable training pattern and read in a relatively small set of training patterns in order to modify properly the structure of the existing class model, i.e. to add (or delete) some of elementary class descriptions.

The latter type of modification, the numerical refinement, adjusts numerical data involved in the class models. As training patterns are coming step-by-step onto the input of the learning system, both likelihoods (frequencies) of elementary descriptions and their bonds are modified according to a suitable reinforcement algorithm. This modification takes place within each step, i.e. even if the current input training pattern is covered by an existing class model. In such a case, the elementary descriptions which match the input training pattern will be reinforced, the others will be weaken. The same procedure is applied to the bonds.

If the frequency (likelihood) of an elementary description declines below a certain threshold γ, then we can interpret it as a weakness (scarcity) of the given elementary description. A human (as a learning system) might forget this piece of knowledge. Our learning system will emulate the same: it will *forget* such a description. Before describing the forgetting process we have to introduce a new concept: *description bases*. We consider two class

description bases: the *top* and the *bottom* one. The top description base contains all elementary descriptions of all class models that the learning system remembers. The bottom description base comprises all forgotten elementary descriptions. The forgetting process looks as follows:

- the elementary description that is to be forgotten will be moved from the top description base to the bottom one, but
- its bonds with remaining elementary descriptions of the same model remain in the top base.

A forgotten elementary description will not participate in further processing, thus saving time required for updating class model(s) as well as saving the memory. More specifically, its frequency (likelihood) will not be modified at all when stored in the bottom description base. Nor its bonds to other elementary description will be changed, but as we have already mentioned, they will retain in the top description base. An illustrative example of a top and bottom description bases is on Fig. 2.

If we, humans, forget something, we are - sometimes - able to retrieve it since we have not actually forgotten a given piece of knowledge completely but our mind has pushed it from its top level down to a bottom level. As soon as a specific situation remind us something we are able to retrieve a forgotten piece of knowledge by withdrawing it from the bottom of our mind. This process is emulated by a so-called *retrieving* algorithm in our learning process. Its principle is following:

- The retrieving process will be invoked if there arises a larger discrepancy between the current class models and the given input training pattern, i.e. if the match accordance is below the threshold δ.
- Because of no sufficient support for the current input training example, the system will retrieve some elementary descriptions from the bottom description base to the top one. It will scan bonds between elementary descriptions at the top description base and those at the bottom description base and retrieve *only those* bottom-base elementary descriptions whose *bonds are greater* than a given threshold ε.
- Those retrieved elementary descriptions which do not match the given input training example will, however, be pushed back to the bottom level, for they have not helped to remove the existing discrepancy. *Only those* which *match* the input example will remain in the top description base and their bonds and frequencies (likelihoods) will be reinforced.

As we have stated at the beginning, the sequential stage has not been implemented nor tested yet. The above description just indicates the interesting aspects of a sophisticated approach to genuine sequential learning. The entire system, however, is under an intensive development.

5. Conclusion

The paper is describing the underlying ideas of both batch and sequential stages of our structure-processing learning system MILEARN. The batch stage, which has been implemented completely, includes all additions and representations that are required for the sequential stage. The principle differences between similar structure-processing learning systems and our batch stage are:

1. Our learning algorithm is being implemented in the programming language Prolog (as a matter of fact, an extension of Prolog, called McPOPLOG [Br88], is being used). We have not developed any language for pattern and class description (such as VL for INDUCE). Rather, we describe patterns and classes by means of Prolog terms exclusively.
2. Our learning system is using deduction as well as association as inference strategies. Furthermore, it utilizes not only the usual development of a discriminant concept description from a *consistent* model to a *complete* (and consistent) one through generalization, but also the opposite scheme going from a *complete* model to a *consistent* (and complete) one through specialization.
3. The learning system forms not one but a given number (`Ncd`) of discriminant class descriptions (models).
4. Elementary descriptions for each discriminant description (model) are accompanied by their *likelihoods* (or *frequencies*).
5. An initial *bond net* among elementary class descriptions is formed.

There are many open problems which will have to be solved during the actual implementation. But even after that, several extensions and modifications should be analyzed and developed. Let us just mention one possible extension. As we already remarked, the learning system finds `Ncd` discriminant class models, when all initial (batch-stage) training patterns are scanned and processed. At present, we require that all class models be consistent and complete, but if this requirement could not be satisfied then the system should at least choose the consistent models with the highest *accuracy*, i.e. those which cover initial training patterns with highest likelihoods (that is perhaps the reason why [Ber88a] may use the term *typicality* of training examples instead of our *likelihood*).

Acknowledgement

This research has been supported by NSERC operating research grant A8034 (project leader I. Bruha) and by NSERC operating research grant OGP0025112 (project leader F. Franek).

References

[Ben87] Bentrup, J.A., Mehler, G.J., Riedesel, J.D.: "INDUCE 4: A program for incrementally learning structural descriptions from examples", Techn. Report UIUCDCS-F-87-958, Univ. of Illinois, Feb. 1987

[Ber88a] Bergadano, F. - Matwin, S. - Michalski, R.S. - Zhang, J.: "A general criterion for measuring quality of concept descriptions", Techn. Rept., AI Center, George Mason Univ., 1988

[Ber88b] Bergadano, F. - Matwin, S. - Michalski, R.S. - Zhang, J.: "Representing and acquiring imprecise and context-dependent concepts in knowledge-based systems", In: Z.W. Ras - L. Saitta (eds.): Methodologies for Intelligent Systems 3, Elsevier Science Publ. Co., 1988

[Br88] Bruha, I.: "AI multilanguage system McPOPLOG: the power of communication between its subsystems", The Computer Journal (accepted Sept. 1988)

[Br89] Bruha, I. - Franek, F.: "On a knowledge-based model of structure learning: methodology and implementation", 1989 European Simulation Conf., Rome, June 1989

[Ce87] Cestnik, B., Kononenko, I., Bratko, I.: "ASSISTANT 86: a knowledge elicitation tool for expert systems". In: Bratko, I., Lavrac, N. (eds.): "Progress in Machine Learning", Wilmslow, Sigma Press, 1987

[Da87] Dalkey, N.: "Models vs. inductive inference for dealing with probabilistic knowledge", Techn. Rept. CSD-870050, UCLA, 1987

[Mi80] Michalski, R.S.: "Pattern recognition as rule-guided inductive inference", IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 4, 349-361, July 1980

[Mi83a] Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (eds.): Machine Learning, an Artificial Intelligence Approach. Tioga Publ. Comp., California, 1983

[Mi83b] Michalski, R.S.: "A theory and methodology of inductive learning", in [Mi83a]

[Mi87] Michalski, R.S., Stepp, R.E.: "INDUCE 3: A program for learning structural descriptions from examples", Techn. Report, Dept. Computer Science, Univ. Illinois, Urbana, 1987

[Pur88] Purswani, F.S.: "A probabilistic reasoning-based approach to machine learning", Techn. Report UIUCDCS-R-88-1475, Dept. Computer Science, Univ. of Illinois at Urbana-Champaign, 1988

[Qui86] Quinlan, J.R.: "Induction of decision trees", Machine Learning, Vol. 1, 81-106, 1986

[Ve85] Venkatasubramanian, V.: "Inexact reasoning with expert systems: a stochastic parallel network approach", 2nd Conf. Artificial Intelligence Applications, Miami Beach, Florida, Dec. 1985

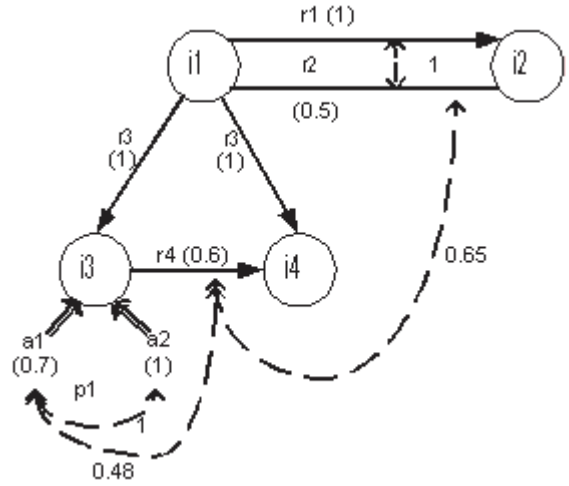

Fig. 1. A fragment of a class description.

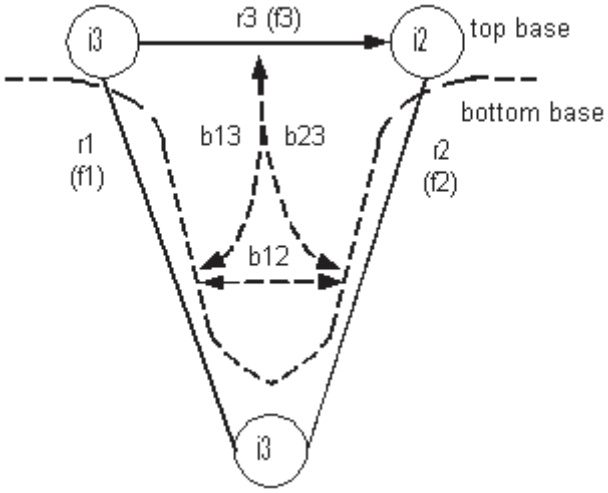

Fig. 2. The top and bottom description bases. Here *i1* to *i3* are individuals, *r1* to *r3* are relational functors, *f1* to *f3* are their likelihoods, *b12* to *b23* are bonds. The dotted line separates the top base and the bottom one.