PERFECT TRAINING SEQUENCES AND THE COSTS OF CORRUPTION - A PROGRESS REPORT ON INDUCTION INFERENCE RESEARCH

R.J. Solomonoff
Oxbridge Research, Box 559, Cambridge, Ma. 02238

Aug, 1982

Abstract: A Training Sequence is an ordered set of problems arranged so that a student or program solving each problem thereby becomes more able to solve subsequent problems in the set. Conceptual Jump Size (CJS) is a measure of the computational cost needed to solve a problem, given that certain information has already been acquired by the student.

A Perfect Training Sequence is one having characteristics that make it particularly easy for certain kinds of machines to solve them. One important characteristic is that the CJS of each problem in the sequence is acceptably small.

A specific kind of perfect training sequence is described along with a program that uses an algorithm by L. Levin to solve the sequence.

We then discuss the effects of corruptions on the training sequence - departures from "Perfectness". Such corruptions can take the form of too few (or actual omission of) examples of critical concepts and/or the misordering of various problems in the sequence. Each of these corruptions lengthens the time needed by the student or program to solve the sequence. We try to devise problem solving strategies that minimize this cost.

More general kinds of problem solving algorithms are then discussed, followed by some remarks about the motivation for this work - its importance as an approach to A.I. - in particular, its bearing upon the problem of representing knowledge.

PERFECT TRAINING SEQUENCES AND THE COSTS OF CORRUPTION

Suppose we give a student a very difficult problem, α_{100}. He tries to solve it for a long time but cannot, because he has not yet acquired the concepts needed for a quick solution.

On the other hand, we can give this student an easy problem, α_0, for which he has already acquired almost all of the needed concepts. In solving α_0, the student acquires one or more new concepts which make it easy to solve α_1, the next problem in the sequence. After he has solved α_1, we present him with $\alpha_2, \alpha_3 \ldots$, each problem exercising old concepts and/or introducing simple new ones. Eventually we give him α_{100}, which he now finds easy to solve, because he has almost all of the needed concepts.

We will call a sequence of problems like this, a "Training Sequence". The present paper will consider the design of such sequences so that the student can learn to work them with minimum effort. Though most of the results discussed apply to both machine and human students, we will consider only machines in detail. In machines, by "minimal effort", we will mean minimal computation cost.

To begin with, problems will consist of question-answer pairs, (Q_i, A_i). Here Q_i and A_i are binary strings, but more generally the symbols may be from any finite alphabet. The student is to find a short program relating Q_i to A_i. More exactly, if M is some reference machine (usually a Universal machine), we want a program, P, such that $M(P, Q_i) = A_i$. Furthermore, we want P to have as few bits as possible. If P is a short program, then it is more likely that the algorithm $M(P, \cdot)$ will extrapolate well - so that if a different Q_j is used, the correct A_j will be produced as output.

Starting with a Tabula Rasa machine, M, having minimal a priori information, L. Levin (1) has devised a simple algorithm for finding short codes, P_0 such that $M(P_0, Q_0) = A_0$.

If P_0 is a solution to the problem, then given M, Q_0 and A_0, Levin's algorithm will find P_0 with a computation cost less than about $2^{|P_0|}W(P_0)$. Here $|P_0|$ is the number of bits in P_0, and is a measure of its information content. $W(P_0)$ is the computation cost of the calculation, $M(P_0, Q_0)$.

We will define $2^{|P_0|}W(P_0)$ to be the Conceptual Jump Size (CJS) from the Tabula Rasa state of M to the solution P_0 of the problem (Q_0, A_0).

After P_0 has been found to solve (Q_0, A_0) we give the system the next problem, (Q_1, A_1). It tries P_0 on this problem. If $M(P_0, Q_1) = A_1$ then it stops. If it does not work, the system tries to find a minimal modification of P_0 that <u>will</u> work.

Let $M'(P_0, X_1) = P_1$. Here M' is a machine that produces modifications in P_0. P_1, the output, is the modification of P_0 that is described by X_1. For this particular kind of machine, $M'(P_0, \Lambda) = P_0$ i.e. the null modification leaves P_0 invariant.

We use short strings, X_1, to produce simple modifications of P_0 as trial values for P_1. Then $M(P_1, Q_1)$ is calculated and compared with A_1. We are looking for a short string, X_1 such that $M(M'(P_0, X_1), Q_1) = A_1$ and $M(M'(P_0, X_1, Q_0) = A_0$ are both true.

As before, we use Levin's search algorithm to find such an X_1. As before, the CJS between the solution P_0, able to work only (Q_0, A_0) and the solution P_1 able to work both (Q_0, A_0) and (Q_1, A_1) is

$$2^{|X_1|}\, W(P_1, Q_0, Q_1)$$

Here $W(P_1, Q_0, Q_1)$ is the cost of computing P_1 from X_1 and trying it out on both Q_0 and Q_1.

As before, the CJS is an upper bound on the computational cost of finding P_1, given P_0.

The system then continues as before, and works problems (Q_2, A_2), (Q_3, A_3) . . . ., etc., to the end of the training sequence.

Training sequences for machines are written much as lessons are prepared for a human student by a very careful teacher. A sequence of problems is devised so that each problem needs few additional concepts beyond the previous problem. Using Levin's formula we can find the CJS between the solutions of successive problems. If in all cases, the CJS is within the practical capacity of our computer, the sequence is satisfactory. If it is not, then we have to break the excessively difficult concepts into several simpler ones by inserting new problems between pairs of problems where the CJS was too large. For any set of solutions to a training sequence it's possible to use Levin's formula to obtain an upper bound to the cost of finding all the solutions.

While the method outlined for writing solvable training sequences is close to workable, the solution algorithm needs to be modified. This is because the first solution that Levin's algorithm finds may not be short enough. One approach to this difficulty is to choose a CJS limit, A, and then, for all problems, search through all possible solutions with CJS $< A$. Of these solutions to a particular problem, we retain the solutions of minimal length.

If we find a problem that has no solution for CJS $< A$, we double A and start over at the beginning, using this larger A value for all of the problems. This doubling is repeated again and again, if necessary, until the whole set of problems is solved. We end up with a value of A that's adequate for all problems but not much larger than is necessary.

There is another difficulty in the proposed system. Occasionally a problem will have an ad-hock solution that is very short, that works for one particular problem, but does not deal with the concept that the problem is designed to illustrate. One way to deal with this is to have many examples of each concept in the training sequence.

To solve training sequences of this sort, we proceed as before, but whenever we come to a problem that demands a new P_i (i.e. the problem has new concepts in it), we first obtain a P_i that will solve the new problem (as well as all of the previous problems), then we see how many subsequent problems it can solve without modification. Let this number be $N_{i,1}$. For each solution, $P_{i,j}$ that we obtain for the i^{th} new concept, we obtain a corresponding $N_{i,j}$. When we have completed the search for the i^{th} concept, we choose the $P_{i,j}$'s for which $N_{i,j}$ is maximum.

If we are very careful in constructing our training sequence it will be solvable by the foregoing technique. We will call a sequence that is so solvable a "Perfect Training Sequence"

Ordinarily, Perfect Training Sequences are difficult to write, and they do not often occur in nature. The commonest ways in which a training sequence deviates from perfection, is that it may leave out important concepts or have too few examples of them. Another common deviation occurs when some of the concepts occur in the wrong order. Each of these corruptions of the sequence makes it more difficult to solve and makes it necessary to modify the original technique that searches for solutions.

The search algorithm that has been considered can be regarded as a "Blind" search. It is an exhaustive search in a region of description space about the position of the previously adequate solution. It is "Blind" in the sense that the choice of new trials is not in any way conditioned by information about the new problem that the old solution failed to solve. In this sense the technique is very much like organic evolution. It is most like the mutation part of the evolutionary mechanism, in which trials of new genotypes are clustered in description space about previously successful genotypes.

In the present search techniques, there has been no effort to model the recombination mechanism of organic evolution, in which partial descriptions of somewhat successful trials are recombined to make new trials. Techniques of this sort could probably increase search efficiency as well as give us a good programmable model of part of organic evolution.

A direction of search that seems even more promising is to consider more goal directed searches. These certainly seem more efficient than pure "blind search" and we have investigated some algorithms of this kind.

So far the kinds of problems considered for training have been the learning of algebraic notation from examples. e.g. To illustrate the operation " + " in RPN we have:

Q: 3.5719, 1.2216, +

A: 4.7935

A typical training sequence would first introduce single unary operators, then nested unary operators, then binary operators, then mixtures of unary and binary operators.

Later we expect to have simple equations to solve, then more complex equations in one unknown, then simple equations in two unknowns, etc.

We have not tried to work difficult problems, since in the present context, they are of no particular interest. A difficult problem differs from an easy one only in that it has a larger CJS, so its solution is expected to take more computation time than an easy problem. The same search algorithm is used in both cases, however, and this is our focus of interest.

What is the value of this study of Perfect Training Sequences? We have noted that they do not occur naturally in the real world and they are difficult to write. Nonetheless they can give us a good understanding of how to write training sequences for human students that facilitate most rapid learning.

Another motivation is that these sequences can be useful models of part of the process of organic evolution. In particular, the CJS concept is closely related to mutation probability.

Another possible application: If we are able to devise usable techniques for dealing with Training Sequences with a fair amount of misordering of problems that are otherwise "Perfect", then sequences of this sort are not so difficult to write and may very well occur in natural environments. $\Rightarrow$ See <u>comment sheet</u>

Perhaps of most importance, the Perfect Training Sequence gives us a formalism by which we can integrate any describable problem solving techniques into a unified system - so we can compare various techniques and optimize the system as a whole. Suppose β is a certain problem solving technique. Then we can devise a training sequence such that for problems at the end of that sequence, Levin's search procedure is essentially equivalent to the technique, β. Furthermore, Levin's algorithm gives us one usable solution to the problem of how to represent knowledge. This representation is in the form of short codes, and these short codes express all of the regularities, all of the useful ideas that we have been able to find in the data.

Footnote 1: L. Levin, "Universal Sequential Search Problems" Problemy Paredachi Informatsii, Vol. 9, No. 3, pp. 115-116, July-Sept., 1973, translation by Plenum Publishing Corp., N. Y. 1975.
While theorem two of his paper is based on Levin's search techniques, he does not explicitly describe the technique itself or his simple proof of its effectiveness. These have been given at various times in lectures at M.I.T.