3/27/01

1

IDSIA: Title, Abstract, of first talk. April 4, (Wed) 11 AM — 130 – 230 r Giovanni.

A ~~Program for~~ System Machine Learning.

~~"The Scientist's Assistant: a Sub Goal for [redacted]~~ High Level ~~Machine Learning."~~

~~The Scientist's Assistant has been a long term subject in our research in AI and Machine learning.~~ We will describe Recent progress in "The Scientist Assistant" — a program for machine learning that we have been working on for many years. Very little of ~~the program~~ it has been embodied in software. Our discussions will ~~be largely of theoretical analyses of the~~ operation of the system ~~—~~ how it learns to solve problems, and some of its apparent difficulties, and our motivation for working on a system of this kind.

~~The system begins with some initially simple techniques for solving problems.~~ We give it so problems, which it solves easily. Having solved these easy problems, we give it somewhat more difficult problems — which it solves using some of the techniques ~~it~~ acquired in solving the initial set. We continue to give the system problem sets of progressive difficulty, ~~until it has acquired the~~ and each time it is able ~~has acquired the desired~~ required ~~level of intelligence.~~ to solve them by using techniques acquired in solving previous problems. We continue this training until it acquires the desired level of intelligence.

~~It sov~~

We begin training the system by giving it an initial set of simple problems. Using a kind of Levin-Universal Search, it is able to solve these ~~simply~~ simple problems, guided by an initially simple probability distribution over possible solutions. Having solved these problems, the guiding probability distribution is modified, in view of the problems solved. We then give it more difficult problems that it is able to solve, ~~using the modified probability~~ ~~using~~ using Levin Search, but guided by the modified probability distribution. ~~This process of problem~~ The probability distribution is then again modified to accommodate the newly solved problems. ~~This alternation ... the alternation of Random modification of the probability distribution and to~~ Learning problems of progressively greater ~~difficulty~~ difficulty is alternated with modification of the probability distribution. A very high level of problem solving expertise ~~has been acquired~~ can be acquired in this manner.

~~[illegible]~~ We will discuss some of the details of program construction, the kinds of problems solved, the method of modification of the probability distribution as well as the capabilitys and [redacted] limitations of our system.

3/22/01 2

antonella@idsia.ch.

D

Abstract

We will describe a system for machine learning that uses Levin's universal search (Lsearch) algorithm to solve problems of progressively increasing difficulty.

First a discussion of the search algorithm — its necessary limitations and how they can be overcome — partly by modifying the algorithm itself and partly by progressive modification of the probability distribution that guides it.

~~Title~~ Title: A System for Machine Learning based on Levin's Universal Search Algorithm.

We will discuss Levin's search algorithm (Lsearch) in various forms and the kinds of problems it can solve.

~~There are necessary limitations~~ its

One limitation that normally limits its use to simple problems — we will show how to overcome this limitation by using a conditional probability distribution to guide the search.

18 By suitably updating this conditional probability distribution, the system is able to integrate its knowledge base of solutions to all of the problems it has solved. The updating algorithm is based on a global credit assignment function.

~~We will describe the updating algorithm~~ the Global credit assignment function that governs the updating algorithm.

In previous papers, we claimed that it was very plausible that this system was within a factor of 4 of optimum, if we restricted the behavior of competing systems in certain ways. If we relax these restrictions and modify our system accordingly, the "factor of 4" becomes closer to "a factor of one" but the plausibility arguments are more uncertain.

If we relax these restrictions on competing systems and suitably augment our own system, it appears that we may be very close to optimum, but the "plausibility arguments" are not as strong as they were formerly.

~~With~~ within a certain restricted class of problem solving systems, the system proposed, we have heuristic plausibility arguments to show it was optimum within a factor of 4. Using a more efficient form of Levin's algorithm we can obtain a new factor of 2. If we are to compare our system with an unrestricted set of problem solving algorithms —

If we augment our system a bit, it appears to be within a factor of 2 of unrestricted learning algorithms solving the same kinds of problems — but the heuristic arguments justifying this claim are weaker than before.

3/22/01

Idsz

.00:12:26 We will discuss this claim in some detail as well as more recent claims of ~~a~~ near optimality under less restricted conditions.

3/27/01 1

IDSIA: Title, Abstract, of first talk April 4, (Wed) 11 AM 130 — 230 Giovanni.

A ~~Program~~ System for Machine Learning.

"~~The Scientist Assistant: a Sub Goal for~~ High Level ~~Machine learning.~~"

~~The Scientist Assistant has been a long term subject in our research in~~ ~~machine learning.~~ We will describe Recent progress in "The Scientist Assistant" — a program for machine learning that we have been working on for many years. Very little ~~of the program~~ it has been embodied in software. Our discussions will ~~be largely of theoretical analysis of the~~ operation of the system — How it learns to solve problems, and some of its apparent difficulties, and our motivation for working on a system of this kind.

~~The system begins with some initially simple techniques for solving problems.~~ We give it ~~some~~ problems, which it solves easily. Having solved these easy problems, we give it somewhat more difficult problems — which it solves using some of the techniques it acquired in solving the initial set. We continue to give the system problem sets of progressive difficulty, ~~~~ and each time it is able ~~has acquired the needed, required level of intelligence~~ to solve them by using techniques acquired in solving previous problems. We continue this training until it acquires the desired level of intelligence.

~~It sov~~

We begin training the system by giving it an initial set of simple problems. Using a kind of Levin Universal Search, it is able to solve these simple problems, guided by an initially simple probability distribution over possible solutions. Having solved these problems, the Guiding probability distribution is modified, in view of the problems solved. We then give it more difficult problems that it is able to solve, ~~they~~ ~~through the problem~~ using Levin Search, but guided by the modified probability distribution. ~~This process of problem~~ The probability distribution is then again modified to accommodate the newly solved problems. ~~This alternation ... the alternation of Random ... modification of the probability distribution and ...~~ Presenting problems of progressively greater ~~difficulty~~ difficulty is alternated with modification of the probability distribution. A very high level of problem solving expertise can be acquired in this manner. ~~has been acquired~~

We will discuss some of the details of program construction, the kinds of problems solved, the method of modification of the probability distribution as well as the capabilities and limitations of the system.

3/22/01

2

antonella@idsia.ch

D

Abstract

We will describe a System for Machine learning that uses Levin's Universal Search Algorithm (Lsearch) to solve problems of progressively increasing difficulty.

First a discussion of the search algorithm — its necessary limitations and how they can be overcome — Partly by modifying the algorithm itself and partly by progressive modification of the probability distribution that guides it.

Title: A System for Machine Learning based on Levin's Universal Search Algorithm.

We will discuss Levin's search algorithm (Lsearch) in various forms and the kinds of problems it can solve.

~~There are necessary limitations~~ its

One limitation that normally limits its use to simple problems we will show how to overcome this limitation by using a ~~probability distribution~~ conditional probability distribution to guide the search.

18 By updating this conditional probability distribution, the system is able to integrate its knowledge base of solutions to all of the problems it has solved. The updating algorithm is based on a global credit assignment function

~~We will describe the~~ ~~the~~ Global credit assignment ~~function~~ that governs the updating algorithm.

In previous papers, we claimed that it was very plausible that this system was within a factor of 4 of optimum, if we restricted the behavior of competing systems in certain ways. If we relax these restrictions and modify our system accordingly,

.26 the "factor of 4" becomes closer to "a factor of one" but the plausibility arguments are more uncertain.

If we relax these restrictions on competing systems and suitably augment our own system, it appears that we may be very close to optimum, but the "plausibility arguments" are not as strong as they were formerly.

Within a certain restricted class of problem solving systems, the system proposed, we have heuristic plausibility arguments to show it was optimum within a factor of 4. Using a more efficient form of Levin's algorithm we can obtain a new factor of 2. If we are to compare our system with an unrestricted set of problem solving algorithms —

If we augment our system a bit, it appears to be within a factor of 2 of unrestricted learning algorithms solving the same kinds of problems — but the heuristic arguments justifying this claim are weaker than before.

3/22/01

Ids2

.00[12.26] We will discuss this claim in some detail as well as more recent claims of ~~[illegible]~~ near optimality under less restricted conditions.

4/3/01 Idea

Talk, $\nvDash \int_{-\infty} \int$ [redacted]

.00 : 94.40

[redacted] dealt with more fully, after we [redacted] discuss [redacted] the estimation of probabilities.

The particular method of probability estimation we use is called Algorithmic Probability.

Suppose we [redacted] are doing sequential prediction! We have a long/Binary Sequence ~~of~~ ~~symbols~~, S and we want to know the relative probability of the next symbol being 1 rather than 0 — ~~S is a sequence of symbols in an Alphabet of which a and b are members~~

Let us assume that we know an a priori distribution on all possible/Binary strings ~~of symbols~~, [redacted] R, ~~from our alphabet~~. P(R) assigns a probability to each such string. Then the relative probabilities of the continuations 1 & 0 will be $\frac{P(S1)}{P(S0)}$. This would solve the sequential prediction problem, if we knew P(R). —

How to find [redacted] P(R).

For a heuristic understanding of what follows, consider Ockham's [redacted] idea: That simpler hypotheses are more likely than complex ones. We will quantify this idea and describe some results on [redacted] [redacted] the accuracy of the resultant system.

To quantify Ockham: Suppose B is a finite binary sequence. — Then D_B is "description" of B if $M_r(D_B) = B$.

Here M_r is an algorithm or [redacted] computing machine that is able to take the Binary string D_B as input, for which it gives B as output. D_B can be regarded as ~~Description~~ Description of B in terms of Machine function M_r.

I use the subscript r, because M_r is to be regarded as a reference machine. Clearly, by changing reference machines, [redacted] the various descriptions for B will change. The shortest description will be the one with fewest bits — containing least information. We could approximate the probability of B by 2^{-D_B}. For more accurate estimation, we use $\sum_i 2^{-D_{B_i}}$. / Also [illegible] theory "[illegible]" Here we are getting the total probability due to all possible descriptions of B. We will call $\sum_i 2^{-D_{B_i}}$ the Algorithmic probability assigned to B by M_r $= P^{M_r}(B)$

4/3/01 IDSIA

.00:95.40; There are a couple of loose ends here: First, what reference Machine to use for our probabilities? There is a group of Machines called "Universal Machines", that are particularly good for describing things. They have the property that if we compare 2 of them, there will always be a constant factor that tells how much they differ from one another.

Two machines M_1, M_2 Then $P^{M_1}(B)$ and $P^{M_2}(B)$ are always within a constant factor $C_{1,2}$ of one another. This constant factor depends on M_1 and M_2, but is independent of B. The constant factor can be quite large — so it usually makes a lot of difference as to what reference Machine one uses.

In our System for Machine learning we will be periodically changing the reference machine as a way to accomodate new things that the system has learned.

Second The method of computing probabilities of strings may seem a bit arbitrary, — and for several years after I thought of this method, I wasn't sure it would work — wasn't sure about the details of the method. However, I finally did work out a proof that the system was very accurate in estimating probabilities from empirical data. $E \sum_{i=1}^{m} (P_i - P_i')^2 \le \frac{1}{2} b \ln 2$: b is some length of the prior of them. if we used a universal reference Machine.

.1E

Third It turns out that $\sum_i 2^{-D_{B_i}}$ is not computable in a finite amount of time, We can't even be sure that we'll have found the shortest code for B! That Algorithmic probability is incomputable may seem like a strong argument against it — but it is not — this incomputability is an essential part of probability — of science itself. → 97.01 → 99.00

.23

The short codes that contribute most to the $\sum 2^{-D_{B,i}}$ correspond to strong regularities in the data. Consider the string $(01)^{1024}$. We can easily program a machine to write that sequence by simply telling it to write 01, 1024 times — which takes 2 bits to say "01" and 10 bits to say 1024 plus a few other bits — a vast compression over the 2048 bits in the original string;

.. In that is a 02 pattern.

.001 96.23 ! The reason is not so much that we can't try all possible codes! — very long codes don't contribute much to the sum. The main problem is that there are certain short strings that we put into the Machine, and the machine runs and runs, and after a long time, we still don't know if it will output B and stop — and there is no survey to tell. → $\boxed{\text{Back to 96.23}}$

In general in science, when you have a batch of data and you are looking for regularities in the data and you want to be used for prediction — you can never be sure that if you spend 10 more minutes looking for regularities would not give one much better than the best you've found yet.

The corresponding thing occurs with ~~random~~ sequences that seem to be "RANDOM" — i.e. no discernable regularities. It is believed by many, that the stock market prices are random — yet there is no finite amount of investigation that could be convincing on this question. On the other [redacted], it becomes hard, once a strong regularity is found, [redacted] very unlikely that the sequence is [redacted] random.

The problem of induction is to obtain good approximations to $\sum_{i=1}^{\infty} 2^{-l_i}$ ← P

This is done by [$\sum_i 2^{-l_i}$ summing over not all of the codes.] ← P'

P' is an approximation to P. The more codes one finds the closer P' is to P.

To maximize P' by finding as many codes as possible [redacted] in the available time, is a [redacted] time limited optimization problem. It is equivalent to having a minimum error (i.e. P−P') in one's estimate, P'.

[redacted] The ~~prediction~~ induction/inference problem and its solution as a time limited optzn problem are very important in the ~~system~~ learning system that I'm describing. One reason is that the problem of updating the probability distribution that guides searches for solutions to problems — is an induction problem.

How is this so?

Consider all of the [problem description$_i$, solution program$_i$, Time$_i$] triplets that have occurred thus far.

[Problem Descr$_i$, Solution$_i$ ~~Program~~ Time$_i$]

4/3/01 IDSIA

.00 : 97.40 ! What we want is a bunch of relatively short codes to describe all of this data.
From a set of codes of this sort, we can then extrapolate the data to say new.

GPD_1 (problem, solution) [redacted] → Probability distribution on Time to solution.

From GPD_1 it is possible by a process of integration, to create

$\boxed{GPD_2\text{(problem, soln.)} \to \text{Probability that this is the fastest solution to the problem.}}$

This last distribution is the Guiding distribution for our search for solutions to new problems.

I will now discuss Levin's universal search [redacted] procedure:

Suppose I'm in a Gambling house and there is a kind of lottery with a ~~two~~ single large prize. ~~A ticket cost #1 and I am allowed to chose any but~~ I am the only customer

Each lottery ticket has a certain probability of winning — which is printed on the ticket.
In the first ~~case~~ kind of lottery, all tickets cost #1. The best ticket to get would be one with max win probability. If it doesn't win choose next largest — and so on.

In the next case, each ticket costs a different amount of Money,
(p_i, m_i) are the probability, cost associated with the i^{th} ticket type.
The best choice to make is the ticket with maximum $\frac{p_i}{m_i}$. You get the maximum probability of winning, per dollar spent. If you continue to buy tickets in $\frac{p_i}{m_i}$ order you are [redacted] certain to ~~[illegible]~~ have least expected money spent before winning.

The Gambling house [redacted] can be transferred into a problem solving environment. Instead of money for a trial, we [redacted] spend TIME.
The ~~best~~ trial to choose is one with [redacted] maximum $\frac{p_i}{t_i}$.

Normally in trying to solve problems, one ~~does~~ may know p_i, but one doesn't know t_i. In this [redacted] case, a very good strategy is Levin's time share strategy — (which I think is likely to be the best possible.)
The way it works: You work on all trials simultaneously, but you work harder on trials with larger p_i values. The rate at which you work on trial, i is proportional to p_i. Suppose we use this strategy and after a while the j^{th} trial gives a solution after spending a total time t_j on it. Since other trials have amounts of time spent on them proportional to their p_is. The i^{th} trial will have $\frac{p_i}{p_j} \cdot t_j$ time spent on it.

4/3/01 IPSIA
Talk

.00:98.40: The total time spent on all trials will be $\sum \frac{p_i}{p_j} t_j = \frac{T_j}{p_j} \sum p_i$

$\sum p_i$ will in general be 1. So the total time needed to solve the problem this way is $\frac{t_j}{p_j}$.

So: this is Levin's time shared search and it finds its as fast as is possible for the kinds of problems in which it is used — i.e. INVersion problems

~~Another application for this search for~~

This technique can also be used for time limited optimization problems. In this case our trials are not programs that attempt to solve inversion problems: They are $\boxed{\text{optimization techniques}}$ each one takes as input, the problem description, $G(X)$ the function to be maximized, and the time available for solution.

As before we have a conditional probability distribution that looks at the problem description and assigns probabilities to various optimization techniques to be applied to that problem. ~~However since we know the time cost for each trial — (say it is T), the best strategy is to try the optimization techniques in p_i order. If~~

Factor of ≈ 71.13

61.35 All into PD.

~~59.3~~ 63.30 on OZ

The way we do the search: say over time limit T for some small τ we spend time $\tau \cdot p_i$ on the ith O.T. We keep track of the O.T. that has best G value and slowly increase τ. When $\tau p_i = T$ for any OT, we stop spending time on all the O.T.'s. When $\tau p_i = T$ for the best O.T., we stop. That is our best value. that O.T. when $\tau p_i = T$ for the best O.T.

Total time $\leq \sum \tau p_i = \tau \sum p_i = \tau$; $\tau = \frac{T}{p_i}$ so time to obtain best soln. in time T, is $\leq \frac{T}{p_i}$ so we have efficiency factor of $\frac{1}{p_i}$ — as we did for INV problems

4/4/01

Talk : [pp 92 – 100]

IDSIA Talk,

.00: : ~~Next~~ $\boxed{\text{review System}}$: Ideas of optimality in system.

① ON OPTIMALITY: Lsrch for ~~...~~ INV problems is ≈ OPTIMUM if All info used in srch is in P.D. ① cutoff ② Heuristics re reordering & technique & Lsrch ③ How get PD's via synthetic Data.

② No learning allowed during ~~...~~ competing srch routines

So we modify system so it ~~..~~ uses time learning featur. Main Search problem is problem of improving GPD

The ~~this~~ system would seem to be much better than a system that allows no learning featur. trials — I have no ideas on how close it could be to optimum

.14 ② On [redacted] Inductiveness, past [redacted] ~~re~~ source ~~...~~ Limited OZ problem: (85.00ff)

· While it may be true for early life of the system, it is not true in general.

If you have a very large mass of data, say information about the stock market, and you want to do prediction ~~...~~ for next day's price, and

.14 you have one hour to make the prediction; you will not be able to process all of the data — i.e. try to find short codes for it. Typically you will consider only the data that seems most relevant and try to describe

~~...~~ that data using ~~...~~ "Short Codes".

~~I have got to solve the trade-off problem between~~ the desirability of ~~using as much data as poss., and the necessary time limits on computation,~~

~~the fact that one can't ... good ... codes~~ find many good codes ~~in a very long data sequence, in a short time~~

There ~~is~~ are two conflicting goals in induction: ~~...~~

~~...~~ It is easier to find regularities in small data sets.

Using large data sets gives more accurate model probabilities, — but takes more time.

③ How system can transcend Lsrch if necessary: That cands can be srch prms doing learning during search.

④ How to get good set of OT's is extensionally set.

9/4/01 IDSIA

Things left out in talk:
" needed for next lecture.

— 1. /Details on how induction is done for BAG, Operator, sequential! What are the codes? ~~Part~~
What are they codes of, etc. ← GPD_1 is of this type

— 2. Some ways in which practical induction is actually done — particularly for Bag & Operator probs.

— 3. Details of update process — for OZ, for INV, for induction (Bag, Operator, Sequential).

• 4. How new (non-Lsrch) ptms are derived (Manton' Marcos/Jurg's ideas on this) — maybe as an Appendix. (How system transcends Lsrch)
Q — Are they really analogous? — do they actually work.?

• 5. ≡ That I made it clear how induction was a OZ problem

○ 6. Discussion of optimality of Lsrch! in what sense is it "optimal"? see 10
• for INV probs
~~○ for OZ probs (...)~~ see 102.04

• 7. Discussion of optimality of system as a whole: The "factor of 2" Question, the "no learning during srch" etc. see (13)

○ 8. Clearer statement of Diffy in Induction as a "Pure OZ problem" How size of corpus enters (do I have a soln?) see, e.g. 100.14 (But I want to work on this w. Weekly Ray.)

~○?9. How to start w. Good set of OTs — from "factor" them into a stoch lang, then extrapolate to set: This can be done (all or partly) by humans.

• I can do easily
○ I can do, but not quite success...
— needs a lot of work.

○ 10. perhaps discuss details of Lsrch for both INV & OZ problems ([redacted] Along w. optimality) ← see (6)

11. That the incomputability of ALP is not a valid criticism of it. Riss uses computable version of ALP & it has serious diffys.

12. Discuss "random" Lsrch. Have any papers been written using actual Lsrch?
.29 Also, other kinds of random srch. (N.B.) My Args about Random Srches only correct if there is 1 or no soln.

13. $^{\text{from (7)}}$ T. Q of Optimality if learning during Lsrch (modifn. of GPD during Lsrch) is allowed.

14. Math & Logical reasoning: Impt. in prob solving, yet not explicitly considered in the system

15. Train Seqs & how to write them "CJS" (maybe not use CJS — its only used a few times).

.36 16. Serious diffy! In converting from GPD_1 to GPD_2: we write $\int_{-\infty}^{+\infty} d\epsilon\ P_1'(\epsilon) \prod_{i=2}^{k} P_i(\epsilon)$ → (112.28)
Hvr. the $P_i(\epsilon)$ are usually highly correlated (§2.09) so this S is way off!

17. OSL: how most induction schemes (≈ MDL) have to be modified to accommodate it.

Q's Juergen was concerned w. assumption in which [redacted] GPD is obtained
That had a short code, but a very long execution time.
Main Q is how long does GPD take to establish [candy P's]