This was written & passed around during the Dartmouth Summer Workshop in Artificial Intelligence, 1956.

This vsn of MS contains comments & corrections. I guess this was mainly written in 1956 : partly at Dartmouth. It is dated (on page 1) Aug 14, 1956. — rjs

An Inductive Inference Machine

Ray Solomonoff
Technical Research Group
New York City
Mailing Address: c/o Grace Solomonoff, 72 Winter St.,Arlington, MA 02474, USA
prettyvivo@gmail.com, http://raysolomonoff.com

Contents

PREFACE

A machine is described which is designed to operate as human beings seem to. Inductive inferences are made by classifying events and the outcomes 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 the 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 is analyzed. Operation is described with almost sufficient detail for programming on a digital computer.

At even the most 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 program 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.

Sections 2, 3 and 4 describe the simplified machine in some detail, as well as its response to a set of examples and problems. Section 1 is rather general, and applies to more complex machines, as well as the simplified type. Sections 5.1, 5.2 and 5.3 are descriptions of some of the heuristic devices and transformations that are used in the more complex machines.

desirable terminology changes:
Element $\longrightarrow$ example
q-element $\longrightarrow$ problem – rjs

For some *very imp. difficulties* with this model, see NIC 135.30 (also NIC 136 –rjs)

1980; Aug 1: my impression is that these difficulties had to do w. U evaluation, that making ($U = pc|w.\ qu.\ CB\ limit$) (???) would solve most if not all of these problems — rjs

1 Introduction

The following is a description of a machine which is designed to learn to work problems in mathematics after being given a series of correctly worked examples. After the machine has been shown several examples; it is able to evolve a method by which they might have all been solved. When the machine is given new problems, it uses this method to work them.

The operation of the machine is meant to follow what the author believes to be a process whereby much abstract learning takes place in man.

A person solving simple problems will try a series of simple methods. He will tend to remember the methods that have been successful and discard those that have been useless. When he is given a new set of more difficult problems, he will first try the methods of solution that have been successful in the past. If these fail, he will combine these once successful methods to obtain new trial methods. The ones that are most often successful will be used on the next set of problems, The process continues in this manner.

The techniques by which methods of problem solving that have been successful in the past, are combined to form new trial methods is I believe, the critical problem in all problem solving and induction. Several methods have been found to be of great applicability

An important guide to scientific methodology is our spoken and written language. For the purposes of problem solving and prediction, we shall define "word" to designate any set of possible or actual objects, or space–time configurations, in the world. An example of a "word" might be "pencil," which designates the set of all possible pencils. Another "word" in this sense would be "a rainy day in Egypt" — which designates the set of all such possible days. Another would be "35° centigrade," which designates all parts of the world at that temperature.

There are two properties of words we shall examine.

The first property is exemplified by "cloudy weather," "rainy weather," "automobile," "gasoline." The property of these words that is useful is that certain of them have a high space–time correlation with certain others. For example, if we observe "cloudy weather," we expect "rainy weather" in the immediate space–time neighborhood — and vice–versa. If we observe an "automobile," we expect "gasoline" to be found, with high likelihood, in a certain spacial orientation with respect to it — i.e., in its gas tank.

The finding of "words" that have a high and useful correlation with other "words" has been, by far, the most important task of science.[1]

The second kind of "word" consists of two or more "words" of the first kind, in a particular space–time orientation with respect to one another, with a probabilistic implication relation between the various parts. An example would be "automobile" — consisting of an "engine," "gasoline tank,"

[1]We should mean by "correlation" that these words enable us to calculate good, useful probability distributions of other words.

"wheels," etc. If we observe part of the "automobile," we expect to find an "engine" in a certain spacial orientation to the part of the "automobile" we have seen. The "automobile" includes various "words" as parts of it, and the presence of any group of such parts probabilistically implies the others. The implication may be unidirectional, however. For example, the presence of most of an "automobile" may make the presence of an "engine" very probable. However, observation of an "engine" implies the existence of the rest of an "automobile" with somewhat less probability.

Associated with a "word" of the second kind is a set of probabilities — one for each part of the word — that gives the probability of that part occurring if the rest of the "word" is observed.

It will be noted that most words that we use are of both types simultaneously. However, in designating them "types one and two," we have drawn attention to different aspects of their use in induction.

It is clear that words of the second kind can be easily used for prediction. Trial words of the second kind can easily be constructed by spacia11y orienting words of the first kind with respect to each other in various ways. These are then tested for usefulness in prediction. The useful "words" are retained and given symbols, like "automobile." The useless ones are never used and are immediately forgotten. The useful words of the second kind are used as trial words of the first kind. They are combined spacio–temporally with other words and trial words of the first kind, to produce trial words of the second kind. These latter are tested for their effectiveness in prediction. The words of the first kind that are effective in producing useful words of the second kind are retained in the language and given symbols — like "engine." Trial words that have not been useful are always discarded. While many abstract operations can be expressed in terms of "words" of the first and second kinds, they are by no means adequate to cover all interesting phenomena. "Words" have been defined to mean sets of possible or actual objects in the real world. To cover more complex phenomena, we will need sets of words. These will correspond to sets of sets of objects. We will also need higher order sets of sets of sets, etc.

In order to avoid certain paradoxes associated with higher order sets, B. Russell has evolved the "theory of types." It is likely, however, that we will have no need to avoid these paradoxes by specia1 logical rules. We can, in a statistical machine, steer clear of such paradoxes in a very natural way by noting that the operations that form them tend to be of little value in prediction. Even if these operations *usually* do turn out to be useful, we shouldn't be surprised if they are occasionally of little value.

In the description of the operation of the machine, the objects that correspond to words of the first kind are called "n-grams." Objects corresponding to words of the second kind are called "prediction n-grams."

An ordered set of n words of the first kind, that is to be made into a word of the second kind, corresponds to what is called an "n-tuple."

A set of instructions corresponds to what is called a "structure" if it tells how to mutually orient in space–time a set of words of the first kind to form a word of the second kind.

The application of the above–mentioned instructions corresponds to the operation of a "structure" upon an "n-tuple" to form a "prediction n-gram."

It should be stated at the present time that the machine described here is intended to be only a preliminary approach to the problem, designed to clarify the general method of operation. The present machine can only learn the most elementary arithmetic operations. Significant operational modification must be made before it can learn anything interesting, though the direction of development seems quite clear at the present time.

It should also be noted that this is not a "progress report" in the usual sense, but merely serves to indicate in what direction work is proceeding and the general approach used. It is not meant to indicate the state of development of the work.

2 General description

2.1 The problem

The machine is at first presented with many examples of correctly worked arithmetic problems. An example consists of a 10 x 7 rectangular array of digits. Each digit may be 0, 1, =, +, s (space) or any of many other mathematical symbols.

The machine is then presented with an example in which one or more of the digits have been omitted. It is the problem of the machine to compute what these digits should be.

Some examples of correctly worked problems presented to the machine are

$$\begin{array}{cccc} = 1\ 0\ 0\ 1 & = 0\ 1\ 1\ 1 & = 1\ 0\ 0\ 0 & = 0\ 0\ 1\ 1 \\ \ \ 1\ 0\ 0\ 1 & \ \ 0\ 1\ 1\ 1 & \ \ 1\ 0\ 0\ 0 & \ \ 0\ 0\ 1\ 1 \\ \sim 0\ 1\ 0\ 1 & \sim 1\ 0\ 0\ 1 & \sim 0\ 0\ 1\ 1 & \sim 0\ 0\ 0\ 1 \\ \ \ 1\ 0\ 1\ 0 & \ \ 0\ 1\ 1\ 0 & \ \ 1\ 1\ 0\ 0 & \ \ 1\ 1\ 1\ 0 \end{array}$$

Each of these examples is presented somewhere in its 7 x 10 array.

A problem might take the form $\begin{array}{cccc} = 0 & 0 & 1 & 0 \\ \square & \square & \square & 0 \end{array}$ or $\begin{array}{cccc} = 0 & 1 & 0 & 0 \\ \square & 0 & 1 & \square \end{array}$.

The machine must then give the digits that are appropriate to each of the blank squares.

2.2 How the machine solves the problem

The most elementary method used utilizes what are called n-grams. An n-gram is a spacial configuration of n digits. Some examples are

$$\begin{array}{c} 1 \\ 0 \end{array}\ , \quad \begin{array}{c} = 1 \\ \ \ 1 \end{array}\ , \quad \sim\ 1\ 1\ , \quad \begin{array}{c} 1\ 0 \\ 1\ 0 \end{array}\ .$$

These n-grams are, respectively, a digram, a trigram, a pentagram, and a tetragram. By designating one of the symbols of an n-gram as the "predicted digit," we can use n-grams for prediction. For example, the n-gram $\begin{array}{c} = 1 \\ \boxed{1} \end{array}$

would be useful in prediction if every time the pair of digits = 1 appeared in this relative position, another 1 appeared below the 1.

For the set of examples shown above, the n-grams

$$\begin{array}{c} = 1 \\ \boxed{1} \end{array}\ , \quad \begin{array}{c} = \boxed{0} \\ \ \ 0 \end{array}\ , \quad \begin{array}{c} \sim 1 \\ \ \ \boxed{0} \end{array}\ , \quad \begin{array}{c} 1\ 0 \\ 0\ \boxed{1} \end{array}\ , \quad \text{and} \quad \begin{array}{c} 1\ \boxed{0} \\ 1\ 0 \end{array}$$

are all useful, since the □ enclosed digit is always implied whenever the rest of the digits of its n-gram appear in an example.

The positive or negative verification of the usefulness of an n-gram in prediction is a relatively routine task, as is the problem of prediction itself, once a set of useful n-grams has been found.

An important, difficult problem of the machine is to find n-grams that are likely to be useful in prediction. One method is to take n-grams that have been useful in the past and place them in various relative spacial orientations. The resultant configurations of digits are n-grams that are of reasonable probability of usefulness.

N-grams may have either or both of two types of usefulness. First, they may be directly useful in prediction of digits. Secondly, they may be useful as components, to be combined with other n-grams, to produce n-grams of the first type. (or n-grams of the 2nd type — rjs)

The methods of spacially orienting various n-grams of the second type involve a special construct called a "structure." Various methods will be devised for obtaining useful "structures."

It is possible to form more complex entities than n-grams, which are also useful in prediction. An important one, is a set of n-grams.

All n-grams correspond to certain subsets of the set of all possible configurations of digits on the 7×10 array. The use of sets of sets, and sets of sets of sets, etc., becomes extremely important in many problems.

3 Detailed Description of Machine Behavior and Operation

3.1 Definitions of terms:

3.1.1 Digit

A *digit* is any one of the mathematical symbols that may occupy a position in a 7 x 10 array. Some digits that will be used are 0, 1, =, $\sim$, +, $\times$, s (denoting space) ,$\bigoplus$, and $\bigotimes$ (denoting Boolean sum and product).

3.1.2 Element

An *element* is a 7×10 rectangular array of 70 *digits*. If there are 9 different possible digits, then there are 9^{70} different possible elements.

3.1.3 Q-Element

An element that has one or more of its digits missing. The completion of these *q elements* is the problem that the machine must solve. An example of a q element is: $\begin{matrix} = & 1 & 0 & 0 & \square \\ & 1 & \square & 0 & 1 \end{matrix}$.

The configuration shown is placed somewhere in a 7 x 10 array.

3.1.4 Interrogation Square

A position of a missing digit in a q element. The *interrogation squares* in the above example are marked by the symbol $\square$.

3.1.5 N-Gram (abbreviation: ngm)

A spacial configuration of N digits. These digits must have coordinates that are integers only. Some *n-grams* are $\begin{matrix}=\\1\\1\end{matrix}$, $\begin{matrix}1&0\\1&0\end{matrix}$, $\begin{matrix}s\\1&0\\0\end{matrix}$. $\begin{matrix}=\\1\\1\end{matrix}$ is a *trigram*; $\begin{matrix}1&0\\1&0\end{matrix}$ and $\begin{matrix}s\\1&0\\0\end{matrix}$ are *tetragrams*. There is a one-to-many mapping from the set of all n-grams to the set of all elements. Each n-gram maps to all elements that contain it. Associated with each n-gram–element pair is a number, which tells how many times the n-gram is contained in the element. For example, let us take the element:

s	s	s	s	s	s	s	s	s	s
s	s	s	s	s	s	s	s	s	s
s	s	=	1	0	0	0	s	s	s
s	s	s	1	0	0	0	s	s	s
s	s	s	s	s	s	s	s	s	s
s	s	s	s	s	s	s	s	s	s
s	s	s	s	s	s	s	s	s	s

(The symbol "s" denotes a blank space.)

The digram $\begin{matrix}0\\0\end{matrix}$ is contained in the element three times; the digram $\begin{matrix}1\\1\end{matrix}$ is contained once; the trigram $\begin{matrix}0&0\\&0\end{matrix}$ twice; the digram $\begin{matrix}0\\s\end{matrix}$ four times, etc.

3.1.6 P n-gram (prediction n-gram; abbreviation: pngm) :

An n-gram that is to be used in prediction. Such an n-gram will have the digit that is predicted enclosed in a square. Some examples of p n-grams are :

$$\begin{matrix}1\\\boxed{1}\end{matrix}, \quad \begin{matrix}=\\\boxed{1}\\1\end{matrix}, \quad \begin{matrix}=&0\\&\boxed{1}\end{matrix}, \quad \begin{matrix}\boxed{0}&0\\0\end{matrix}, \quad etc.$$

The p digram $\begin{matrix}1\\\boxed{1}\end{matrix}$ corresponds to the statement that every time the digit 1 appears, the digit 1 should appear just below it. Similarly the p trigram $\begin{matrix}=&0\\&\boxed{1}\end{matrix}$ states that whenever the pair of digits = 0, appear, then a 1 should appear just below the zero.

In general, most p n-grams are not accurate in their predictions. The p n-grams $\begin{matrix}1\\\boxed{1}\end{matrix}$ and $\begin{matrix}=\\\boxed{1}\\1\end{matrix}$ will, at the beginning, be fairly useful in prediction. The p trigram $\begin{matrix}=&0\\&\boxed{1}\end{matrix}$ will never be useful in prediction.

If the machine were given the q element

s	s	s	s	s	s	s	s	s	s
s	s	s	s	s	s	s	s	s	s
s	s	=	□	0	0	0	s	s	s
s	s	s	1	0	0	0	s	s	s
s	s	s	s	s	s	s	s	s	s
s	s	s	s	s	s	s	s	s	s
s	s	s	s	s	s	s	s	s	s

then the p trigram $= \boxed{1}$ would correctly predict that the occupant of the interrogation square should be 1. The p digram $\boxed{0}$ would incorrectly predict that the occupant of the interrogation square should be 0.

3.1.7 N-gram Count

As time goes on, various elements are presented to the machine as correctly worked examples of arithmetic problems. These include correctly completed q elements that had been presented to the machine as questions to be answered. The total number of times that a certain n-gram has appeared in all elements presented to the machine up to time t, will be called the *n-gram count* of that n-gram at time t. Often an n-gram will appear several times in an element. If m elements have been presented to the machine up to time t, then the n-gram count of certain simple n-grams [e.g. 0 1] may be many times as large as m.

3.1.8 P N-gram count

If one removes the square surrounding the predicted digit of a p n-gram, then an n-gram results. The n-gram count of this n-gram, at time t will be called the *p n-gram count* of that p n-gram at time t.

3.1.9 Cases

The number of *cases* of a p n-gram, at time t, is the number of times that this p n-gram has been applicable to predictions of the contents of interrogation squares. A p n-gram may have more than one case in a q element if that q element contains more than one interrogation square.[2]

3.1.10 Consistency

From a p n-gram, remove the prediction digit and the square surrounding it. Substitute for the prediction digit each of the possible digits, in turn, other than the prediction digit of that p n-gram. If the n-gram counts of each of the resultant n-grams, up to time t, are zero, then we will say that the p n-gram of interest is consistent at time t. *Consistency* of a p n-gram means that no counter-examples to its predictions have been observed.

[2]There is much disc'n of the rel. imp. of cases and count in the U of p ngms in Dartmouth Notebook

3.1.11 N-tuple (abbreviation: ntp)

An ordered set of n-grams and p n-grams. The total number of n-grams and p n-grams is N.

Some examples are $\begin{pmatrix} 1 & 0\ 1 & = 1 \\ 1\ , & 1\ , & 1 \end{pmatrix}$ which is called a "triple," $\begin{pmatrix} = , & 1 & 0 & 1 \\ & 1\ , & 0\ , & 1\ , \end{pmatrix} 0$ which is called a "pentuple," $\begin{pmatrix} \sim , & 0\ 0 \\ & 0\ 0 \end{pmatrix}$ which is called a "pair" or "double," and $\begin{pmatrix} 1\ 1 \\ 1\ 1 \end{pmatrix}$ which is called a "singlet" and consists of a single n-gram or p n-gram.

3.1.12 Structure (abbreviation: str):

A set of positive integers placed at various integral coordinate positions on a two-dimensional Cartesian grid. Some examples of structures are:

4	1		2
2			
	2		

, | 1 | 2 |, | 1 | 1 |
|---|---|
| 2 | 2 |

, | | 3 | | 7 |
|---|---|---|---|
| 1 | | 5 | |

,	3	2	2	2

.

3.1.13 Multiplication:

An n-tuple *multiplied* by a structure yields a set of n-grams or p n-grams, The set may contain one or more members, or no members at all. To perform the multiplication, select one digit from each n-gram or p n-gram of the n-tuple. Place the chosen digit of the i-th component of the n-tuple in all of the coordinate squares of the structure that are occupied by the digit i. Each such digit should be surrounded by the properly oriented digits of the rest of its n-gram or p n-gram.

If the resultant configuration does not cause any of the digits to overlap,[3] we will have an n-gram or p n-gram. If they overlap, we reject the result.

Choosing digits from the components of the n-tuple in various ways will result in different n-grams or p n-grams after multiplication by the structure. The set of all such resulting n-grans or p n-grams constitutes the result of multiplying the n-tuple by the structure. For examples, see appendix I.

If | 1 | 2 | is a possible str (structure), the rules for finding "close by" ones may have to be modified from those rules in 3.2.1.3.1

3.1.14 Utility (abbreviation: U).

After the machine has operated for some time, it will be found that certain p n-grams will have been very useful in prediction, and others have been less so. It is possible, in various ways, to assign a value to the effectiveness of a particular n-gram in prediction. As a simple approximation, we may use the "average number of cases per unit time," or its log, as such a measure. We may, in addition, postulate that p n-grams that are not consistent are to have $U = 0$, since they will not be used in prediction in the machine that is being described at present.

[3] It would be well to allow them to overlap if the overlapping digits are identical. If they are not identical, we discard the result. The str | 1,2 | | 2,5 | | 1 | becomes a permissible type of str., e.g. | 1,2 | $\times \begin{pmatrix} 1\ 0\ , & 1 \\ & 1 \end{pmatrix} = \begin{pmatrix} 1\ 0 \\ 1 \end{pmatrix}$ or $\begin{pmatrix} 1 \\ 1\ 0 \end{pmatrix}$

(Modify Appendix 1 if this is used — also give more examples)

In the operation of the machine, p n-grams will continually be created by combining strs, ntps and ngms in various ways.[4] These methods of combination will be described later in some detail. We would like to assign Us to strs, ntps and ngms, so that, when they are combined to form pngms, the ultimate U of this pngm will be well approximated by a kind of average of the Us of the strs, ntps, and ngms from which it was constructed.

If a pngm has just been created and has had no cases yet, its U will be determined by the Us of the components from which it was constructed. As the machine is given more q elements and more cases of this p n-gram arise, the value of U will drift from its initial "a priori" value and be controlled to a greater extent by empirical information on its utility in prediction.

The Us will be assigned to ngms, strs and ntps so as to optimize the "a priori" approximations of the ultimate Us of the pngms that they are combined to form. An example will be given of a simplified situation in which Us are assigned to structures and n-tuples.

Let S_1 be a set of structures and let U_{S_i} be their as yet undetermined utilities. $(i = 1, 2, \ldots, m)$.

Let N_j be a set of n-tuples and U_{N_j} be their utilities, also unknown as yet. $(j = 1, 2, \ldots, r)$.

Let U_{ij} be the utility that was observed for the p n-gram that is formed through multiplication of N_j by S_i. This U_{ij} should be obtained after the machine has been in operation for a long time, so that U_{ij} is close to the ultimate U.

We want to choose our U_{S_i}s and U_{N_j}s so that U_{ij} is approximated by $U_{S_i} + U_{N_j}$ as well as possible. We have $m+r$ quantities to adjust and mr quantities to optimize. This is a problem which is solvable by known means. Since m and r may be large, it may be necessary to use some ingenuity in arriving at a reasonable solution.[5] It is, however, quite adequate for the values of the Us to be very approximate and first approximations to the Us are also available, so that computation may not be out of the question.

A simple approximation that seems to work is :

$$U_{S_i} = \frac{1}{r}\sum_{j=1}^{r} U_{ij} - \frac{1}{2mr}\sum_{j=1}^{r}\sum_{i=1}^{m} U_{ij} \tag{1}$$

$$U_{N_j} = \frac{1}{m}\sum_{i=1}^{r} U_{ij} - \frac{1}{2mr}\sum_{j=1}^{r}\sum_{i=1}^{m} U_{ij} \tag{2}$$

3.2 Mode of operation

3.2.1 Steady state operation:

3.2.1.1 The machine uses five memory banks

1. Contains pngms. With each pngm is contained its pngm count, the number of cases that it has had, its U, and the a priori U which was obtained from the ngms, strs, ntps or pngms, from which this pngm was constructed. This *memory bank* is called the "pngm memory."

   Are 'counts' ever actually *used* anywhere? — rjs

2. Contains ngms, their ngm counts, and their Us.

3. Contains ntps and their Us. ← [This also *includes* the info of 2.]

[4] This is true in a slightly more advanced machine than the one whose rules are described in 3.2.1.6

[5] Not so ingenious – the solutions shown may indeed be optimum.

4. Contains strs and their Us.

5. Contains all elements and q elements that have ever been presented to the machine. This memory bank is called the "array memory."

3.2.1.2 How the Machine Makes Predictions. To make predictions, the machine examines the interrogation square of the q element, finds a p n-gram that fits the problem, and makes the prediction that corresponds to that p n-gram. Upon very rare occasions, there will be two p n-grams that fit the same interrogation square, and these p n-grams will give different predictions. In such a case, the probability distribution between the two predictions may be made proportional to the ratio of the respective counts of the two p n-grams.

Such cases will be rare because as soon as the correct answer is known, one of the opposing p n-grams will lose its consistency — and that particular pair of p n-grams will never disturb one another again.

For example, take the q element $\begin{matrix} \sim & 1 & 0 & 1 & 1 \\ \square & 1 & 0 & 0 & \end{matrix}$

Suppose we have the p n-grams $\sim \begin{matrix} 1 \\ \boxed{0} \end{matrix}$ and $\begin{matrix} 1 \\ \boxed{1} \end{matrix}$, both being consistent up to the present time. Both give different predictions for the interrogation square.

If the p n-gram count of $\begin{matrix} 1 \\ \boxed{1} \end{matrix}$ is 9 and that of $\sim \begin{matrix} 1 \\ \boxed{0} \end{matrix}$ is 7, then the probability of the prediction 1 may be taken to be 9/16, that of the prediction 0, 7/16. It should be noted that, in certain cases, the predictions of the machine will be poor, because of inadequate history. In such cases, there is nothing that can be done to improve a particular prediction very much. If there are competing p n-grams, so that a probability distribution must be given, rather than a single prediction, we have an example of a case in which the best prediction possible is not a very good one.

3.2.1.3 How the Machine Makes New Abstractions from Old. In addition to making predictions, the machine is at all times combining and transforming n-grams, structures, and n-tuples to produce new n-grams, structures, n-tuples and, ultimately, p n-grams.

To describe these processes, let us first assume that the machine has been operating for some time and has a large supply of useful pngms, ngms, strs and ntps.

Some important methods of combination and transformation are:

Only ones that are used in the examples should be given in detail.

3.2.1.3.1 Old Structures are Transformed into New Structures. Before describing these transformations, mention should be made of what will be called the "distance" between two structures.[6]

The application of this notion of "distance" is that, if there is a certain empirically observed U associated with a certain structure, then the a priori Us associated with structures that are a small "distance" from it are not much smaller than the empirical U of the original structure. The a priori U will, in general, be a decreasing function of the "distance" from the original structure.

3.2.1.3.1.a New structure is "close" to the old one in a simple geometric sense. Some examples are:

[6]Most of the ideas of distance used in 3.2.1.3.1 will have to be modified after a few real problems are examined.

$S_1 \equiv \boxed{1}\boxed{2}\boxed{3}$ is "close" to $S_2 \equiv$ $\begin{array}{|c|}\hline 3 \\ \hline \end{array}$ $\begin{array}{|c|c|}\hline 1 & 2 \\ \hline \end{array}$ and to $S_3 \equiv \boxed{1}\boxed{2}\boxed{\ }\boxed{3}$.

It is easy to devise a measure for such "closeness". A possible measure might be: if two structures are identical, except that an integer i, in the first, is in a different position from the same integer i, in the second, than the "distance" between these two structures is the Euclidean distance between the centers of the squares in which these two integers occur.

Using this definition of "distance," the "distance" between S_1 and S_2 is 1, between S_1 and S_3 is 1, between S_2 and S_3 is $\sqrt{2}$.

3.2.1.3.1.b New structure consists of the old structure with an additional integer added. For example, the old structure might be $S_4 \equiv \boxed{1}\boxed{3}\boxed{2}$, the new $S_5 \equiv$ $\begin{array}{|c|c|c|}\hline 1 & 3 & 2 \\ \hline \end{array}$ $\begin{array}{|c|}\hline 1 \\ \hline \end{array}$

"Distance" between old and new structures would be a function of (1) the Euclidean distance between the new integer and the closest integer of the old structure and (2) the numerical difference between the new integer and the one numerically closest integer of the old structure. The "distance" between S_4 and S_5 would then be (1, 0).

It should be noted that the "distance" defined In 3.2.1.3.1.a, has one component, while that defined in 3.2.l.3.1.b has two. These two kinds of "distance" will never be compared to one another.

For a "distance" that has two components, it may be convenient to use some single combination of the components for a priori U determination, say a weighting coefficient for one of them.

It should be noted that "distance" is being used as the inverse of the intuitive notion of "closeness," and isn't meant to have necessarily any properties that a "metric" has — e.g., these "distances" need not satisfy the triangle inequality.

3.2.1.3.1.c New structures may be created from old by the multiplication of an ordered set of n structures by a single structure in the normal manner of multiplication of an n-tuple by a structure, Some examples are :

$$\boxed{1}\boxed{2} \times \left(\begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline \end{array}, \boxed{2}\boxed{3} \right) = \begin{array}{|c|c|c|}\hline 1 & 2 & 3 \\ \hline 2 \\ \cline{1-1} \end{array} \text{ and } \begin{array}{|c|}\hline 1 \\ \hline \end{array} \begin{array}{|c|c|c|}\hline 2 & 2 & 3 \\ \hline \end{array}$$

$$\boxed{1}\boxed{1} \times \left(\boxed{1}\boxed{1} \right) = \boxed{1}\boxed{1}\boxed{1}\boxed{1}$$

3.2.1.3.1.d Creation of new structures from old by rotations and/or reflections, Some examples:

$$\begin{array}{|c|c|}\hline 1 & 2 \\ \hline 1 \\ \cline{1-1} \end{array} \rightarrow \begin{array}{|c|c|}\hline 2 & 1 \\ \hline & 1 \\ \cline{2-2} \end{array} \quad \text{by reflection.}$$

$$\boxed{1}\boxed{2} \rightarrow \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline \end{array} \rightarrow \boxed{2}\boxed{1} \rightarrow \begin{array}{|c|}\hline 2 \\ \hline 1 \\ \hline \end{array} \quad \text{by rotations.}$$

3.2.1.3.2 Creation of New n-tuples from Old n-tuples and n-grams

This is $\overset{c}{x}$ for ngmsts, pngmsts & ntpsts.

3.2.1.3.2.a Juxtaposition of n-tuples or n-grams or p n-grams with n-tuples, or n-grams or p n-grams, to obtain new n-tuples.

For example : $\begin{matrix}\sim & 1\\ & \boxed{0}\end{matrix}$ and $\begin{matrix}\boxed{0}\\ 1\end{matrix}$ combine to form the n-tuple $\left(\begin{matrix}\sim & 1\\ & \boxed{0}\end{matrix}, \begin{matrix}\boxed{0}\\ 1\end{matrix}\right)$, also

$\sim, \begin{matrix}\boxed{1}\\ 0\end{matrix}, \begin{matrix}1\\ 1\end{matrix}$ combines with $(=, \alpha)$ to form $\left(\sim, \begin{matrix}\boxed{1}\\ 0\end{matrix}, \begin{matrix}1\\ 1\end{matrix}, =, \alpha\right)$.

3.2.1.3.2.b Permutation, omission, repetition,

This can be done by the str. — or perhaps it had better be done here, since in more complex T.M.s, many imp. things are done with ntpsts directly.

Binary permutations are the simplest, and therefore result in the least expected decrement in U.

For example $(\alpha, \beta, \gamma, \delta) \rightarrow (\alpha, \delta, \gamma, \beta)$.

Any more complex permutation is expressible as the result of one or more binary permutations, but the a priori U decreases with each extra binary permutation that is needed.

The simplest omission consists of omitting one component:

e.g. $(\alpha, \beta, \gamma, \delta) \rightarrow (\alpha, \gamma, \delta,)$. Omissions of greater than one component can be made by repeated application of a single component omission operator.

The simplest kind of repetition is the single repetition: e.g. $(\alpha, \beta, \gamma, \delta) \rightarrow (\alpha, \beta, \beta, \gamma, \delta)$.

More complex repetitions may be expressed as the result of several simple repetitions.

E.g. $(\alpha, \beta, \gamma) \rightarrow (\alpha, \beta, \beta, \gamma) \rightarrow (\alpha, \beta, \beta, \beta, \gamma)$
or

$(\alpha, \beta, \gamma) \rightarrow (\alpha, \alpha, \beta, \gamma) \rightarrow (\alpha, \alpha, \beta, \beta, \gamma)$.

3.2.1.3.3 Creation of new n-grams and p n-grams through multiplication of an n-tuple by a structure (See Appendix II for examples)

3.2.1.3.4 Creation of a p n-gram from an n-gram or p n-gram by addition of an interrogation square. Some examples are:

$$\begin{matrix}\sim & 1 & 0 & 1\\ & 0 & 1 & 0\end{matrix} \rightarrow \begin{matrix}\sim & 1 & 0 & 1\\ & 0 & \boxed{1} & 0\end{matrix} \text{ or } = \begin{matrix}\boxed{1} & 1 & 0\\ 1 & 1 & 0\end{matrix} \rightarrow = \begin{matrix}\boxed{1} & 1 & 0\\ 1 & 1 & \boxed{0}\end{matrix}.$$

3.2.1.3.5 Creation of a p n-gram from an n-gram or p n-gram by omission of an interrogation square. This is the inverse process of the previous. Some examples are:

$$\begin{matrix}\sim & 1 & 0 & 1\\ & 0 & \boxed{1} & 0\end{matrix} \rightarrow \begin{matrix}\sim & 1 & 0 & 1\\ & 0 & 1 & 0\end{matrix} \text{ or } = \begin{matrix}\boxed{1} & 1 & 0\\ 1 & 1 & \boxed{0}\end{matrix} \rightarrow = \begin{matrix}\boxed{1} & 1 & 0\\ 1 & 1 & 0\end{matrix}.$$

3.2.1.3.6 Some inversion techniques.

3.2.1.3.6.a P n-grams or n-grams "divided" by structures to yield n-tuples.

If S_0 is a structure, Nt_0 an n-tuple, Ng_0 an n-gram and $Ng_0 = S_0 \times Nt_0$;

Then we may write $Ng_0 \div S_0 \equiv Nt_0$ as a *definition* of $Ng_0 \div S_0$.

For a given S_0 and Ng_0 there will usually *not* exist a value of $Ng_0 \div S_0$. If one value of $Ng_0 \div S_0$ does exist, usually there will be several values of Ng_0 for which $Ng_0 \div S_0$ is the same. For example,

$$\text{If} \quad S_0 = \boxed{1}\boxed{2}, \quad Nt_0 = \begin{pmatrix} \alpha, & 1 \\ & 0 \end{pmatrix}$$

$$\text{Let} \quad Ng_1 = \begin{matrix} 0 \\ \alpha \ 1 \end{matrix}, \quad Ng_2 = \begin{matrix} \alpha \ 0 \\ 1 \end{matrix}$$

$$\text{Then} \quad Ng_1 \div S_0 = \begin{pmatrix} \alpha, & 0 \\ & 1 \end{pmatrix} = Nt_0$$

$$\text{Also} \quad Ng_2 \div S_0 = \begin{pmatrix} \alpha, & 0 \\ & 1 \end{pmatrix} = Nt_0$$

3.2.1.3.6.b P n-grams or n-grams divided by n-tuples to yield structures. As in (a), if $Ng_0 = S_0 \times Nt_0$ then $Ng_0 \div Nt_0 \equiv S_0$, by definition.

Often if $Ng_0 \div Nt_0$ does exist (and it usually will not) then it will have several values, For instance, as in (a) :

$$\begin{matrix} 0 \\ \alpha \ 1 \end{matrix} \div \begin{pmatrix} \alpha, & 0 \\ & 1 \end{pmatrix} = \boxed{1}\boxed{2} \text{ or } \begin{matrix} & \boxed{2} \\ \boxed{1} & \end{matrix}$$

$$\begin{matrix} \alpha \ 0 \\ 1 \end{matrix} \div \begin{pmatrix} \alpha, & 0 \\ & 1 \end{pmatrix} = \boxed{1}\boxed{2} \text{ or } \begin{matrix} \boxed{1} & \\ & \boxed{2} \end{matrix} \quad .$$

3.2.1.3.6.c N-tuple "divided"[7] by an n-tuple to yield a new n-tuple.

If $Nt_0 = (\alpha, \beta)$ is is an n-tuple and (α) and (β) are both n-tuples, then we define

$$Nt_0 \overset{L}{\div} (\alpha) \equiv (\beta) \quad \text{(division on the left)}$$

$$Nt_0 \overset{R}{\div} (\beta) \equiv (\alpha) \quad \text{(division on the right)}$$

Some examples:

$$\text{Let} \quad Nt_0 = \begin{pmatrix} 1 & , & 0 \\ 1 & , & 0 \end{pmatrix}, \quad Nt_1 = \begin{pmatrix} 1 & , & 0 \\ 1 & , & 0 \end{pmatrix}, \quad Nt_2 = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

$$\text{Then} \quad Nt_0 \overset{R}{\div} Nt_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{also} \quad Nt_0 \overset{L}{\div} Nt_1 = (=).$$

In the present case

$$Nt_0 \overset{L}{\div} Nt_2 \equiv \begin{pmatrix} 1 & , & 0 \\ 1 & , & 0 \end{pmatrix} \overset{L}{\div} \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

is meaningless. Usually two n-tuples cannot be divided into one another in this way. When the n-tuple obtained as a result of this division is a "single" or "mono–tuple," either a p n-gram or n-gram may be obtained through this division process.

[7] Elias' review objected to the use of the term "divided" – this will become clear if I use the term cartesian division and define it for manipulations of ntpsts.

3.2.1.4 Assigning Prior and Posterior Utilities to Various Abstractions and Prediction Methods. Another process that is going on at all times[8] is the assignment of Us to all ngms, ntps, strs and pngms. These Us must be continually reevaluated, as new problems and data are given to the machine.

A partial description of this process of U evaluation was given in section 3.1.13. A more detailed description will be given here.

Each method of creating pngms will have associated with it a method of getting the a priori Us associated with these pngms — also a weight to be given to this a priori U, when combining it with empirical data,

For an example of this weight determination, suppose that U_{S_I} is the U of str S_i, U_{N_j} the U of ntp N_j. Then the a priori U of the pngm $S_i \times N_j$ will be $f(U_{S_i},\ U_{N_j})$, where f is a function that will be optimized for the operation of multiplying strs by ntps. At first, we will let $f = AU_{S_i} + BU_{N_j}$ and optimize A and B. As the machine gets more data, it will be possible to use more coefficients — say

$$f = AU_{S_i} + BU_{N_j} + CU_{S_i}^2 + DU_{S_i}U_{N_j} + EU_{N_j}^2 \ldots$$

The process of using infinite coefficients can be optimized

For this discussion we will assume $f = AU_{S_i} + BU_{N_j}$. Let U_{ij} be the empirically observed U of the pngm $S_i \times N_j$.

$$\text{Then} \quad \sigma^2 = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\left(U_{ij} - (AU_{S_i} + BU_{N_j})\right)^2$$

is the mean square error of $AU_{S_i} + BU_{N_j}$ viewed as a prediction of U_{ij}.

The weight to be assigned to $AU_{S_i} + BU_{N_j}$ as an a priori prediction of U_{ij} is then

$$W_{ij} \approx \frac{AU_{S_i} + BU_{N_j}}{\sigma^2}\,.$$

More explicitly, if τ is the total number of interrogation squares that have appeared on all q elements up to the present time, and C_{ij} is the total number of cases of the pngm $S_i \times N_j$, then the value given to U_{ij} is

$$U_{ij} = \frac{C_{ij} + W_{ij}(AU_{S_i} + BU_{N_j})}{\tau + W_{ij}}\,.$$

It will be noted that, as $\tau \to \infty$, $U_{ij} \to \frac{C_{ij}}{\tau}$ as desired. Also, for $\tau = 0$, $U_{ij} = AU_{S_i} + BU_{N_j}$ and thus $AU_{S_i} + BU_{N_j}$ is, indeed, the a priori value of U_{ij}.

The above discussion is not an exact one, and was given to illustrate the use of weights in the application of a priori Us to the computation of true Us. A more accurate method of computation is available. It is discussed in Appendix II, and in the Dartmouth Notebook, Dart 20).30.

Another aspect of U computation is illustrated by the creation of new structures from old by choosing structures that are a short "distance" from the old ones. If U_{S_i} is one of the Us of structure S_i, and S_j is a structure at a distance d_{ij} from S_i, then the corresponding a priori U to be associated with S_j is $k_{ij}U_{S_i}$, where k is a decreasing function of d_{ij}. We may write $k_{ij} = e^{-\alpha d_{ij}}$, then find an α such that this a priori U is as good an approximation as possible to the empirically obtained Us — averaged over all pairs of neighboring structures upon which one has data.

[8] In the simple model, this is *not* parallel to other activities

3.2.1.5 Equations for operation of a Simplified Machine We will now describe the behavior of a simplified machine that has pngms, ngms, strs and ntps, but has only five ways to produce new ones from old. Let these methods be written[9]

1. str, ntp → pngm
2. ngm_1, ngm_2 → ntp
3. str, ngm → ntp
4. str → str
5. ngm → pngm

Let us further suppose that we only have binary ntps and strs; that the product of a str and ntp always exists and is single valued; that the fourth operation is single valued — producing a single str from another single str. See 3.2.1.3.3.

Operation (1) may be normal multiplication of an ntp by a str to obtain a pngm.

Operation (2) may be simple juxtaposition of two ngms to obtain an ntp. See 3.2.1.3.2.a.

Operation (3) may be division of an ngm by a str to obtain an ntp. See 3.2.1.3.6.a.

Operation (4) may be creation of a new str by taking one that is some fixed distance (say one unit) from the old one. See 3.2.1.3.1.a.

Operation (5) may be the creation of a new pngm by adding an interrogation square to an ngm. See 3.2.1.3.4.

For the first operation,

3.2.1.5.1

$$\sigma^2 = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}\frac{\left(\frac{C_{ij}}{\tau} - (A_1 U_{S_i} + B_1 U_{Nt_j})\right)^2}{\left(1 + \frac{C_{ij}}{\tau\sigma^2}\right)^2}$$

U_{S_i} is the U of the $i-$th str and is to be determined.

U_{Nt_j} is the U of the $j-$th ntp and is to be determined.

A_1 **and** B_1 are constant coefficients and are to be determined.

τ is the number of interrogation squares that have occurred up to the present time

S_i is the $i-$th str.

Nt_j is the $j-$th ntp.

U_{ij} is the U of the png that is formed by $S_i \times Nt_j$.

[9] the order of these ops should be changed – see example 2, Section 4.1

C_{ij} is the number of cases of the pngm that is formed by $S_i \times Nt_j$, provided this pngm is consistent. Otherwise $C_{ij} = 0$.

m is the number of strs in the memory.

n is the number of ntps in the memory.

σ^2 is the mean square deviation of U_{ij}. from the a priori U, which is $A_1 U_{S_i} + B_1 U_{Nt_j}$.

$A_1,\ B_1,\ U_{S_i}(i = 1, \ldots, m),\ U_{Nt_j}(j = 1, \ldots, n)$ are all to be adjusted so that σ^2 is minimal.

3.2.1.5.2 The value of U_{ij} is taken to be

$$U_{ij} = \frac{(A_1 U_{S_i} + B_1 U_{Nt_j})\frac{C_{ij}}{\tau\sigma^2} + C_{ij}}{\frac{C_{ij}}{\tau\sigma^2} + \tau}$$

If σ^2 is small, then

$$\frac{C_{ij}}{\tau} = A_1 U_{S_i} + B_1 U_{Nt_j}$$

and we may write[10]

$$W_{ij} \approx \frac{A_1 U_{S_i} + B_1 U_{Nt_j}}{\sigma^2}$$

$$U_{ij} \approx \frac{W_{ij}(A_1 U_{S_i} + B_1 U_{Nt_j}) + C_{ij}}{W_{ij} + \tau}$$

After we have found optimum values of A_1, B_1 and the U_{S_i}s and U_{Nt_j}s, we may proceed to the second and third operations.

3.2.1.5.3

This is utterly silly! The 2 $\sum$s can be minimized *separately.* **NO**, U_{Ng_j} are common to both

$$E \equiv \sum_{i=1}^{r}\sum_{j=1}^{r}\left[(A_2 U_{Ng_i} + B_2 U_{Ng_j}) - U_{Nt_{ij}}\right]^2 + \sum_{i=1}^{m}\sum_{j=1}^{n}\left[(A_3 U_{S_i} + B_3 U_{Ng_j}) - U'_{Nt_{ij}}\right]^2$$

$A_2,\ B_2,\ A_3,\ B_3$ are constants to be determined.

U_{S_i} is the U of S_i and has already been fixed.

Ng_i is the $i-$th ngm

U_{Ng_i} is the U of Ng_i[11]

r is the total number of ngms in the memory.

$U_{Nt_{ij}}$ is the U of the ntp (Ng_i, Ng_j)

$U'_{Nt_{ij}}$ is the U of the ntp formed by "dividing" Ng_j by the str S_i.

E is defined by the equation. It is the total error in the a priori estimates of U.

We are then to determine $A_2,\ B_2,\ A_3,\ B_3$ and all of the U_{Ng_i}s so that E is minimized.

[10] the meaning of W_{ij} is not clear. Also why go thru this trouble? Why not use the prev formula for U_{ij} directly?

[11] the $U_{Nt_{ij}}$ and $U'_{Nt_{ij}}$ are obtained from "operation 1" (and then) equ. 3.2.1.5.1

NOTE the operation of Dart.Thinking Machine is examined & some serious diftys are found in NIC 135 & NIC 136, particularly NIC 136.35

Hey! what about the 5th operation?? This happens to be imp't
Not so imp: Firstly it may not be an imp't operation, 2ndly, we can simply find an optimum K' so that $U_{pngm_i} \approx K' U_{Ng_i}$ (where $pngm_i$ is derived from Ng_i by adding an interrogation square.)
NIC 16.30 suggests a better way: adding another $\sum$ term to 3.2.1.5.3 .

3.2.1.5.4 For the fourth operation,[12]

$$E_1 \equiv \sum_{i=1}^{m} (U_{S_i^o} - K U_{S_i})^2$$

This is *silly!* usually the dist btw s_i^o and s_i = dist. btw. s_i and s_i^o; In such cases K *must* be 1 by symmetry considerations. *NO*, see below.

S_i is the $i-$th str.

S_i^o is the str that results from the application of the fourth operation upon S_i.

$U_{S_i^o}$ is the U of S_i^o.

K is a constant to be adjusted.

E_1 is defined by the equation and is the total error in the a priori estimates of U.

We must adjust K so that E_1 is minimal.

3.2.1.5.5 Value of K A solution is

$$K = \frac{\sum_{i=1}^{m} U_{S_i^o} U_{S_i}}{\sum_{i=1}^{m} U_{S_i}^2} \quad .$$

3.2.1.6 Rules for Operation of a Simplified Machine Let us review the operation of this simplified machine. There are essentially three tasks that the machine must perform. First of all, it must make predictions. In order to do this, it must create new pngms, ngms, strs and ntps from old ones. It must also reevaluate and keep up to date the U values of all old and new pngms, ngms strs and ntps.

Expressed briefly, when the machine is given an element, it places this element in its array memory and waits for further inputs. When it is given a q element, it places the q element in its array memory, then updates the *consistency* of all pngms affected by this q element and recent elements. It also updates Us of all pngms, ngms, strs and ntps affected.

It then scans through its pngm memory to see if it can make the required prediction. If it can, it does so, and stops.[13] If it can't, it makes new pngms that fit the q element, tests them for consistency and updates all Us and consistencies. It repeats this subroutine of forming new pngms that fit, testing them, and updating until it finds a relevant pngm that is consistent — at which time it makes a prediction and stops.

In greater detail, the rules of operation are:[14]

[12] There is much question about just how useful this K is – in partic, what good this error criterion is.
[13] It *shouldn't* stop here – see $\propto$ 149.28 for imp't comments.
[14] A flow diagram of operations should be included.

1. When the machine is given a new element, it stores this element in its "array memory" and waits for the next input.

2. When the machine is given a new q element, it first stores the q element in its array memory, Next, it scans through the array memory, from the present q element, back to, but not including, the previous q element. It notes which pngms have become inconsistent because of the elements and q element scanned.

   It then searches through its pngm memory to see if it has any consistent pngms that apply to the interrogation square of interest. If it has, it makes the appropriate prediction, and recalculates the C_{ij}s that have been affected by the new-found inconsistencies and/or new pngm cases. If there is any ambiguity it first goes to rule 5 to resolve it, then proceeds to rule 3. If there is no ambiguity, it goes to rule 3 directly.

3. (a) If the C_{ij}s have been changed, the Us of pngms, ntps and strs are recomputed in accord with equations 3.2.1.5.1 and 3.2.1.5.2, and the machine goes to rule 3.b.

   (b) The machine recomputes the Us of ngms, in accord with equation 3.2.1.5.3, then goes to rule 3.c.

   (c) K, of equation 3.2.1.5.4, is re–optimized and the machine goes to rule 3.d.

   (d) If a prediction has been made for the present q element, the machine stops and waits for the next input. If not, it goes to rule 4.

4. The machine constructs new pngms, ngms, ntps and strs of high a priori U, that apply to the interrogation square in question, using the methods of section 3.2.1.5. It devises several new appropriate pngms that it does not have in its pngm memory and places them in that memory. It then scans through the entire array memory to see if any of the new pngms are consistent. If any of them are, it makes the appropriate prediction. (In case of ambiguity, it refers to rule 5.) It then recalculates the C_{ij}s that have been affected by the new found inconsistencies and/or new pngm cases and goes to rule 3, independently of whether or not a prediction has been made.

5. If[15] the predictions of two pngms conflict, take the prediction of the one with most cases. If the case numbers are equal, take the one with the greatest count. If the count numbers are equal, no prediction should be made. If the case numbers are equal and greater than zero, the prediction accuracy may be expected to be poor.[16]

3.2.2 Initial Values of Prediction Parameters

In the previous sections we have outlined the "steady state" operation of the machine. The processes described assume that the machine already has a stock of pngms, ngms, ntps and strs, along with

[15] I think that the Us of the 2 conflicting pngms should be compared. Had this been considered in th. Dart. Notebook discussion of this ambiguity?
It would seem **not!**
There is a discn of this pt. from a new pt. of view in $\propto$ 278.11.
Dart.Notebook 94 has mention of the above decision procedure but only a little discussion....
Dart.Notebook 56 has mention of the above decision procedure but no discussion.

[16] At this point mention loop around rule 3 and 4 until problem is solved.

their Us, and any other needed parameters. To start off the machine, we will have to give it some elementary abstractions to build upon. A set that is probably adequate to start with is:

1. Each digit is made an ngm with a priori U of $1/d$, d being the number of different digits. Initial values of W must also be assigned.

2. The str $\boxed{1}$, is assigned an initial U of, say, 0.5.

   We will also have to assign some initial values to the various combination parameters. For instance, the "K"[17] of equation 3.2.1.5.4 as well as A_1, A_2, A_3, B_1, B_2, B_3 that occur in equations 3.2.1.5.2 and 3.2.1.5.3 may be given the value 0.5. In a similar way we may assign almost arbitrary initial values to various parameters that occur, exercising no great degree of care. These initial values become unimportant after the machine has been given several elements and q elements, since the values are soon modified so as to optimize them with respect to this empirical data.

4 Examples of machine operation

Having described in a little detail the operation of the machine, we will now analyze, in a general way, its response to particular training sequence of elements and q elements.

The first part of the training sequence will consist of elements of the general form:

$$\begin{array}{cccccccccccccccccccccc} = & s & 0 & 1 & 1 & = & 1 & 1 & 0 & 0 & 1 & = & 0 & = & s & s & 0 & \square & 0 & 0 & = & 1 & 1 & 0 & 1 & 1 \\ & s & 0 & 1 & 1 & ' & & 1 & 1 & 0 & \square & 1 & ' & & 0 & ' & & s & s & \square & 1 & 0 & \square & ' & & 1 & 1 & 0 & 1 & 1 \end{array} .$$

They will be presented anywhere in the 7×10 array. The digits following the "=" are a random sequence of 0s, 1s and spaces, with the row below having the same sequence of 0s, 1s and spaces, as that following the "=" digit, except for $\square$s.

4.1 Twenty sample problems

1. Suppose that the first example presented to the machine is the element $\begin{array}{cccc} = & 1 & 0 & 1 \\ & 1 & 0 & 1 \end{array}$. Following rule (1) of section 3.2.1.6, it stores this element in its array memory and waits for the next input.

2. Next, let us present $\begin{array}{cc} = & 0 \\ & 0 \end{array}$ $\begin{array}{c} \square \\ 1 \end{array}$. By rule (2) of 3.2.1.6, it again stores this in its array memory.

   Following the rest of rule 2, we note that the machine has, initially, no pngms at all — only the ngms 0, 1, and the str $\boxed{1}$. There are not yet any C_{ij}s to change. The machine goes to 3.a, does nothing then skips to 3.d, does nothing again, and goes to rule 4.

   Rule 4 is most critical, since it is at this point that new pngms are created.

   From the ngrms 0, 1, =, the machine creates the pngms $\boxed{0}$, $\boxed{1}$, and $\boxed{=}$ by operation 5 of 3.2.1.5. For simplicity, we may assign a U attenuation factor of unity[18] to the process of operation 5.

[17] section 4.1.2 says unity for K.

[18] section 3.2.2.2 says .5

To illustrate the operation of the instructions, we will assume that we stop with the set of pngms we have just made, and scan through the array memory to test *consistency*. If $\boxed{0}$, for example, were consistent, this would mean that every square of every array presented to the machine (other than interrogation squares) was occupied by 0. Clear1y neither $\boxed{0}$, $\boxed{1}$, nor $\boxed{=}$ are consistent, so the machine makes no prediction. In any case we go back to step 3. We *do* have some new Cs (case numbers of pngms), since three new pngms have been created. Rules 3.a, b, and c do not, however, require that anything be done, since none of the transformation and combination rules that result in equations 3.2.1.5.2, 3.2.1.5.3, and 3.2.1.5.4 have yet been made use of.

Since no prediction has been made yet, we go from 3.d to step 4 again and devise some new pngms.

We may use the combination rules of 3.2.1.5 in a rather systematic manner to create new pngms, ngms, strs and ntps.

Operation 1 isn't of much help yet. The only ntps we have are (0), (1), (=), ($\boxed{0}$), ($\boxed{1}$), ($\boxed{=}$). The only str we have is 1, which leaves all of the ntps essentially invariant. It creates no new pngms or ngms. (str x ntp.)

Operation 2 creates the set of ntps that is the Cartesian product of the set of all ntps with itself. We obtain $(0,0)$, $(0,1)$, $(=,\boxed{1})$, $(0,\boxed{=})$, etc. — 36 ntps in all.

Operation 3. Again for lack of a useful str, we can do nothing that obtains anything new.

Operation 4 obtains some new, useful strs. From $\boxed{1}$ we obtain $\boxed{1\,|\,1}$, $\begin{array}{|c|}\hline 1\\\hline 1\\\hline\end{array}$, and $\begin{array}{|c|}\hline 1\\\hline\end{array}\!\!\begin{array}{|c|}\hline 1\\\hline\end{array}$, all of which are within $\sqrt{2}$ of $\boxed{1}$. We can go a slightly greater distance and obtain $\boxed{1\,|\,2}$, $\begin{array}{|c|}\hline 2\\\hline\end{array}\!\!\begin{array}{|c|}\hline 1\\\hline\end{array}$, $\begin{array}{|c|}\hline 2\\\hline 1\\\hline\end{array}$, $\begin{array}{|c|}\hline 2\\\hline\end{array}\!\!\begin{array}{|c|}\hline 1\\\hline\end{array}$, $\boxed{2\,|\,1}$, etc.

Operation 5 yields nothing new, since we have no new ngms yet.

With these new strs, we can start through the list of operations again and obtain pngms.

Operation (1) now yields a very large set of pngms and ngms of reasonably large a priori U. Some examples:

$$\boxed{1\,|\,1}\times 1 = \begin{array}{cc} 1 & 1\end{array}\,, \qquad \boxed{1\,|\,2}\times(=,\boxed{1}) = (=\boxed{1})$$

$$\begin{array}{|c|}\hline 1\\\hline 1\\\hline\end{array}\times 1 = \begin{array}{c} 1\\ 1\end{array}\,, \qquad \begin{array}{|c|}\hline 1\\\hline 2\\\hline\end{array}\times(1,\boxed{1}) = \begin{array}{c} 1\\ \boxed{1}\end{array}$$

$$\begin{array}{|c|}\hline 2\\\hline\end{array}\!\!\begin{array}{|c|}\hline 1\\\hline\end{array}\times(\boxed{=},1) = \begin{array}{c} 1\\ \boxed{=}\end{array}\,.$$

Since our q element is $\begin{array}{cc} = & 0 \\ 0 & 1\end{array}\,\square$, only the pngms $0\ \boxed{1}$, $\begin{array}{c} \boxed{=}\\ 0\end{array}$, $\begin{array}{c}\boxed{0}\\ 0\end{array}$, etc., are relevant.

Of the rest of the operations, only (5) yields pngms, but, in trying it out on the ngms just generated, we find that we obtain no *new* pngms.

We have enough pngms to try again for a consistent one to cover the q element of interest.

Following the machine's instructions, we scan through the array memory to find if any of the relevant pngms are consistent. The only consistent ones are $\begin{matrix}\boxed{1}\\1\end{matrix}$, $0\ \boxed{1}$, and $\begin{matrix}&\boxed{1}\\0&\end{matrix}$. We then make the prediction 1 for the q element and go on to step 3 again.

1. For step 3.a we observe that there are now many new C_{ij}s. All of them, however, are zero, except those of and $\begin{matrix}\boxed{1}\\1\end{matrix}$, $0\ \boxed{0}$, and $\begin{matrix}&\boxed{1}\\0&\end{matrix}$. Most of the C_{ij}s are zero because they are inconsistent. A few, like $\begin{matrix}1\\\boxed{1}\end{matrix}$ and $\begin{matrix}\boxed{0}\\0\end{matrix}$, are *consistent*, but they have no *cases* yet.

Let us examine the methods by which our three pngms were created.

confusion btw $\boxed{1}$ in strs, $\boxed{1}$ in predict. ngms

$$\begin{matrix}1\\1\end{matrix} = \begin{matrix}\boxed{1}\\\boxed{2}\end{matrix} \times (\boxed{1}, 1) = \begin{matrix}\boxed{2}\\\boxed{1}\end{matrix} \times (1, \boxed{1})$$

$$\begin{matrix}\boxed{1}\\1\end{matrix} = \begin{matrix}1\\1\end{matrix} \text{ with } \square \text{ added, through operation 5}$$

$$\begin{matrix}1\\1\end{matrix} = \begin{matrix}\boxed{1}\\\boxed{2}\end{matrix} \times (1,1) = \begin{matrix}\boxed{2}\\\boxed{1}\end{matrix} \times (1,1) = \begin{matrix}\boxed{1}\\\boxed{1}\end{matrix} \times (1)\,.$$

$$0\ \boxed{1} = \boxed{1}\boxed{2} \times (0, \boxed{1}) = \boxed{2}\boxed{1} \times (\boxed{1}, 0)$$

$$0\ \boxed{1} = 0\ 1\ , \quad \text{with } \square \text{ added}$$

$$0\ 1 = \boxed{1}\boxed{2} \times (0,1) = \boxed{2}\boxed{1} \times (1,0)\,.$$

$$0\ \begin{matrix}\boxed{1}\\&\boxed{2}\\&\boxed{1}\end{matrix} = \begin{matrix}\boxed{2}\\\boxed{1}\end{matrix} \times (0, \boxed{1}) = \begin{matrix}\boxed{1}\\\boxed{2}\end{matrix} \times (1,0)$$

$$0\ \begin{matrix}\boxed{1}\\&\square\end{matrix} = 0\ \begin{matrix}1\\&\square\end{matrix} \quad \text{with } \square \text{ added}$$

$$0\ \begin{matrix}1\\&\boxed{2}\\&\boxed{1}\end{matrix} = \begin{matrix}\boxed{2}\\\boxed{1}\end{matrix} \times (0,1) = \begin{matrix}\boxed{1}\\\boxed{2}\end{matrix} \times (1,0)\,.$$

Through equation 3.2.1.5.1 the U values of

$$\begin{matrix}\boxed{1}\\\boxed{1}\end{matrix},\ \begin{matrix}\boxed{1}\\\boxed{2}\end{matrix},\ \begin{matrix}\boxed{2}\\\boxed{1}\end{matrix},\quad \boxed{1}\boxed{2},\ \boxed{2}\boxed{1},\quad \begin{matrix}&\boxed{2}\\\boxed{1}&\end{matrix},\ \begin{matrix}&\boxed{1}\\\boxed{2}&\end{matrix},$$

$$\text{and } (1),\ (\boxed{1}, 1),\ (1, \boxed{1}),\ (1,1),\ (0, \boxed{1}),\ (\boxed{1}, 0),\ (0,1),\ (1,0)$$

will increase, while the Us of all other strs and ntps that were tried will decrease. More accurate values of A_1 and B_1 can now be calculated.

Equation 3.2.1.5.2 will now give the new values of U of $\begin{matrix}\boxed{1}\\1\end{matrix}$, $0\ \boxed{1}$, and $\begin{matrix}&\boxed{1}\\0&\end{matrix}$, as well as those of any other pngms that have been tried.

For step 3.b, we compute the Us of the ngms that were used, through equation 3.2.1.5.3. Also, more accurate values of A_2, B_2, A_3, B_3 can now be calculated.

For step 3.c we reoptimize the value of K, in accord with equation 3.2.1.5.4. The machine then stops and waits for the next input.

3. Suppose the next input is $\begin{matrix}=0&0&1&1\\0&\square&1&1\end{matrix}$. The machine goes through the same routine as before.

   The only consistent pdigms that can be devised are $\begin{matrix}0\\\boxed{1}\end{matrix}$ and $\begin{matrix}0\\\boxed{0}\end{matrix}$. Since the predictions given by them conflict, we go to rule 5, which tells us to compare their case numbers. They are both zero. Next we compare their counts. They are both zero, so we make no prediction.

   It should be noted that $0\ \boxed{1}$ and $\begin{matrix}&\boxed{1}\\0&\end{matrix}$, which caused difficulty during the last example, are no longer *consistent*.

4. The next example is $\begin{matrix}=&0&0&1&1&0\\&0&0&\square&1&0\end{matrix}$

   We note that the pngm $\begin{matrix}0\\\boxed{1}\end{matrix}$ which caused difficulty during the previous example, is now inconsistent, The only *consistent* pdigm is $\begin{matrix}1\\\boxed{1}\end{matrix}$ and so our prediction is 1.

5. The next example is $\begin{matrix}=&1&0&0&1\\&\square&0&0&1\end{matrix}$.

   The only consistent pdigm is $\begin{matrix}1\\\boxed{1}\end{matrix}$ and so our prediction is 1.

   The next set of examples will teach a new operation to the machine.

6. $\begin{matrix}\sim&1&0\\&0&1\end{matrix}$

7. $\begin{matrix}\sim&0&0&1\\&1&1&0\end{matrix}$

8. $\begin{matrix}\sim&0&1&0&1\\&1&0&1&0\end{matrix}$

9. $\begin{matrix}\sim&1&1&0\\&0&0&1\end{matrix}$

10. $\begin{matrix}\sim&\square&1&1\\&1&0&0\end{matrix}$

When the machine is presented with example 10, it soon finds that there are no pdigms that are *consistent* for examples 1 through 10. In particular, all of the pdigms that were of high U for examples 1 through 5 are now inconsistent. As a result, all of the Us of ngms, strs and ntps that were a result of these pdigms being useful, are reduced considerably. The machine starts again from scratch. Since all pdigms are inconsistent, it must try ptrigms next.

The only consistent ptrigm that is relevant is $\sim \boxed{0}\ 1$. The resultant prediction is therefore zero.

The ptrigm of interest was derived in the following manner:

$$\sim \begin{matrix}0\\1\end{matrix} = \boxed{1}\boxed{2} \times \left(\sim, \boxed{\begin{matrix}0\\1\end{matrix}}\right) = \boxed{2}\boxed{1} \times \left(\boxed{\begin{matrix}0\\1\end{matrix}}, \sim\right)$$

$$\text{also} \quad \sim \boxed{\begin{matrix}0\\1\end{matrix}} = \sim \begin{matrix}0\\1\end{matrix} \quad \text{with } \square \text{ attached, and}$$

$$\sim \begin{matrix}0\\1\end{matrix} = \boxed{1}\boxed{2} \times \left(\sim, \begin{matrix}0\\1\end{matrix}\right) = \boxed{2}\boxed{1} \times \left(\begin{matrix}0\\1\end{matrix}, \sim\right).$$

By this process we see that

$$\boxed{1}\boxed{2}, \quad \boxed{2}\boxed{1}, \quad \left(\sim, \begin{matrix}0\\1\end{matrix}\right), \quad \left(\begin{matrix}0\\1\end{matrix}, \sim\right), \quad \left(\sim, \boxed{\begin{matrix}0\\1\end{matrix}}\right), \quad \left(\boxed{\begin{matrix}0\\1\end{matrix}}, \sim\right)$$

all increase in U.

Also $\boxed{\begin{matrix}1\\2\end{matrix}}, \boxed{\begin{matrix}2\\1\end{matrix}}$, $(0,1)$, $(1,0)$, which are the strs and ntps by which $\begin{matrix}0\\1\end{matrix}$ was formed, increase in U.

It should be noted that in normal machine operation the order of decreasing a priori U is not necessarily monograms, digrams, trigrams etc. It is only when there are no lower order ngms that have been found to be useful, that this is true. For example, if the digram $\begin{matrix}1\\0\end{matrix}$ and the monogram 1 were of equal U, then multiplying them both by the str $\boxed{1}\boxed{1}$ would yield the ngms $\begin{matrix}1&1\\0&0\end{matrix}$ and $1\ 1$ respectively. These two ngms would have equal a priori U. In the present case, the machine is young and inexperienced and it has no digrams of high U, so all ngm and pngms must be built up rather arduously from the basic digits. It is this condition that explains why the a priori U of an ngm is, at this stage of machine education, a decreasing function of n.

Another important point that should be noticed is the method that has been used to obtain pngms to fit a given q element. The method consists of applying a set of combination and transformation rules to a set of pngms, ngms, ntps and strs to obtain a new set of pngms, roughly in order of decreasing a priori U. Usually a large number of pngms are obtained, and from this large set the machine selects out those that apply to the q element of interest. This procedure is, then, an exhaustive search, and as such, is not particu1ar1y efficient.

The problem of obtaining a suitable pngm is at once a "well defined problem" of the first and second kinds.[19] We have a set of operations. We may take any of this set and put them in any order. We then have a definite criterion to decide whether the result (if any) of these operations is a pngm that fits the q element. The definiteness of this criterion makes it a problem of the first kind. If we have two pngms that fit the q element, we can compare their a priori Us to determine which is better. For this reason we have a problem of the second kind — since we want to find a pngm of high U.

11. For this example, let us go back to the "=" operation and see how the machine behaves with $\begin{matrix} = & 1 & 0 & 1 & 1 \\ & \square & 0 & 1 & 1 \end{matrix}$. As before, there are no consistent pdigms. The only consistent ptrigm is $\begin{matrix} = & 1 \\ & \boxed{1} \end{matrix}$ and so the prediction is 1.

    The strs $\boxed{1}\boxed{2}$ and $\boxed{2}\boxed{1}$ are used again, since

$$\begin{matrix} = & 1 \\ & \boxed{1} \end{matrix} \quad = \quad \boxed{1}\boxed{2} \times \left(=, \begin{matrix} 1 \\ \boxed{1} \end{matrix} \right) \quad = \quad \boxed{2}\boxed{1} \times \left(\begin{matrix} 1 \\ \boxed{1} \end{matrix}, = \right).$$

    Also, the str $\begin{matrix} \boxed{1} \\ \boxed{1} \end{matrix}$ is useful, since $\begin{matrix} 1 \\ 1 \end{matrix} = \begin{matrix} \boxed{1} \\ \boxed{1} \end{matrix} \times (1)$.

12. The next example is $\begin{matrix} \sim & 1 & \square & 0 & 1 \\ & 0 & 1 & 1 & 0 \end{matrix}$.

    There are no *consistent*, compact ptrigms, so the machine tries new strs. The str $\boxed{1}\boxed{2}$ has been useful, so neighboring strs are tried. The str $\boxed{1}\boxed{\ }\boxed{2}$ is close and proves useful.

$$\boxed{1}\boxed{\ }\boxed{2} \times \left(\sim, \begin{matrix} \boxed{0} \\ 1 \end{matrix} \right) = \begin{matrix} \sim & \text{Ƃ} & \boxed{0} \\ & & 1 \end{matrix}.$$

    (Notation note: The symbol "Ƃ" in a pngm indicates that this pngm does not concern itself with the digit that appears in the square occupied by "Ƃ". The symbol is used as a spacing device.)

    Since $(\sim, \begin{matrix} 0 \\ 1 \end{matrix})$ has already proved useful, the pngm that has been constructed is of high a priori U.

    The prediction is 0.

13. The next example is $\begin{matrix} = & 1 & 0 & 0 & 1 \\ & 1 & \square & 0 & 1 \end{matrix}$.

$$\begin{matrix} 0 \\ 0 \end{matrix} = \begin{matrix} \boxed{1} \\ \boxed{1} \end{matrix} \times (0).$$

[19] Define "well-defined problems of 1st and 2nd kinds"

$$\begin{pmatrix} = & \exists & 0 \\ & & \boxed{0} \end{pmatrix} = \boxed{1}\boxed{\ }\boxed{\ }\boxed{2} \times \begin{pmatrix} =, & 0 \\ & \boxed{0} \end{pmatrix}.$$

.

$$\begin{pmatrix} = & \exists & 0 \\ & & \boxed{0} \end{pmatrix} = \begin{pmatrix} = & \exists & 0 \\ & & 0 \end{pmatrix} \quad \text{with } \square \text{ added.}$$

The ntp $\begin{pmatrix} =, & 0 \\ & \boxed{0} \end{pmatrix}$ is found to be of very high U since there are many *cases* of its use.

E.g.: $\begin{pmatrix} = & 0 \\ & 0 \end{pmatrix} = \boxed{1}\boxed{2} \times \begin{pmatrix} =, & 0 \\ & \boxed{0} \end{pmatrix}$ and $\begin{matrix} = & \exists & 0 \\ & & 0 \end{matrix} = \boxed{1}\boxed{\ }\boxed{2} \times \begin{pmatrix} =, & 0 \\ & 0 \end{pmatrix}$.

14. The next example is $\begin{matrix} \sim & \square & 1 & 0 & 0 \\ & 0 & 0 & 1 & 1 \end{matrix}$.

The ptrigm $\sim \begin{matrix} \boxed{1} \\ 0 \end{matrix}$ is used. The ntps $\begin{pmatrix} \sim, & 1 \\ & 0 \end{pmatrix}$ and $\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \sim$ increase in U.

15. The next example is $\begin{matrix} \sim & 0 & 0 & 1 & 1 \\ & 1 & 1 & \square & 0 \end{matrix}$. It is possible for the machine to create the pngm

$$\begin{matrix} \sim & \exists & \exists & 1 \\ & & & \boxed{0} \end{matrix} = \boxed{1}\boxed{\ }\boxed{\ }\boxed{2} \times \begin{pmatrix} \sim, & 1 \\ & \boxed{0} \end{pmatrix}$$

since $\boxed{1}\boxed{\ }\boxed{\ }\boxed{2}$ is close to $\boxed{1}\boxed{\ }\boxed{2}$, but the pngms $\begin{matrix} 0 & 1 \\ 1 & \boxed{0} \end{matrix}$ and $\begin{matrix} 1 & 1 \\ \boxed{0} & 0 \end{matrix}$ have higher a priori U since they are easily constructed from high U digms, and strs

$$\begin{matrix} 0 & 1 \\ 1 & 0 \end{matrix} = \boxed{1}\boxed{2} \times \begin{pmatrix} 0 & 1 \\ 1, & 0 \end{pmatrix} \quad \text{and}$$

$$\begin{matrix} 1 & 1 \\ 0 & 0 \end{matrix} = \boxed{1}\boxed{2} \times \begin{pmatrix} 1 & 1 \\ 0, & 0 \end{pmatrix} = \boxed{1}\boxed{1} \times \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

In any case, the prediction is 0 .

16. The next example is $\begin{matrix} \sim & 0 & 0 & 1 & 1 & 1 \\ & 1 & 1 & 0 & \square & 0 \end{matrix}$.

The pngms $\begin{matrix} 1 & 1 \\ \boxed{0} & 0 \end{matrix}$ or $\begin{matrix} 1 & 1 \\ 0 & \boxed{0} \end{matrix}$ are used, and so the prediction is 0 .

17. The next example introduces the machine to a new operation:

$$\begin{matrix} \oplus & 1 & 1 & 0 & 0 & 1 \\ & 0 & 1 & 1 & 0 & 0 \\ & 1 & 1 & 1 & \square & 1 \end{matrix}.$$

The p tetragrams $\begin{matrix} 1 & 0 \\ 1 & \boxed{0} \end{matrix}$ and $\begin{matrix} 0 & 0 \\ \boxed{1} & 1 \end{matrix}$ both fit the q element, but $\begin{matrix} 1 & 0 \\ 1 & \boxed{0} \end{matrix}$ is inconsistent, so the prediction is 1.

18. The next example is

$$\begin{array}{cccccc} \oplus & 1 & 0 & 0 & 1 & 1 \\ & 0 & 0 & 1 & 1 & 0 \\ & 1 & 0 & 1 & \square & 1 \end{array}.$$

There are no compact p tetragrams that fit.

The p tetragrams of highest a priori U, that fit, are

$$\begin{array}{cccccccccccccccccc} & 1 & & & 1 & & & & 1 & & & & 1 & & & & 1 & \\ & 1 & , & & 1 & , & & & 1 & , & & & 1 & , & & & 1 & . \\ 1 & \boxed{1} & & & \boxed{1} & 1 & & 0 & \text{Ǝ} & \boxed{1} & & 0 & \text{Ǝ} & \boxed{0} & & 1 & \text{Ǝ} & \text{Ǝ} & \boxed{1} & & 1 & \text{Ǝ} & \text{Ǝ} & \boxed{0} \end{array}$$

In case of conflict, we go to rule 5. None of the above p tetragrams have any cases, but the first two each have counts of one, so they determine the prediction, which is 1.

19. This example is

$$\begin{array}{cccccc} \oplus & 1 & 0 & 1 & 1 \\ & 0 & 1 & 0 & 1 \\ & 1 & 1 & \square & 1 \end{array}.$$

The consistent p tetragms of highest count are $\begin{array}{ccc} & 1 & \\ 1 & \boxed{1} & \end{array}$ and $\begin{array}{ccc} & 1 & \\ & \boxed{1} & 1 \end{array}$, so the prediction is 1.

20. The last detailed example is

$$\begin{array}{cccccc} \oplus & 1 & 0 & 0 & 1 & 0 \\ & 1 & 1 & 0 & 0 & 1 \\ & 1 & \square & 0 & 1 & 1 \end{array}.$$

The consistent p tetragrams of highest count are

$\begin{array}{cccc} + & \text{Ǝ} & 0 & \\ & & 1 & \\ & & \boxed{1} & \end{array}$, $\begin{array}{ccc} & 0 & \\ & 1 & \\ 1 & \boxed{1} & \end{array}$, and $\begin{array}{ccc} & 0 & \\ & 1 & \\ \boxed{1} & \text{Ǝ} & 1 \end{array}$. The prediction is 1.

Further Examples

Further examples would involve learning operations of the type

$$\begin{array}{cccccc} \otimes & 1 & 0 & 1 & 0 \\ & 0 & 0 & 1 & 1 \\ & 0 & 0 & 1 & 0 \end{array}.$$

For these examples, p hexagrams of the types $\begin{array}{cc} 0 & 1 \\ 1 & 0 \\ 0 & \boxed{0} \end{array}$ and $\begin{array}{cc} 0 & 1 \\ 1 & 0 \\ \boxed{1} & 1 \end{array}$ would probably be discovered.

If, however, a sufficiently large number of examples of $\oplus$ and $\otimes$ are given, the machine must occasionally use pngms of the type

$$\begin{array}{cccc} \oplus & \text{Ɜ} & \text{Ɜ} & 1 \\ & & & 0 \\ & & & \boxed{1} \end{array}$$

since the p hexagrams would be inapplicable.
A case of such inapplicability is

$$\begin{array}{cccc} \oplus & 0 & 1 & 1 \\ & 1 & 1 & 0 \\ & 1 & 1 & \square \end{array}$$ since the p hexagram $$\begin{array}{cc} 1 & 1 \\ 1 & 0 \\ 1 & \boxed{1} \end{array}$$ is not consistent, nor is $$\begin{array}{cc} 1 & 1 \\ 1 & 0 \\ 1 & \boxed{0} \end{array}$$.

After many examples of $\oplus$ and $\otimes$, we may present the machine with ordinary arithmetic addition. Some examples are

$$\begin{array}{cccccc} + & 1 & 1 & 0 & 1 & 0 \\ & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 1 \end{array}$$ and $$\begin{array}{cccccc} + & 0 & 1 & 1 & 0 & 1 \\ & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \end{array}$$

The first two lines are the numbers to be added. The fourth line is their sum, and the third line contains the "carries."

This particular method of presenting addition leads to pngms of the type $\begin{array}{c} 1 \\ 0 \\ \boxed{0} \\ 0 \\ 1 \end{array}$ and $\begin{array}{c} 1 \\ 0 \\ \boxed{0} \\ 0 \\ 1 \end{array}$ etc.

A very interesting problem for the machine would be to present it with addition problems without "carry" lines. Some examples are : $\begin{array}{cccccc} + & 1 & 1 & 0 & 1 & 0 \\ & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \end{array}$ or $01101 + 11010 = 100111.$

This particular problem has not yet been investigated at any length. It is not certain whether the machine would learn addition in this form, using the present methods of creating new pngms, or whether new methods are needed.

4.2 Discussion of machine response to training sequences.

Let us examine the response of the machine to the examples. In general, the machine always tries to make predictions using the simplest possible compact pngms.

In example 15 we would like the machine to learn the connection between the symbol "$\sim$" and the digm $\begin{array}{c} 1 \\ 0 \end{array}$, but the machine chooses a simpler method of prediction. This method is again used in example 16. In examples 18 and 19, the association between the symbol $\oplus$ and the ngms of interest is not made because there are more compact, consistent pngms available. In example 20, the association between $\oplus$ and the pngm $\begin{array}{c} 0 \\ 1 \\ 1 \end{array}$ is made, but the resultant pngm is only one of several that will serve equally we11.

In general, the machine will not use a particular pngm for prediction unless it is the simplest method available. It has, of course, its own idea as to which method is "simplest". If, eventually, the machine is given problems in which it is *necessary* to recognize the logical connection between, say "$\sim$" and $\begin{matrix}\text{“}0''\\ 1\end{matrix}$, it will do so. But until then it will continue to use "simpler" prediction methods.

5 Program for future work

5.1 Some new definitions & a modification, giving a powerful machine language

First of all, several new definitions will be introduced. They are:

1. Pngm set : an unordered set of pngms. An example might be:

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

2. Ngm set: an unordered set of ngms. An example might be:

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

3. Ntp set: an unordered set of ntps. An example might be:

$$\left(\sim\ ,\ \begin{matrix}1\\ 0\end{matrix}\right), \left(\sim\ ,\ \begin{matrix}0\\ 1\end{matrix}\right).$$

4. Str set: an unordered set of strs. An example might be:

$$\boxed{1}\boxed{2}, \quad \boxed{1}\boxed{\ }\boxed{2}, \quad \boxed{1}\boxed{\ }\boxed{\ }\boxed{2}, \quad \boxed{1}\boxed{\ }\boxed{\ }\boxed{\ }\boxed{2}.$$

5. Perhaps most important of all is the *redefinition* of the ntp, so that a component of an ntp may be a pngm, pngm set, ngm, ngm set, str, str set, another ntp, or an ntp set.[20]

This latter definition makes the language of the machine able to express almost anything. It is particularly useful in expressing mathematical ideas. We may, for example, define sets of objects, sets of sets of objects, sets of sets of sets of objects, etc.

The integer, r, may be defined as the set of all sets of r objects — the set of all r-tuples. The concept "one greater than" may be defined as the set of all pairs of sets, such that the first set is the set of all r-tuples; the second set, the set of all $r+1$-tuples.

The concept "integer" may be defined as the set of all sets that represent integers.

[20] If a member of an ntp may be an ntp or an ntpst, then we have not defined anything. We can, however, define a ntp and ntpst *recursively* legitimately.

5.2 New abstraction combination methods – concept of function

Some additional combination methods will be used. An important one is the idea of a "function." It combines two ntp sets to produce a new ntp set. If (α_i, β_i) and (γ_j, δ_j) $[i = 1, \ldots, n]$, $[j = 1, \ldots, m]$ are two ntp sets, then we may define a new ntp set $(\alpha_i, \beta_i, \delta_j)$ in which all i, j pairs are taken for which $\beta_i = \delta_j$.[21]

Also, Boolean sums and products of sets can be used to form new sets.[22]

5.3 Modification of the machine for probabilistic/non-deterministic problems

Another important development is the modification of the present machine, so that it will work problems in which the correct answer is not a single digit, but a probability distribution over several digits. Aside from being able to work problems that the present machine cannot work, it will have the advantages of

1. Being independent of occasional errors in the information given to it. The presently described machine can have much of its prediction ability destroyed by a single error in the input data.
2. Such a machine is readily adaptable to the problem of modifying its own methods of prediction, since the efficacy of a new method of prediction is a probabilistic question.

5.4 Future problems to be solved by the machine

Some of the problems that will be investigated are:

1. Subtraction.
2. Omission of the "carry" line for both addition and subtraction.
3. Multiplication.
4. Division.
5. Change to "linear" notation, e.g. $10 + 101 = 111$.
6. Combining operations by use of parenthesis.
7. Literal solution of algebraic equations,
8. Interpolation, extrapolation of functions.
9. Differentiation, Integration.
10. Literal solution of differential equations.
11. Theorem proving.
12. Translation of English into logically tractable form.
13. Translation of logically tractable notation into English.

[21] This operation is obtainable via other operations, however.

[22] Not all boolean ops. are expressible in terms of "and" and "or". But the set of operations "and", "or" and "not" ($\equiv$ complementation) *is* complete. — as is the joint denier.

5.5 Machine Training Sequences as Training Sequences for Children

A possible early application of the present work is to devise proper training sequences for children, At the present time, there is little conscious recognition of the sizes of the various "logical jumps" that are necessary in learning some new material. Analysis of machine learning makes all of this very explicit and clarifies the problem in an important way.

6 An evaluation of this machine investigation as a study tool

Using the set of rather simple abstraction–creating devices described in 3.2.1.5, the machine is able to learn simple arithmetic operation up to addition and subtraction. With the additional aid of the new definitions of section 5.1 and the new abstraction–creating devices of Section 5.2, it is expected that the machine language will be powerful enough to express any problem and any heuristic device.

The advantages of this approach to the artificial intelligence problem are:

1. The operation of the device is fairly simple, in the sense that it needs no special apparatus to deal with important problems of great difficulty. This contrasts strongly with special machines for chess, checkers and theorem proving.

2. With this meager apparatus, it becomes possible to start work *immediately* on what is believed to be the heart of the artificial intelligence problem: i.e how to get new good ideas from old ones.

3. It is possible to work on problems that are as simple, or as difficult, as desired. Working simple problems is very useful, since the basic heuristic devices used tend to be clearer. As one proceeds to more difficult problems, the devices needed can be more easily guessed at, if one has experience with simpler problems.

4. The machine is particularly adapted to work on the problem of improving itself. The rules that express the methods of creating new abstractions from old ones are, in many cases, identical to the abstractions themselves. This statement, however, is only true in machines that have the complexity that is introduced through the new definitions of section 5.1.

It should be noted that the abstraction–creating devices described are extremely general and were not at all chosen for the particular problems given as examples. It is likely that, when the machine has worked its way up through mathematical theorem proving, it will need no additional heuristic devices to learn to play chess — any more than a man would in similar circumstances.

It is possible that the heuristic devices that have already been chosen will be close to sufficient for most problems we are interested in.

At a suitable level of development, it should be possible to teach the machine to answer simple questions in English. This particular problem had been one of the earlier approaches of the author to the artificial intelligence problem.

Elias' reviewer's comments suggests that including more examples of multiplication would be expedient.

A Appendix I: Some examples of multiplication of strs by ntps

$$\boxed{1}\boxed{2} \times (A, B) = AB$$

$$\begin{matrix}\boxed{2}\\ \boxed{1}\end{matrix} \times (A, B) = \begin{matrix} B \\ A \end{matrix}$$

$$\begin{matrix}\boxed{2}\\ \boxed{1}\boxed{\ }\boxed{1}\end{matrix} \times (A, B) = \begin{matrix} B \\ A \quad A \end{matrix}$$

$$\boxed{1}\boxed{2} \times \left(A, \begin{matrix} B \\ C \end{matrix}\right) = \begin{matrix} AB \\ C \end{matrix} \quad \text{and} \quad \begin{matrix} B \\ AC \end{matrix}$$

$$\begin{matrix}\boxed{1}\\ \boxed{2}\end{matrix} \times \left(\begin{matrix} A \\ B \end{matrix}, CD\right) = \begin{matrix} A \\ B \\ C \quad D \end{matrix} \quad \text{and} \quad \begin{matrix} A \\ B \\ C \quad D \end{matrix}$$

$$\begin{matrix}\ \ \boxed{2}\\ \boxed{1}\end{matrix} \times \left(\begin{matrix} 1 = \\ \alpha \end{matrix}, \ 3 \ 1\right) = \begin{pmatrix} 3 & 1 \\ 1 = & \\ \alpha & \end{pmatrix} \text{ and}$$

$$\begin{pmatrix} & 3 & 1 \\ 1 = & & \\ \alpha & & \end{pmatrix} \text{ and } \begin{pmatrix} 1 = & 3 & 1 \\ \alpha & & \end{pmatrix} \text{ and } \begin{pmatrix} 3 & 1 \\ 1 = & \\ \alpha & \end{pmatrix} \text{ and } \begin{pmatrix} & 3 & 1 \\ 1 = & & \\ \alpha & & \end{pmatrix}$$

B Appendix II: Improved Formulae for Utility Computations

It is desirable to epproximate the Utility, U_{ij}, of pages 11 and 16, by

$$U_{ij \text{ apriori}} = U_{S_i} \times U_{N_j}$$

rather than

$$U_{ij \text{ apriori}} = U_{S_i} + U_{N_j}.$$

One reason for use of the product form of apriori Utility, is that in the additive form, if a certain structure has high U, then if it is multiplied by *any* ntp, the result would be of high apriori U.

This state of affairs is undesirable and is alleviated by the multiplicative formula, in which *both* U_{S_i} and U_{N_j} are important in determining the apriori value of U_{ij} — independently of the relative sizes of U_{S_i} and U_{N_j}.

Another desirable characteristic of the multiplicative form is that apriori probabilities can, in general, be only estimated to "orders of magnitude." The employment of variance of the apriori distribution of $\ln U_{ij}$ to partly characterize the distribution of $U_{ij \text{ apriori}}$ is in good accord with the usual state of our apriori knowledge of U_{ij}.

A third and most important desirable characteristic of this method may be observed by comparing it with $U_{ij \text{ apriori}} = U_{S_i} + U_{N_j}$. In this latter method, approximations to U_{ij} are obtained which are optimum in the mean square sense, averaged over all U_{ij}s. This means that large values

of U_{ij} apriori will have a good "fractional accuracy," but small values of U_{ij} apriori will have *very poor* "fractional accuracy."

An important result of this disparity, is that if one is looking for pngms of highest possible U, that fit a particular q element, and one has exhausted the pngms of high U, then the accuracy in choosing the optimum pngm among many pngms of low U, will be *very poor*.

On the other hand, if $\ln U_{ij}$ apriori is optimized in the mean square sense, then the fractional accuracy of the U_{ij} apriori's will be independent of how large the U_{ij} apriori is, which is the condition we desire. This will enable us to choose the pngm of highest U_{ij} from a set of pngms, with an accuracy that does not decrease, as we exhaust the pngms of high U_{ij}.

To obtain the values of U_{ij} from our empirical data, C_{ij} and σ, assume that the apriori distribution of $\ln U_{ij}$ is

$$f(x)dx = \frac{(-x)^r}{r!M^{r+1}}e^{x/M}.$$

$f(x)dx$ is the probability that $x \leq \ln U_{ij} \leq x + dx$. This distribution is of mean, $M(r+1)$ and variance, $M^2(r+1)$. We want mean $\ln U_{S_i} + \ln U_{N_j}$ and variance σ^2 so we set

$$r \equiv \frac{\ln U_{S_i} + \ln U_{N_j}}{\sigma^2} - 1,$$

$$M \equiv \frac{\sigma^2}{\ln U_{S_i} + \ln U_{N_j}}.$$

A Poisson distribution is used for $f(x)$, since the range of x is $(-\infty, 0)$.

After we have obtained the empirical data, C_{ij} and τ (these symbols are defined on page 16) the aposteriori probability distributions for U_{ij} become, using Bayes' theorem on inverse probability,

$$g(x) = Ae^{C_{ij}x}(1 - e^x)^{\tau - C_{ij}}(-x)^r e^{x/M},$$

where $g(x)dx$ is the aposteriori probability that $x \leq \ln U_{ij} \leq x + dx$ and A is a normalization constant.

To obtain suitable values for the U_{S_i}s and U_{N_j}s and σ, we will again use Bayes' Theorem.

This time we will assume uniform apriori distributions for the $\ln U_{S_i}$s, $\ln U_{N_j}$s and for σ. This iterated use of Bayes' Theorem is certainly open to question.

It is believed by the author, however, that this method is legitimate in the present case and a fairly good way of expressing most of our intuitive apriori knowledge about σ, the U_{ij}s, U_{S_i}s, and the U_{N_j}s

The aposteriori probability distribution for σ, the $\ln U_{S_i}$s and $\ln U_{N_j}$s is given by

$$h(\ln U_{S_1}, \ln U_{S_2} \ldots \ln U_{S_m}, \ln U_{N_1}, \ln U_{N_2} \ldots U_{N_m}, \sigma)$$

$$= h(V_{S_1} \ldots V_{S_m}, V_{N_1} \ldots V_{N_m}, \sigma)$$

$$= B \underbrace{\int_{-\infty}^{0} \cdots \int_{-\infty}^{0}}_{mn \text{ integrals}} \prod_{i=1}^{m} \prod_{j=1}^{n} \left(\frac{\exp(C_{ij} \ln U_{ij})(1 - \exp(\ln U_{ij}))^{\tau - C_{ij}}}{r!M^{r+1}} (-\ln U_{ij})^r \exp(\frac{\ln U_{ij}}{M}) d \ln U_{ij} \right)$$

$$= B \int_{-\infty}^{0} \cdots \int_{-\infty}^{0} \left[\prod_{i=1}^{m} \prod_{j=1}^{n} \frac{\exp(C_{ij} V_{ij})(1 - \exp(V_{ij}))^{\tau - C_{ij}}}{\Gamma\left(\frac{(V_{S_i} + V_{N_j})^2}{\sigma^2}\right) \left(\frac{\sigma^2}{V_{S_i} + V_{N_j}}\right)^{\left(\frac{V_{S_i} + V_{N_j}}{\sigma}\right)^2}} (-V_{ij})^{\frac{(V_{S_i} + V_{N_j})^2}{\sigma^2} - 1} \exp(V_{ij} \frac{V_{S_i} + V_{N_j}}{\sigma^2}) dV_{ij} \right]$$

B is a normalization constant,

$$r! \equiv \Gamma(r+1)\,, \quad V_{ij} \equiv \ln U_{ij}\,, \quad V_{S_i} \equiv \ln U_{S_i}\,, \quad V_{N_j} \equiv \ln U_{N_j}\,, \quad \exp(x) \equiv e^x\,.$$

To explain the integral, let
"A" represent the statement "The apriori distribution of $\ln U_{ij}$ is of mean $\ln U_{S_i} + \ln U_{N_j}$ and variance σ^2."
"B" $\equiv$ "The true values of the $\ln U_{ij}$s are in the hypercube bounded by the $\ln U_{ij}$s and the $\ln U_{ij} + d \ln U_{ij}$s ."
"C" $\equiv$ "The empirical data $C_{11}, C_{12} \ldots C_{21}, C_{22} \ldots C_{mn}$, will be observed."
The "true" values of the U_{ij}s, are those that would be observed at $\tau = \infty$.

The integral is the probability that "if A, then C."

The integrand times $\prod_i \prod_j dU_{ij}$ is the probability that both "if A then B"; and "If B then C." This quantity is the product of the probabilities of two independent events. The first probability is of "if A then B" and is

$$\prod_i \prod_j \left[(-\ln U_{ij})^r e^{(\ln U_{ij})/M} d \ln U_{ij} \right]$$

The second probability is of "if B then C," and is

$$\prod_i \prod_j \frac{U_{ij}^{C_{ij}} (1 - U_{ij})^{\tau - C_{ij}}}{r! M^{r+1}}$$

The integral is of the form

$$\prod_k \left(\int_0^\infty e^{-\alpha_k x} (1 - e^{-x})^{\beta_k} x^{\gamma_k} dx \right)$$

in which β_k is always an integer. Although it immediately integrable by expanding $(1 - e^{-x})^{\beta_k}$, the general behavior of the function, h, is not readily perceived.

Since h is not a delta function, but, is in general, a rather broad distribution, it will not give us single values of $U_{S_1}, \ldots, U_{N_n}$ and σ. To obtain usable values of these quantities, we can use the coordinates for which h is maximized, or we may use the "expected values" of each of the quantities. Probably the latter would be a better set of parameters to use, though the question may have to be decided on the basis of which set of values is most easily obtained mathematically,

It should be noted that the behavior of the function, h, need only be examined for certain values of its arguments, and of the C_{ij}s and τ. In particular, the cases in which the sample size is small and the apriori distribution is broad, must be investigated. If τ and C_{ij} are large, good approximations are easy to make.

It is also important to note that all ranges of values of $\ln U_{ij}$ are not of equal interest. It will probably be convenient to consider all values of $\ln U_{ij}$ below a certain value, say — 20, as being

equal to this value — being essentially minus infinity. This cut-off point will be dictated largely by computer memory size and/or speed.

It is likely that the rather complex formula that has been derived for determining σ, the U_{S_i}s and the U_{N_j}s will not be used directly to determine these quantities. It is however, useful to check various approximate formulae that might be suggested. A particularly simple set of approximation formulae that may be sufficiently accurate is

$$\ln U_{S_i} = \frac{1}{n}\sum_{j=1}^{n} \ln \frac{C_{ij}}{\tau} - \frac{1}{2mn}\sum_{j=1}^{n}\sum_{i=1}^{m} \ln \frac{C_{ij}}{\tau}$$

$$\ln U_{N_j} = \frac{1}{m}\sum_{i=1}^{m} \ln \frac{C_{ij}}{\tau} - \frac{1}{2mn}\sum_{j=1}^{n}\sum_{i=1}^{m} \ln \frac{C_{ij}}{\tau}$$

Errata Letter

Technical Research Group 17 Union Square, West New York 3, New York

December 28, 1956
Dr. Herbert A. Simon, Dept. of Indust. Admin., Carnegie Tech, Pittsburgh, Pa.

Dear Dr. Simon:
Enclosed are some errata sheets for "An Inductive Inference Machine" by R.J. Solomonoff.
Also enclosed are a preface and an additional appendix to the report.

Sincerely,
Ray Solomonoff

Publication Notes, 2026, (Ed. Grace Solomonoff.)

Errata and Comments

1. The errata, preface and appendix II have all been incorporated into this version of the manuscript.
2. There are no errata or marginal notes for appendix II or the preface, and they may have mistakes.
3. Manuscript has some marginal notes, probably by Ray

History

1. This is the first known full report of a general machine learning system. Some other programs had been written, but only for specific applications, such as a checkers game.
2. It was written during the Dartmouth Summer Workshop on Artificial Intelligence, 1956, where Ray gave two lectures on it.
3. It was circulated at the IRE Convention 1956, and sent to various math departments. A much shorter version was published by the IRE in 1957, but this full version was unpublished until now.

4. The Dart Notebook referred to, was also written during the summer and has more general thoughts. It is in the Dartmouth Archives section of raysolomonoff.com.