1

CONSTRAINED INFERENCE - ZENNA TAVARES

Inference is the process of deriving conclusions from premises assumed to be true. We can view inference as a generalisation of the evolution of a program. If we view our program as a dynamical system, which computes by transitioning from one state to another in discrete time. This kind of computation is a special case of inference, where we have complete knowledge of the program state at time t and we want - at least in principle, often we are only concerned with observing part of the program - complete knowledge of the program at time $t+1$. Inference allows us to ask more powerful quesitons about our program.

1.1 $\sigma - machines$

Envision a collection of autonomous machines, each collecting, transforming, and propagating beliefs about values. Each machine propagates independently, ignorant of the activity of others, and as a consequence must itself be prepared to recieve information from arbitrary sources.

Formally each of these machines - which henceforth we will call σ-machines, or simply as nodes for convience - compose a triple (Π, S_v, S_c, χ, T). The name χ of a machine is its unique identifier; it has no structure. Each value sockets $s_v \in S_v = (V)$ is a container of a value, they cannot be conditioned upon, and can only be propagated from, not to. Variable socket $s \in S = (B, private)$. Beliefs B is a relation on $setoftype(type) \times \mathbb{R}$. $private \in \{0, 1\}$ is boolean flags defining whether or not a socket may be accessed by machines other than this machine.

The policy set Π defines the operation of a node. It is immutable, invariant to time, and is what distinguishes two primitive nodes from another. It forms a sequence of statements of the form: given that I receive beliefs of a certain type at a certain socket, I will propagate beliefs of a certain type to a certain (other) socket. The policy has juristriction only over its own sockets, and non-private sockets of other machines. More precisely, a policy set is an ordered set of functions $\Pi = (\pi_0, .., \pi_j)$ where $\pi_j : \mathcal{P}(S_i) \to \mathcal{P}(S_l)$ where $\mathcal{P}(X)$ represents the power set of X. That is, π is a function mapping o or more sockets and associated data of that type to another buffer and associated data.

Machines communicate through propagation of beliefs from sockets within themselves to sockets of other machines. This is achieved by means of a directory socket, s_d. s_d contains relations on $S_r \times S_w \times$

$W \times C$, where S_r is the set of sockets within the machine's read-juristriction, S_w those sockets in its write-juristriction, C is the set of channels, and W is a set of real-valued weights. In words, when beliefs are incident on a particular socket in s_d, they are instantaneously routed to the corresponding socket in the relation, on a particular channel and with a particular strength. A directory is special in that it defines the propagation routing, and hence without it a machine may not propagate. But it is still an ordinary socket, and each relation is accompanied with a weight representing its uncertainty.

1.2 DYNAMICS, COMPUTATION AND DYNAMICS

The above description portrays sigma machines as computing some process over time. In contrast to common specifications of procedural systems, a program is not specified by initial conditions, but in general by a set of temporal propositional formulae which condition, or give evidence about the state of the program at different times. We then infer values of interest, at a times of interest, under the constraints given by the evidence (taking into account its reliability), and the belief revision rule occuring at every timestep.

$\sigma - machinesareorganisedintooneoftwokindsofgroup :< i > propagatory < /i >, or < i > propositionalensembles < /i >.Apropagatoryensembleisitself\sigma - machineinahigherlevel < i > encapsulatingensemble < /i >; precisely, itisabeliefb \in B$ in a socket of another $\sigma - machines.Itcontainssocketsoftypepropositionandquery.Eachpr machines.$

A propositional ensembles

Each proposition provides some evidence about machines in the propagatory ensemble, that is true within under some time qualifications, for instance $X = 5$ at t_0.

This relationship between $\sigma - machineandensemblesisclearlyrecursiveandinfinite; Thisinfintienessdoescauseproblems, needtoremedyit)Anensemblesigmamachineha$ S_i Sigma machines can propagate any machine provided the beliefs are type consistent. This includes higher level sigma machines, and it is this property that enables conditional execution and a deep reflective capacity.

There is a duality in the interpretation of an ensemble. On the one hand an ensemble represents the causal relationship between uncertain values. But these causal relationships are not implicit; they are defined only in terms of the computations it takes to transform one set of beliefs into another. So time When we get rid of all uncertainty, and consider all new information as true, then this model collapses to familiar notions of procedural computation, where time represents some imperative to execute some operation.

1.3 AN ELEMENTARY MODE

An infon space is a directed graph $G = (N, E)$ comprised of a set N of nodes, where $n \in N = \{0,1\}^*$ is a finite length binary string. Each node may act as a value, function, or both, in a manner determined by the set E of directed edges. Each edge $e \in E = (x, y, t)$ is directed from x to y, and of type $t \in \{argument, output\}$. In well formed spaces, nodes and edges always occur in quintuples of the form $\left(n_0, e_0^{argument}, n_1, e_1^{output}, n_2\right)$ where e^t denotes an edge of type t, $n_0 = tail(e^{argument})$, $n_1 = head(e^{argument}) = tail(e^{output})$ and $n_2 = head(e^{output})$. In words, arcs always appear in pairs, one of each type, and with the head node of an $argument$ edge also the tail node of an $output$ edge.

The naming of edge types hints towards the semantics of this graph structure; n_1 applies itself as function to the value n_0, the output of which replaces the current value of n_2. In an elementary infon space, a node may have only one incident incoming edge, rendering all functions unary. Function application in this respect is explained easiest by example. Consider the simplest possible network composed of a single quintuple $G = (N = \{n_0, n_1, n_2\}, E = \left\{e_0^{argument}, e_1^{output}\right\})$, with edge connectivity as described in the previous paragraph, and where $n_0 = n_2 = (0)$. Denoting the set $F_{a,b}$ as all functions mapping binary strings a to a binary strings b, $|F| = 2^{|b|2^{|a|}}$. A node acting as a function must be sufficiently long (and not longer) to uniquely encode all functions from its arguments to its output, $|n_y| = \log_2(|F_{x,z}|) = \log_2\left(2^{2^y b}\right) = b2^a$. In this example $|n_1| = 2$, and let us define it as (01).

A network evolves over discrete time $t = t_1, .., t_n$. Actually decoding a function from the binary string is then done by simply looking up a substring of the function string, which is of length equal to the output string, and at a position indexed by the natural number encoded in the argument. In this example we are looking for a substring of n_1 of length $|n_2| = 1$ and at position encoded by $(0) = 0$. Indexing a binary string by convention from right to left, we can conclude thatn_2 will become 1.

But as stated in our desiderata we intend to go further than just forward evolution. We want to ask questions such as, given n_x has binary value b_y at time t, what value could n_y have been at $t-1$ to have caused this? More generally we wish to state a set of premises: statements about values of particular nodes at particular times, and infer, when possible, the values of other nodes conditioned on these premises being true. We can achieve this by propagating constraints.

Here we have three limited incarnations of our desiderata. We have an implicit fault tolerance, it is We have a form of inference through constraint propagation And we have a form of reflection, afforded by the uniform representation of values and functions

A uniform representation of values and functions, combined with arbitary toplogy by use of a graph, enables this model to generalise a number of others. Cellular automata are similar idealisations of physical systems, composed of a regular uniform lattice of discrete variables evolving over discrete time. The value of the variable at one cell at t_i is a (global) function of the values of variables at cells in its neighbourhood in the previous step t_{i-1}. Boolean networks, first proposed as a mathematical model of genetic networks and later applied to a diversity of biological systems, generalise cellular automata. The graphical formulation allows arbitrary connectivity, as the lattice like structure of celluar automata a special case. However infon spaces generalise further by permitting arbitrary network topology, provided the network is well formed. Arbitrary topology is only possible if arbitrary update functions are allowed, which is made possible by embedding the update functions in the graph itself through use of the uniform representation of nodes and edges. This property, enables two further generalisations. If function specifications are embedded in the graph and are treated no differently to any other piece of data, it implies that functions can vary over time.

1.3.1 *Generalisations*

We can will make three generalisations to the model, each of these generalisations could be in isolation. The first extends infons to strands. The second makes infons probabilistic The third relaxes the unary requirement.

- Sigma machines - What do we want as the output of an inference? A constraint equation to be solved? The equivalent of the equation in ensemble form? A sample of values? An enumeration of values? The posterior probability distribution? – IT seems like the constraints in some form may be useful, a sample satisfying the constraints would be useful, a sample satisfying the constraints and from the posterior distribution would be even more useful, the entire posterior would be the most useful but is likely intractable in all but the simplest of cases.

- Should nodes explicity store probability distributions or not? The most likely? – - Do we need to represent prior information, and if so how? - Should the directory and label be immutable or mutable, should it be just another socket? - Do we need to differentiate between values and variables in some form? - How to handle channels

The most appealing and most general approach seems to be to suggest that an ensemble is just a set of named nodes. We compute then by conditioning temporally, and querying. The problem is that we need some representatiion of a condition. The obvious representation is to use the same representation as our program, i.e. ensembles. But then we the same problem. The solution must be eithe rdont use the

same representation, or use the same representation but with some constraints or something., or allow unconditioned structure. But if you allow unconditioned structure, what does it mean? If I alllow it in the condition, and hence allow it in the actual ensemble, then what does it mean in the actual ensemble.

One approach would be to say that WITHIN a node, we have temporally qualified values, and then when we condition we just probide more evidence. Then what it means is abundantly clear.

i.e. we don't say x = y + z. We say x = y + z @ t=0. In other words we are distinguishing between initial constraints and further constraints. This makes it very possible to give further constraints which do not coincide with the initial constraints. So what is the point of further constraints, well there are none in principle, except for code reus

Where is this evidence stored? HOw can a node compuite over time and store evidence about its values? Well one way to look at is is this evidence distribution extended through time. This is what they store. Another suggestion is that it is stored in socket of the meta node. But this seems to be just deferring the problem

There is a difference between an ensenble and evidence. An ensemble is a set of machines and set of evidence nodes. Each evidence node must contain one temporal formulae ss

BIBLIOGRAPHY

[1] N Goodman, V Mansinghka, D Roy, K Bonawitz *Church: a language for generative models* 2012.

[2] RA Paige, O Danvy *Automatic program development: A tribute to Robert Paige* 2008:

[3] CE Freer, DM Roy, JB Tenenbaum *Towards common-sense reasoning via conditional simulation: legacies of Turing in Artificial Intelligence* 2012 Arxiv:

[4] A Turing *On computable numbers, with an application to the Entscheidungsproblem* 1936: Proceedings of the London mathematical society.