Making Fast Informative Queries with Learned Propagations

by

Yewen Pu

B.A., University of California Berkeley (2012)
M.S., Massachusetts Institute of Technology (2015)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
September 2019

Author .....
Department of Electrical Engineering and Computer Science
August 30, 2019

Certified by .....
Armando Solar-Lezama
Associate Professor of Computer Science and Engineering
Thesis Supervisor

Certified by .....
Leslie Pack Kaelbling
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Making Fast Informative Queries with Learned Propagations

by

Yewen Pu

Submitted to the Department of Electrical Engineering and Computer Science
on August 30, 2019, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

In an informative querying problem, one achieves a certain objective by issuing a series of queries to an oracle and receives a series of observations in return. It is a challenging task because the queries need to account for the uncertainties of the oracle, while being informative to the objective at hand. While successful algorithms have been developed for a range of querying tasks, these algorithms can be slow to compute and in some cases, intractable. A common Achilles's heel of these prior works is their reliance on the computation over the space of oracle functions itself during inference time. As a result, when the space of oracle functions becomes complex, these approaches become computationally infeasible.

In this thesis, we explore an alternative approach to informative query selection. Rather than computing over the space of oracle functions, we learn a propagation function that, given a set of past observations, predicts future queries' outcomes directly. We show that by leveraging the propagation function, one can perform a range of informative querying tasks that were previously intractable. To this end, we prescribe a general method of informative querying with learned propagation: In meta-learning time, a propagation function is trained to learn the relationships between observations, and at inference time, a task specific acquisition function is constructed to leverage the propagation in making informative queries.

Thesis Supervisor: Armando Solar-Lezama
Title: Associate Professor of Computer Science and Engineering

Thesis Supervisor: Leslie Pack Kaelbling
Title: Professor of Computer Science and Engineering

Acknowledgments

To You :

Thank you, the reader, for being here. This thesis is dedicated to you.

Personal Thanks :

Thanks to my co-authors and collaborators whom have shown me what it looks like to struggle against difficult problems and sharing their unique and philosophical ways of framing and tackling these problems. I will wield these wisdoms responsibly.

Thanks to Armando for being very patient with me of 4 years not publishing anything (LOL) and still supported me. Without this support I would have never get to do the research that I believed in, and this thesis will not be possible.

Thanks to Leslie for taking me in when I was a clueless student and teaching me the important lesson of description over prescription, i.e. "You cannot solve a problem that you have not first formally defined".

Thanks to Josh Tenenbaum for being on my thesis committee and convincing me that $humans > computers$ and showing me how to entertain an audience.

Thanks to my friends who gave me their valuable times, catch phrases, and memes. Graduate school would have been very lonely otherwise.

Thanks to twitch chat, POGGERS.

Thanks to my parents, Xi Wen and Hongtu Pu for being patient with my extended years of education and not making actual money (haha).

Thanks to Meng Sun, for showing me how to make strong points, endless critiques that helped me grow and mature, and showing me the meaning of a e s t h e t i c. You are truly my best friend.

Contents

List of Figures

List of Tables

Chapter 1

20 Questions

When people ask me what I do for research, I tend to start with "Have you heard the game called 20 questions?". Indeed, it is a good idea to start this thesis by considering the game of 20 Questions. Doing so, we can intuitively understand the underlying principles of this thesis work: making fast and informative queries.

The Rules of 20 Questions

The rule of 20 Questions is simple, I think of a secret object (for instance, a turtle), and you have to guess its identity by asking a series of yes/no questions (for instance, does it fly?) to which I can only answer yes or no. The goal is for you to correctly guess the secret object with as few questions as possible, up to 20 questions. Despite the enormous number of objects one can think of, when played against a real person, they usually can divine the hidden object with a series of well-chosen questions, with an average game-length of about few minutes. Let's go over a small game.

Table 1.1: I first generated the tables then screenshoted them into figures

	swim	climb	fly	hair	shell	>4 legs	slow
armadillo				x	x		x
cricket		x				x	
dragonfly		x	x			x	
duck	x		x	x			
earth worm							x
jellyfish	x					x	x
monkey		x		x			
octopus	x	x				x	
seal	x			x			
sloth		x		x			x
snake	x	x					
turtle	x				x		x

Figure 1-1: Table of animals and attributes, 'x' means I answer yes, otherwise no

A Small Game of 20 Questions

Rather than thinking of an arbitrary secret object, I am going to limit myself to the set of following animals: octopus, duck, seal, snake, turtle, jellyfish, sloth, monkey, armadillo, earth worm, dragonfly, cricket. And, instead of allowing you ask any arbitrary questions about these animals, I am limiting myself to only answering a very small number of specific questions, with their response recorded in Figure 1-1

Given this particular set of animals and their attributes, what would be the best strategy for you to figure out which animal I have in mind? Specifically, since every game must have a first question, what would be the best *first* question to ask? It turns out that the globally optimal first question is actually quite difficult [14], but a good strategy in practice is to divide the set of possible objects in half with every question [15, 4] as much as possible. With that in mind, the first question you should ask is "does it swim?" Because precisely half the animals listed here can swim, this first question cut the set of possible animals in half, irrespective of my answer of yes or no. Continuing with this line of logic, you can arrive at the following decision-tree 1-2. To play the game thus amounts to following the branches of the decision tree, going left whenever I answer yes, and going right whenever I answer no. As you can see, one can arrive at the correct animal with at most 4 questions.

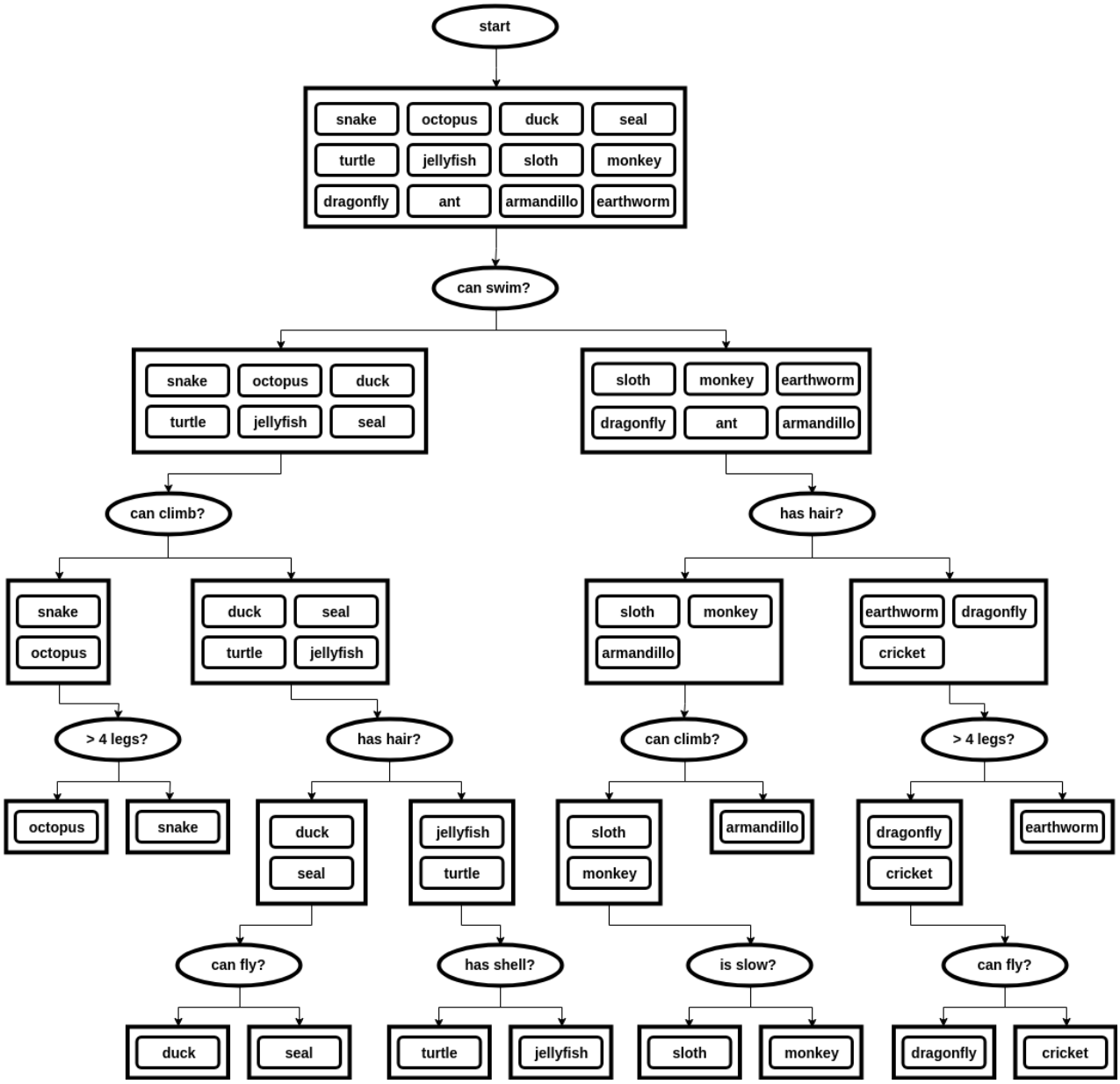

Figure 1-2: A Decision Tree on how the short 20 Question game can be played. Go left on every decision node (circle) if the answer is yes, otherwise go right.

A Naive Algorithm For Playing 20 Questions

The game-tree given in Figure 1-2 was created by yours truly painstakingly by hand. Could we automate the game-playing process with an algorithm? More precisely, given a set of past question-answer pairs (for instance, $\{(swim?, yes), (climb?, no)\}$), the algorithm must decide what is the next-best question to ask, from the set of remaining questions (for instance, $\{> 4\ legs, hair, fly, shell, slow\}$). Hopefully you already have in mind a working algorithm, I will just confirm it quickly:

Generalised Binary Search A naive solution to the problem of 20 questions is generalised binary search [15]. This algorithm works by evenly dividing up the set of remaining animals. Let the set of remaining animals be D. Then, for every question q, we check if this question most evenly split D in half, by counting the number of animals which q would result in a 'yes', and subtract it with the number of animals which would result in a 'no'. When the difference is small, the question q is good.

While this is a perfectly valid algorithm, it becomes inefficient if the set of animals D becomes large. As the algorithm needs to *count* the number of 'yes' elements from the set, the run-time of this algorithm will grow linearly with the size of D. While this may not be a problem for this particular game with only 12 animals, the problems of interest in this thesis often contain combinatorially many items (for instance, 10^{23}), which renders this naive algorithm intractable.

Making Fast Queries in 20 Questions by Propagation

One salient property of the aforementioned naive algorithm is that the time it spends thinking up the next query is proportional to the number of remaining elements in D. In particular, it would spend the most time asking the first question, and gradually speed up as the number of remaining elements in D diminishes. In contrast, human rarely spend any time on the first question: "does it swim?" is intuitively a very good first question, and we barely spend any time coming up with it. Because we have good *prior* knowledge that, of all the well-known animals, roughly half can swim, we do not have to explicitly count all the animals that could swim, unlike the algorithm.

Our prior knowledge also aids us in querying the right questions beyond the first one. Consider a particular point during a 20 Questions game where it is already established that the animal can breath in water, and your friend Julio asks "Does it have feathers?". You would probably be upset, because clearly an animal that can breath in water cannot be a bird, which is the only kind of animal that has feathers. A human player would ask question they are *most uncertain* about, one that cannot be readily deduced by previous answers. Most importantly, this reasoning process can happen *without* explicitly enumerating over all the possible animals.

Querying with Propagation Rather than reasoning over the set of animals via enumeration, the prior knowledge of different kinds of animals and the interactions of their attributes allows the human player to *propagate* information of *past* question-answer pairs to outcomes of *future* questions *directly*. Thus, if one is to emulate how human is able to make fast informative queries, it may be a good idea to build models that can directly propagate past observations to future queries.

In conclusion, the ability to propagate past observations to future observations plays an important role in making fast, informative queries. This ability stems from prior knowledge on how observations interacts with each other. This thesis is an attempt of realizing this intuition, by first learning a propagation function on how observations interact, then using the propagation to make informative queries.

Chapter 2

Introduction

This thesis addresses the problem of making informative queries, where an agent interactively queries for observations from an oracle such that these observations optimize a certain objective. For example, in the game of 20 questions, the agent interactively queries the oracle to deduce the identity of the hidden object. Informative querying is a challenging task because conditioned on the observations collected so-far, the agent needs to query for a new observation that maximizes the objective while under the uncertainty of the oracle.

Informative querying problems have a range of applications, including active diagnostics [8, 3, 9, 12, 18], coreset selection [5, 10, 1, 6], and bayesian optimisation [20]. While successful algorithms have been developed for these domains, these algorithms are either slow to compute or they make specific assumptions about the oracle function. A common Achilles's heel of these prior approaches is their reliance on reasoning over the uncertainties over the space of oracles. For instance, in the naive algorithm outlined for 20 questions, this amounts to explicitly counting over the space of possible items to select the best question to ask. As a result, when the space of oracles becomes sufficiently large and complex (for instance, of combinatorial complexity), these approaches become intractable to compute.

In this thesis, we explore an alternative approach to informative query selection with propagation. A propagation is a function that maps outcomes of past observations to outcomes of future queries directly, bypassing the expensive reasoning step

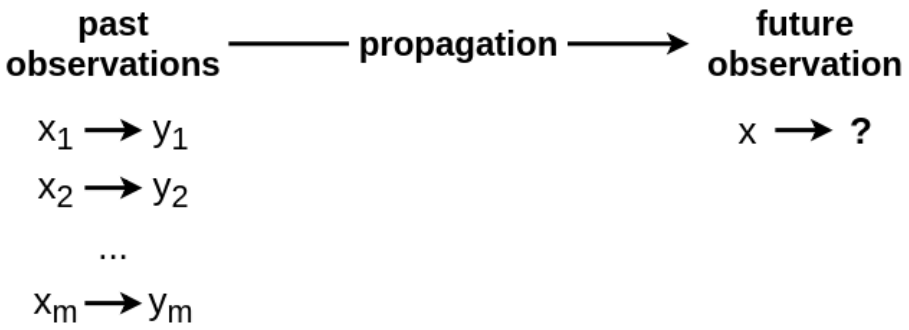

Figure 2-1: The propagation function

over the space of oracles. For instance, in 20 questions, given the observation that the animal can breath under water, one can use propagation to answer if it is likely to have feathers (the answer is no), and in turn decide that the question about feathers is a bad question. We show that by propagating past observations to future outcomes, one can solve a range of informative querying applications that were intractable.

2.1 Propagating Directly

How might one model the propagation function that maps outcomes of past observations to future observations? Let the set of past observations be $\bar{o} = \{(x_1, y_1) \cdots (x_m, y_m)\}$, where x_i denotes a *query*, y_i denotes an *outcome*, and (x_i, y_i) together denotes an *observation*, the propagation function will take these past observations and predict the outcome on a future query x (see Figure 2-1). Typically, propagation is computed by first performing an *induction* step, where one fits a hypothesis class F that is consistent with the past observations, followed by a *deduction* step that applies the learned hypothesis F to the new query point x to predict its outcome: $y = F(x)$. [1]

Rather than performing a pair of induction and deduction, the process of *transduction*[7] seeks to predict the future outcome directly. In this work, the propagation function is learned in meta-learning time and modeled with a neural network. Figure 2-2 compares the differences between induction, deduction, and transduction in the context of computing propagation.

[1] for ease of understanding, F is a single deterministic function for now

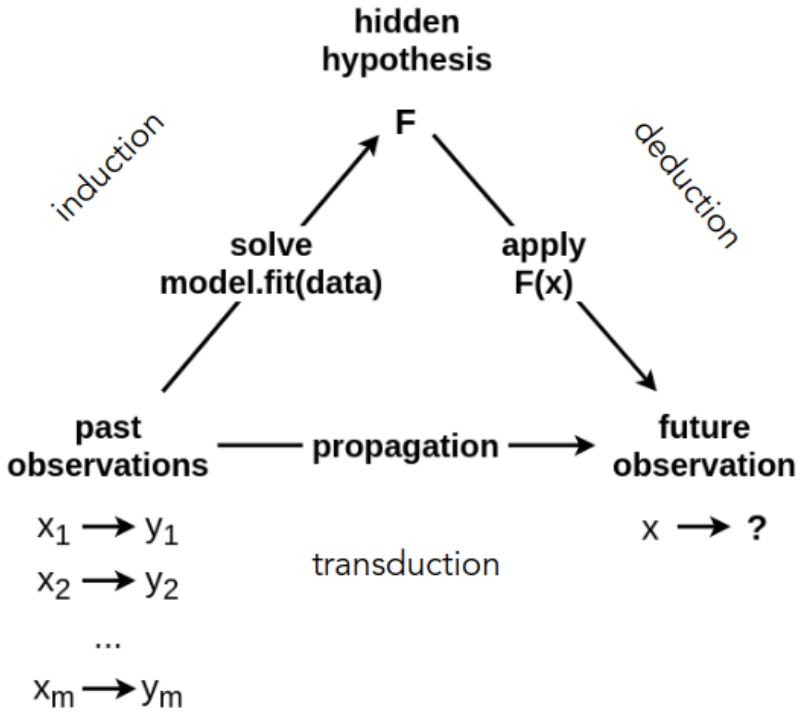

Figure 2-2: Induction, Deduction, Transduction

As one can see, rather than going "up" and "down" by performing induction followed by deduction, transduction goes "across" directly. Although both approaches can compute the same propagation function, modeling propagation with transduction has the distinctive advantage of allowing the learning model (a neural network, for instance) to pick up low-level patterns (Figure 2-3) that exists naturally between observations. Additionally, propagating the observation directly saves time as the induction step is often expensive, involving fitting a parameterized model.

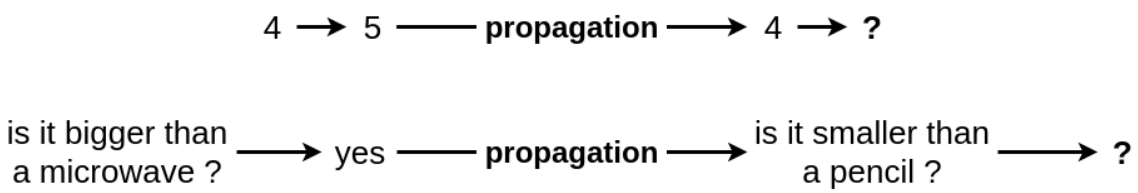

Figure 2-3: Modeling propagation directly can more easily learn simple patterns. In the first task, the answer is clearly 5, and in the second task, the answer is clearly no. Both answers can be deduced without solving for a consistent oracle hypothesis.

In this work, the propagation function is trained with meta-learning. In meta-learning time, we sample an oracle function from the hypothesis space, and perform mock queries to the sampled oracle to obtain a dataset of past observations and future observation, and a neural-network is trained to predict the outcome of the future observation conditioned on the past observations. In a sense, rather than performing computation over the hypothesis space at inference time, we perform *sampling* of the hypothesis space at training time, and the learned propagation distills the information as a pattern matching mechanism, amortizing the computation at inference time.

2.2 Informative Queries with Propagation

With a learned propagation function, one can perform informative queries in a range of applications by defining a selection criteria (an acquisition function) over the space of queries. Given a query, the querying agent first predicts its outcome by calling the propagation function, then, depending on the result of propagation, rank this query against other queries and selecting the best query to ask. Here, we will briefly explain one applications that can be solved with a learned propagation, preference elicitation.

2.2.1 An Example Application: Preference Elicitation

Preference elicitation, like 20 questions, is a problem of active diagnostics, where the agent issues a series of queries with the goal of diagnosing, or inferring, the identity of the oracle. Unlike 20 questions where the oracle is a single undisclosed item, in preference elicitation, the oracle is a hidden preference of multiple items. We use the sushi preference dataset [11], which records preferences among 10 kinds of sushi of 5000 surveyed Japanese residents. The agent is to discover the hidden preference (i.e. one of 10! permutations of 10 sushis) by asking the person a series of binary comparison questions (i.e. "do you like tuna more than shrimp?").

Propagation The neural-network propagation function takes in a set of past observations (pairwise comparison queries with answers) and predicts the answers to fu-

ture pairwise comparison queries. Figure 2-4 shows the learned propagation network propagating past observations to future queries. Here, each i, j coordinate denotes a pairwise query of whether sushi item i is more preferable to sushi item j. Green dots denote past observations with an outcome of *yes* (the green dot on 2nd row, 3rd column means $eel > tuna$). Red dots denote past observations with an outcome of *no* (the red dot on the 5th row and 1st column means $urchin < shrimp$). And the shading is the neural network's prediction on outcomes to future queries, with brighter indicating a more likelihood of *yes*. By looking at the figure, we note that The learned propagation function was able to successfully infer the following facts:

- **anti-symmetry** $(urchin < shrimp) \xrightarrow{propagate} (shrimp > urchin)$.
- **transitivity** $(fat\ tuna < tuna) \wedge (tuna < squid) \xrightarrow{propagate} (fat\ tuna < squid)$
- **educated guesses** The learned propagation function is fairly certain that this person likes squid more than urchin, $squid > urchin$, despite the fact that this inference cannot be made with logical deductions from the observations so far.

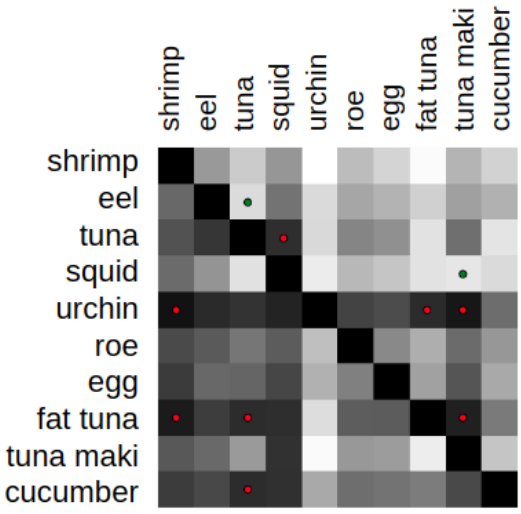

Figure 2-4: The learned propagation function propagating pairwise comparisons

Query Selection Given the result of propagation, the agent simply select the pairwise query with the *highest uncertainty* (i.e. the most gray coordinate) to query

next. We can show that querying with a learned propagation can out-perform classical sorting algorithms such as merge-sort and quick-sort, which lack the ability to make educated guesses unlike the propagation function trained on the sushi dataset.

2.2.2 Other Applications

In addition to preference elicitation, this thesis also addresses the following informative querying tasks with learned propagation, all with the same flavor of first propagating the outcomes of the queries conditioned on past observations, then selecting the query with the **most** _____ outcome. Here is a brief overview:

Active Diagnosis In active diagnosis, the agent interactively query the oracle with the objective of uncovering the oracle's identity. We show that the most informative next query is the one that the propagation function has the **most uncertainty**.

Representative Subset Selection for Program Synthesis In Program Synthesis, one seek to construct a program that perfectly fits a set of input-output examples as a symbolic regression task. One challenge of program synthesis is in the encoding of a large numbers of input-output examples. We can use propagation to select a coreset of input-output pairs that is sufficient to explain the rest of the input-output pairs, by iteratively selecting the **least likely** example.

Bayesian Optimization In bayesian optimization, the agent interactively query the oracle with the objective of maximizing the outcome while also being frugal with the number of queries. The query agent uses the learned propagation to predict future outcomes conditioned on past observations, and select according to the UCB criteria for the **most optimistic** query.

Before elaborating on our approaches, we will look at some related works and explain how this thesis work differs from them.

Chapter 3

Related Works

Informative querying problems have been studied extensively, as it naturally arise in situations where one needs to interact with an unknown oracle function. In this chapter, we demonstrate how this thesis is a departure from these prior approaches. This thesis work is best contrasted to prior approaches by two characteristics: The use of a meta-learned transduction model for computing propagation, and the fact that we use meta-learning to take advantage of the specific observation relationships, but do not assume an explicit form of the observation generation process.

3.1 Prior Works that Uses Induction

As stated in the introduction, we model the propagation function as a transduction process, going from past observations to future observation predictions directly. In contrast, typical prior approaches takes a induction-deduction approach, where a parameterised model is fitted to the past observations, then this model is used to infer the outcome of future queries. While both approaches can compute the same propagation, and in turn, used for query selection, using transduction has the advantage of being faster to compute by avoiding to fit a parameterised model.

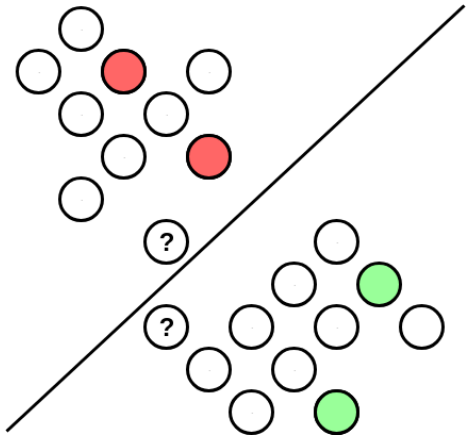

Figure 3-1: In boundary based active learning, one query for the unlabelled point closest to the current decision boundary.

3.1.1 Boundary Based Active Learning

The most well-known instance of informative querying is boundary based active learning [22]. In this setting, one has a large dataset of unlabelled data, and one wishes to fit a good decision boundary by iteratively querying for which new data point is to be labelled. The general approach here is to fit a decision boundary with the labelled data, and query for the unlabelled point closest to the decision boundary. See Figure 3-1. This is an instance of induction (fitting a parameterised decision boundary) and deduction (points that lie closest to the decision boundary) process.

3.1.2 Version Space Algebra

Rather than solving for a single hypothesis from the model class like boundary based active learning, in version space algebra [13], or by extension, committee based active learning [19], one maintains a *set* of models consistent with the observations so-far, and query for the observation that would maximally reduce the set of current models (The naive solution to 20 Questions is an instance of this scheme). See Figure 3-2.

3.1.3 CEGIS

In the field of program synthesis, one is often confronted with an overwhelmingly large set of input-output examples which one needs to synthesize a program that is

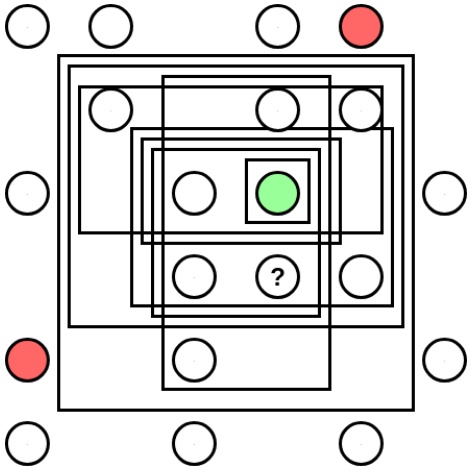

Figure 3-2: In version space algebra, or committee based active learning, one seeks an query whose result would be maximally disagreeable among the current set of consistent models. Here, the task is to solve for a rectangle whose interior contains only green points, and the rectangles drawn here represents the set of currently valid rectangles. The question-marked input is the most disagreeable, as half the rectangles contain it and the other half do not.

consistent to the set. As the synthesizer scales poorly with the number of examples, one likes to select a subset of examples to run the synthesizer on. Counter Example Guided Inductive Synthesis (CEGIS) [21] is the most popular algorithm to select this subset of examples. It works by starting with a random subset of examples, then iteratively alternates between synthesizing a consistent program, and adding a *counter example* that contradicts the current program. See Figure 3-3. Like the boundary-based active learning approach, CEGIS is a process of induction (first solving for a consistent program) and deduction (using the program to select a counter-example).

3.2 Prior Works that Assumes Generation Processes

The propagation function is learned by training on samples of observations to learn the relationships between them. In theory, one can fit to any observation generation process by selecting an appropriate NN architecture, and training on the set of observations. In contrast, prior works that models the propagation function usually assumes a particular forms of observation generation process, which prohibits them from faithfully model the propagation function when these assumptions are not met.

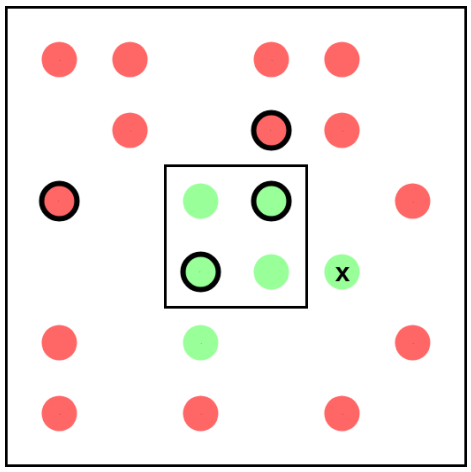

Figure 3-3: To select a subset of examples, CEGIS first solves for a consistent program (here, a rectangle where only the green examples lay inside of) from the current subset (circles with dark boarders), and add an example that contradicts the current program (the green example with the cross on it).

3.2.1 Coreset Selection

In coreset selection, one seek to select a subset of observations, called a coreset, to represent the original set of observations, which are often too big to be stored. Typical coreset selection algorithms assume a particular observation generation process, such as they are piece-wise linear [1, 6], or are generated from a logistic regression model [5, 10]. See Figure 3-4.

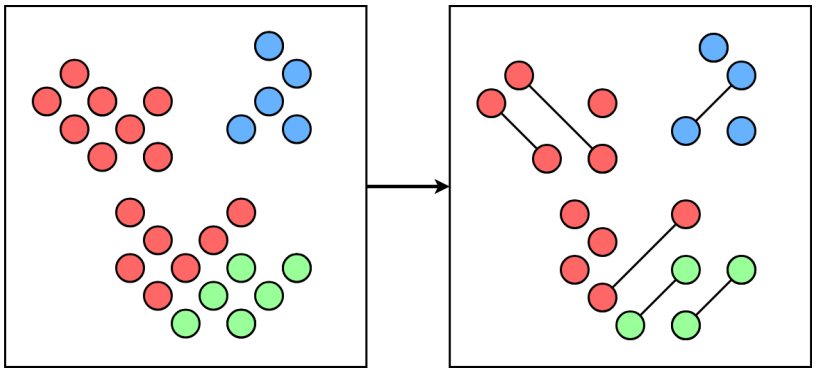

Figure 3-4: By assuming the observations are piece-wise linear and only keeping the end-points, coreset selection picks a subset of observations that can be used to re-construct the original observations

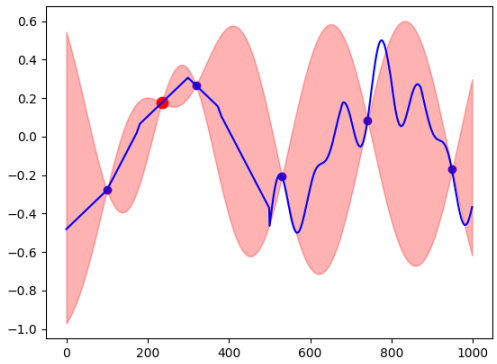

Figure 3-5: By assuming the observations are jointly gaussian, GP can compute the predictive posterior distribution, which is used to select the next query

3.2.2 Gaussian Process

Gaussian process (GP) [20] is the go-to algorithm for solving bayesian optimisation problems. In Gaussian process, one assumes that any subset of observations drawn from the oracle are jointly gaussian. Using this assumption, GP can compute the predictive posterior distribution (a form of propagation function on the domain of real-numbered outcomes), which can be used to select the next query. However, it maybe the case that the observations are not jointly gaussian where GP will have troubles modeling the predictive posterior faithfully. See Figure 3-5.

The rest of the thesis will formally define and address these aforementioned applications, and solve them with a learned propagation.

Chapter 4

Preliminaries

The required formalism for this thesis is thankfully not at all abstruse. Here is a quick re-cap of the concepts we'll be using.

Entropy This thesis deals with making informative queries. One way of measuring information is through the use of entropy H. Entropy is a measurement of uncertainty, the greater the uncertainty, the larger the entropy. Let X be a discrete random variable capable of taking on values x with probability $P(X = x)$, its entropy is given by the following formula:

$$H(X) = -\sum_{x} logP(X = x)P(X = x)$$

For example, if $coin_1$ has 2 sides but it is rigged so that it *always* lands on head: $P(coin_1 = Head) = 1$ and $P(coin_1 = Tail) = 0$, its entropy is

$$H(coin_1) = -(log(1) * 1 + log(0) * 0) = 0$$

Which coincides with our intuition that there is no uncertainty about this coin. A $coin_2$ that gives head half the time has the following entropy:

$$H(coin_2) = -(log(\frac{1}{2}) * \frac{1}{2} + log(\frac{1}{2}) * \frac{1}{2}) = -(-1) * (\frac{1}{2} + \frac{1}{2}) = 1$$

In another word, a fair coin has precisely 1 "bit" of information. With 6 sides instead of 2 sides, one would expect a fair die to contain more uncertainty. Indeed, a fair 6-sided die has the following entropy:

$$H(die) = -log(\frac{1}{6}) * \frac{1}{6} * 6 \approx 2.5849$$

If the random variable X is not discrete but continuous, we simply switch the summation with integration:

$$H(X) = -\int_x logP(X = x)P(X = x)dx$$

In this thesis, when a random variable can be left unspecified to be discrete or continuous, we will discuss it in the continuous setting using integration.

Conditional Entropy Just like how a probability can be made into a conditional probability, an entropy can be made into a conditional entropy. This can be done in 2 different ways:

If a probability is itself a conditional probability, one can compute its entropy the same way by using the conditional probability. The entropy of a conditional distribution $P(X|Y = y)$ is simply:

$$H(X|Y = y) = -\int_x logP(X = x|Y = y)P(X = x|Y = y)dx$$

On the other hand, one can compute the so-called "conditional entropy" between two *random variables* X and Y as follows:

$$H(X|Y) = \mathop{\mathbb{E}}_{y\sim P(Y)}[H(X|Y = y)] = \int_y P(Y = y)H(X|Y = y)dy$$

That is to say, the entropy of a random variable X, conditioned on another random variable Y, is the expected value of the entropy $H(X|Y = y)$ under $P(Y)$.

Mutual Information Mutual information measures how much knowing the outcome of a random variable reduces the uncertainty of another random variable, and vice-versa. Given two random variables X and Y, their mutual information is:

$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

In other words: Without any knowledge, a random variable X has a certain amount of uncertainty $H(X)$. However, if we can observe the outcome of Y, the uncertainty on X is reduced, becoming instead a conditional entropy $H(X|Y)$. The reduction of entropy between $H(X)$ and $H(X|Y)$ is the mutual information between X and Y. Note the above relationship is provably symmetrical, i.e. one can compute the mutual information either by conditioning X with Y or conditioning Y with X.

KL divergence KL divergence measures how two distributions P and Q differs:

$$\text{KL}(P||Q) = \int_x P(X=x) log(\frac{P(X=x)}{Q(X=x)}) dx$$

The KL divergence between P and Q is 0 if P and Q are identical distributions. Note that you can re-write the KL divergence as an expectation over $P(X)$:

$$\begin{aligned}
&\text{KL}(P||Q) = \int_x P(X=x) log(\frac{P(X=x)}{Q(X=x)}) dx \\
&= \mathop{\mathbb{E}}_{x \sim P(X)} [log(\frac{P(X=x)}{Q(X=x)})] \\
&= \mathop{\mathbb{E}}_{x \sim P(X)} [logP(X=x) - logQ(X=x)]
\end{aligned}$$

Chapter 5

Making Informative Queries

We describe the problem of making informative queries in a formal sense, along with formal definitions of some of its applications that this thesis addresses.

5.1 The Observation Process

We start by explaining the observation process, central to this process are the possible queries $x \in X$, the possible observation outcomes $y \in Y$, and the oracle function $F(\cdot, \theta) : X \longrightarrow Y$ capable of taking in a query and returning an observation, depending on which particular oracle $\theta \in \Theta$ it was parameterized with.

Of the entities mentioned so-far, X and Y are sets, F is a deterministic function, and only θ is a random variable drawn from a distribution $\theta \sim P(\Theta)$. That is to say, all source of uncertainty in this setting can trace its roots to the unknown identity of θ we instantiated the oracle function F with[1]. Figure 5-1 shows the interactions of these entities in a graphical model view: the query x which can be simply considered as a fixed parameter, is turned into a random variable $Y_x = F(x; \Theta)$ once passing through the function F due to the uncertainty introduced by not knowing which θ that F was parameterized with. We may use $F(x, \Theta)$ and Y_x interchangeably, whichever form is more illustrative for the problem at-hand. Note that, while there are uncertainties over the identity of the oracle function F, F is still a *function* in

[1] in later applications the uncertainty can also come from a noisy channel

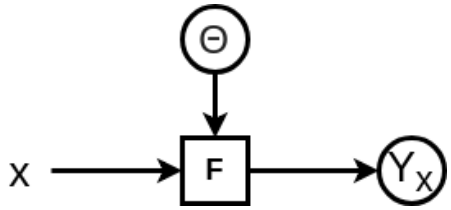

Figure 5-1: Function $F(\cdot, \theta)$ changes a parameter x into a random variable Y_x due to the uncertainty induced by $\theta \sim P(\Theta)$

that repeated querying the same input x would yield a consistent observations y:

$$P(F(x, \Theta) = y | F(x, \Theta) = y) = 1$$

5.2 The Informative Querying Problem

In an informative querying task, the goal is to return a set of observations by iteratively querying an oracle function F, while ensuring the resulting observations optimize a certain objective (for instance, achieving the maximum outcomes). As the oracle function is unknown a-priori, the querying algorithm must trade-off exploring and understanding the oracle function with exploiting to maximize some objective.

Let us use $\bar{o}$ to denote the set of observations $\bar{o} = \{(x_1, y_1) \dots (x_m, y_m)\}$. An informative querying problem is typically solved by Algorithm 1.

Algorithm 1: The Successive Querying Process

```
1 SUCCESSIVE QUERYING (F, X);
  Input  : oracle function F, input domain X
  Output: collected observations ō
2 ō ⟵ {}
3 while not terminate(ō) do
4 |   x = select(X, ō)
5 |   y = F(x)
6 |   ō ⟵ ō ∪ (x, y)
7 end
8 return ō
```

Central to the selection process is deciding which query to issue next, conditioned

on previous observations (line 4 in Alg 1). For instance, the selection process can be random, but it would not be informative. Making informative query is often realized by a well crafted *acquisition function*.

5.3 Acquisition Function

An acquisition function $a_{\bar{o}}(x)$ is a function that is parameterized by past observations $\bar{o}$. It takes in a query point x, and measures the "goodness" of this query point with respect to a certain metric (such as improving the best returns so-far or maximizing information gain).

Once an acquisition function is defined, the informative query is selected on the condition that the query x optimizes (argmin or argmax) the acquisition function.

$$x_{query} = \underset{x \in X}{\operatorname{argmax}}\, a_{\bar{o}}(x)$$

This query is then issued to the oracle function, obtaining an observation y in return. The new observation (x, y) is then added back into $\bar{o}$ and the process is repeated until some termination condition is met, usually achieving a certain optimization objective or reaching a certain budget. The resulting $\bar{o}$ is returned, see Figure 5.3.

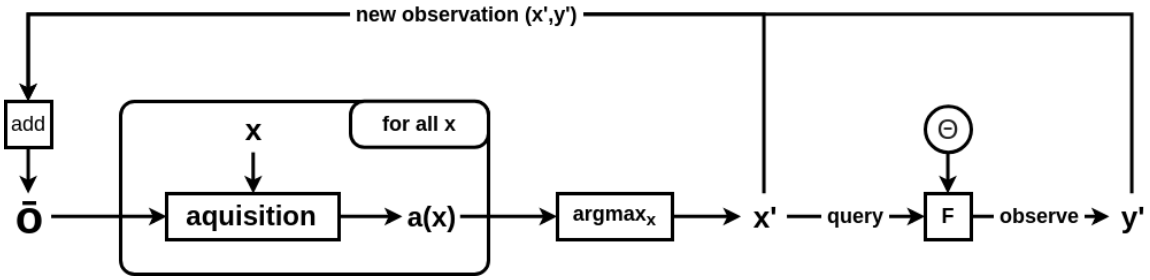

Figure 5-2: Successive Querying With Acquisition Function

5.4 Applications of Informative Queries

We now discuss three important applications that can be framed as informative querying problems, and highlight the difficulties in their typical solutions.

5.4.1 Active Diagnostics

A well known application of informative query selection is active diagnostics. In active diagnostics, the goal is to infer the hidden parameter $\theta \in \Theta$ with as few observations $F(x_1, \Theta) \dots F(x_n, \Theta)$ as possible, where these observations are made adaptively. Given past observations $\bar{o}$, one selects the next best query by maximizing mutual information between the query's outcome Y_x and the hidden parameter Θ.

Acquisition Function Formally, the acquisition function for active diagnostics is:

$$a_{\bar{o}}(x) = I(\Theta, Y_x | \bar{o})$$

Using this acquisition function, one can select the most informative query by maximizing over the query space $x \in X$:

$$\operatorname*{argmax}_x I(\Theta, Y_x | \bar{o})$$

Typically, the optimal selection is computed by re-writing the definition of mutual information as a difference of entropy reduction:

$$\operatorname*{argmax}_x H(\Theta | \bar{o}) - H(\Theta | Y_x, \bar{o})$$

Which is to say, the mutual information between Θ and Y_x can be measured by the reduction of entropy of Θ given Y_x. Note that the first term $H(\Theta|\bar{o})$ is constant over x, thus we can omit it within the argmax_x operator, obtaining:

$$= \operatorname*{argmin}_x H(\Theta | Y_x, \bar{o})$$

However, this acquisition function is difficult to compute because it involves an entropy computation over the space of possible functions Θ, which is intractable in practice if Θ is sufficiently complex (for instance, of combinatorial complexity).

5.4.2 Representative Subset Selection

In representative examples selection, one *starts with* a large, existing set of observations $\bar{o}_{big}$. One is also armed with a computationally expensive solver *solve* that can take a set of observations $\bar{o}$ and infers a satisfying $\theta \in \Theta$:

$$\theta = solve(\bar{o}) \Longrightarrow \bigwedge_{(x,y)\in\bar{o}} F(x,\theta) = y$$

The goal is to solve for a θ that satisfies the big set of observations $\bar{o}_{big}$. However, due to the size of $\bar{o}_{big}$, one cannot afford to run the solver on it directly. One can think of the representative subset selection problem as the opposite of the active diagnostic problem: In active diagnostic, the challenge is we do not have enough observations to pin-point the hidden value of θ; In representative subset selection, we have *too many* spurious observations that renders the expensive inference process *solve* intractable.

Therefore, one would like to select a *representative subset* $\bar{o}_{repr} \subseteq \bar{o}_{big}$ such that:

$$\bigwedge_{(x,y)\in\bar{o}_{repr}} F(x,\theta) = y \Longrightarrow \bigwedge_{(x,y)\in\bar{o}_{big}} F(x,\theta) = y$$

Which is to say, any θ that is consistent with $\bar{o}_{repr}$ will also be consistent with $\bar{o}_{big}$. If such a representative subset can be found, one can run the solver on the representative subset $solve(\bar{o}_{repr})$, which would still yield a consistent θ without incurring the expensive computation cost.

Counter Example Guided Inductive Synthesis (CEGIS) Typically, problem of representative subset selection is formulated as a informative querying problem with the CEGIS algorithm: Starting with an empty set of representative examples $\bar{o} = \{\}$, the CEGIS algorithm runs the *solve* function on the current set of observations,

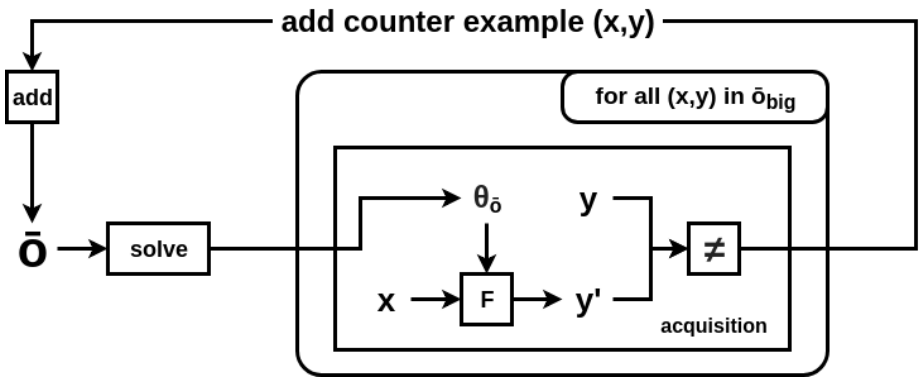

Figure 5-3: Successive Querying with CEGIS

obtaining a *candidate hypothesis* $\theta_{\bar{o}} = solve(\bar{o})$. Then, the algorithm checks whether θ is consistent with every item in the big set of observations $\bar{o}_{big}$: If there is a particular observation (x, y) that is inconsistent with θ, i.e. $y \neq F(x, \theta)$, this *counter example* is added to $\bar{o}$. Conveniently, when the CEGIS algorithm terminates, we obtain θ that is consistent with $\bar{o}_{big}$. The CEGIS algorithm is shown in Figure 5.4.2

Acquisition Function For CEGIS The acquisition function for CEGIS can be then summarized as:

$$a_{\bar{o}}((x, y)) = \begin{cases} 0 & \text{if } F(x, \theta_{\bar{o}}) = y \\ 1 & \text{if } F(x, \theta_{\bar{o}}) \neq y \end{cases} .$$

Note that the acquisition function is over observations (x, y) for the representative subset problem, as we starts with a set of observations already, there is no need to query the function F for more outcomes.

The drawback of using CEGIS as part of the acquisition function is that one is still invoking the expensive solver $solve(\bar{o})$ for every counter-example added to $\bar{o}$. So in case of $\bar{o}_{repr}$ needing to be large, CEGIS can be inefficient simply on the basis of calling the expensive *solve* function many times. Furthermore, the set $\bar{o}_{repr}$ returned by CEGIS is in fact *not* representative. For instance, the solver may return a $\theta_{\bar{o}}$ that is coincidentally consistent with $\bar{o}_{big}$ with just a single element in $\bar{o}$.

5.4.3 Bayesian Optimization

One of the most prominent applications of informative queries selection is the problem of bayesian optimization. In bayesian optimization, one wishes to optimize a unknown and expensive to query function $g(\cdot) : X \longrightarrow \mathbb{R}$ by selecting the optimal input x^*.

$$x^* = \underset{x}{\operatorname{argmax}}\, g(x)$$

The goal is to discover the optimal input x^* with as few queries as possible, since g is expensive to query. To apply our formalism, let: $g(\cdot) = F(\cdot, \theta)$, where the $\theta \in \Theta$ represents uncertainties over the identity of g.

Bayesian optimization is typically solved with gaussian process (GP). Under the GP formulation, the informative querying process is realized by the computation of a *predictive posterior*, which itself is a gaussian:

$$P(Y_x|\bar{o}) = \mathcal{N}(\cdot|\mu(x, \bar{o}), \Sigma(x, \bar{o})^2)$$

Which is to say, given a set of past observations $\bar{o}$, and a new query point x, a GP will compute the mean $\mu(x, \bar{o})$ and variance $\Sigma(x, \bar{o})^2$ of the possible outcomes at x. Various acquisition functions that can be constructed from this posterior. The typical acquisition functions with a predictive posteriors are probability of improvement, expected improvement, and upper confidence bound. For this thesis, we will consider only the acquisition function of upper confidence bound (UCB) [2] .

Upper Confidence Bound The UCB acquisition function is:

$$a_{\bar{o}}(x) = \mu_(x, \bar{o}) + \Sigma_(x, \bar{o})$$

Which can be readily computed from the predictive posterior's $\mu_(x, \bar{o})$ and $\sigma_(x, \bar{o})$.

The success of GP lies in choosing the right kernel function which will lead to an accurate predictive posterior estimation. However, it is possible that for a specific class of functions, the GP assumption that any finite observations of the function

forms a multinomial gaussian distribution is too restrictive and prevents one from exploiting the underlying structures of the bayesian optimization problem at hand.

In this chapter, we formally defined the informative querying problem, and how they are typically solved by constructing a suitable acquisition function. We also observed that the typical solutions are either inefficient (in the number of queries) or slow (in terms of computation time).

Chapter 6

Propagation

In this chapter we will formalize the definition of propagation in the most general sense, explain how it is trained, give some details on its encoding, and explain some advantages of the particular way the propagation is constructed.

6.1 Propagation Probability

We start by defining the propagation probability. The propagation probability takes in *past observations* and infers the outcomes of a *future query*. Formally, given a set of past queries $x_1 \dots x_m$ and their corresponding observations $F(x_1, \Theta) = y_1 \dots F(x_m, \Theta) = y_m$, we would like to infer the distribution of outcome $F(\cdot, \Theta)$ on a new query x. Similar to how Y_x is a short-hand for $F(x, \Theta)$, the observation pair (x_i, y_i) is a short-hand for denoting the specific *event* that the random variable $F(x_i, \Theta)$ takes on the specific value y_i. The propagation probability can be written in several ways:

$$
\begin{aligned}
& P(Y_x|\bar{o}) \\
:=& P(Y_x|\{(x_1, y_1) \dots (x_m, y_m)\}) \\
:=& P(F(x, \Theta) = y|F(x_1, \Theta) = y_1 \dots F(x_m, \Theta) = y_m)
\end{aligned}
$$

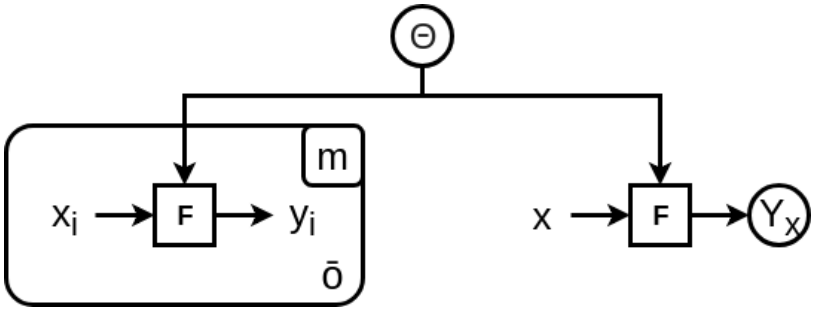

Figure 6-1: Given m past observations, what is the distribution on a future observation Y_x? Note the two random variables in this model is Θ and Y_x

The distribution Y_x maybe computed exactly by integrating over Θ:

$$\begin{aligned}
&P(Y_x = y|\bar{o}) \\
=&P(F(x,\Theta) = y|F(x_1,\Theta) = y_1 \dots F(x_m,\Theta) = y_m) \\
=&\int_\theta \mathbb{1}[F(x,\theta) = y]\ P(\Theta = \theta|F(x_1,\Theta) = y_1 \dots F(x_m,\Theta) = y_m)d\theta
\end{aligned}$$

Given the past observations $\bar{o}$, we integrate over parameters θ from the posterior distribution $P(\Theta = \theta|F(x_1,\Theta) = y_1 \dots F(x_m,\Theta) = y_m)$ (induction), and check whether $F(x,\theta) = y$ (deduction). However, when confronted with a complicated space of Θ, such integral is intractable. Instead, we *approximate* the propagation probability.

6.2 Propagation Function

Rather than computing the propagation probability $P(Y_x|\bar{o})$ exactly, we will approximate it with a *propagation function* Q_ω that is a function approximator with parameter ω (i.e. a neural network). This propagation function would take as input a set of past observations $(x_1, y_1) \dots (x_m, y_m)$, an additional query input x, and compute an approximate to the propagation probability.

$$P(Y_x|\bar{o}) \approx Q(\cdot|x,\bar{o}) \tag{6.1}$$

Figure 6.2 shows the inputs and output of a propagation function:

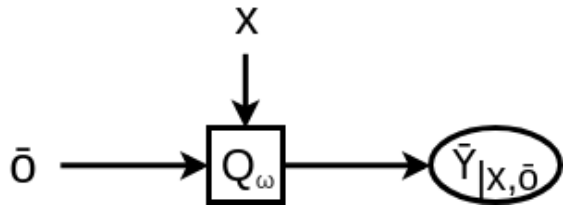

Figure 6-2: Given past observations $\bar{o}$ and a new query input x, the propagation function approximate the propagation probability

6.2.1 Approximating the Propagation Probability

To make sure the propagation function faithfully approximate the propagation probability, we minimize the KL-divergence between the true propagation probability $P(F(x,\theta) = \cdot|\bar{o})$ and the approximate propagation function $Q(\cdot|x,\bar{o})$

$$\underset{\omega}{\operatorname{argmin}} \underset{y}{\mathrm{KL}}(P(F(x,\Theta) = y|\bar{o}) \;||\; Q(y|x,\bar{o})) \tag{6.2}$$

Note that the propagation probability $P(F(x,\Theta) = y|\bar{o})$ is a conditional probability, where a different distribution corresponding to *each* of its possible conditionals $\bar{o}$, which leads to inefficiencies in approximation[1]. Therefore, rather than creating a propagation function *for each* of its possible conditions, we create a *single* propagation function that minimizes the expected KL divergence, under a suitable known distribution of $\bar{o}$ and future query x.

Putting everything together, the objective for Q_ω is as follows:

$$\underset{\omega}{\operatorname{argmin}} \underset{\bar{o},x}{\mathbb{E}}[\underset{y}{\mathrm{KL}}(P(F(x,\Theta) = y|\bar{o}) \;||\; Q_\omega(y|x,\bar{o}))] \tag{6.3}$$

This objective is difficult to directly optimize because we need to sample from $\bar{o}$, which preclude us from sampling the pairs x_i, y_i independently, because we might sample certain combinations of x_i, y_i for which no functions $F(\cdot,\theta)$ is admissible. In addition, we also need to compute the KL-divergence over all $y \in Y$.

[1]If in practice $\bar{o}$ always has less than 10 elements, it is pointless to make Q_ω to approximate instances of $\bar{o}$ having more than 10 elements

We will now massage the objective into a form that is tractable with stochastic optimization. First, we re-write the KL divergence as an expectation:

$$\begin{aligned}
&\operatorname*{argmin}_{\omega} \mathop{\mathbb{E}}_{\bar{o},x}[\mathop{\mathrm{KL}}_{y}(P(F(x,\Theta)=y|\bar{o}) \;||\; Q_\omega(y|x,\bar{o}))] \\
=&\operatorname*{argmin}_{\omega} \mathop{\mathbb{E}}_{\bar{o},x}[\mathop{\mathbb{E}}_{y\sim P(F(x,\Theta)=y|\bar{o})}[logP(F(x,\Theta)=y|x,\bar{o}) - logQ_\omega(y|x,\bar{o})]]
\end{aligned}$$

Note that the term $logP(F(x,\Theta)=y|\bar{o})$ does not depend on ω, which we can ignore inside the $\operatorname{argmin}_\omega$ operator, omitting this term we obtain:

$$\begin{aligned}
&= \operatorname*{argmin}_{\omega} \mathop{\mathbb{E}}_{\bar{o},x}[\mathop{\mathbb{E}}_{y\sim P(F(x,\Theta)=y|\bar{o})}[-logQ_\omega(y|x,\bar{o})]] \\
&= \operatorname*{argmax}_{\omega} \mathop{\mathbb{E}}_{\bar{o},x}[\mathop{\mathbb{E}}_{y\sim P(F(x,\Theta)=y|\bar{o})}[logQ_\omega(y|x,\bar{o})]]
\end{aligned}$$

Next, we re-write the expectation over $y \sim P(F(x,\Theta)=y|\bar{o})$ as an integral:

$$\operatorname*{argmax}_{\omega} \mathop{\mathbb{E}}_{\bar{o},x}[\int_y P(F(x,\Theta)=y|\bar{o})\; logQ_\omega(y|x,\bar{o})\; dy]$$

We then create a joint distribution between $F(x,\Theta)$ and Θ, before integrating it away, followed by a few re-writes to re-arrange the conditional probabilities:

$$\begin{aligned}
&= \operatorname*{argmax}_{\omega} \mathop{\mathbb{E}}_{\bar{o},x}[\int_y \int_\theta P(F(x,\Theta)=y,\Theta=\theta|\bar{o})\; logQ_\omega(y|x,\bar{o})\; d\theta\; dy] \\
&= \operatorname*{argmax}_{\omega} \mathop{\mathbb{E}}_{\bar{o},x}[\int_y \int_\theta \frac{P(F(x,\Theta)=y,\Theta=\theta,\bar{o})}{P(\bar{o})}\; logQ_\omega(y|x,\bar{o})\; d\theta\; dy] \\
&= \operatorname*{argmax}_{\omega} \mathop{\mathbb{E}}_{\bar{o},x}[\int_y \int_\theta \frac{P(F(x,\theta)=y,\bar{o}|\Theta=\theta)P(\Theta=\theta)}{P(\bar{o})}\; logQ_\omega(y|x,\bar{o})\; d\theta\; dy]
\end{aligned}$$

We re-write the expectation under $\bar{o}$ as an integral as well:

$$\begin{aligned}
&= \operatorname*{argmax}_{\omega} \mathop{\mathbb{E}}_{x}[\int_{\bar{o}} \int_y \int_\theta P(\bar{o}) \frac{P(F(x,\Theta)=y,\bar{o}|\Theta=\theta)P(\Theta=\theta)}{P(\bar{o})}\; logQ_\omega(y|x,\bar{o})\; d\theta\; dy\; d\bar{o}] \\
&= \operatorname*{argmax}_{\omega} \mathop{\mathbb{E}}_{x}[\int_{\bar{o}} \int_y \int_\theta P(F(x,\theta)=y,\bar{o}|\Theta=\theta)P(\Theta=\theta)\; logQ_\omega(y|x,\bar{o})\; d\theta\; dy\; d\bar{o}]
\end{aligned}$$

Note that $F(x, \Theta)$ and $\bar{o}$ are conditionally independent given $\Theta = \theta$:

$$= \underset{\omega}{\text{argmax}} \underset{x}{\mathbb{E}}[\int_{\bar{o}} \int_{y} \int_{\theta} P(F(x, \Theta) = y | \Theta = \theta) P(\bar{o} | \Theta = \theta) P(\Theta = \theta) \ log Q_\omega(y | x, \bar{o}) \ d\theta \ dy \ d\bar{o}]$$

Note that $P(F(x, \Theta) = y | \Theta = \theta)$ is an indicator function, having a probability of 1 when $y = F(x, \theta)$. Thus, as we integrate over all possible values of y, the only mass will be on the instance when $y = F(x, \theta)$. This allows us to drop the integral over y by substituting the instance of y within Q_ω with $F(x, \theta)$:

$$= \underset{\omega}{\text{argmax}} \underset{x}{\mathbb{E}}[\int_{\bar{o}} \int_{\theta} P(\bar{o} | \Theta = \theta) P(\Theta = \theta) \ log Q_\omega(F(x, \theta) | x, \bar{o}) \ d\theta \ d\bar{o}]$$

We rewrite the integration over $\bar{o}$ as two separate integrations, first over all the possible x values $\bar{x}$ then over all the possible y values $\bar{y}$, and re-write $\bar{o}$ accordingly.

$$= \underset{\omega}{\text{argmax}} \underset{x}{\mathbb{E}}[\int_{\bar{x}} \int_{\bar{y}} \int_{\theta} P(\bar{o} | \Theta = \theta) P(\Theta = \theta) \ log Q_\omega(F(x, \theta) | x, \bar{o}) \ d\theta \ d\bar{x} \ d\bar{y}]$$
$$= \underset{\omega}{\text{argmax}} \underset{x}{\mathbb{E}}[\int_{\bar{x}} \int_{\bar{y}} \int_{\theta} P(\bar{y} | \bar{x}, \Theta = \theta) P(\bar{x}) P(\Theta = \theta) \ log Q_\omega(F(x, \theta) | x, \bar{o}) \ d\theta \ d\bar{x} \ d\bar{y}]$$

Similar to how knowing θ makes $P(F(x, \Theta) = y | \Theta = \theta)$ an indicator function over $F(x, \Theta)$, knowing θ also makes $P(\bar{y} | \bar{x}, \Theta = \theta)$ an indicator function over $\bar{y}$: picking up a mass of 1 only when $y_i = F(x_i, \theta)$ for every $(x_i, y_i) \in \bar{o}$ and 0 otherwise. Thus, we can drop the integration over $\bar{y}$ over the indicator $P(\bar{y} | \bar{x}, \Theta = \theta)$:

$$= \underset{\omega}{\text{argmax}} \underset{x}{\mathbb{E}}[\int_{\bar{x}} \int_{\theta} P(\Theta = \theta) P(\bar{x}) \ log Q_\omega(F(x, \theta) | x, \bar{o}) \ d\theta \ d\bar{x}]$$

Rewriting all integrations as their corresponding expectations, we finally[2] obtain:

$$\underset{\omega}{\text{argmax}} \underset{\theta, \bar{x}, x}{\mathbb{E}} [\ log Q_\omega(F(x, \theta) | x, \bar{o})] \tag{6.4}$$

[2]I'd like to thank Kevin Ellis for making this long derivation possible over 6 cups of tea

6.2.2 Training the Propagation Function

The above objective can be optimized in a stochastic fashion: First, we sample an arbitrary parameter θ, a set of queries to be observed $\bar{x}$, and a future unobserved query x. Then, we compute $\bar{o}$ by applying $F(\cdot, \theta)$ on $\bar{x}$, and compute $F(x, \theta)$. We task the propagation function Q_ω to maximize the log likelihood over the outcome $F(x, \theta)$, conditioned on the past observations $\bar{o}$ and the query point x. See Figure 6.2.2.

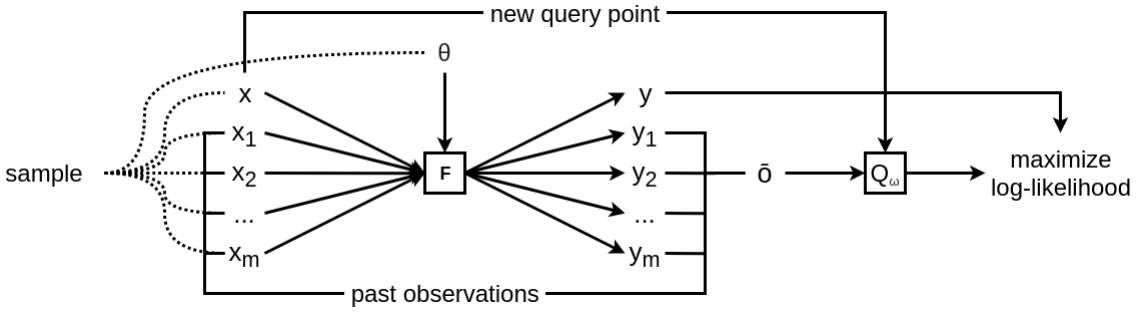

Figure 6-3: Training Q_ω via stochastic optimization by sampling $\theta, \bar{x}, x$

Access to F and $P(\Theta)$ or Samples of $P(\Theta)$ In the most ideal case, one has access to both F and generator of oracles, or samples of, $P(\Theta)$ itself. This scenario tends to arise in simulated settings where one can generate their own data. Having access to F and $P(\Theta)$ means one can construct a query-able oracle function $F(\cdot, \theta)$ which can be queried interactively for any given input x for as many times as needed.

What's remain is to construct an appropriate distribution for for $P(\bar{x})$ and $P(x)$ so that one can successfully train a propagation function Q_ω using the stochastic optimization scheme outlined earlier. The following scheme has worked well:

- let N be the maximum number of queries one expects to make at inference time
- randomly sample a number $k \sim Uniform(\{0 \cdots N-1\})$
- sample $\bar{x}$ as k instances of $x_i \sim Uniform(X)$
- sample x as $x \sim Uniform(X)$

One can also skew the distribution of $x, \bar{x}, \bar{o}$ to match the distribution that would happen at inference time, but in practice random sampling works well.

Access to Only Past Observations In practice, one may not have full access to the full generative process of Θ, but instead, series of observations from prior samples of $F(\cdot, \theta) \sim P(\Theta)$. This is often the case where the oracle function F cannot be simulated, and one must rely on past-draws of observations for meta-learning.

Specifically, let $\bar{o}_1 \cdots \bar{o}_M$ be M sets of past observations corresponding to samples from $F(\cdot, \theta_1) \cdots F(\cdot, \theta_M)$. The following sampling scheme works well in practice:

- sample at random a set of observations $\bar{o}_j \sim Uniform(\{\bar{o}_1 \cdots \bar{o}_M\})$
- sample at random x and y without replacement $(x, y) \sim \bar{o}_j$
- sample a number $k \sim Uniform(0, len(\bar{o}_j) - 1)$
- sample k items without replacement from $\bar{o}_j$ as $\bar{o}$
- return $x, \bar{o}, y$ as training sample for Q_ω directly

The ability to train a propagation function without access to the observation generating function is a consequence of modeling propagation with transduction, which obviates the need of compute over the data generation (via induction) process.

6.2.3 Encoding the Propagation Function

So far we have discussed the query space X, outcome space Y, observations $\bar{o}$, and Q_ω as abstract mathematical objects. In this section, we will detail some encoding templates for these components that have worked well in practice.

Indexable X, Multiclass Y We first explain the simplest architecture for handling instances where the query space X can be indexed[3] $X = \{x^1 \cdots x^M\}$, and a multi-class outcome space $Y = \{1 \cdots N\}$.

[3]finite, and not of combinatorial complexity. i.e. about 1000 possible queries

We start by appending the outcome space with an additional value 0 to denote an unobserved outcome $Y = \{0, 1 \cdots N\}$. To encode the observations $\bar{o}$, one can enumerate for every index i and check the outcome of its corresponding query x^i: Either it is observed, i.e. $(x^i, y^i) \in \bar{o}$, or it is unobserved, i.e. the outcome of x^i is unknown 0. We can then concatenate these values together into a single vector $\bar{o} = [y^1 \cdots y^M]$, where each of the y^i can be of a value $0, 1 \cdots N$

Rather than encoding a future query x and passing it into Q_ω, in the case of an indexable query space X, it is feasible to have Q_ω outputs all the predicted outcomes across all future queries x^i simultaneously $Y = [y^{1'} \cdots y^{M'}]$. See Figure 6.2.3.

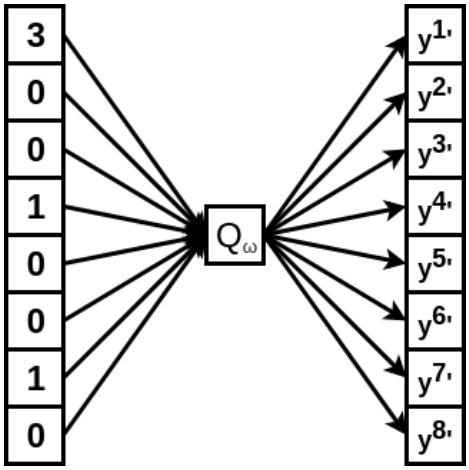

Figure 6-4: A computation of Q_ω under the indexable X, multi-class Y setting, where $M = 8$ and $\bar{o} = \{(x^1, 3), (x^4, 1), (x^7, 1)\}$

Q_ω in this case can simply be a fully connected MLP, and each predicted outcome $y^{i'}$ is a categorical distribution over N items (the output cannot be 0).

Factorized X In certain applications, it may be the case that the query space X has certain structures that allows the propagation probability to be factorized:

$$P(Y_x|\bar{o}) = P(Y_x|rel_x(\bar{o}))$$

Here, rel_x is a function that takes in $\bar{o}$ and returns only the relevant observations in $\bar{o}$ with respect to the query point x. For instance, it could be all observations

that lies within a certain neighborhood of x. In this case, one can similarly factorize the computation of Q_ω. Figure 6.2.3 shows an instance of a factorizable Q_ω in the indexable X multi-class Y setting:

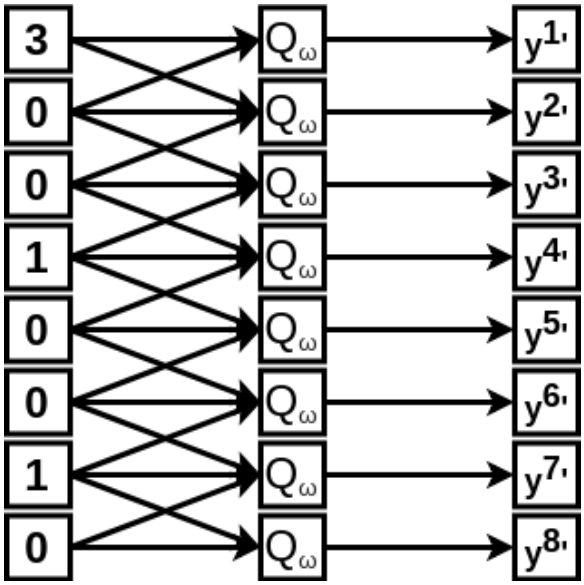

Figure 6-5: A factorized computation of Q_ω under the indexable X, multi-class Y setting, where $M = 8$ and $\bar{o} = \{(x^1, 3), (x^4, 1), (x^7, 1)\}$

Continuous X and Y We now discuss the case where X and Y are continuous values. Since X and Y are uncountable, one cannot use the indexable scheme outlined earlier. Instead, we will encode each observation $(x_i, y_i) \in \bar{o}$ and the future query x as individual items. Further, we will assume that the output distribution y is a gaussian distribution, and output its corresponding $\mu_{x,\bar{o}}$ and $\sigma_{x,\bar{o}}$ accordingly.

Figure 6.2.3 shows the encoding scheme for the continuous X and Y setting.

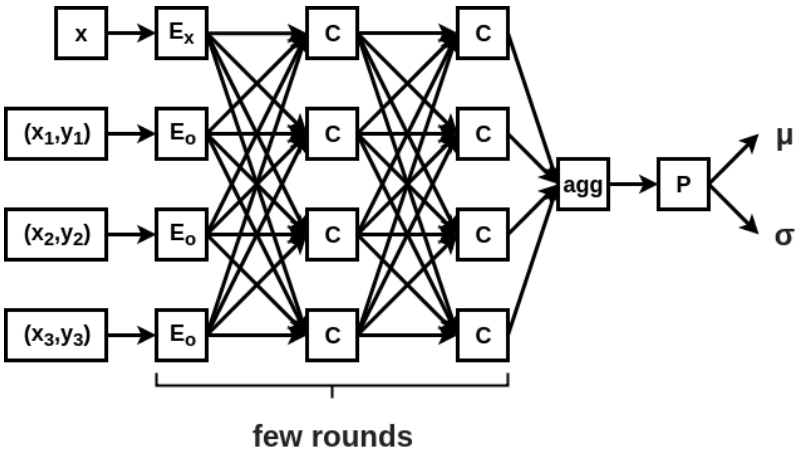

Figure 6-6: A transformer style encoding: E_x denotes the encoder for the query, E_o the encoder for an observation, C denote a communication module that pools information from other items together, agg is a generic set-invariant aggregator (max or mean), and the final prediction module P output the predicted mean and std(or co-variance in case of multi-dimensional gaussian)

As one can see, the input query x and the observations $(x_i, y_i) \in \bar{o}$ are allowed to intermix in several rounds of computations, before being aggregated together, and a prediction module outputs the predicted mean μ and standard deviation σ (or co-variance matrix, in case of multi-dimensional output) on the predicted value y. One can implement this as a transformer neural network or any structures that allows the exchange of information between the query x and other observations x_i, y_i.

6.3 Propagator

A propagator is a learned *propagation function* Q_ω conditioned with a *support set* of past observations $\bar{o}$. We write a propagator as follows:

$$Q_\omega(\cdot_y|\cdot_x, \bar{o})$$

Here, $\cdot_x$ and $\cdot_y$ denote the fact that it is a function over both the future query point x and a particular outcome event $y \in Y$. Figure 6-7 depicts the propagator.

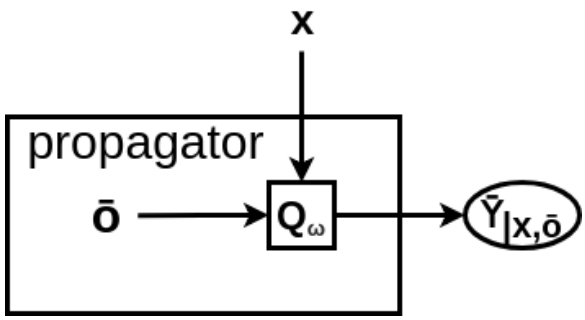

Figure 6-7: A propagator consists of a propagation function Q_ω and a support set $\bar{o}$

Complimentary Parts The most distinctive dichotomy is that the propagator consists of a parametric function Q_ω, which is *learned* during meta-training, and a collection of non-parametric observations $\bar{o}$, which are *collected* during inference. By learning a good propagation function Q_ω, the propagator can extrapolate future outcomes accurately given only a few past observations $\bar{o}$. On the other hand, inaccuracies of Q_ω can be alleviated at inference time by providing the propagation function with a bigger support set of $\bar{o}$. In the worst case, the propagator can always fail gracefully by devolving into the identify function over observed supports.

Chapter 7

Informative Queries with Propagation

The learned propagation function Q_ω can be applied in a range of informative querying tasks. To recap, in a informative querying task, the goal is to return a set of observations $\bar{o}$ by interactively querying an oracle function F, while ensuring the resulting observations $\bar{o}$ optimize a certain objective.

Across all tasks, we learn a *single* propagation function at meta-learning time. At inference time, we build a task-specific acquisition functions around the learned propagation function. The generic iterative querying process is shown in Figure 7-1.

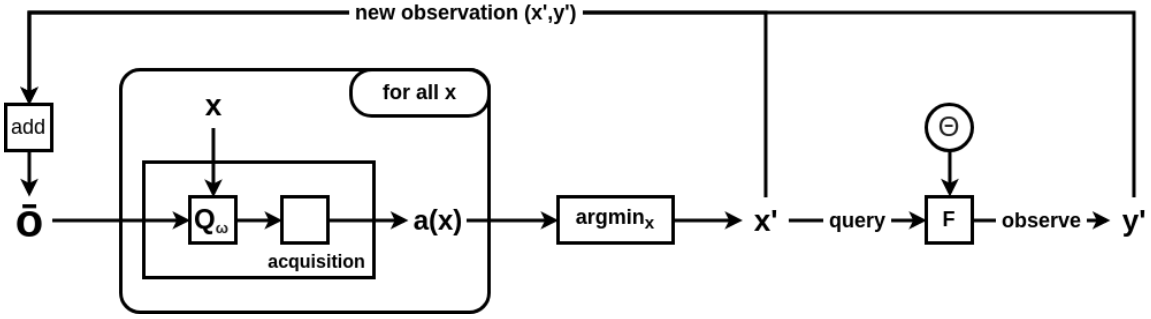

Figure 7-1: Informative Query Selection with Propagation

In the beginning of the querying process, the propagator is uninformed, having an empty set of supports $Q_\omega(\cdot_y|\cdot_x, \{\})$. However, as more queries and observations are being made, the support $\bar{o}$ grows and the propagator becomes more informed by being able to extrapolate between the future query x and a larger $\bar{o}$.

We now discuss the application of the propagator on querying problems, giving

formal proofs that the using the propagator is a faithful approximation to the **gold-standard** acquisition function.

7.1 Active Diagnostics

In active diagnostics, the goal is to infer the hidden parameter $\theta \in \Theta$ with as few observations $F(x_1, \Theta) \dots F(x_n, \Theta)$ as possible, where these observations are made adaptively. Constructing the optimal set of queries is intractable. In practice, given a set of past observations $\bar{o}$, one selects the next best query by maximizing mutual information between the query's outcome Y_x and the hidden parameter Θ. Formally, the **gold standard** acquisition function of active diagnostics is:

$$a_{\bar{o}}(x) = I(\Theta, Y_x | \bar{o})$$

Which is to say, given observations $\bar{o}$, select a query x such that the mutual information between its outcomes, Y_x, and the distribution of functions, Θ, is maximized.

In prior works, the optimal selection is computed by re-writing the definition of mutual information as a difference of entropy reduction:

$$\operatorname*{argmax}_x H(\Theta|\bar{o}) - H(\Theta|Y_x, \bar{o})$$

Which is to say, the mutual information between Θ and Y_x can be measured by the reduction of entropy of Θ given Y_x. Note that the first term $H(\Theta|\bar{o})$ is constant over x, thus we can omit it within the argmin_x operator, obtaining:

$$= \operatorname*{argmin}_x H(\Theta|Y_x, \bar{o})$$

However, this acquisition function is difficult to compute because it involves an entropy computation over the space of possible functions Θ, which is intractable in practice if Θ is sufficiently complex (for instance, of combinatorial complexity).

Using the Propagator In this thesis work, we opt for a symmetric, but different re-write of mutual information:

$$\underset{x}{\operatorname{argmax}}\, H(Y_x|\bar{o}) - H(Y_x|\Theta, \bar{o})$$

Which is to say, the mutual information between Θ and Y_x can also be measured by the reduction of entropy of Y_x given Θ. Let's focus on the second term:

$$H(Y_x|\Theta, \bar{o}) = \int_\theta H(Y_x|\Theta = \theta, \bar{o}) d\theta$$

Note that once given $\Theta = \theta$, the outcome Y_x is uniquely $F(x, \theta)$ which is to say $\forall\theta.\ \ H(Y_x|\Theta = \theta, \bar{o}) = 0$. Therefore $H(Y_x|\Theta, \bar{o}) = 0$. This observation allows us to drop the second term in our acquisition function, obtaining:

$$\begin{aligned} a_{\bar{o}}(x) =& H(Y_x|\bar{o}) \\ =& H(Q_\omega(\cdot_y|x, \bar{o})) \\ =& \begin{cases} -\sum_y Q_\omega(y|x, \bar{o}) log Q_\omega(y|x, \bar{o}) & \text{if discrete Y} \\ \frac{1}{2}\ln(2\pi e \sigma^2_{x,\bar{o}}) & \text{if continuous Y modeled as a gaussian} \end{cases} \end{aligned}$$

As a result, the acquisition process is simply going over all $x \in X$, and select the query that maximizes the uncertainty of $H(Q_\omega(\cdot_y|x, \bar{o}))$, which, if Q_ω faithfully approximates the propagation probability, is provably the optimal 1-step query for maximizing mutual information. Once a new observation $F(x, \Theta) = y$ is observed, it can be added back into the set of observations $\bar{o}' = \bar{o} \cup F(x, \Theta) = y$. Thus, we have justified that for the problem of active diagnostics, one should query the *most uncertain* query under the current propagation, which provably approximates the gold standard acquisition function.

7.2 Representative Subset Selection

In representative examples selection, one *starts with* a large, existing set of observations $\bar{o}_{big}$. One is also armed with a computationally expensive solver *solve* that can take a set of observations $\bar{o}$ and infers a satisfying $\theta \in \Theta$:

$$\theta = solve(\bar{o}) \implies \bigwedge_{(x,y)\in\bar{o}} F(x,\theta) = y$$

The typical approach to solve this problem is by using the CEGIS algorithm. In CEGIS, the acquisition function is computed by first solving the observations collected so far $\theta_{\bar{o}} = solve(\bar{o})$ then applying the following acquisition function:

$$a_{\bar{o}}((x,y)) = \begin{cases} 0 & \text{if } F(x,\theta_{\bar{o}}) = y \\ 1 & \text{if } F(x,\theta_{\bar{o}}) \neq y \end{cases}.$$

The drawback of using CEGIS as part of the acquisition function is that one is still invoking the expensive solver $solve(\bar{o})$ for every counter-example added to $\bar{o}$. So in case of $\bar{o}_{repr}$ needing to be large, CEGIS can be inefficient simply on the basis of calling the expensive *solve* function many times. Furthermore, the set $\bar{o}_{repr}$ returned by CEGIS is in fact *not* representative. For instance, the solver may return a $\theta_{\bar{o}}$ that is coincidentally consistent with $\bar{o}_{big}$ with just a single element in $\bar{o}$. Having under-representative sets of example can be a source of instability for the *solve* function.

Using the Propagator Consider the following set of parameters $\Theta_{\bar{o}}$:

$$\Theta_{\bar{o}} = \{\theta \in \Theta | \bigwedge_{(x',y')\in\bar{o}} F(x',\theta) = y'\}$$

Which denotes the set of $\theta \in \Theta$ that is consistent with every observation $(x', y') \in \bar{o}$. Abusing notation slightly, we use $\bar{o}(\theta)$ to denote the *event* that θ is consistent with $\bar{o}$.

We consider the following **gold standard** acquisition function:

$$\operatorname*{argmin}_{(x,y)} a_{\bar{o}}(x,y) = |\Theta_{\bar{o}\cup\{(x,y)\}}|$$

Which is to say, we seek to add an observation (x, y) such that, once added, minimizes the set of still consistent parameters. Performing this operation successively will prune away the greatest number of inconsistent parameters each turn, until finally, no more parameters can be pruned, and thus the resulting set $\bar{o}$ is representative. At first glance it seems like an impossible task: Solving for a single consistent $\theta_{\bar{o}}$ is already challenging, let alone *count* the number of consistent parameters! However, if we divide the acquisition function by $|\Theta_{\bar{o}}|$, which is a constant given $\bar{o}$, we obtain:

$$\begin{aligned}
a_{\bar{o}}(x,y) =& \frac{|\Theta_{\bar{o}\cup\{(x,y)\}}|}{|\Theta_{\bar{o}}|} \\
=& \frac{|\Theta_{\bar{o}\cup\{(x,y)\}}|/|\Theta|}{|\Theta_{\bar{o}}|/|\Theta|} \\
=& \frac{P(\bar{o}(\theta) \wedge \{(x,y)\}(\theta))}{P(\bar{o}(\theta))} \\
=& P(\{(x,y)\}(\theta) \mid \bar{o}(\theta)) \\
=& P(F(x,\theta) = y \mid \bar{o})
\end{aligned}$$

And wait a minute, that is *exactly* the propagation probability, which one can approximate well with the propagator. Which is to say, the acquisition function is:

$$\begin{aligned}
&\operatorname*{argmin}_{(x,y)} |\Theta_{\bar{o}\cup\{(x,y)\}}| \\
=& \operatorname*{argmin}_{(x,y)} P(F(x,\theta) = y \mid \bar{o}) \\
\approx& \operatorname*{argmin}_{(x,y)} Q_\omega(y|x,\bar{o})
\end{aligned}$$

And selecting under this acquisition amounts to picking the *least likely* example, provably approximating the gold standard acquisition function.

7.3 Bayesian Optimization

The most direct application of the propagator is bayesian optimization: Since the predictive posterior distribution has exactly the same form as the propagation probability, we can replace it with a learned propagator within the acquisition function:

Upper Confidence Bound Upper confidence bound explicitly addresses the exploration exploitation trade-off of understanding the oracle function and maximizing the returns on the queries. As stated earlier, upper-confidence-bound (UCB) has the following acquisition function:

$$a_{\bar{o}}(x) = \mu_{x,\bar{o}} + \sigma_{x,\bar{o}}$$

Which we can directly replace with $\mu_{Q_\omega(x,\bar{o})}$ and $\sigma_{Q_\omega(x,\bar{o})}$ predicted by the propagator.

Chapter 8

Case Studies

In this chapter we show empirical results of using the learned propagator in informative querying tasks outlined earlier: active diagnostics, representative subset selection, and bayesian optimisation. All case studies are lifted from my previous publications [16] and [17] [1], please refer to the original document for more details.

8.1 Active Diagnostics

In this section we evaluate the performance of our active diagnosis algorithm on several active diagnosis problems. Here we outline the problems and their characteristics.

Battleship A variant of the classic battleship game, where the goal is to infer the configurations of all 5 ships on a 10 by 10 board with as few queries to the board as possible. The hypothesis space consists of approximately 2^{38} different configurations, and each query is a (x,y) coordinate that results in "hit" or "miss".

Sushi A problem of preference elicitation, where the goal is to learn the full preference order of an user on 10 different sushi types. The hypothesis space consists of 10! potential full rankings, and each query is a pair-wise comparison between 2 sushi types that results in "yes" or "no".

[1] so you have some more cool pictures to look at

Network We consider a fault localization task on a network of 100 nodes organized as a tree, where each link has 2% chance of failure. The task is to learn an efficient scheme for querying pair-wise connectivities to localize the failure with as few queries as possible. The hypothesis space consists of all 2^{100} combinations of failures. The query is a pair-wise connectivity check, restricted to all 99 direct link checks and 300 additional pairs of fixed nodes with measurement equipment.

8.1.1 BattleShip Variant

The task is to infer the locations of all the ships with as few queries as possible. The board is a 10 by 10 grid, with 5 ships of size $2 \times 4, 1 \times 5, 1 \times 3, 1 \times 3, 1 \times 3$ placed randomly with arbitrary horizontal or vertical orientations. Adjacent placements are allowed, but the ships may not overlap with each other. See Figure 8-1 for an example board. The agent selects the most confusing query and updates its belief with the resulting observation. Figure 8-1 illustrates the propagated belief space at various numbers of observations. Note that without any observations, our model predicts that a ship is more likely to be located near the center of the board; In the case of 22 observations, our model queried a "hit" on the lower left without completely observing the 1×5 long ship in the middle of the board.

Once the observations are collected, We consider two kinds of hypothesis delivery schemes to infer the hidden identity of the oracle function (i.e. the board configuration of the ships). In the first scheme we deliver the probabilities of all the observations using the propagator Q_ω, implicitly inferring the hidden locations of the ships. In the second scheme we use a constraint solver which solves for a board configuration that is consistent with the collected observations so-far.

For comparison we consider 2 baseline algorithms: The random sampler **rand** samples unseen coordinates at random. The **sink** algorithm samples unseen coordinates at random, and when it has found a hit, it queries all its neighboring coordinates until no "hit" coordinates can be found, then resumes random sampling. Both the random and sink algorithms initially mark all coordinates as "miss" and update them to "hit" when a hit has been recorded.

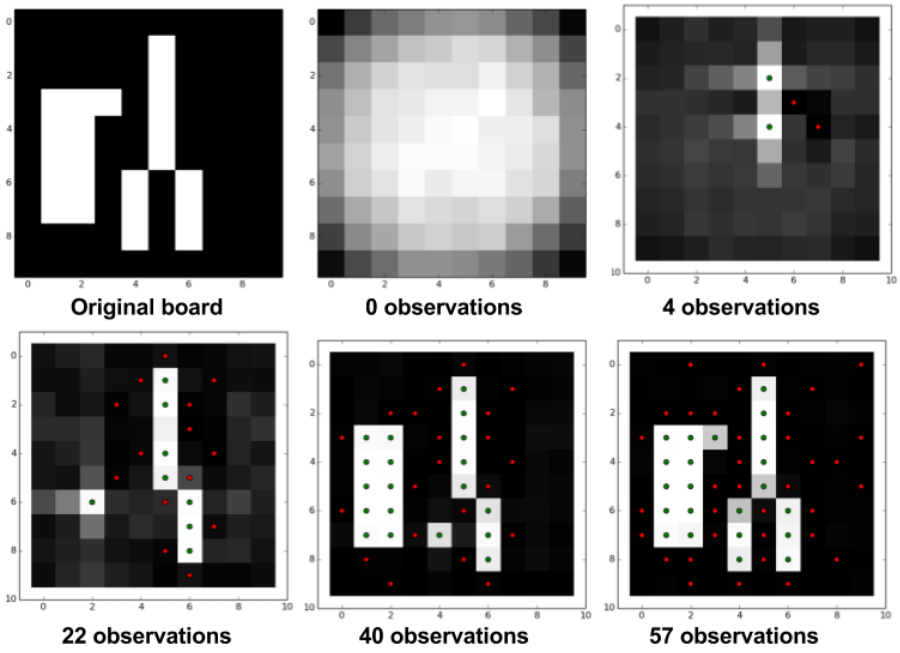

Figure 8-1: Belief space of observations on a particular board at various numbers of observations. Intensity indicates the probability of a coordinate being a hit, and colored dots indicates past observations: green for "hit" and red for "miss"

To evaluate performance under the first hypothesis delivery scheme, we let the propagator output the maximum likelihood guess for each coordinate. Accuracy is measured as a fraction of correctly guessed coordinates. Figure 8-2 compares accuracy across all coordinates as a function of number of observations; an accuracy of 1.0 means all coordinates are guessed correctly. In this experiment, we consider 3 different variant of the propagation function (all of the indexable X, multi-class Y kind): **oc_1** is the single hidden-layer fully-connected model originally described, **oc_0** is the 0-layer neural-network model (logistic regression), and **oc_cnn** contains a convolutional neural-network layer before a fully-connected hidden layer. As we can see, the random algorithm improves linearly as expected, the sink heuristic performs better than random, and our approach OC performs the best, with **oc_1** and **oc_cnn** performing better than the logistic regression **oc_0**. Figure 8-3 considers the same experiment except a 10% observation error is introduced, under this condition all 3 variants of the OC similarly, and more robust to noise than the baseline algorithms.

To evaluate the performance under the second hypothesis delivery scheme, we

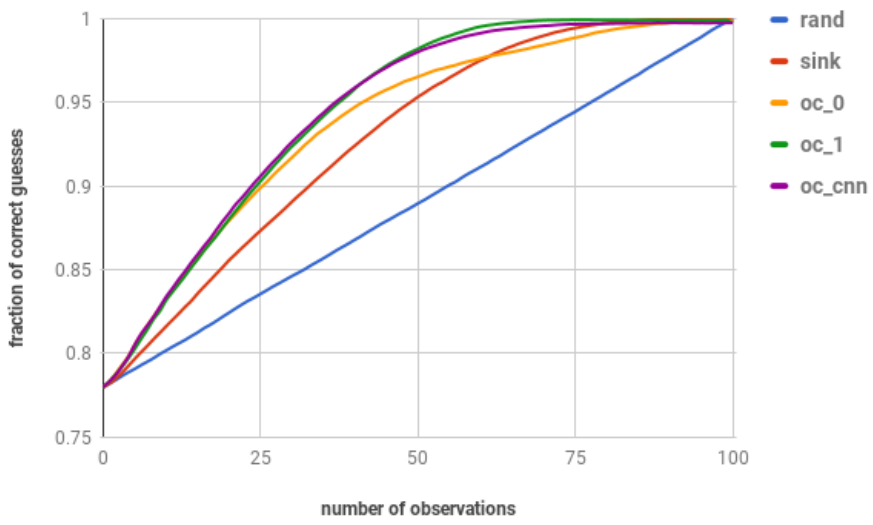

Figure 8-2: Comparison of our algorithm **oc** against the 2 baselines. Accuracies are averaged over 1000 randomly generated boards

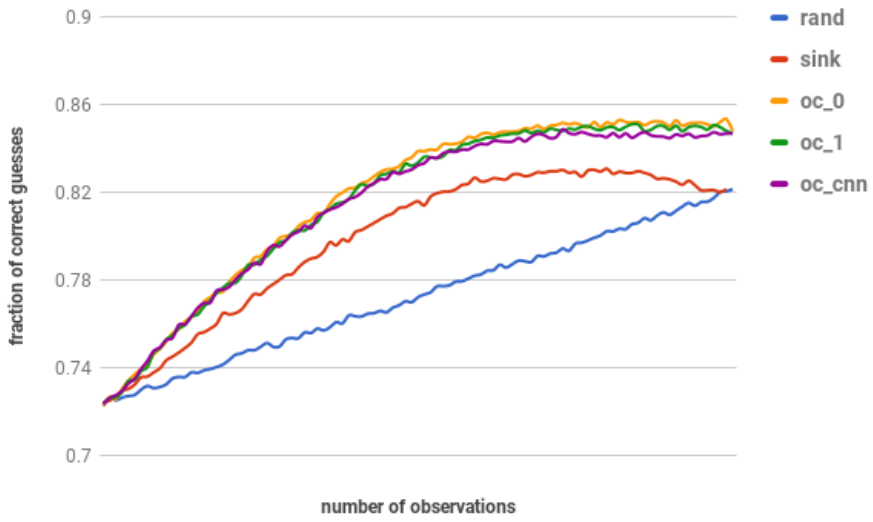

Figure 8-3: Comparison of our algorithm **oc** against the 2 baselines. Accuracies are averaged over 1000 randomly generated boards with 10% chance of observation error

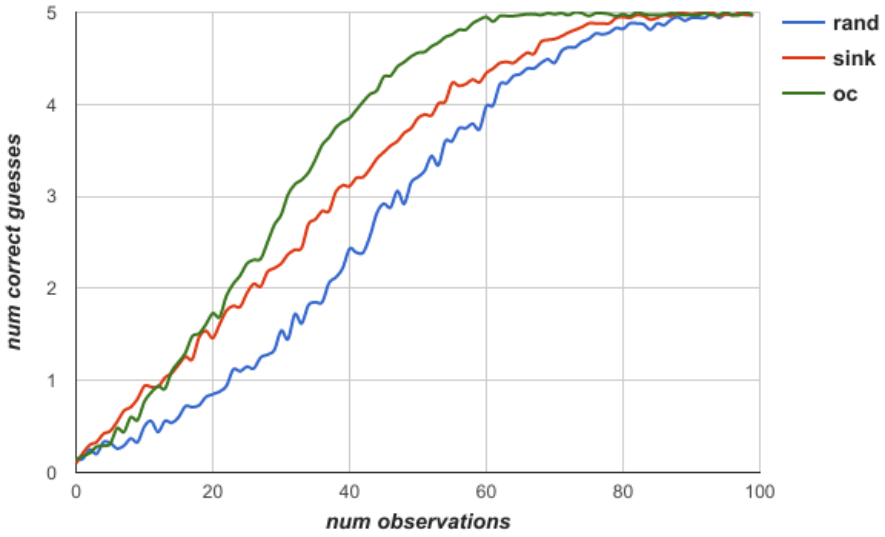

Figure 8-4: Comparison of our algorithm **oc** against the 2 baselines when using a constraint solver for hypothesis delivery. Accuracies are averaged over 100 randomly generated boards. Only the single-hidden-layer propagation network is considered here.

use a constraint solver to produce a hypothesis in the form of ships' locations and orientations, using observations collected by **rand**, **sink** and **oc** as constraints to the hypothesis. We then measure accuracy by the number of correctly predicted ship's locations and orientations: 5 means all 5 ship's locations and orientations are correctly produced by the constraint solver given the observations collected. Figure 8-4 compares number of correctly guessed ships as a function of number of observations. As we can see our algorithm OC performs the best, followed by sink with random performing the worst. Note that constraint solvers are known to produce arbitrary hypotheses that satisfy the constraint in an under constrained system, yet despite this, OC was able to consistently out perform the baseline algorithms.

8.1.2 Sushi Preference Elicitation

The sushi data set [11] contains 5000 user preferences for 10 kinds of sushi expressed in *full rankings*. The dataset also contains feature vectors describing each individual type of sushi and describing the users, which in this study we omit. Our task is the following: Given a new user, how to infer the full ranking of this user with as few

pair-wise comparison queries as possible? Naively, this is a sorting problem where one can obtain the full ranking of any permutation of items with $O(n \log n)$ comparisons. However, the preference orderings are not uniformly random: a preference of eel over tuna may indicate a user's liking of cooked sushi over raw sushi. We evaluate accuracy by using the Kendall correlation: a value of 1.0 means all pair-wise orderings of our prediction and the ground truth agree with each other, and a value of -1.0 indicates all pair-wise orderings are in disagreement.

There is no related work on this dataset that attempts to discover the full preference of a new user based on pair-wise queries to the new user. Thus, for comparison we consider various in-place sorting algorithms as baseline. Each time a pair-wise comparison question is made, we take a snapshot of the current array and extract pair-wise orderings from it.

The 5000 user preferences are split into a 2500 preferences training set and 2500 preferences testing set. In order to train our propagation function to handle novel permutations not seen during the training set, we augment the training set by a set of randomly sampled permutations in addition to the 2500 preferences.

For this experiment, we can measure performance directly as Kendall correlation without delivering an explicit ordering as the hypothesis (If one wishes one can easily compute a full ordering from all pair-wise orderings). Figure 8-5 compares our observation collection algorithm **oc_0** and **oc_1** against various in-place sorting algorithms: BubbleSort **bsort**, QuickSort **qsort**, and MergeSort **msort**. As we can see, even without any observations OC was able to obtain a Kendall correlation of 0.3, indicating the underlying distribution for preferences is not uniform; by comparison, the baseline sorting algorithms make no assumptions on the underlying distribution, thus starts with a correlation around 0. Our scheme was able to infer the full ranking of any user in 26 queries, beating the performance of **qsort**, which takes 40 queries.

We also considered the case where the query has a 10% chance of error, the result is shown in Figure 8-6. Note the OC algorithm is robust in the presence of noise as it improves its pair-wise preferences incrementally with each observation (a property also shared by **bsort**) whereas deterministic baseline such as **qsort** and **msort** sorts

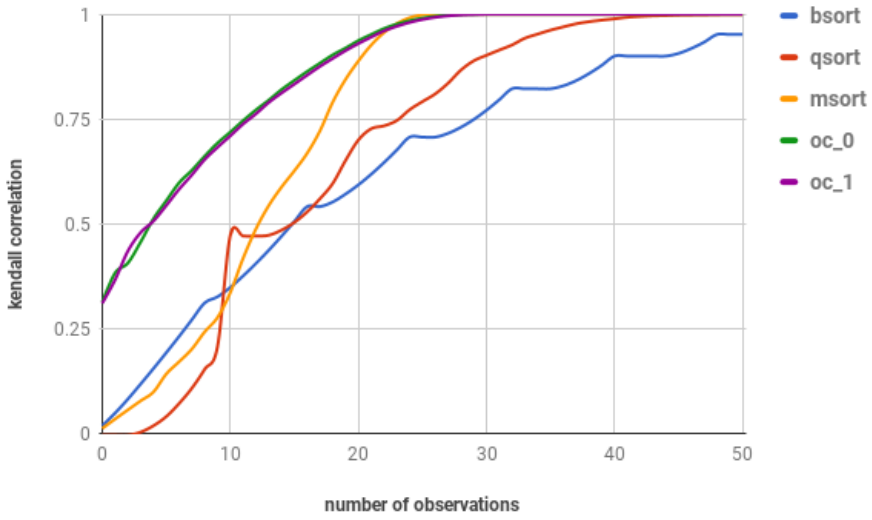

Figure 8-5: Kendall correlation as a function of number of queries averaged across 2500 testing examples

the preferences assuming all observations are perfect.

8.1.3 Network Fault Localization

We consider the task of fault localization on a network of nodes organized as a tree. The network without any failure is shown in Figure 8-7. There are 100 nodes in this network with 99 direct links, forming a spanning tree. The hypothesis space consists of failure cases where each direct link has a probability of 2% of failure. The query is a pair-wise path connectivity check, restricted to all 99 direct link checks and 300 additional pairs of fixed nodes.

For hypothesis delivery, we use the learned propagation function Q_ω to output the probability of failure on the 99 directly connected pairs of nodes. To measure accuracy, we use the maximum likelihood guess for these pairs and measure the fraction of correctly diagnosed link failures.

We compare our approach with the naive localization scheme **rand** that randomly picks an unobserved direct link and checks if it is disconnected. The random scheme will always be able to diagnose all the link failures in 99 observations. However, using propagation can leverage the network structure to learn an efficient querying

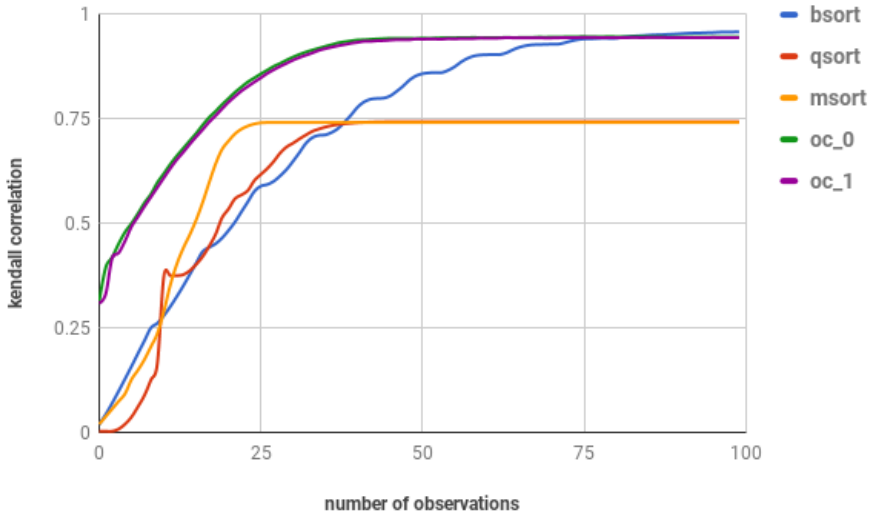

Figure 8-6: Kendall correlation as a function of number of queries averaged across 2500 testing examples, where each query has a 10% chance of error

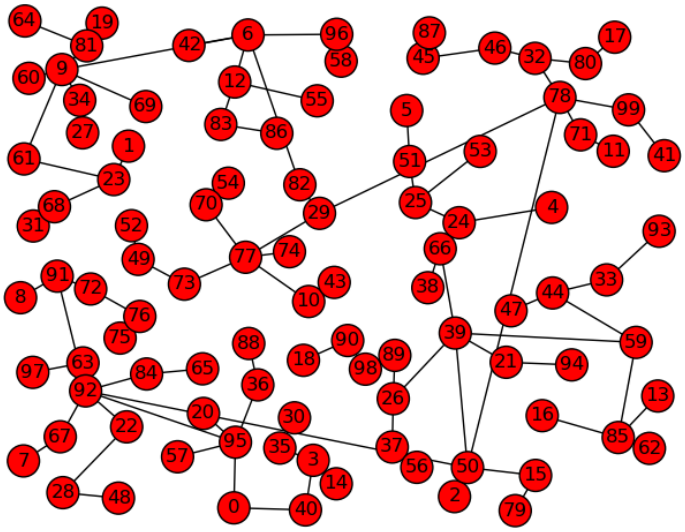

Figure 8-7: The network without any failure

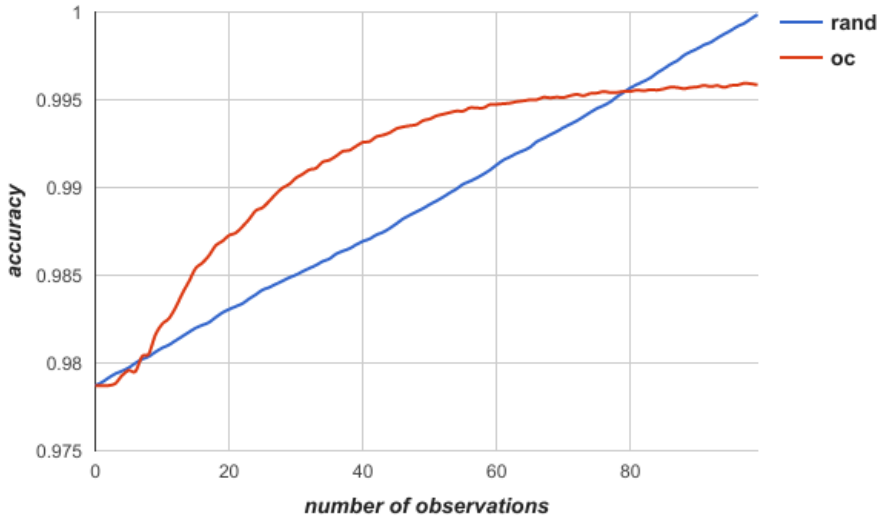

Figure 8-8: Accuracy of link failure diagnosis averaged across 1400 random instances

scheme by utilizing connectivity queries. Figure 8-8 shows accuracy of using propagation (labeled **oc**) against the random algorithm. We see **oc** outperforms the random algorithm most of the time, reaching 99.5% accuracy in 60 observations. Using propagation performs worse than the random scheme for less than 10 observations because the agent using propagator does not query the direct links for the first few observations: The 300 additional pairs can yield more information without pin-pointing an exact failure. Using propagation again performs worse than the random scheme past 80 observations. This is a consequence of the 1-step greedy approach: the agent maximizes information gain one observation at a time, but the set of observations selected by the random approach (all direct links) is the optimal set of size 99.

8.2 Representative Subset Selection

The efficacy of using a learned propagator for representative subset selection is evaluated against two criteria: First, the representativeness of our selected subset is explicitly measured (the 1st experiment); Then, the time/stability improvement of using such a subset is measured against strong baselines (the 2nd and 3rd experiment).

8.2.1 Representative Subset for Total Orderings

This experiment explicitly measures the representativeness of the subset selected by our algorithm on the task of ordering synthesis: Given a dataset of pair-wise ordering relations, $\bar{o}_{big} = \{a < b, a < c, b < d, c < d, d > a, c > a\}$, the task is to synthesize any total-ordering that is consistent with $\bar{o}_{big}$, for instance, (a, b, c, d) or (a, c, b, d) both are consistent orders with respect to $\bar{o}_{big}$. This task is useful because the optimal representative subset can be constructed as a Hasse diagram by pruning transitive relations: $\bar{o}_{repr} = \{a < b, a < c, b < d, c < d\}$. Thus, we can measure both the representativeness and optimality of our selection algorithm based on propagator. In this experiment, we set $n = 10$ and give our selection algorithm a dataset of size 30% to 100% of all possible pair-wise orderings. The neural network for the propagation function is again the indexable X, multi-class Y type.

We test the representativeness of our approach against the following baselines: **cegis**, **random x**(randomly select x percent of dataset)[2], and **hasse** (the optimal construction). The measurement of average subset size and fraction of representative subsets is shown in Figure 8-9. As we can see, **our** approach selects about twice as many examples as the optimal subset, and 85% of the times our subsets are representative. By contrast, **cegis** and **rand35** fails to discover any representative subset, while **rand80** discovers representative subset only 30% of the times despite sampling 80% of the total data. Figure 8-10 visualizes the chosen pair-wise orderings on a particular dataset **all**. This dataset specifies a unique total-ordering, which **hasse** was able to concisely represent with the minimal representative subset (bottom). **our** approach also discovers a representative subset, albeit with a few extra redundant relations. By contrast, **cegis** and **rand35** fail to discover a representative subset, as their subsets lack the relationship between elements 7 and 0, which are adjacent in the total ordering.

[2]we use 35% because it matches our average subset size

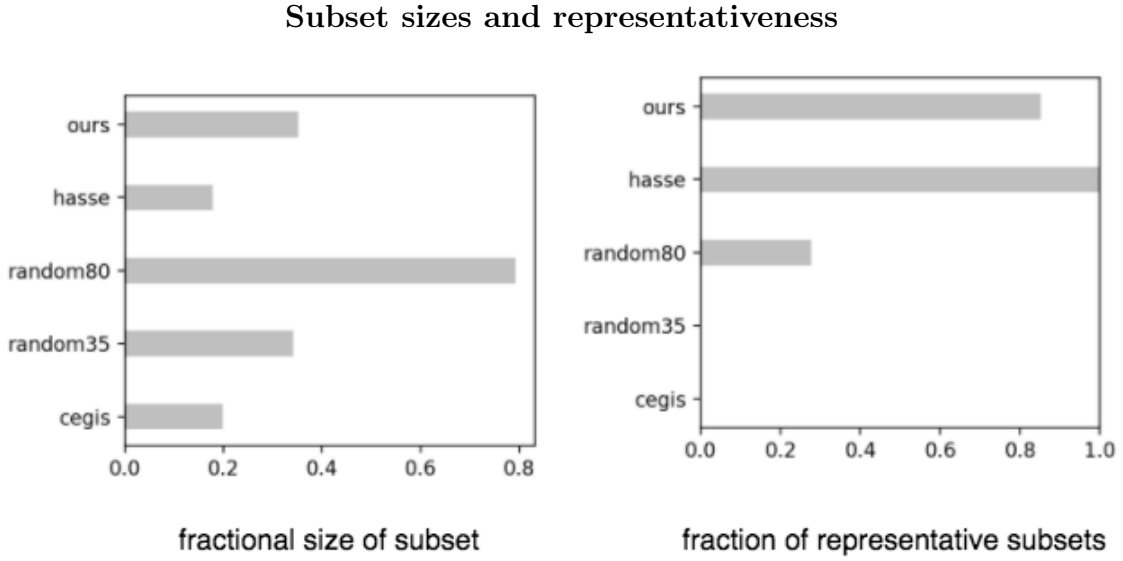

Figure 8-9: Our approach discovers representative subsets 85% of the times while sampling 2× the optimal subset size. Measured on 500 datasets drawn from randomly sampled total orderings

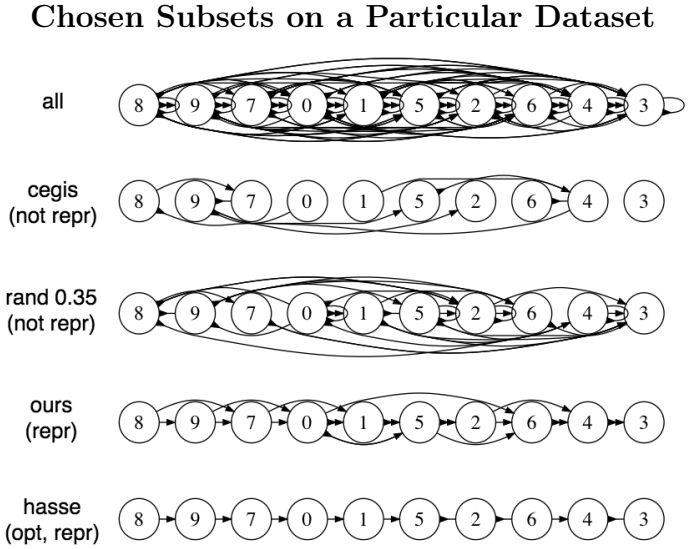

Figure 8-10: Chosen subsets on a particular dataset **all**. A subset is representative if it contains all adjacent pair-wise ordering

8.2.2 DFA Synthesis

This task is to synthesize a deterministic finite-state automaton (DFA) from a set of accepted and rejected strings. The space of hypothesis contains DFA of 6 states over a binary alphabet of 0 and 1 with a single accept state. The hypothesis space of total possible DFAs is of size $6^{12} = 2.18 \times 10^9$. 1000 strings of variable length between 5

and 10 were provided as the dataset for each synthesis task, the experiment consists of 400 tasks. We use the factorized propagation architecture for this task, where given a new example string *str*, its predicted outcome of accept or reject is conditioned on only the top 10 closest prefix and suffix matching examples from the subset $\bar{o}$. The neural network architecture is a simple feed-forward neural network that predicts the accept/reject label of *str* directly.

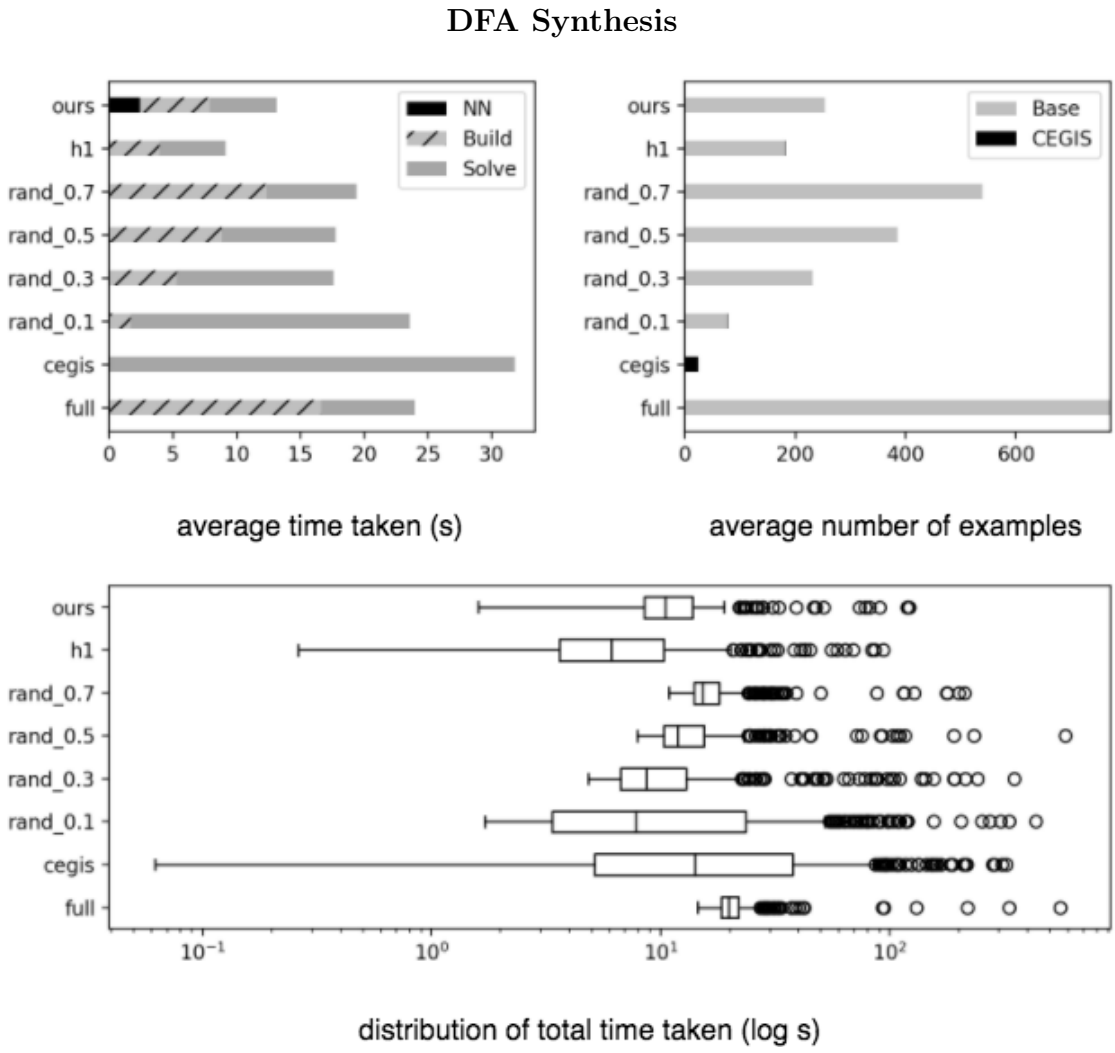

Figure 8-11: Time performance on DFA synthesis. **our** approach nearly matches the crafted heuristic **h1**, which constructs a suffix-tree over the entire dataset $\bar{o}$, and out-performs all other baselines.

We measure performance against the following: **full** (all examples in $\bar{o}$), **cegis**,**rand_x** (initialize CEGIS with a random x fraction of data), **h1** (a heuristic that construct

a suffix-tree over the entire dataset, see supplementary). Figure 8-11 (top,left) shows the comparison of performances in average time. As we can see, the heuristic subset collection **h1** performs best on average, but **our** approach (querying the least likely example predicted by the propagator) comes in close. As we can see, **ours**, **h1**, **full** have similar solve time, which we can infer that **our** approach and **h1** have found a well-constraining representative subset. This is in stark contrast to **cegis** which explodes in solve time with hardly any examples chosen. In terms of stability (Figure 8-11 (bot)), **our** approach also closely matches that of the heuristic, whereas all other algorithms (except **full**) suffers big variance in total time, likely a result of performing synthesis on under-representative subsets. Figure 8-11 (top,right) shows the average number of examples in the collected subset, we see that using a learned propagator outperforms randomly selected subsets of any size.

8.2.3 Programmatic Drawing Synthesis

We evaluate our approach on 250 randomly sampled 32×32 pixel renderings created from a diagram drawing function. The program synthesis task is to synthesize a program that can recover the diagram, where each pixel is an input-output example (See Figure 8-12). The drawing function has a parameter space of size 1.31×10^{23}. We use a propagator to estimate the probability of a pixel being white conditioned on the pixels selected so-far ($\bar{o}$). We again use the factorized propagation function, where each pixel prediction depends only on a 7×7 sliding window centered on it. The predicted pixel values from the propagator is shown in Figure 8-13.

We measure performance against the following: **full** (all examples are added), **cegis, rcegis, acegis** (different CEGIS flavours on how the counter examples are selected: canonical top-left most pixel, random, and a fixed but arbitrary order), **rand+cegis** (instantiate CEGIS with a random 20% subset), and **h1+cegis** (a heuristic that adds a pixel if any pixel within a 5×5 window has a different value). The results are shown in Figure 8-14. As we can see, **our** approach with a learned propagation performs best on average, beating all competitors on average time. One unexpected outcome is that **cegis** performs very well on this domain: We postulate

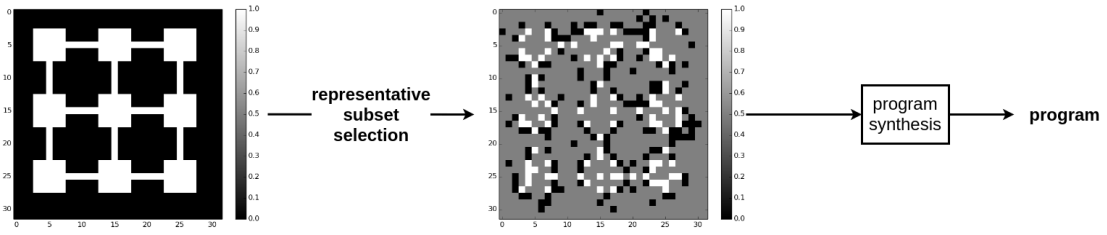

Figure 8-12: Given a 32x32 canvas where each pixel is an input-output example (i.e. the entire canvas is $\bar{o}_{big}$), we use a learned propagator iteratively select the least-likely example to construct a representative subset ($\bar{o}_{repr}$), which is then fed into the solver

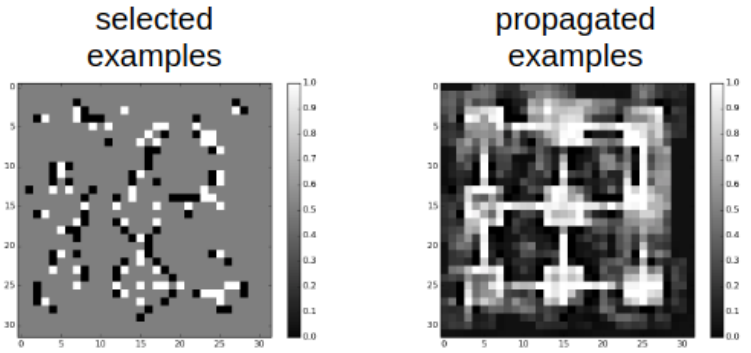

Figure 8-13: Given the selected pixels, the learned propagator predicts the rest

that the top-left-most counter-examples chosen by **cegis** happen to be representative as they tend to lay on the boundaries of the shapes, which is well suited for the drawing DSL domain. However, such coincidence is not to be expected in general: By making the counter example be given at random, or given at a fixed but arbitrary ordering, **rcegis** and **acegis** were unable to pick a representative set of examples and suffer in overall time. In terms of variance (Figure 8-14 (bot)), our approach was able to match the variance of **full** (clear representative) and **h1**+**cegis** (also representative as a 5×5 sliding window can distinguish squares and lines perfectly). However, **our** approach was able to discover representative subsets with a much smaller number of examples (Figure 8-14 (top, right)). Overall, our approach improves synthesis time and stability by providing the solver with a representative subset upfront.

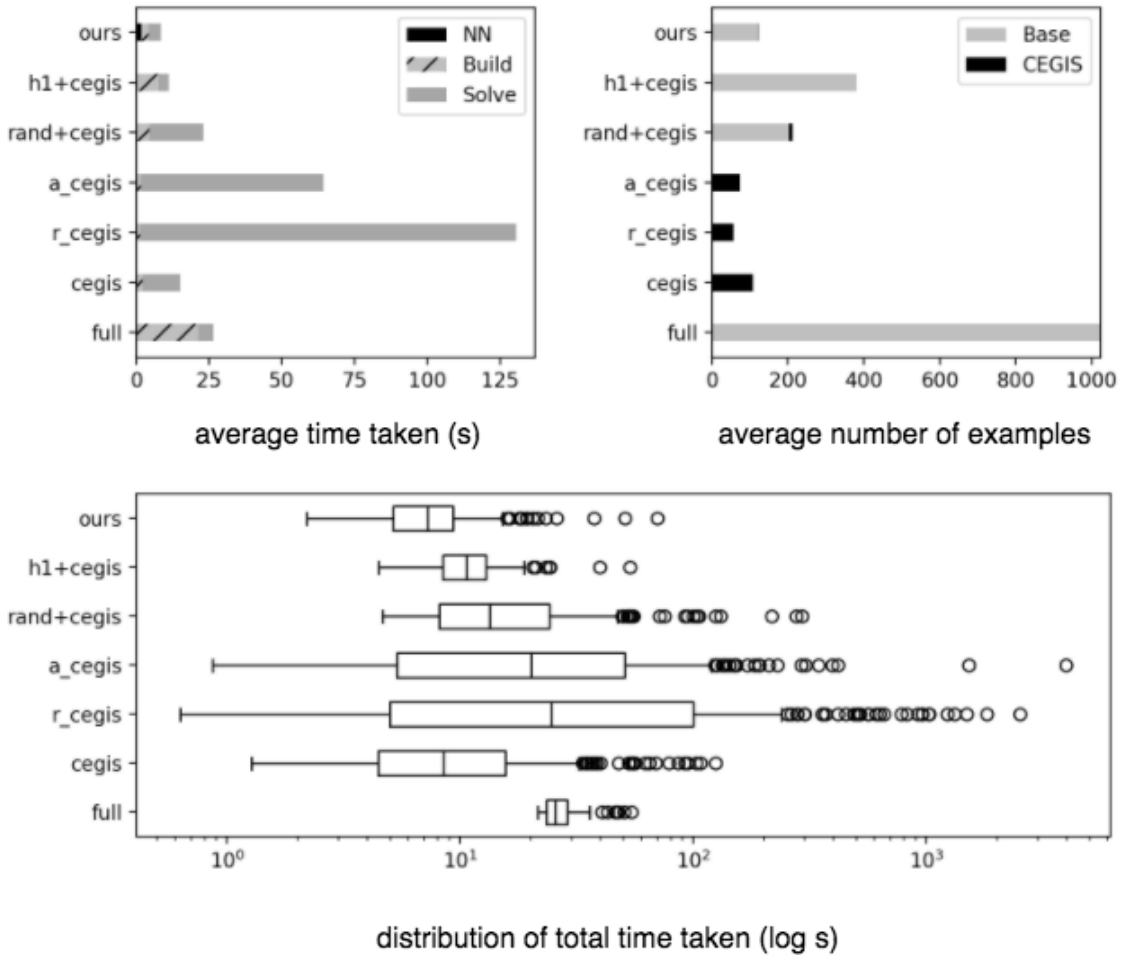

Figure 8-14: Time performance on programmatic drawing synthesis. **our** approach with propagation is best in average time, and achieves similar stability as **full** and **h1+cegis** with much fewer samples.

8.3 Bayesian Optimization

In this preliminary study, we investigate whether a learned propagation model can perform better than a gaussian process with an uninformed kernel. The task is to achieve the highest outcome with as few queries as possible. The space of oracle functions consists of continuous, peak-like functions that maps $[0, 1]$ to the real numbers. Moreover, the function consists of two correlated parts, where the left part $[0, 0.5]$ consists of straight line peaks numbering between 1 and 3, and the right part $[0.5, 1]$ is identical to the left part, except with some sinusoidal noise added. See Figure 8-15.

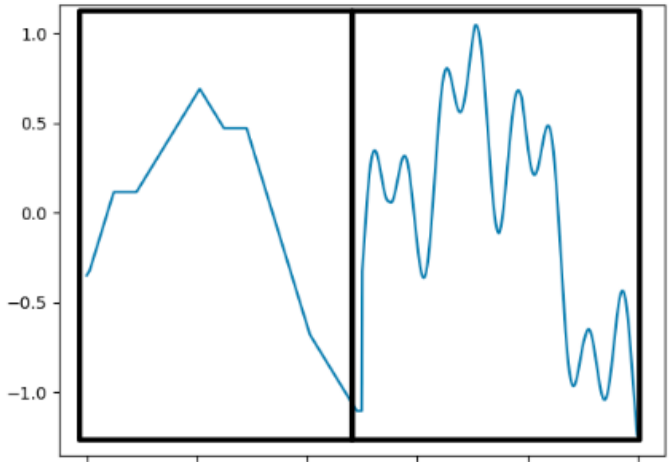

Figure 8-15: The oracle function consists of two mirroring parts, where the second part is identical to the first part except for an added sinusoidal noises

Figure 8-16 shows the comparison of using a gaussian process' as the predictive posterior, versus using a learned propagation function (using the continuous X, continuous Y architecture). Here, the blue line is the ground-truth function, and the red region denotes the area of 1 standard deviation predicted by either GP (left) or the learned propagation (right). As one can see, the learned propagation can model the correlations between the left-half of the function and the right-half, while the uninformed GP cannot model this. In a trial of 100 randomly sampled oracle functions, an upper confidence bound algorithm using the learned propagation achieves the highest outcome of 0.481, while achieving only 0.386 using GP.

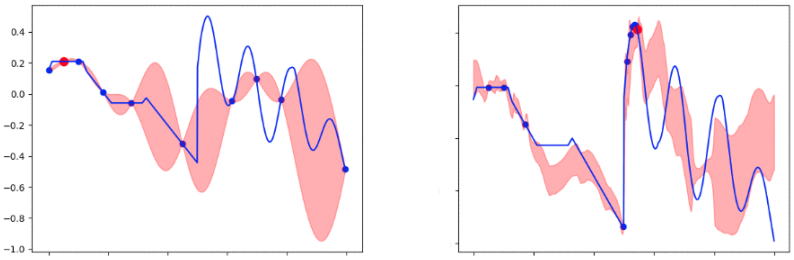

Figure 8-16: By using a learned propagator, the agent can make better posterior predictions on the function values, leading to better queryings for the optimal point

Chapter 9

Conclusion

In conclusion, this thesis attempts to use a learned propagation function to make informative queries, similar to how human make informative queries by relating past observations to future queries' outcomes. We've demonstrated that the propagation function can be readily learned by sampling observation data and forcing a neural network model to learn the relationships between these observations. Once the propagation function is learned, it can be adapted to form a suitable acquisition function, which provably approximates gold-standard next-best query. We've demonstrated that the learned propagation has superior performances on various informative querying tasks, either by making better queries, or achieving faster computation time.

Future Works One exciting future work is to amortize away the acquisition function itself, i.e. have a policy $\pi_{\bar{o}}$ that can directly select the next-best query. Also learning the space of observation functions F is a fascinating question, i.e. learning what set of questions one should use as communication channel between the agent and the oracle: Humans play 20 question so well precisely due to our flexibility in the kind of questions we get to ask, so let's learn to induce these questions!

If you've reached this far, I want to say: Thank you for reading my PhD thesis! As a disembodied piece of information embedded into these pages, I'm so happy someone would take their time and read it. :). Go forth and propagate! I should sleep it's late lol . . .

Bibliography

[1] Pankaj K Agarwal, Sariel Har-Peled, and Kasturi R Varadarajan. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52:1–30, 2005.

[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[3] Gowtham Bellala, Jason Stanley, Suresh K Bhavnani, and Clayton Scott. A rank-based approach to active diagnosis. *IEEE transactions on pattern analysis and machine intelligence*, 35(9):2078–2090, 2013.

[4] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[5] Trevor Campbell and Tamara Broderick. Bayesian coreset construction via greedy iterative geodesic ascent. *arXiv preprint arXiv:1802.01737*, 2018.

[6] Gereon Frahling and Christian Sohler. A fast k-means implementation using coresets. *International Journal of Computational Geometry & Applications*, 18(06):605–625, 2008.

[7] Alexander Gammerman, Volodya Vovk, and Vladimir Vapnik. Learning by transduction. In *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, pages 148–155. Morgan Kaufmann Publishers Inc., 1998.

[8] Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011.

[9] Alex Holub, Pietro Perona, and Michael C Burl. Entropy-based active learning for object recognition. In *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–8. IEEE, 2008.

[10] Jonathan Huggins, Trevor Campbell, and Tamara Broderick. Coresets for scalable bayesian logistic regression. In *Advances in Neural Information Processing Systems*, pages 4080–4088, 2016.

[11] Toshihiro Kamishima, Hideto Kazawa, and Shotaro Akaho. A survey and empirical comparison of object ranking methods. In *Preference learning*, pages 181–201. Springer, 2010.

[12] Andreas Krause and Carlos Guestrin. Submodularity and its applications in optimized information gathering. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(4):32, 2011.

[13] Tessa Lau, Steven A Wolfman, Pedro Domingos, and Daniel S Weld. Programming by demonstration using version space algebra. *Machine Learning*, 53(1-2):111–156, 2003.

[14] Hyafil Laurent and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17, 1976.

[15] Robert Nowak. Generalized binary search. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, pages 568–574. IEEE, 2008.

[16] Yewen Pu, Leslie P Kaelbling, and Armando Solar-Lezama. Learning to acquire information. *arXiv preprint arXiv:1704.06131*, 2017.

[17] Yewen Pu, Zachery Miranda, Armando Solar-Lezama, and Leslie Pack Kaelbling. Selecting representative examples for program synthesis. *arXiv preprint arXiv:1711.03243*, 2017.

[18] Irina Rish, Mark Brodie, Natalia Odintsova, Sheng Ma, and Genady Grabarnik. Real-time problem determination in distributed systems using active probing. In *2004 IEEE/IFIP Network Operations and Management Symposium (IEEE Cat. No. 04CH37507)*, volume 1, pages 133–146. IEEE, 2004.

[19] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294. ACM, 1992.

[20] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[21] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. *ACM Sigplan Notices*, 41(11):404–415, 2006.

[22] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66, 2001.