0.1 ABSTRACTION

The challenges of inductive inference - extracting generalisations from observations - has extended from the philosophical to practical. Instead of how can we justifiably select a single hypothesis from a possible infinite number of alternatives, we question how we could plausibly do it with the astonishing efficiency and correctness humans possess. Although the rules of conditional probability state how to update our beliefs in hypotheses conditioned on evidence, it tells us only declaratively, and leaves us with no guidance as to how hypotheses should be constructed in the first place. In anticipation of the distinction emphasised in this essay, we describe conditional probability and in particular Bayes' Rule as the *logic* of a program, while the the *control* or *procedure* remains a topic of continued research.

0.2 BROADER IMPACT

0.3 CORE AIM 1

We The former o

So many good ideas - This is why I am here THe connective tissue - We want a complete account of inductive inference - The declarative account has been formalised within the laws of probability, in particular bayes rule - Perhaps within or adjunct to this declarative component is the representational one. - There are many things a representation for the kinds of theories we are able to construct should satisfy, in particular compositionality - As in all algorithmic problems, the representation and procedure are mutually constraining, the search/-construction becomes more difficult in complex domains? – This is because we attempt to apply most methods attempt to exploit some local syntatic transformations. In the representations we are considering, the semantics of the structure is not robust to local syntatic perturbations. This property holds in other domains - The solution, at least in general is clear, either seek representation for which this propery does hold, or avoid search based on syntatic analysis; prefer instead semantic exploration. – Fortunately there is a rich and diverse field of the semantic analysis of programs

- In Tomers work we're trying to find a theory explaning some data – I would suggest its more like we're trying to find a theory about soem abstraction of the data – For instance hearing the rain could allow us to ask, why is it raining at such a time, why is it raining so heavily - What is a theory for such abstractions? - Tomer posits theories as distinct from models, theories are logical relations/programs on abstract properties, and a model is basically a mpaping from these predicates to abstract properties of the world. – So that if we're dealing with magnets we can form the theory that there is some property

that two objects must have for their interaction to be true, separately or simultaneously from determining what that property might be. We know that property to be magnetism, but it could for instance be that the weight is greater than x, or colour or anything. This seems reasonable a child might first see if the colour is the determining factor. Something like it being magnetic is not computable from our observations. There's a latent cause. – MY innovation: These properties can be thought as abstraction

: We can think of this as an inference problem in an abstract domain – We are trying to infer the abstract domain, the concretization function – and the Theory

As Schultz points current theories are incomplete, we generate things which are highly constrained by rich information sources. We are faced again then with a how what question What - Do we have a declarative statement of the problem, how is this represented How - How can we perform this efficiently

I suggest these constrainted sampling from a probalistic program These constraints are predicates on constraints from some distribution. Represented as programs How again returns semantic analysis of these programs, and abstract interpretation

1

CONSTRAINED INFERENCE - ZENNA TAVARES

- Inductive inference is process of deriving causes of observed phenomena - This necessitates a representation for causes - Computer program as theory - Induction in the computer program as theory domain necessitates a different approach to common methods (WHY?)

Is there a differene between inductive inference and just inference?

We address the problem of tractably drawing exact samples from conditional probability distributions. In particular we propose a formalism for constructively incorporating constraints into generative models for probabilstic inference. The presented approach builds upon formalisms of inference as conditional execution of a program, and analysis of such programs with respect to their formal semantics.

Main points Inference algorithms for probabilstic program Constructing hypotheses Framing Problem

Goals: Provide an algorithmic account and efficient implementation of inference in highly structured probabilstic models

Goals: Frame the problem of theory induction as one of program

Goal three: Abstract Interpration

1.1 INTRODUCTION

Probabilstic inference has - NOTE:DID NOT ESTABLISH SELF DID IT - established itself as a means of reasoning in domains subject to uncertainty. Although the rules of conditional probability state how to update our beliefs in hypotheses conditioned on evidence, it tells us only declaratively, and leaves us with no guidance as to how hypotheses should be constructed in the first place. In anticipation of the distinction emphasised in this essay, we describe conditional probability and in particular Bayes' Rule as the *logic* of a program, while the the *control* or *procedure* remains a topic of continued research.

Inference typically refers to finding the expectation of some function with respect to a probability distribution, computing the posterior distribution, or drawing from it. Numerous approaches exist, but our focus is on Monte Carlo methods whose strengths lie in their applicability to high dimensional distributions, where direct sampling becomes intractable. Inference by means of random perturbations to hypotheses is often inefficient however; we would prefer to not waste resources proposing candidate hypotheses which violate sensible constraints. In other words, we would like to make good guesses.

The objective of this study is to explore conceptual and practical methods of making good guesses by incorporating constraints *con-*

structively into generative models. We frame construction in constrast to testing; to construct is (ideally) to not test. We build upon recent interpretations of probabilstic inference in computational terms, in particular the QUERY operation, which performs universal inference in the probabilstic programming language Church [1]. We suggest a new constructive perspective CONSTRAIN, which given a generative model manifested as a stochastic *prior program* P, and a conditioning predicate C aims to construct a new program P^* which samples only values which adhere to our condition.

Universal inference [1] is analagous to notions of universality in computation. Informally it refers to the ability to perform conditional simulation of any computable probabilstic generative process.

The feasibility of this aim is vulnerable to skepticism; how can we construct only constrained proposals without testing to see if they satisfy our constraints? It is unlikely this objective can be achieved in full generality, in some cases we have no alternative than to generate and test. Furthermore we shall see that relaxations of the above maxim may lead to more pragmatic implementation strategies; to construct is to test less perhaps, or to test only partially and incrementally. Precisely what distuiginshes one constraint from another in terms of the extent to which it can be implemented constructively, is a topic of great interest and speculated on in our conclusion. Yet, we can motivate the plausibilty of constructive inference by appealling to common experience: consider the ease with one can compose a paragraph that rhymes, generate a polynomial expression, or draw an acyclic graph. Each of these examples entails a logical constraint which we satisfy without hinderance. While it is risky to draw conclusions about the computational hardness of a problem based on our subjective difficulty when solving it, each of the above examples has a known efficient algorithm. We conjecture the automatic discovery of algorithms as a means towards conditional sampling, is necessary for any general inference procedure approaching optimal efficiency.

First we will summarise QUERY, a computational theory of probabilstic inference. Then we formalise the key concept of this paper, CONSTRAIN, in terms of QUERY. Two approaches to implementing CONSTRAIN are then described in detail.

1.1.1 *Query*

QUERY formalises probabilistic inference as a *program* in a corresponding model of computation. Developed first as a primitive function in the probabilstic programming language Church [1], QUERY has been described in terms of probabilstic generalisations of the λ-calculus and Turing machine, both classical models of computation.

In its original formulation, QUERY is a higher order function which accepts as input a stochastic expression to be evaluated and a set of predicate conditions. In lisp (clojure) notation:

Notation: defn defines a function. The following term is the function name and terms inside square brackets [] are argument names. Łetís used to assign names to values, conceptually similar to assignment of values to variables in imperative languages

```
(defn query [exp pred]          ; Define function
  (let ((val (eval exp))        ; sample from model, call it val
```

```
(if (pred val)              ; If val satisfies conditions
    val                     ; Then.. return it
    (query exp pred)))))    ; Otherwise try again
```

In [3] a formulation in terms of a probalistic turing machine (PTM) is given. A Turing machine [4] is a mathematical abstraction of a machine, which may read, write and seek access on a finite collection of infinitely long binary tapes. Prior to execution, its input is loaded onto one or more of its tapes, and the output is the content of its tapes after the machine halts. A probabilistic Turing machine (PTM) is a Turing machine equipped with a tape consisting of a sequence of independent random bits, which is accessible to the Turing machine as a read only randomness source.

QUERY is a PTM which takes two inputs, a prior program P and a conditioning predicate C. Both P and C are themselves encodingings of PTMs that take no input. QUERY generates a sample from P. Then, if C is satisfied this sample is outputted, otherwise the process is repeated.

It should be of little surprise that both these formulations are equivalent, as they both perfom the function of drawing samples from a prior distribution conditioned on C, using rejection. While rejection sampling provides an simple and intuitive understanding of the meaning of QUERY, it is of course grossly inefficient for the majority of non-trivial problems.. Much research in inference is in looking for tractable approximations and alternatives.

1.1.2 *Constrain*

The semantics of CONSTRAIN can be readily understood in terms of QUERY. Expanding on the lisp definition of QUERY given above, CONSTRAIN is a higher order function expecting a prior program P, and taking *constraint* C^*. Uncertain evidence, or equivalently soft constraints are not considered here. Hence a condition C and constraint C^* can be used almost interchangeably, but are differentiated in terms of hardness; constraints are logical and must be true. CONSTRAIN simply returns a function of no arguments which is QUERY with all its arguments evaluated.

```
(defn constrain [exp pred]        ; Define a function
  (fn [] (query expr pred)))      ; Return a 0-ary function
```

While QUERY returns a sample given a model and condition, CONSTRAIN returns a function of no arguments which calls QUERY. In deterministic programs, a function with all its arguments evaluated is simply a value which evaluates to itself, and little is typically gained from taking a functional perspective. In stochastic programs however the output of CONSTRAIN implictly defines a conditional distribution. This alone differs little from QUERY, as we have only described

the semantics. We differentiate with the objective that that C^* is conditioned on constructively; we wish to refrain from applying it as a predicate to fully formed samples, i.e., testing, and instead exploit its semantics to find a more efficient means. The rest of this study is devoted into proposed means of doing this.

2

CONSTRAINED GENERATIVE MODELS

First consider a simplified instance of the problem: our generative model defines a distribution over linear transformations of a random vector of uniformly distributed real values, where the number of variables and its parameters to each random variable (both lower and upper bound) are fixed and known ahead of time. For example:

A constraint involves the conjunction and disjunction of any number of linear equalities and inequalities on these samples. Both the generative model and the constraint are represented as programs in a limited functional language, which importantly is side-effect free and lacks general recursion.

Both a logical and geometric perspective brings some clarity; our constraint program C implicitly defines a logical formula composed of the conjunction and disjunction of a fixed number of literals, each literal representing an inequality of the form $Ax \leqslant b$ (note we can always convert an equality to two inequalities, as well as convert a greater than relation to a less than through negation). For instance the expression:

```
(if (> x1 10)                  ; If val satisfies conditions
    true                       ; then our constraint is satisfied
    (or (> x2 10) (> x1 2)))   ; Otherwise if x2>10 or x1>2
```

Defines a logical formula of the form:

$$x_1 > 10 \vee (x_1 \leqslant 10 \wedge x_2 > 10) \vee (x_1 \leqslant 10 \wedge x_1 > 2) \tag{1}$$

The extraction of this logical expression from the program is important, and the first step of the proposed method. The formula is in disjunctive normal form: one of an infinite number of equivalent alternatives but unique in that it is the disjunction of a minimal number of clauses, where each clause is the conjunction of a number of literals. Each clause can be thought of as a set of local conditions which when all are simultaneously true, will cause the constraint to be satisfied. Geometrically, each literal is a linear inequality defininig a half-space which splits $\mathbb{R}^n$ into two, and designates one side of the split as feasible and the other infeasible. The entire clause is the intersection of a number of these half-spaces and is thus a convex polyhedron. The entire formulae in this form can then be viewed as dividing $\mathbb{R}^n$ into a number of possibly overlapping convex polyhedrons, within which our sample is permitted to lie.

Both the generative model G, and condition C define constraints on samples. Since only linear transformations of random variables

are permitted within G, it defines a d dimensional interval - a hyperrectangle - within the sample space $\mathbb{R}^d$. P carves a set of convex polyhedra within the this hyperrectangle.

The problem of constrained inference in this setting is then reduced to discovering these convex polyhedra, and efficiently sampling from them without deviating from.

2.1 A MOTIVATING EXAMPLE: MOTION PLANNING

Planning covers a number of problems critical to robotics, computer graphics and artificial intelligence. Here we consider one instance of the problem, which is to find a path between two points or regions within a two or three dimensional space, which avoids collision with any obstacles. While numerous approaches exist, sampling based methods such as rapidly exploring random trees (RRT) and probabilistic road maps (PRM) have become established as efficient and practical methods. Broadly, they rely on efficient collision detection to find feasible points followed by growth of potential paths between these points. However, performance can degrade rapidly in complex environments with narrow passages, where the probability of sampling a point within a the passage is relatively small.

One application of constrained inference is a straightforward generalisation of the problem statement: from all possible paths within a region, sample a . More formally we seek to construct a path $\{x_0, y_0, x_1, y_1, .., x_n, y_n\}$. Obstacles can be defined as polyhedra, but for simplicity and without loss of generality we will assume them to be isothetic (axis aligned) rectangles or boxes, and represented in h-space.

The obstacle avoidance constraint can be stated precisely by asserting that no edge $\{x_i, y_i, x_{i+1}, y_{i+1}$ may cross the boundary of an obstacle. This is a non-linear problem, which can be seen intuitively by observing that the constraints on any point depend on the previous point.

A similar linear problem is to enforce that each vertex does not fall within an obstacle. While necessary, this alone is insufficient since the vertices of an edge may not collide with an obstacle but the path between them may cross the obstacle.

I suggest this can be overcome by incorporating two extra details. First we ensure that the maximum edge length is within some radius. This is a linear problem. Then we increase the number of points.

Effectively we have found a linear equivalent of a nonlinear problem, with the trade off that we increase the dimensionality of the problem.

And in first order logic logical form

...

In a functional programming language, we get: Note one power of using a programming languages, our program is now of constant length

Hence the trade-off is severe. For every new point we add we will gain two new dimensions, and described in more detail in the following paragraph, you'll get b more polytopes.

Niavely scales very badly. But we can ask many questions, are all these polytopes important, perhaps some contain others. Perhaps some while logically true are geometrically impossible. Perhaps we can have a coarser representation of these convex polytopes. Perhaps there is structure in the way in which they are growing

{Algorithms

2.2 PROGRAM ANALYSIS

Program analysis if or and functions primitive recursion symbolic exection

2.3 SAMPLING

Polytope Sampling Abstraction Our primary results are methods in ensuring the correctness , we analyse these points in detail after a first motivating example.

We then analyse P to extract further constraints on the samples. These constraints will define regions of convex polyhedra We sample from these regions

Theorem: Polytope sampling is correct.

Theorem: we can sample from disjoint polytopes by first selecting a box and then sampling within that box.

Difficulties arise primarily from two sources, first is avoiding bias and second is handling complexity.

2.4 HANDLING COMPLEXITY

Problem Extraction of convex polyhedra from program - number of convex polyhedra typically exponential in number of variables - We need tocounter act this, first pass: – Subsumption/redundancy, some expressions are redundant – infeasible, some expressions when in a conjunction become infeasible — These sets of expressions appear in sets of regularity

2.4.1 *Inconsistency*

Logical inconsistency denotes infeasibility of the conjunction. We can determine this efficiently using linear programming.

2.4.2 *Subsumption*

Handling this logically Geometrically

2.5 ABSTRACT INTERPRETATION

- Previous methods may help in removing redundant clauses but number of polytopes may and likely will still be exponential - Joining means placing an abstraction around some terms

2.5.1 *Exploiting Regularities*

- In many problems the convex polyhedra are positioned with some regularity - Can we exploit this - parameterised boxes

2.6 OVERLAPPING POLYHEDRA

Covering of these convex polyhedra Accounting for overlapping convex polyhedra - Overlapping polyhedra cause 2 problems – 1. Bias, if not accounted for – 2. Inefficiencies when covering

2.6.1 *Removing overlap*

First solution is to find intersection between two convex polyhedra. Problem 1. This would require conversion to vertex representation which can be exponential 2. Aand Bwould not necessarily be convex any more.

Hence we are more interesting in dissecting the abstraction

Dissecting exactly The problem then can be stated as given a set of overlapping intervals in the domain, we wish to disect these into a set of non-overlaping intervals. It's not clear how hard this problem is, and it is little studied in higher than two or three dimensions.

It should be clear that must be at least as hard as Klee's problem, which is to fine the volume of the union of a set of boxes.

Naive approach is to form a grid of every bound for every box in every dimension. This results in aworse case peroramnce, which is exponential in d.

There are two areas in which we could relax this problem. First we can permit some overlap, in accodinace with preceeding paragraph. If an exact solution is infeasible we would like to be able to tune how much overlap, such that it can be managed. 2. We can tolerate some overextension into free space because 1) Such space may be removed in our divisive stage 2) anypoint landingin free space will be a rejected sample. Undesirable, but not changing the distribution

Our output sensitive growth algorithm.

Problem 1: In order to be able to disect boxes we ned to know which boxes are overlapping (Do we?), how ca we do this efficiently.

Problem 2: How can we make the expansion of the algorithm constant time

2.6.2 *Permitting overlap*

A simple way to permit overlap is to assign every element of the set to a single abstraction. Numerous reasonable ways exist to do this, one obious way is to enforce a total order on the abstraction, such that any point is assigned to the most dominant abstraction which encapulates that point

If a sampled point as not selected by first selecting its dominant abstraction it is discard. The problem with this is that we are wasting points, perhaps a large percentalge of points. What we can do instead is have a cache for each abstraction

PROBLEMTOSOLVE: Does This method work with abstractions as it does with polytopes.

PROBLEMTOSOLVE: How can we find a good assignment

2.7 NOTES

1. Perhaps I need to abstract away the covering problem from the other problems 2. The

2.8 RESULTS

2.9 NON-LINEAR CASE

2.10 CONCLUSION

Here the probabilstic inference framework QUERY was given a constructive perspective with CONSTRAIN. We proposed two methods for implementing CONSTRAIN based on program transformations and symbolic evaluation. We suspect there will be many difficulties with such an implementation, but preliminary evidence suggests even crude approaches to generating more constrained proposals can have dramatic effects on inference performance.

Discussion has been limited to a single constraint C^*, since multiple constraints can be found conjoining each individual constraint together. However, there may be important practical implications in both of the above methods for the order of the conjunction, e.g $A \wedge B$ vs $B \wedge A$. There may for instance be a constraint, only visible with a semantic understanding of the programs which represent A and B, such that $A \implies B$. Moreover, A and B may have an internal struc-

ture such that there is a more efficient compiliaton of the two together than just considering each in order.

The central idea of this paper, constructivism, is still open to interpretation. How formally can we differentiate between constructive and non-constructive inference. An appealing possibility is to frame constructivism as the synthesis of an optimal algorithm to sample from the conditional distribution. The cause of variation in the difficulty in which constraints can be adhered to constructively can then be framed in this light; a constraint which is difficult to implement constructively is one where there does not exist (or it is difficult to find) an algorithm which implements it efficiently.

But optimality must be defined with respect to some criterion, and there exist many. An asymtotically optimal algorithm for instance is one which for large inputs performs at worst a constant factor worse than the best possible algorithm. But constant factors can be important, and the worst case is not always indicative of real world performance. Hence, often times even if such an algorithm is known, it is not used regularly.

In spite of these issues the synthesis perspective on inference is appealing, and seeks to appeal to the intuition that whether explicitly or otherwise, almost all approaches to probabilstic inference are implemented on computers as probabilistic programs. The synergy between the fields of program semantics and probability is likely to yield continued insight.

2.11 ABSTRACT INTERPRETATION

Abstract program denotes computations in some universe of objects. Abstract interpration of programs consists in using that denotation to describe computations in another universe of abstract objects, so that the results of the abstract execution give some information on the actual computations.

Abstract interpretation is typically used to prove numerical properties of teh program. Requirement suc as "all the array indexes are within the correct bounds, " "division by zero is impossible" to complex loop inariants. Moreover, numerical properties may help other kind of analyses, such as termination analyses, timing analyses, shape analyses, stiring cleanness analyses, and so on.

has been employed as a program analysis technique to answe

Let's say I have a list of objects, I might only be interested in the size of the list. This is an abstraction of the list

2.12 NOTES

Theory as program as theory sources Ulman 2012 - Children seem to go form fragments of observed data to rich knowledge of the world

- Children learning much like a kind of science - Children organise their knowledge into intuitive theories, abstract coherent frameworks that guide inference and learning within particular domains, allowing them togerenalise eidence toneew examples make prediction and plan effective interventiosn on te owrld. - Two Questions - What and how, what is how this – What (declarative component): which representationn can capture the form and content? – How (algorithmic component): maximise posterior probability

Framework Theory T set hypothesis space and prors for models Data determines model likelihood Bayes rule combines these facts universal theory is a grammar defining prior over spae of theories A theory is a horn clause which is conjunction of terms implies something, where each term is a predicate expressing an attribute or relation on entities in the domain, e..g f(X), S(X,Y)

Two predicate types: core/surface e.g. interacts(X,Y) <- f(X) and g(X)

In the magnetism case there are a numr of things First you need to figure that there is some abstract property of these objects, so it is like constructing the abstract domain and abstract map,

And then it is finding a theory which generates a theory constructed on these abstrations

- In constructing a theory, the learner introduces abstract predicates via new laws, or new roles in existing laws, and thereby essentially creates concepts. - Predicates may be typed

- Theory prior is probabilstic context-free Horn clause grammar – Generates possible horn clauses – Templates implemented as terms in the grammar – Templates can represent notions such as transitivity

– Model likelihood comes from assumption that we are observing randomly sampled (with replacement) true facts

There are many stories here - Theory induction as hierarchical probabilstic inference. WHere does the hierarchy come in? Read the technical overview. Then there is probabilstic inference in structured models ala Church. What's the difficultference and what's the connection. Is all Probablistic programming inductive inference? What divides inference from inductive inference.

- Schultz - Finding new facts – Child gen hypothesis about how the world works, and that the childs actions are guided towards evaluating these hypotheses - Principles of induction have implication for, but do not directly address problems equally critical to learning: search, exploration - We routinely generate new ideas without having access to new data

Me- It's not out data we are trying to explain, it's an abstraction of our data. - Thinking of new ideas goes beyond a search problem. Why? – How to explore? - Sometimes however, no single action will support information – Models by gopnik tenenbaum etc explain how to select among competing hypotheses, not how to construct these

in the first place. - EFFICIENTLY SAMPLED HYPOTHSES IS NOT THE SAME AS CONSTRUCTING THEM ME – isn't it? - However with only minimal constraints on simplicity, grammaticality, and previously productive templates, changes to hypotheses generated by random variation seems at best inefficient. - More importantly our minds seem to have access to rich sources of information that could better constrain the process of hypothesis generation that current approaches do not exploit. - error maps

- One possibility is that hypothesis generation is constrained not only by our abstract error maps of particular problems, but by our abstract representations of criteria for good explanations in general.

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.