Preprint from 1957
IRE convention Record,
Section on Information
Theory

AN INDUCTIVE INFERENCE MACHINE

R. J. Solomonoff
Technical Research Group, New York 3, N. Y.

SUMMARY

The machine described is designed to operate as human beings seem to. Inductive inferences are made by classifying events and the outcome of these events within suitable categories. The inductive inference in the individual case is made on the basis of the average behavior of events within the category used.

Accuracy of inference is largely dependent upon how good the categories are. Most of science can be viewed as attempts to find useful ways to categorize phenomena.

The inductive inference machine takes categories that have been useful in the past and, by means of a small set of transformations, derives new categories that have reasonable likelihood of being useful in the future. These are then tested empirically for usefulness in prediction and these new useful ones are combined with old useful categories to create newer ones. These in turn are tested and the process is repeated again and again.

A simplified machine was devised to illustrate the operation of such devices. Since it utilizes only part of a more complete set of transformations it performs only relatively simple learning tasks. Its behavior in learning to perform some arithmetic operations on the basis of a set of correctly worked examples has been analyzed. Operation was described with almost sufficient detail for programming on a digital computer.

At even an elementary level of complexity, recognition of structural similarities and performance of substitutions become natural developments of the heuristic devices that are used. At a slightly more complex level, relations, sets, and hierarchies of sets develop.

The simplified machine that is described here operates on problems in which there is one and only one correct answer. By making the machine operate on statistical training sequences and asking for probability distributions as answers to problems, sensitivity to errors in input data is decreased. It is also possible to programme such a machine to work on the problem of improving itself.

Using the more complete set of transformations, it is expected that these machines will ultimately be able to prove theorems, play good chess and answer questions in English. A preliminary analysis of the relationship of these devices to the work of Chomsky on English grammar indicates that these machines would probably be able to recognize the difference between a "grammatically correct" and a "grammatically incorrect" sentence in Chomsky's best approximation to English, providing the machine was given a training sequence of grammatically correct sentences.

1. Introduction

The following will be a description of a machine designed to perform inductive inferences. Although its methods of operation were suggested by considerations of human thought and problem solving, the machine is not meant to be a model of human thought processes.

While we speak of "a machine", we must realize that no such machine has been constructed. Instead, a set of instructions have been prepared by which certain problems in inductive inference can be solved. In reference 1, Section 3.2.1.6, these instructions are given with almost sufficient detail for programming on a general purpose digital computer. The simplified program given is able to work problems that are solvable using only the heuristic devices described in Sections 2.1 and 2.2 of the present report.

We will describe informally the operation of the machine, using the devices described in the simplified program of reference 1, as well as more powerful devices, which are expected to enable the machine to solve problems of far greater complexity.

2. Problems presented to the machine and methods of solution

2.1 Use of N-grams and Prediction n-grams

Information is presented to the machine in the form of correctly worked examples in arithmetic. At first, examples illustrating the meaning of "equality" are given:

```
= 1 0 1 1 0 ,   = 0 0 1 1 1 0 ,   = 1 0 1,  = 0 1 10
  1 0 1 1 0       0 0 1 1 1 0       1 0 1     0 1 10
```

Here, we have the equality sign, followed by a row of zeros and ones. The row of zeros and ones is repeated in the row below, indicating "equality".

After several examples of this type, we present the machine with problems of the form:

```
= 1 0 0 1 ,   = 0 1 1 1 0 ,   = □ 1
  1 □ 0 1       0 1 1 □ 0       0 □
```

Here, we want the machine to decide upon the most probable digit to fit into the empty squares.

For these problems, a very elementary prediction scheme can be used. Shannon[3] has shown how we may predict words or letters in English by the use of "n grams". An "n gram" is a sequence of n letters. If we want to predict the next letter in the phrase, "Today we ar", we can study the relative frequencies of the "digrams" ra, rb, rc, rd, re, etc. If the digram "re" is 5 times as frequent as the digram "ra", then we may conclude that the final letter in the phrase is 5 times as likely to be "e" as "a".

We may obtain more accurate predictions by using larger n grams. If we use "12-grams" of the type "Today we ar□", the limitations on the terminal letter are more severe than when we used digrams. Note that we consider spaces as contributing to the 12 characters of the "12-gram".

Returning to the arithmetic problems, we may extend the "n-gram" concept to 2 dimensions. For the problem

= 1 0 0 1

1 □ 0 1

the machine may use several possible 2-grams. Some of them are

0 0

0 , 1 1 , 1 , 1 0 , 0 0 .

For prediction purposes it uses "Prediction n-grams", in which the character that is predicted is surrounded by a square. Some "Prediction 2-grams" are

0 0

[0] , 1 [1] , [1] , [1] 0 , [0] 0

From our series of examples of "equality", it is found that the prediction 2-gram $\begin{matrix}0\\[0]\end{matrix}$ is consistent with the examples, because whenever the symbol "0" has appeared, the symbol "0" has appeared below it. On the other hand, the Prediction 2-gram, 1 [1] is not consistent, because when the symbol "1" has appeared, the symbol to the right of it has sometimes been "1", sometimes "0". We shall define "Consistency" of a Prediction n-gram to indicate that all of its predictions, as applied to the examples given to the machine, have been correct.

By analysis similar to the above, the machine finds that the Prediction n-grams $\begin{matrix}0\\[1]\end{matrix}$, [1] 0 and [0] 0 , are all Inconsistent. For this reason, it uses the Prediction 2-gram, $\begin{matrix}0\\[0]\end{matrix}$, to give us the prediction "0" for our first example.

For the problem

= 0 1 1 1 0

0 1 1 □ 0

The only Prediction 2-gram that is relevant, that has been Consistent in the past, is the Prediction 2-gram,

1

[1]

so the prediction is "1".

2.2 Use of Ntuples and Structures.

For some more complex problems, it is possible to use n-grams, but not of a compact type. Consider the problem

= 0 1 0 0

□ 1 0 0

If we want the machine to realize that the Prediction 2-gram is intimately related to the " = " sign, we would want it to discover the Prediction 3-gram

= 0

[0]

It is natural for the machine to try this Prediction 3-gram, since both "0" and " = " are physically close to the character to be predicted. However, if the machine is given the problem

= 1 0 0 1 1

1 0 □ 1 1

the discovery of the Prediction 3-gram

= a a 0

0

isn't very likely, unless special methods are used. (Here the symbol a is used as a spacer and is not a proper part of the Prediction 3-gram).

A very effective special method involves the use of Structures and Ntuples.

An Ntuple, is an ordered set of n objects. In the present context, these objects may be N-grams, Prediction n-grams, or a mixture of both. In Section 2.4 we shall allow these objects to be less restricted in nature. Some examples of Ntuples are:

$$\left(\begin{matrix}[1]\\0\end{matrix}, \begin{matrix}[1]\\0\end{matrix}, =\right); \left(+, a, \begin{matrix}1\,[0]\\1\;\;1\end{matrix}, =\right); \left(\begin{matrix}1\\\;\end{matrix}, \begin{matrix}0\\\;\end{matrix}\right)$$

The first is a 3-tuple; the next a 4-tuple; the last a 2-tuple.

A Structure is a set of instruction for taking the members of an Ntuple, and moving them around with respect to another in a certain way, so as to produce a new N-gram or Prediction N gram, that is a combination of the old ones.

A Structure can be defined by a coordinate grid, with integers in various squares of the grid.

An example of a Structure is:

2
1

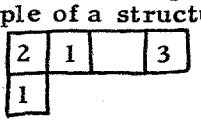
To operate on the 2-tuple, (1, 0) with this structure, we place the first member of the 2-tuple over the 1, and the second member over the 2, to obtain the N-gram

$$\begin{matrix}0\\1\end{matrix}$$

If we operate on the 2-tuple (0 1, =) with this structure, there is some ambiguity. We can place either the "0" or the "1" over the "1" of the structure. Thus we obtain either

$$\begin{matrix}=\\0\ 1\end{matrix} \quad \text{or} \quad \begin{matrix}=\\0\ 1\end{matrix}$$

In the case of more complicated Structures and N-tuples, there may be overlap of some of the N-grams or Prediction n-grams. If the characters that overlap are identical, there is no problem, but if the overlapping characters differ, the resultant configuration is discarded. Another example of a structure is



We may use it to operate on the 3-tuple

$$(1,\ \overset{\alpha}{1}\ ,\ \alpha\ \beta\ )$$

to obtain

$$\begin{matrix}\alpha\\1\ 1\ \alpha\ \beta\\1\end{matrix}\ ,\ \begin{matrix}\\\alpha\ 1\ \alpha\ \beta\\1\end{matrix}\ ,\ \begin{matrix}\alpha\\1\ 1\ 1\ \ \alpha\ \beta\\1\end{matrix}\ ,\ \begin{matrix}\\\alpha\ 1\ \ \alpha\beta)\\1\end{matrix}$$

Another interesting kind of Structure is

It operates on the 3-tuple ($\epsilon\alpha$, β, $\overset{\alpha}{\boxed{\Delta}}$) to obtain the single Prediction N-gram

$$\begin{matrix}\epsilon\ \ \alpha\\\boxed{\Delta}\end{matrix}$$

The β does not appear, since the figure "2" does not appear in the Structure.

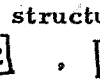
Returning to our problem, the machine finds it very natural to construct the 2-gram

$$(=\ ,\ \begin{matrix}0\\0\end{matrix}\ )$$

This is because the Prediction n-gram $\begin{matrix}0\\0\end{matrix}$ has been useful in prediction in the past, and is likely to continue to be so. The character " = " appears because any individual character that has appeared in examples has fair likelihood of being useful.

Of course the machine constructs many other 2-grams, which are as likely apriori to be useful as is the above, but they are soon discarded, since they do not yield Prediction n grams that turn out to be useful in prediction.

The structures

 ,  and  are all fairly simple, and therefore good structures to try.

If operates on (= , $\begin{matrix}0\\0\end{matrix}$) the Prediction 3-gram = $\begin{matrix}0\\\boxed{0}\end{matrix}$ is obtained, and it turns out to be useful in prediction.

For this reason, the machine remembers that and (= , $\begin{matrix}0\\0\end{matrix}$) have been a useful Structure and 2-gram, respectively, and it tends to use them in the future, in constructing trial Prediction n-grams.

To solve the problem

$$= 1\ 0\ 0\ 1\ 0$$
$$1\ \square\ 0\ 1\ 0$$

The machine will use the structure,  , (which is "close" to the useful structure) to operate on the 2-tuple

$$(=\ ,\ \begin{matrix}0\\0\end{matrix}\ )$$

to obtain the very useful Prediction 3-gram

$$= \begin{matrix}\beta\ 0\\\boxed{0}\end{matrix}$$

2.3 The Use of N-gram Sets, Prediction n-gram sets and N-tuple sets.

Problems solvable through the use of N-grams, Prediction N-grams, Structures and N-tuples, are of very limited complexity. Suppose the machine had been given examples of equality, using only the symbols "0" and "1". We then give the machine the problem:

$$= 2\ 1\ 0\ 1$$
$$\square\ 1\ 0\ 1$$

It is unable to solve this problem, since it cannot yet have learned the Prediction 3-gram,

$$= \begin{matrix}2\\\boxed{2}\end{matrix}$$

A way out of this and many other difficulties is afforded by the use of sets of the devices of the previous sections.

An "N-gram set," is an unordered set of N-grams. An example of such an N-gram set is:

$$\left[\begin{matrix}0 & 1 & 2 & 3 & = & +\\ 0\ , & 1\ , & 2\ , & 3\ , & =\ , & +\end{matrix}\right]$$

A "Prediction N gram set" is an unordered set of Prediction N grams. An example is

$$\left[\begin{matrix}= 0 \\ \boxed{0}\end{matrix}\ ,\ \begin{matrix}= 1 \\ \boxed{1}\end{matrix}\ ,\ \begin{matrix}= \boxed{0} \\ 0\end{matrix}\ ,\ \begin{matrix}= \boxed{1} \\ 1\end{matrix}\right]$$

An "N-tuple set" is an unordered set of N-tuples. An example is

$$\left[\begin{pmatrix}=, & 0\\ & 0\end{pmatrix};\ \begin{pmatrix}=, & 1\\ & 1\end{pmatrix};\ \begin{pmatrix}=, & 0\\ & 0\end{pmatrix};\ \begin{pmatrix}=, & 1\\ & 1\end{pmatrix}\right]$$

A 2-tuple set corresponds to what we would

ordinarily call a "binary relation". For example, the relation "larger than" would correspond to the set of all ordered pairs of objects for which the first is "larger than" the second.

We may operate on N-tuple sets with Structures, to obtain N-gram sets or Prediction n-gram sets. If the structure $\boxed{1}\boxed{2}$ operates on the N-tuple set that was given as an example, the Prediction n-gram set

$$\left[= \begin{matrix} 0 \\ \boxed{0} \end{matrix}, \quad = \overset{0}{\boxed{0}}, \quad = \begin{matrix} 1 \\ \boxed{1} \end{matrix}, \quad = \overset{1}{\boxed{1}}, \quad = \begin{matrix} \boxed{0} \\ 0 \end{matrix}, \quad = \overset{\boxed{0}}{0}, \quad = \begin{matrix} \boxed{1} \\ 1 \end{matrix}, \quad = \overset{\boxed{1}}{1} \right]$$

is obtained.

The Cartesian (or Direct) product of two sets, yields a 2-tuple set. For example, the Cartesian product of the 1-gram set $[0, \ 1, \ 2]$ and the Prediction 2-gram set

$$\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$$

is the 2-tuple set

$$\left[(0, \begin{matrix} 0 \\ 0 \end{matrix}), (0, \begin{matrix} 1 \\ 1 \end{matrix}), (1, \begin{matrix} 0 \\ 0 \end{matrix}), (1, \begin{matrix} 1 \\ 1 \end{matrix}), (2, \begin{matrix} 0 \\ 0 \end{matrix}), (2, \begin{matrix} 1 \\ 1 \end{matrix}) \right]$$

It is also useful to take the Boolian product of various sets. This can be done in at least two very different ways. The first way results in a set, the elements of which are those that are common to the two product sets. This type of product of the 2-gram sets

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \end{bmatrix} \text{ and } \begin{bmatrix} 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 \end{bmatrix}$$

would be the 2-gram set

$$\begin{bmatrix} 2 & 3 \\ 2 & 3 \end{bmatrix}$$

To form the other kind of Boolian product, we first note that to every N gram set, there is a corresponding set of examples to which that N-gram set applies. The correspondence is reciprocal, though there are certain N gram sets that are equivalent to each other. For example, if only the characters 0, 1 and = have ever occurred, then the N gram sets

$$\begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0, & 0 & 1, & 0 & =, & 1 & 0, & 1 & 1, & 1 & = \end{bmatrix}$$

are equivalent.

This second kind of Boolean product of two N gram sets results in an N gram set such that the set of examples to which it corresponds, is the set of examples held in common by the two sets of examples that correspond to the two product N gram sets.

It is important to note that these two kinds of Boolean products are significantly different, and are used to accomplish different effects.

It is possible to show that all of the operations described up to the present time, are fairly limited in the kinds of Prediction n gram sets that they can produce. It might be thought that the Boolean sum operation would help in this respect. While it does, indeed, make possible the formation of any Prediction n gram set that is conceivable, this operation produces too many Prediction n gram sets that are of no value in prediction, and hence the Boolean sum operation is of little value.

An operation that does enable the machine to transcend the limits of its previous operations, is the "Occurrence operation". If G is an N-gram set, then the notation $\nexists G$, denotes an N gram set, the members of which are those N grams of G, which have each had at least one example occur to which they applied.

The number of N-grams in $\nexists G$, is a non-decreasing function of time.

If G consists of the N-gram set

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 0, & 1, & 2, & 3 \end{bmatrix}$$

and the only examples that the machine has been given are

$$\begin{matrix} = 0\,1\,0\,0 \\ 0\,0\,0\,0, \end{matrix} \quad \begin{matrix} = 0 \\ 0, \end{matrix} \quad \begin{matrix} = 1\,1\,0 \\ 1\,1\,0 \end{matrix} \text{ and } \begin{matrix} = 0\,1\,0 \\ 0\,1\,0 \end{matrix}$$

then the N-grams that form $\nexists G$, are

$$\begin{matrix} 0 \\ 0 \end{matrix} \text{ and } \begin{matrix} 1 \\ 1 \end{matrix} .$$

As soon as the example

$$\begin{matrix} = 0\,3\,1\,0 \\ 0\,3\,1\,0 \end{matrix}$$

is given to the machine, $\nexists G$ contains the N-grams

$$\begin{matrix} 0 \\ 0, \end{matrix} \quad \begin{matrix} 1 \\ 1 \end{matrix} \text{ and } \begin{matrix} 3 \\ 3 \end{matrix} .$$

Let us return to the original problem, and see just how the above devices help solve it. The machine is required to formulate the Prediction n-gram $\left(= \begin{matrix} 2 \\ \boxed{2} \end{matrix} \right)$ when it has never been given an example with the N-gram $\left(= \begin{matrix} 2 \\ 2 \end{matrix} \right)$ in it.

At the outset, the machine starts with certain built-in N-gram sets, from which it constructs all of the others. Such a built-in N-gram set, is V, which contains all of the characters that the machine will be ever shown. In the present case,

$$V \equiv [+, =, \square, 0, 1, 2, 3 \ \ldots]$$

The machine forms the 3-tuple set

$$\square \overset{C}{\underset{X}{}} = \overset{C}{\underset{X}{}} V$$

Here the symbol $\overset{C}{\underset{X}{}}$ indicates the Cartesian product. The 3 tuple set formed, will be

$$[(\square, =, +); \ (\square, =, =); (\square, =, \square); \ (\square, =, 0); \ (\square, =, 1); \ (\square, =, 2) \ldots \text{ etc. }]$$

Operating on this 3 tuple set, with the Structure

$$\begin{array}{|c|c|} \hline 2 & 3 \\ \hline \multicolumn{2}{|c|}{1,3} \\ \hline \end{array}$$

the machine obtains the Prediction n-gram set

$$\left[\begin{matrix} =+ \\ \boxed{+} \end{matrix}, \begin{matrix} == \\ \boxed{=} \end{matrix}, \begin{matrix} =\square \\ \boxed{\square} \end{matrix}, \begin{matrix} =0 \\ \boxed{0} \end{matrix}, \begin{matrix} =1 \\ \boxed{1} \end{matrix}, \begin{matrix} =2 \\ \boxed{2} \end{matrix} \ \ldots \text{etc.} \right]$$

This Prediction n-gram set contains the Prediction n-gram $\begin{matrix} =2 \\ \boxed{2} \end{matrix}$, which is what is required.

Note that even before the problem

$$\begin{matrix} = & 2 & 1 & 0 & 1 \\ \square & 1 & 0 & 1 \end{matrix}$$

had appeared, the above Prediction n-gram set had been quite useful in prediction since it contained the two Prediction n-grams

$$\begin{matrix} =0 \\ \boxed{0} \end{matrix} \quad \text{and} \quad \begin{matrix} =1 \\ \boxed{1} \end{matrix}$$

Another limitation of the devices of the previous section is afforded by the problem of normal arithmetic addition. If we had presented addition examples and problems to the machine in the following form, it would have been able to learn to work the problems with these limited devices:

$$\begin{matrix} + & 1 & 1 & 0 & 1 & 1 \\ & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{matrix} \qquad \begin{matrix} + & 0 & 1 & 1 & 0 \\ & 1 & 0 & 1 & 0 \\ \square & \square & \square & \square \\ \square & \square & \square & \square & \square \end{matrix}$$

In the example, the first two rows of digits are the numbers to be added (in binary notation). The fourth row is their sum, and the third row contains the "carries".

This machine would have found Prediction n-grams like

$$\begin{matrix} 1 \\ 1 \\ 1 \\ \boxed{1} \end{matrix} \quad , \quad \begin{matrix} 1 \\ 1 \\ \boxed{1}0 \end{matrix} \quad , \quad \begin{matrix} 1 \\ 0 \\ 1 \\ \boxed{0} \end{matrix} \quad , \quad \begin{matrix} 0 \\ 1 \\ \boxed{1}1 \end{matrix} \quad \text{and} \quad \begin{matrix} 0 \\ 0 \\ \boxed{0}1 \end{matrix}$$

were sufficient to work addition problems.

Suppose this machine had learned to work addition problems in this way, and we then gave it some examples and problems like

$$\begin{matrix} +0\,1\,0 \\ 1\,1\,1, \\ 1\,0\,0\,1 \end{matrix} \quad \begin{matrix} +1\,1\,0\,1 \\ 1\,0\,1\,1, \\ 1\,1\,0\,0\,0 \end{matrix} \quad \begin{matrix} +0\,0\,1 \\ 0\,1\,1, \\ 1\,0\,0 \end{matrix} \quad \begin{matrix} +1\,1\,0\,0\,1 \\ 0\,1\,1\,1\,0, \\ \square\square\square\square\square \end{matrix}$$

The machine would not be able to learn to work these problems unless it had had an inordinately large number of examples. This is because any digit in the sum may depend upon many digits in the two numbers to be added that are quite distant from it.

The method by which N gram sets, Prediction n gram sets and N tuple sets may be used to learn to solve these more difficult addition problems, has been worked out to some extent, but will not appear in the present paper.

2.4 Higher order sets

One of the limitations of the transformation and combination methods discussed thus far, is their inability to deal with higher order sets -- sets whose members are themselves sets. Such devices are of much importance in mathematics, in logic, and in science. There are some logical difficulties that may arise if sets are allowed to be members of themselves. If such logical difficulties result in a set's being of less value in prediction, the machine will tend not to use that set. If the logical difficulties do not influence its value in prediction, the machine will act as though the logical difficulties did not exist.

The method by which the machine is made able to deal with higher order sets, is to redefine "N tuple" so that any member of an N tuple may be a Prediction N gram, Prediction N gram set, N gram, N gram set, N tuple or N tuple set. This, unfortunately, defines an N tuple in terms of itself, a practice which is occasionally legitimate, but is not so in the present case.

This difficulty is resolved by defining an N tuple recursively, i.e. any member of an N tuple may be a Prediction N gram, Prediction N gram set, N gram, N gram set, or anything that has been previously shown to be an N tuple or an N tuple set.

An N tuple set is, as before, an unordered set of N tuples.

2.5 Non-deterministic problems

Up to this point, the machine described can only work on problems in which there is one and only one correct answer. A more common type of problem, is one in which there are several possible answers, each having its own probability of being correct, with more than one correct answer being sometimes possible.

Examples of such problems are language translation, predictions of economic behavior, weather prediction and information retrieval.

One immediate advantage of such machines is that small errors in input data do not disturb machine operation. This is in marked contrast to the previous type of machine, in which a single small error in input data could completely destroy the machine's ability to work large classes of problems.

A preliminary analysis has been made of this more complex type of machine. It is very likely that such a machine would be able to recognize sentences conforming with Chomsky's[5] "phrase structure" grammar, after having been given a large set of sentences written in this grammar. The abstractions it would form, would very probably be those that Chomsky used. In the case of the more complex, "transformational" grammar, only a little analysis of machine behavior has been done. However, it appears likely that the machine could learn to recognize sentences in this grammar. Very probably the machine would invent abstractions differing from the ones Chomsky uses, their exact nature depending upon the order in which examples were given to the machine. The

machine would sometimes categorize as "highly improbable" sentences which Chomsky would regard as grammatically correct, but false or meaningless.

2.6 The concept of Utility

In the operation of a deterministic inductive inference machine, "Utility" values are assigned to each abstraction used. For example, Prediction n grams or Prediction n gram sets that are "Consistent" in their predictions (i.e. there are no counter examples to their predictions) are assigned a high Utility. If they are not Consistent, they are given Utilities of 0. Abstractions like Structures or N tuples or N gram sets, etc., are given Utility values proportional to the frequency with which they are successful in being used to create Consistent Prediction n grams or Prediction n gram sets.

The Utilities of abstractions are then used to get the apriori probability of Consistency of a newly created Prediction n-gram or Prediction n-gram set. Combinations of abstractions which are unlikely to lead to Consistent Prediction n grams or Prediction n gram sets, are not made.

2.7 A General description of the mode of operation of the machine

The machine starts out with a small set of N grams, Prediction N-grams, Structures, N tuples, N gram sets, Prediction N-gram sets and N tuple sets, along with apriori Utility values assigned to all of them.

Using the set of combination and transformation rules that have been partially described above, the machine creates a new set of abstractions from the old set, selecting out only those combinations and transformations that have a sufficiently high apriori Utilities - as given by the Utility transformation rules. The new abstractions are then evaluated empirically, in view of their relation to the examples and problems that have occurred up to that time. In this way a more reliable empirical, aposteriori, Utility is assigned to the abstractions.

All of the abstractions available, are then recombined as before, and the machine retains the new abstractions of sufficient apriori Utility. The process then continues as before, and is repeated again and again. Each time, new abstractions are created and their Utilities are evaluated in terms of the examples and problems.

The machine obtains a constantly increasing number of abstractions, all of which are of fairly high Utility with respect to the examples and problems given.

To solve a problem, the machine searches through its Consistent Prediction N grams and Prediction N gram sets, in an attempt to find one that fits the problem. If more than one fits, and give different answers, the answer associated with the abstraction of highest Utility is taken. If the conflicting Utilities are fairly close, then the answer given will not be very reliable, but this will not often occur if a reasonable number of examples are given to the machine.

3. Present state of development of the machine

A program has been written for the operation of a machine that has N grams, Prediction n grams, Structures and N tuples, but not sets of these abstractions. This program is described in reference 1, Section 3.2.1.6. A detailed description of the operation of the machine in dealing with a set of specific arithmetic examples and problems is given in reference 1, Section 4, as well as in reference 2. This program is almost sufficiently detailed for simulation on a digital computer. Some work must be done on Utility evaluation, however, before this is possible. There is some discussion in reference 1, of methods of Utility evaluation -- particularly in Appendix II.

Some detailed analysis has been made of the response of a more complex machine to problems involving arithmetic addition without explicity written "carries". Such a machine uses abstractions of the type discussed in Section 2.3 of the present paper.

A less detailed analysis has been made of a machine for use with stochastic problems, and its ability to recognize "grammatically correct" sentences in the sense of Chomsky's grammars bas been studied. For more detail see Section 2.5 of the present paper.

4. Important problems that must yet be solved

4.1 The problem of searching for Consistent Prediction n-grams and Prediction n gram sets that fit a particular question is probably the most difficult one. It is, however, a "well defined intellectual problem"[4] in the sense that there is a definite criterion for deciding whether a proposed solution is right or wrong. The original inductive inference problem is not "well defined" in this sense. This search problem is formally quite similar to the problem of proving a mathematical theorem.

Methods by which such "well defined" problems may be solved, are discussed by McCarthy[4], Minsky[6], and Newell and Simon[7]. The construction of the trial "characters" and "methods"[6] appears to be the main difficulty. It is believed that use of abstraction generation methods, such as those of the present paper, will solve this problem.

4.2 Although a set of rules for the manipulation of the sets of Section 2.3 has been drawn up, they have not yet been sufficiently investigated.

4.3 The problem of assigning Utility must be worked out in greater detail.

4.4 The operating program of the stochastic machine of Section 2.5 has been investigated at

great length, but no specific design has yet been decided upon.

4.5 Although the methods that have been investigated for generating new abstractions from useful old ones may be adequate, this adequacy must be tested further.

4.6 The problem of realizing physically an inductive inference machine, has been investigated only a little. Very probably there will be no difficulty in storing all input data for relatively rapid access. A human being, receiving information at the high rate of 10 bits per second, 8 hours a day, for 1 year, receives only about 10^8 bits. It is likely that a computer can operate fairly intelligently with less than this amount of input data. A photoscopic storage device with a capacity of 3×10^7 bits and a mean access time of 25 milliseconds is now under development by the International Telemeter Corp. Photographic storage can be used, since erasing is unnecessary. It is also desirable to have most parts of the machine operate simultaneously, so as to save time, as the human brain seems to. Little work has been done on the design of this type of large scale parallel digital computer.

5. Conclusions

A program has been written for a computer to learn to work simple arithmetic problems after being shown a set of correctly worked examples. Less complete work has been done on programs for more complex problems.

6. Acknowledgement

The author sincerely acknowledges the importance of many enlightening discussions with M. Minsky and R. Silver. Much recent work on the subject of the present paper was supported by the Rockefeller Foundation, at the Artificial Intelligence Project at Dartmouth in 1956.

References

1. Solomonoff, R.J. "An Inductive Inference Machine" August 14, 1956. A privately circulated report.

2. Solomonoff, R.J. "An Inductive Inference Machine" to be published in Information and Control, a new international scientific journal.

3. Shannon, C.E. "Prediction and Entropy of Printed English" Bell System Technical Journal, 1951, 30, pp 50-64.

4. McCarthy, J. Automata Studies (Shannon, McCarthy, editors) Princeton U. Press, 1956, p 177.

5. Chomsky, A.N. "Three Models for the Description of Language" IRE Transactions on Information Theory Vol. IT-2, No. 3, Sept. 1956, pp 113 to 124.

6. Minsky, M.L. "Heuristic Aspects of the Artificial Intelligence Problem" 17 December 1956 Group Report 34-55, M.I.T., Lincoln Laboratory, pp I-1 to I-24.

7. Newell, A and Simon, H.A. "The Logic Theory Machine" IRE Transactions on Information Theory Vol. IT-2, No. 3, Sept. 1956 pp 61 to 79.