Lecture 1: Algorithmic Probability

Ray J. Solomonoff

rjsolo@ieee.org http://world.std.com/˜rjs/

Introduction

The Subject of this Series of Lectures is: Strong A.I. — what it is, how we may achieve it — apparent obstacles to achieving it — and how to overcome these obstacles.

First: What is "strong A.I."? My own personal interest is in a machine that is at least as good as a human in solving problems in Math, Science, and most aspects of Engineering. Eventually, through its ability to improve itself, and through the action of Moore's law, it will be able to greatly transcend humans.

The Critical capabilities:

1. Ability to solve most problems as well as humans do.

2. That it is embodied in a technology to which Moore's law applies — i.e. that the amount of computing we get for a dollar, doubles every year or so.

3. It has the ability to work effectively on the problem of self improvement — both in software and in hardware — so the "every year or two" in Moore's law becomes progressively shorter and shorter. At a certain point in the life of the machine we begin to get very rapid improvement and a very intelligent machine.

The result is an extremely rapid increase in computation speed in a relatively short time — accompanied by a corresponding dramatic increase in problem solving capability.

How do we get Strong A.I.? If you give a good scientist a problem to solve, he is able to bring to bear on its solution, everything that he has ever learned in a very effective way. This is probably an optimum way of learning.

We now have an extremely good understanding of learning in a large area of human endeavor... probabilistic prediction. We know how to bring all past experience into play to make near optimum predictions in the sciences. [1] We are able to apply this technique to the most difficult aspects of the sciences... inventing new theories and revision of old theories.

I will show how these prediction techniques can be extended to solve more general kinds of scientific problems.

[1] "Near optimum prediction" is not always very useful prediction: Best prediction of the toss of a coin may remain near 50–50: for chaotic processes like long term weather prediction, the best prediction can be of very low precision.

There is a natural question at this point. The scientific community has been working on A.I. for at least 50 years with no apparent great breakthroughs: why do I feel we are now so much closer to a solution to the problem.

In the past, there have been several obstacles to achieving strong A.I. Among the most prominent:

1) Inadequate understanding of learning/prediction: (I will regard these two as identical for the purposes of our discussion).

2) In induction models: use of an inadequate set of functions.

3) In search for models: Inadequate search techniques, inadequate diversity in search space.

Still, many years after the solutions to these problems were available to the A.I. Community, strong A.I. has not appeared! — Why is this?

Associated with the good models for induction is the need to search over a large space of functions; these functions are candidates for good induction models. We have a very good criterion for the merit of a particular induction function, but we still have to *find* the good functions.

To solve this problem, several tools have for some time, been available to the A.I. community:

1) Levin's search procedure: (I'm using Levin Search in a more general way than originally proposed) This enables us to avoid spending excessive time testing functions (partial recursive functions) that don't converge.

2) PPM: Prediction by partial Matching. This is a very fast, surprisingly precise method for approximate induction. It is possible to improve this method considerably — but at the moment there seems to be a very large speed penalty.

3) Discovery of Grammars of Stochastic Context Free Grammars. A relatively recent solution to an old problem that enables us to get approximate induction over a significantly larger, more relevant, function space than ever before.

4) Genetic programming and its recent enhancements. A potentially adequate technique for very good induction that has in the past, been plagued by inadequacies that we can now address with some degree of confidence.

I've mentioned some of the past inadequacies of A.I. research that we presumably now know how to deal with. I've listed a few techniques that I expect will be useful in attaining the goal. how do we put this all together to get "Strong A.I."?

First we note that intelligence is not obtained in a vacuum: Just as Newton "stood on the shoulders of Giants", our artificial intelligence will "stand on the shoulders" of everyone who has contributed to our modern scientific enterprize.

In what we will call "Phase I", this is accomplished by using "Training Sequences" of problems that embody the intelligence of the Scientific Community. The Training Sequence contains *all* of the information we ever put into the machine. We never "tell" the machine anything. It learns only by discovering solutions to problems and by discovering regularities in those solution.

We start out with very simple problems: the machine tries various possible solutions in order of complexity — using Levin's search technique to decide how much time to spend on a trial. After it has solved a fair number of simple problems, it looks at regularities in the solutions to these problems, to guide future searches. To start off, it will use PPM, PSG discovery and possibly a type of Genetic programming to search for "regularities". Using these regularities, it is able to solve harder problems — which it uses to find more "regularities" to solve more difficult problems — and so on.

At a certain point, the machine will have learned enough to usefully work on the problem of improving its search techniques. At this point it enters "Phase II".

Because of the limited search techniques used in Phase I, the system was severely limited in the kinds of problems it could solve. These limitations do not exist in Phase II — which is able to discover arbitrarily complex search techniques.

While Phase I was limited to a certain broad class of prediction problems, Phase II is able to work on any well defined problems. It is limited only in our ability to write good training sequences and our ability to give it the information processing capacity to solve its problems. At this point we are well on the road to "Strong A.I."

The first lecture will describe Algorithmic Probability. This will be followed by a discussion of how it is used in practical problems.

I'll talk about Context Free Grammar (CFG) discovery, Prediction by Partial Matching (PPM) and Genetic Programming, and their roles as components of Strong A.I.

The last lecture will go into the details of Phase I and Phase II, showing how the components fit together to create "Strong A.I.".

1 Probability and Induction

What is probability?

"An estimate of what is to occur in the future". Also necessary is a measure of confidence in the prediction: As a negative example consider an early A.I. program called "Prospector". It was given the characteristics of a plot of land and was expected to suggest places to drill for oil. While it did indeed do that, it soon became clear that without having any estimate of confidence, it is impossible to know whether it is economically feasible to spend $100,000 for an exploratory drill rig. The moral is that predictions by themselves are usually of little value – it is necessary to have confidence levels associated with the predictions. Probability is one way to express this confidence.

What is Induction?

Prediction is usually done by finding inductive models. These are deterministic or probabilistic rules for prediction. We are given a batch of data – typically a series of zeros and ones, and we are asked to predict any one of the data points as a function of the data points that precede it.

In the simplest case, let us suppose that the data has a very simple structure:

0101010101010...

In this case, a good inductive rule is "zero is always followed by one; one is always followed by zero". This is an example of deterministic induction, and deterministic prediction. It is 100% correct every time!

There is, however, a common kind of induction and prediction that is not as reliable. Suppose we are given a sequence of zeros and ones with very little apparent structure. The only noticeable regularity is that zero occurs 70% of the time and one appears 30% of the time. More generally, inductive algorithms will give a probability for each symbol in a sequence, as a function of the previous symbols.

One measure of "Merit" of an induction algorithm is the probability that it assigns to the entire data set. This will be the product of the probabilities assigned to each of the symbols: $S = \prod p_i$.

We want as large a value of S as possible. The maximum value S can have is one, which implies perfect prediction.

2 ALP, MDL and Stochastic Complexity

An important application of symbol prediction is text compression. If an induction algorithm assigns a probability S to a text, we can devise a coding method that can re-create the entire text using just $-log_2 S$ bits. If $S = 1$, then $-log_2 S = 0$ and the induction algorithm will be able to reproduce the text with no extra information. The bit cost of the text will be the number of bits in the description of the algorithm.

If M is the name of the induction algorithm, then we will use $|M|$ to represent the number of bits in the description of M. If $S > 0$ then the code of the text is in two parts: first, the code for M of $|M|$ of bits, then the code for the probabilistic description of the data, which is $-log_2 S$ bits. The total number of bits in the compressed code is $|M| - log_2 S$ bits.

We can use this length of compressed code as a "figure of merit" of a particular induction algorithm. Ideally we want an algorithm that will give good prediction: i.e. Large S, and small $|M|$. $|M| - log_2 S$, the figure of merit, should be as small as possible.

Usually there are many inductive models available. Our "figure of merit" can be used to select the best inductive model to use. Early approximations were made by Van Heerden in 1963; later in 1968 Wallace described MML (minimum message length) and in 1978 Rissanen wrote about MDL (minimum description length). MML and MDL are very similar – I'm not sure I know what the difference is.

MML, MDL and Stochastic Complexity may all be regarded as approximations of Algorithmic Probability (ALP) — which was first described much earlier — by Solomonoff in 1960. Instead of selecting a single "best" inductive hypothesis, ALP uses a weighted sum of *all* possible hypotheses.

In finding good hypotheses for empirical data, we have two kinds of possible errors: Overfitting, and Underfitting.

In the case of Overfitting we obtain very good probabilities for the corpus – large S – by using a very large, complex hypothesis – i.e. large $|M|$. Predictions of future data tend to be poor.

In Underfitting we use very simple hypotheses: very small $|M|$, but the probabilities assigned to the corpus are small – S is too small. Predictions of future data tend to be poor.

By choosing a hypothesis that minimizes $|M| - log_2 S$ we avoid both Underfitting and Overfitting to a large extent, and obtain very good predictions for future data.

It should be noted that in general, it is impossible to find with any certainty, the true Minimum Description Length model – there is an infinity of models to be tested. At any particular time in the search, we will know the best one so far, but we can't ever be sure that spending 10 more minutes of search will not give a *much* better model.

This strict "incomputability" of MDL is usually dealt with by restricting the space of models to be considered – so one *can* make a complete search in a finite time.

We have briefly mentioned ALP as a weighted sum of all possible models. More exactly:

$$P_M(a_{n+1}|a_1, a_2 \cdots a_n) = \sum 2^{-|M_i|}\ S_i\ M_i(a_{n+1}|a_1, a_2 \cdots a_n) \tag{1}$$

Here $M_i(a_{n+1}|a_1, a_2 \cdots a_n)$ is the probability assigned by the ith model to the predicted $(n+1)$th symbol of the sequence, in view of the previous part of the sequence. S_i is the probability assigned by M_i (the ith model) to the known data. $2^{-|M_i|}$ is something like the a-priori probability assigned to the model M – the probability we assign to M *before we see the data.*

This is apparently a more complex, more time consuming way to compute probabilities – (though actually, summing a bunch of probabilities, rather than picking the largest, uses about the same computing resources).

What advantages does ALP have? For one, it's very accurate. Say we had some data that was generated by an *unknown* probabilistic source, P. Not knowing P, we use instead, P_M, the Algorithmic Probabilities of the symbols in the data. How much do the symbol probabilities computed by P_M differ from their true probabilities, P? The Expected value with respect to P of the total square error between P and P_M is bounded by $-1/2 lnP_0$.

$$\underset{P}{E} \sum_{m=1}^{n} (P_M(a_{m+1} = 1|a_1, a_2 \cdots a_m) - P(a_{m+1} = 1|a_1, a_2 \cdots a_m))^2 \quad \leq \quad -\frac{1}{2}\, lnP_0$$

$$lnP_0 \approx kln2 \tag{2}$$

P_0 is the a-priori probability of P. It is the probability we would assign to P if we knew P, but *hadn't seen the data.*

k is the Kolmogorov complexity of the data generator, P. It's the shortest binary program that can describe P, the generator of the data.

This is an extremely small error rate. The error in probability decreases more rapidly than $1/n$.

So . . . rapid convergence to correct probabilities is one big feature of ALP. Like MDL, ALP is incomputable.

A common approximation is Rissanen's Stochastic Complexity. Instead of summing over *all* models, he picks a space of models of interest and sums over that space only.

In using ALP. we always make approximations. Rissanen's picking a space of models is one way to approximate. Another way is to not pick the set of models a-priori, but have the set to be considered come out of the search itself.

What we want is a set of models of maximum total weight. A set of this sort will bring us as close as possible to ALP and give best predictions.

To obtain such a set, we devise a search technique that tries to find, in the available time, a set of Models, M_i, such that the total weight,

$$\sum_i 2^{-|M_i|} S_i \tag{3}$$

is as large as possible.

Apart from accuracy of probability estimate, ALP has for AI another, more important value: It gives a very good *understanding* of the data.

In the area of AI that I'm most interested in – Incremental Learning – this diversity of explanations is of major importance. The way the System works: At each point in the Life of the System, it is able to solve with acceptable merit, all of the problems it's been given thus far. We give it a new problem – usually its present Algorithm is adequate. Occasionally it will have to be modified a bit. But every once in a while it gets a problem of real difficulty and the present Algorithm has to be seriously revised. At such times, we try using or modifying *once sub-optimal algorithms.*

If that doesn't work we can use parts of the sub-optimal algorithms and put them together in new ways to make new trial algorithms.

It is in giving us a broader basis to learn from the past, that the value of ALP lies. A very conventional scientist understands his science using the *single* "current paradigm" – A more creative scientist understands his science in very many ways, and can more easily create new ways of understanding when the need arises for *Theory Revision.*

To review a bit: Thus far, I've described ALP and shown how it differs from MDL and Stochastic Complexity. I've discussed its precision and a little bit of its "incomputability" and why I use ALP in AI.

The probability distribution for ALP that I've shown is called "The Universal Distribution for sequential prediction".

There are two other universal distributions I'd like to describe.

Suppose we have a corpus of discrete objects, each described by an finite string a_j:

Given a new finite object, what is the probability that it is in the previous set? In MDL, we consider various algorithms that assign probabilities to strings. (We might regard them as *Probabilistic Grammars.*) We would use for prediction, the grammar, M, for which

$$|M_i| - log_2 S_i \tag{4}$$

is maximum.

Here $|M_i|$ is the number of bits in the description of the algorithm M_i.

$S_i = \prod_j M_i(a_j)$ is the probability assigned to the entire corpus by M_i.

To obtain the ALP version, we simply sum over all models as before, using weights $2^{-|M_i|} S_i$

This kind of ALP has an associated convergence theorem giving very small errors in probability. As noted, this approach can be used in linguistics. We can use $|M_i| - log_2 S_i$ to assign a likelihood that the data was created by grammar, M_i.

There is another kind of problem that I'll be discussing later at some length. Suppose you have a bunch of question answer pairs, Q_i, A_i: Given a new question, Q_{n+1}, what is the probability distribution over possible answers, A_{n+1}? Equivalently, we have an unknown analog and/or digital transducer, and we are given a set of input/output pairs Q_i, A_i – For a new input Q_i, what is probability distribution on outputs? Or, say the Q_i are classification data for mushrooms and the A_i are whether they are poisonous or not.

As before, we hypothesize an operator $O^j(A|Q)$ that is able to assign a probability to any A given any Q:

The ALP solution is

$$\begin{aligned} P(A_{n+1}|Q_{n+1}) &= \sum_j a_0^j \prod_{i=1}^{n+1} M^j(A_i|Q_i) \\ &= \sum_j a_n^j M^j(A_{n+1}|Q_{n+1}) \\ a_n^j &= a_0^j \prod_{i=1}^{n} M^j(A_i|Q_i) \end{aligned} \tag{5}$$

The ALP system has a corresponding theorem for small probability of error. There is a corresponding MDL version and Stochastic Complexity version. In MDL we pick the model of maximum weight. In Stochastic Complexity we choose a subspace of models to sum over. In ALP we try to find a set of models of maximum weight in the available time.

3 Incomputability

Just how does incomputability arise and how do we deal with it?

Recall that for ALP. we added up the predictions of all models, using suitable weights:

$$P_M = \sum_{i=1}^{\infty} 2^{-|M_i|} \; S_i M_i \tag{6}$$

Here M_i gives the probability distribution for the next symbol as viewed by the ith model. Just what do we mean by these models, M_i?

There are at least three kinds of functions that M_i can be.

1. Primitive Recursive
2. Total Recursive
3. Partial Recursive

Primitive Recursive Functions are very well behaved. It is possible to enumerate them recursively – which make the summation from 1 to infinity meaningful. For each value of its argument, it is always possible to know if the function has a value or not. Primitive recursive functions are definable by *Do* loops in which the parameters of the loop are primitive recursive or compositions of primitive recursive functions. Just about all of the functions used in Science and Engineering are primitive recursive. It took many years before the Akerman function was discovered – a function that increases so rapidly that it could not possibly be primitive recursive.

Partial recursive functions are definable by recursion equations. They include all primitive recursive functions, and like primitive recursive functions, they are recursively enumerable, so the summation to infinity is meaningful.

Some partial recursive functions do not have values (they are not defined) for certain values of their arguments. The thing that causes trouble is that for certain arguments of certain of the functions, it is impossible to know if the function is defined for those arguments!

The way it works: Each partial recursive function is defined by a computer program. You plug in the argument and often the program will quickly print an output and stop. For other values of the argument, the program will take a long time – but will eventually print an output and stop.

For other argument inputs, the machine will compute without output for a long time. It *may* eventually print and stop. – But it may calculate forever and never stop. Furthermore, it is impossible to tell if it will eventually stop, or not. If you've had no output for a month or so, there is no good reason to believe it will not print and stop in the next 10 minutes!

The total recursive functions are a subset of partial recursive functions in which we know which arguments for which each function eventually prints an output. While the total recursive functions are enumerable (since they are a sub-set of a recursively enumerable set), they are not themselves *recursively* enumerable – There is often no way to tell if a function is total recursive or not.

In searching over possible induction models, MDL and Stochastic Complexity confine themselves to primitive recursive functions. In my own work on incremental learning, I plan to search over the space of partial recursive functions. It involves spending much time making trials on functions that don't have any output.

Since I am interested in obtaining the most "weight" of predictive models for a given computation time – this would seem like a very counterproductive approach!

The main attraction of partial recursive functions is that they enable very compact descriptions of important, useful functions. Furthermore, these functions are expressible as simple combinations of other primitive recursive functions, so they are easy to learn.

Many partial recursive functions can be expressed more cheaply (less memory and time) as primitive recursive functions using *Do* loops, but in the form of primitive recursive functions they would be far more complex and have much lower a-priori probability than their associated partial recursive functions. It may be cost effective to compile some of the partial recursion functions as primitive recursive functions.

The forgoing discussions have been largely theoretical. From a practical point of view, the computability of primitive recursive versus partial recursive functions may not differ very much. While a primitive recursive function will always converge for the arguments that it is supposed to converge for, the amount of time for convergence can be *very large* — well beyond the time available. In which case there would be no *practical* difference between such a function and a partial recursive function.

Another mitigating fact about partial recursive functions is that after you have used one a great deal and found it converges for all values found, it becomes *very likely* that it will continue to converge for new arguments. This property is particularly desirable in predictive functions that have been successful over a large set of historical examples.

However, the trade off between partial and primitive recursive functions is likely to be determined empirically rather than theoretically.

4 Subjectivity

The subjectivity of Probability resides in a-priori information – the information available to the statistician before he sees the data to be extrapolated. This is independent of what kind of statistical functions we use. In ALP this takes the form of our "Reference Computer". Recall our assignment of a $|M|$ value to an induction model – It was the length of the program necessary to describe that function. In general, this will depend on the machine we use . . . its instruction set. Since the machines we use are Universal – they can imitate one another – the length of description of programs will not vary wildly between most reference machines we might consider. But nevertheless, using small samples of data (as we often do in AI), these differences between machines can modify results considerably...

Consider the evolution of a-priori in a scientist during the course of his life. He starts at birth with minimal a-priori – but enough to be able to learn to walk, to learn to communicate and his immune system is able to adapt to certain hostilities in the environment. As he grows older he solves new problems and incorporates the problem solving routines into his a-priori tools for problem solving. This continues through out the life of the scientist – as he matures, his a-priori information matures with him.

In making predictions there are several common ways to insert a-priori information. First, by restricting the set of induction models to be considered. This is certainly the commonest way. Second, by consciously assuming certain probability distributions over certain parameters based on past experience with such parameters. Third, we note that much of the information in our sciences is expressed as definitions — modifications of language. ALP, or approximations of it, avails itself of this information by using these definitions to help assign code lengths to model descriptions.

References

[1] Cover, T. and Thomas, J. *Elements of Information Theory*, Wiley and Sons, N.Y., 1991 – Good treatments of statistics, predictions, etc.

[2] Li, M. and Vitányi, P. *An Introduction to Kolmogorov Complexity and Its Applications*, Springer-Verlag, N.Y., 1993.

[3] Li, M. and Vitányi, P. *An Introduction to Kolmogorov Complexity and Its Applications*, Springer-Verlag, N.Y., 1997.

– Li, Vitányi, 1993, 1997 – Starts with elementary treatment and development. Many sections very clear, very well written. Other sections difficult to understand. Occasional serious ambiguity of notation (e.g. definition of "enumerable"). Treatment of probability is better in 1997 than in 1994 edition.

The following papers are all available at the website: world.std.com/˜rjs/pubs.html.

[4] "A Preliminary Report on a General Theory of Inductive Inference," 1960.

[5] "A Formal Theory of Inductive Inference," *Information and Control*, Part I: 1964.

– "A Preliminary Report . . ."and "A Formal Theory... " give some intuitive justification for the way ALP does induction.

[6] "The Discovery of Algorithmic Probability," 1997. – Gives heuristic background for discovery of ALP. Page 27 gives a time line of important publications related to development of ALP.

[7] "Progress in Incremental Machine Learning; Revision 2.0," Oct. 30, 2003. – A more detailed description of the system I'm currently working on. There have been important developments since, however.

[8] “The Universal Distribution and Machine Learning,” 2003,– Discussion of irrelevance of incomputability to applications for prediction. Also discussion of subjectivity.

[9] “Three Kinds of Probabilistic Induction: Universal Distributions and Convergence Theorems,”— Discusses their error rates