Methods for Teaching Hard Learning Problems

by
Carl Trimbach
B. S., Rutgers University, 2012
Sc. M., Brown University, 2017

A dissertation submitted in partial fulfillment of the
requirements for the Degree of Doctor of Philosophy
in the Department of Computer Science at Brown University

Providence, Rhode Island
October 2021

This dissertation by Carl Trimbach is accepted in its present form by
the Department of Computer Science as satisfying the dissertation requirement
for the degree of Doctor of Philosophy.

Date _____ _____

Michael L. Littman, Director

Recommended to the Graduate Council

Date _____ _____

Eli Upfal, Reader

Date _____ _____

Stephen Bach, Reader

Approved by the Graduate Council

Date _____ _____

Dean of the Graduate School

Vita

Carl Trimbach graduated Magna Cum Laude from Rutgers University in 2012 with a B.S. in Electrical and Computer Engineering. Under the mentorship of Dr. Rajiv Gandhi he studied for two years, attending classes at Rutgers University, Princeton University, and the University of Pennsylvania. In 2014, he began working toward his Ph.D. in Computer Science under Dr. Michael Littman.

This thesis is lovingly dedicated
to the memory of my stepfather,
Joe Hall.

Acknowledgements

First, I want to acknowledge Michael Littman, my advisor. With his guidance and advice, I have learned not only how to conduct research, but also how to be a better mentor and co-worker. I am eternally grateful to have had the opportunity to learn from him and benefit from his vast experience. The knowledge, experience, and confidence I gained while working with Michael are things I will rely on for the rest of my career. Thank you, Michael, for being the most sincere, caring, and understanding advisor and mentor.

I also owe thanks to the members of my committee, Stephen Bach and Eli Upfal. Their genuine interest in my work and the insightful conversations that resulted are a huge factor in any amount of success I achieve.

I would like to acknowledge my brilliant coauthors: Amy Greenwald, Joe Austerweil, James MacGlashan, Stephen Brawner, Royal Wang, Jun Ki Li, Mark Ho, and Elizabeth Hilliard. Working with these people on various projects was both inspiring and rewarding. I learned a lot from each of them and they have all become an important part of my time at Brown.

The entire Computer Science department is also deserving of recognition, particularly, Jane McIlmail, Lauren Clarke, Lori Agresti, and all of the various chairs during my years at Brown.

Rajiv Gandhi deserves more of my thanks than I can express. Without him, I quite literally would never have considered doing my PhD. Without his support and advisement I would have never had the means nor the courage to even attempt applying to graduate school. He materialized countless opportunities for me and others that directly resulted in me discovering an interest in machine learning and, more generally, computer science research. The entirety of my work at Brown was accomplished with the intention of trying to make him proud.

My friends have been an endless source of support and encouragement. Among them are Gwen LaMastro, Justin Spinozzi, Ronnie LaMastro, Andrew Babbin, Colin Johnson, and especially, my boyfriend, Nicky Castle.

My siblings, and four closest friends, all deserve acknowledgement as well. Joseph Trimbach, Jenna and Dave Robertson, and Gabrielle Trimbach have all loved, supported, and inspired me to continue along this path even when it seemed impossibly hard to do so.

Finally, I am unable to accurately express the depth of my gratitude to my four amazing parents. I consider myself infinitely lucky to have been blessed with such incredible parents and then blessed twice over with equally as wonderful step parents. Firstly, I acknowledge my mother, Joyce Hall,

who has always given me more unconditional support and love than I thought was humanly possible. I also thank my father, Julius Trimbach, who believes in my abilities and potential more deeply than even seems reasonable. My stepmother, Vicki Trimbach, has my gratitude for having always shown a sincere interest in all of my work and for never missing an opportunity to ask about it. Lastly, I want to thank my stepfather, Joe Hall, to whose memory this document and all of my work is dedicated. His words and actions incessantly reminded me of how proud he was of me, how much he loved me, and how supportive he was. The strength with which he lived his life inspired me daily and sustained my determination to complete this document.

Contents

List of Tables

List of Figures

Abstract of "Methods for Teaching Hard Learning Problems" by Carl Trimbach, Ph.D., Brown University, October 2021.

In both supervised and reinforcement settings, there exist learning problems that are hard due to having high computational or sample complexity. Researchers have shown, using standard models, that this issue causes certain classes of problems to be provably unlearnable. We propose new learning models that incorporate both iterative data augmentation techniques and human intervention to overcome these issues. In the supervised learning setting, we extend and enhance the standard learning model to include an iterative, round-based learning structure. Additionally, we introduce a benevolent, knowledgeable teaching agent to the model. With these adjustments, we are able to show theoretically that certain unlearnable classes are efficiently teachable. We achieve improved results for previously studied classes and present new results for another class in our model. Additionally, we provide evidence that non-expert human users can play the role of the teaching agent effectively. Inspired by our work in the supervised setting, we leverage the same intuitions to create a reinforcement-learning model that takes advantage of an iterative round-based learning structure. We are able to train agents to learn arbitrarily complex behaviors induced by temporally defined goals in a simple grid world. We are able to show that reinforcement given by non-expert humans is sufficient to get the learning agent to exhibit the desired behavior.

Chapter 1

Introduction

1.1 Motivation

One of the motivating goals of artificial intelligence is to create competent, useful learning agents that are capable of near-human or better performance. Even at a high level, however, comparing learning agents to humans feels like an unfair comparison. The power and compute ability of machines, especially in distributed systems common in today's world or looking forward to advancements in areas like quantum computing, seems to overshadow that of humans. This fact seems to put artificial agents at a huge advantage. On the other hand, however, there are some respects in which machines are at a great disadvantage to humans. For example, the ideal artificial agent would be expected to work "out of the box" with solutions for a variety of problems. These agents don't have the same benefit of a lifetime of real world experience, a massive amount of help from a support group of others like family and friends, nor an innate instinct about how to act and adapt in a changing environment. For these kinds of reasons, expecting human-level or better performance from our artificial agents seems unreasonable. Writing a script for life experience or hard coding intuition is not a feasible solution. While the comparison may be an unfair one, the goal of creating agents that behave and make decisions that are more "human-like" is still an important one. To have artificial agents that are accepted and trusted within our society, we need to, among other things, make them feel like a natural fit.

One natural path to achieving this goal is to train our artificial agents in a way that is inspired by the way humans, themselves, learn. My work takes inspiration from this idea. While I don't claim my algorithms directly mirror any particular real world learning scenario, they do take inspiration from concepts like human learning, curriculum design, animal training, and other natural concepts. The main idea behind this work is that it is possible (and often times advisable) to have an agent learn the solution to a problem in an organized, structured, logical way. Decomposing a problem into smaller sub-problems that are easier to understand and more accessible to learn is one aspect of the human learning process that should be formally introduced into a machine-learning paradigm.

Take, for example, our educational system in which we teach children math, science, language, etc. In a math class, students might learn about a complex concept like long division. The process is explained to them, but they are also shown multiple worked examples to get a thorough understanding of how the process of long division operates. These worked examples are analogous to the training data we might show a learning agent. Like the student, the agent sees a set of data points (math problems) along with a set of labels (the worked answer). In machine learning, we expect that, with enough data, the agent will be able to extrapolate the rule that determines how to relate the data points to the label, just like students are expected to acquire the knowledge of how to apply the process of long division to arbitrary problems. Later, we evaluate the success of the students via methods like homework assignments and tests by providing them problems without worked solutions. We expect that, at this point in time, the student is able to work through the answers independently. Similarly, we can assess the success of the learning agent by providing it unlabeled test data that we expect the agent to label. This scenario illustrates the process of supervised machine learning.

However, we can extend the scenario further to help motivate and inspire our work on Iteratively Moderated Probably Approximately Correct Teaching (IMPACT). For now, we will discuss IMPACT at a high, intuitive level and leave formal definitions for Chapter 3. One indispensable feature of teaching students a more complex concept, like long division, is the fact that, in the past, the students have already learned all the important requisite concepts to be able to grasp the components that make up long division. For example, more basic skills like addition, subtraction, multiplication, and memorization of simple division facts are all skills that would allow a student to build up to carrying out the full process of long division. The idea here is we don't expect students to understand or even be capable of learning complex concepts without first mastering the more fundamental concepts that underlie it. In that same vein, IMPACT teaching does not attempt to have learners learn complex concepts without first being trained on simpler, easier-to-learn concepts. The IMPACT model takes this idea and applies it to the standard (PAC) learning model to create a new model where agents learn over time, starting with simpler, easily learnable concepts to build a solid foundation for more complex and intricate concepts.

Another concept from education that inspires aspects of our model is called "instructional scaffolding". Scaffolding in this sense is the practice of a teacher providing plenty of support while teaching a new concept. As the student gains competency, the teacher's role diminishes and the student becomes more independent. Successful scaffolding in our models relies heavily on the ability of the teacher, be they human or artificial, to design appropriate curricula for the student or agent. One school of thought says that the best way to teach a student is to understand what they already know and teach them something that is difficult enough that they cannot figure it out alone, but simple enough that they are able to learn it with aid from a teacher and their current understanding. This difficulty "sweet spot" of real-world learning problems is known as the *zone of proximal development* Vygotsky [1980], a phrase coined by the developmental psychologist Lev Vygotsky. A diagram illustrating the zone of proximal development can be seen in Figure 1.1.

Concretely, this zone of proximal development contains all the knowledge that a person is able to be taught presently, but not learn independently. Attempting to teach information that is contained in the innermost circle does nothing to aid the learner, since it could have been learned independently, anyway. Consequently, they are no more capable of learning other information than they were before. An example would be teaching a child who already knows how to multiply and divide to count to ten. Similarly, attempting to teach a learner something from the outermost ring is also inconsequential as the learner will be unable to understand knowledge that is so far out of reach, even with assistance from a teacher. This situation is analogous to trying to teach calculus to a toddler who is only just learning to count. Attempting to teach a learner something in the zone of proximal development, however, can be very effective. It is akin to teaching long division to a student who already has a solid understanding of multiplication and division facts. They have the requisite knowledge and the ability to grasp and apply the new concept. As a learner gains more knowledge and expands their understanding, the two innermost circles from Figure 1.1 grow and consume more and more of the outermost circle. We will attempt to apply this idea to our work in Chapter 4. Finding this zone is also an interesting question from a machine-learning standpoint. The analogy to the "zone of proximal development" in machine learning can be thought of as the boundary of a hardness class. That is, anything that an agent can learn with an IMPACT-style teacher, but cannot learn otherwise comprises a class of concepts that is IMPACT teachable, but not independently learnable. Discovering more about this class of concepts is another motivation for studying IMPACT.

While the analogy between machine teaching and the learning model we have as humans in a society is informative and inspiring, it is not perfect nor does it encapsulate all the intricacies of machine teaching. One big disadvantage that our machine-teaching model has that humans do not is the inability to directly communicate with the learner. In our society, the direct communication through speaking and writing is fundamental to education. It would be exceedingly hard to teach students with whom you do not share a language. However, it is often the case, and certainly the expectation, that the average lay-person is not a computer scientist or programmer who is able to write code to directly communicate their intent to artificial agents. Even for those who can code, explicitly stating exactly what is desired is often not an easy task. We can, however, address these issues by again drawing on inspiration from the our world to inform the design of artificial agents. Specifically, we can look to things like animal training or teaching basic skills to infants for inspiration on how to handle teaching agents with whom we do not share a direct line of communication.

There are many parallels between both the task and the methods we use for training animals and for training artificial agents in an IMPACT-like way. One of the first ideas that comes to mind when people think about training animals is behavior reinforcement. This idea highlights that rewarding good behavior or punishing bad behavior can help bring about more or less of that behavior, respectively. Positive reinforcement, be it in the form or treats, praise, or just attention, can be a huge motivator for our pets to continue to perform some desirable behavior. This reinforcement can serve as a stand-in for language when we train animals that will not fully understand what we say when we speak to them. This idea of using positive and negative reinforcement to convey a notion

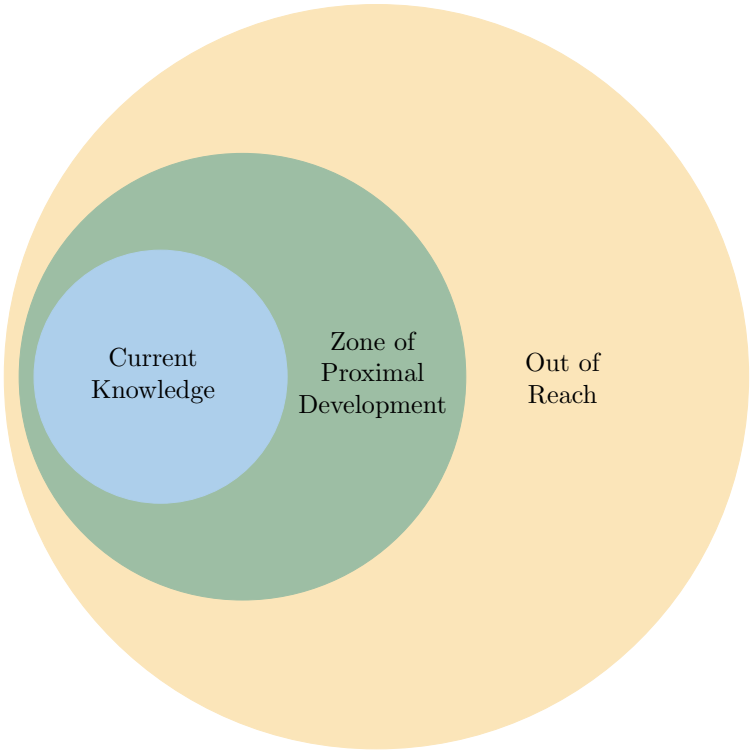


Figure 1.1: The above diagram illustrates the zone of proximal development of a learner. The innermost blue circle represents the space of information that a learner currently knows or can learn without assistance. The middle green circle denotes the zone of proximal development. It is the set of knowledge a learner can attain with assistance. The outermost yellow circle represents all of the knowledge that is out of reach. Even with assistance, a learner cannot (yet) understand this information. As new knowledge is gained, the zone of proximal development expands.

of desirability to an agent is used heavily in Chapter 7. Just like with humans, it is often necessary to slowly build skills over the course of an entire curriculum when doing things like training a dog to follow new commands. Typically, to teach a puppy to roll over, you first teach it to sit, then to lay down, and then to roll on its back, for example. As before, these small skills help establish a knowledge base that the animal can draw onto begin to master harder, more complex skills like rolling over on command. Chapter 7 explores this idea, as we try to teach robots complex skills over time using reinforcement techniques we mentioned earlier.

Similar to animals, infants do not have the language skills to communicate plainly by speaking or writing. This fact, too, complicates the process of teaching them even basic skills needed for independent living. However, evidenced by the fact that society has existed for many generations, they are able to learn all the skills needed to survive. Many of the skills people learn before being old enough to be taught are learned through mimicry. Parents, friends, siblings, and other people interact with and around the infant and the infant is able to mimic their actions and slowly build up the skills to navigate and explore the world. Teaching through example is exactly the framing we use to present the problem of training complex cleaning rules to robots in Chapter 6. In our work, the close connection to natural, relatable teaching methods help us to present teaching tasks to lay-people in an understandable way. Society's means of teaching the next generation, our pets, or each other not only motivate our work, but also help to involve and explain it to non-experts.

Having explored all of these sources of inspiration for our work and discussed how it motivates the problems we explore and the solutions we develop, it is important to remember that this link is intuitive and high level. We do not imply here or elsewhere in our work that our methods and algorithms are in any way an implementation of the real world or an accurate model of human or animal learning. Instead, our work has been *inspired* and *motivated* by the natural world. Our theorems and experimental results are not presented as proof of how humans or animals interact with their environments. Instead, we want to showcase that we can draw analogy to the problems and issues we face in real world learning scenarios. Like the artificial agents at the focus of our work, we can draw on our experience of this real world learning we have encountered, and explore the potential of drawing inspiration for our research.

1.2 Teaching Artificial Agents

As we will see in Chapter 2 and Chapter 4, the issues we discussed become particularly relevant when we begin looking at problems that are in the zone of proximal development. That is, we know that there are interesting, useful learning problems that exist that are too difficult for a learning agent to solve without help. The focus of our work is to formalize what types of help are most effective, how to apply the ideas we discussed that are inspired by real world examples, and how to measure and evaluate these ideas in meaningful ways. We explore problems both that are hard in the theoretically provable sense as well as challenging applications that take a meaningful step towards real world systems. We will identify several such problems, propose an algorithm that incorporates

assistance via a teaching entity, and evaluate the effectiveness and efficiency of those algorithms. While each of the problems we tackle is distinct in its details, and interesting for its own reasons, we provide a standard set of tools that can be applied to these and similar problems. *Through the use of tools such as iterative methods, compositional concepts, and human intervention, we can create learning systems that are competent and capable of solving provably hard learning problems.*

1.3 What to Expect

In the the following Chapter 2, we will provide the relevant background knowledge needed to understand the learning problems and solutions we discuss in this dissertation. Chapter 3 will formally define the IMPACT model, the single most significant contribution of our work. We will discuss the process of creating and developing this model. Additionally, we establish a template for creating algorithms in this setting and proving their correctness and efficiency. Next, we will introduce and discuss an important learning problem known as binary classification in Chapter 4. In that chapter we present three distinct classes from the literature that we will teach an artificial agent accompanied by their respective proofs of correctness, using the templates from Chapter 4. Chapter 5 introduces a new class for us to teach. Again we present proofs for our algorithm with Chapter 4's templates. This time, however, we also provide simulations of our algorithm that we run as empirical evidence of the efficacy of our model.

Moving to Chapters 6 and 7 signals a subtle shift to more human-centered, behavioral research. Chapter 6 continues to explore problems in supervised learning, but this time includes a real, non-expert human as part of the process. We evaluate the possibility of lay people to play the role of the teacher in our model. In Chapter 7, we move away from supervised learning and into the space of reinforcement learning. In this chapter we present an IMPACT-style approach to the problem of teaching an agent to have behavior motivated by an arbitrarily complex logic formula.

Finally, Chapter 8 gives a high-level overview of our major contributions, an comparison of our work to some closely related work in the literature, and a potential path for future work.

Chapter 2

Background

This work as a whole builds on fundamental concepts in machine learning like classification, PAC learning, and machine teaching. We discuss the relevant pieces of these concepts in this chapter. Other background information that is relevant to a specific chapter's contents can be found within the associated chapter.

2.1 Classification and PAC Learning

The most widely used learning model for classification problems in machine learning is called the Probably Approximately Correct (PAC) learning model. In the PAC learning model, a learner goes through two phases of the learning process. The first is the training phase. During training, the learner is provided with a set of examples that have been labeled by some rule, called a concept. This concept is unknown to the learner and is the target of learning. The learner is also provided with a set of possible labeling rules, called a hypothesis space, from which it can choose a hypothesis. The goal of this phase is for the learner to choose a hypothesis that not only performs well on the labeled training data it has seen, but also generalizes to other data that it has yet to encounter. A diagram of this interaction can be seen in Figure 2.1. As part of the learning process, it might be advantageous for the learner to suffer some mismatches on the training data to generalize better. Sometimes, the learner's hypothesis space does not even contain a hypothesis that will correctly label all the training data. In such cases, the difference between the learner's hypothesis' performance on the training data and the true labels is called *training error*. After the learner has selected a hypothesis, the testing phase begins. In testing, the learner is provided with additional examples, this time without labels. The hypothesis chosen by the learner in the training phase is used to label these new examples. Then, the learner is evaluated on how well the labels given by its hypothesis match the true labels of the testing examples. The discrepancy is called *testing error*. Sometimes it is useful to consider how much worse the learner's hypothesis performs on testing data than it did on training data. This metric is called *generalization error*, because it tells us how much additional error the learner incurs when being evaluated on data it has not seen compared to data for which it

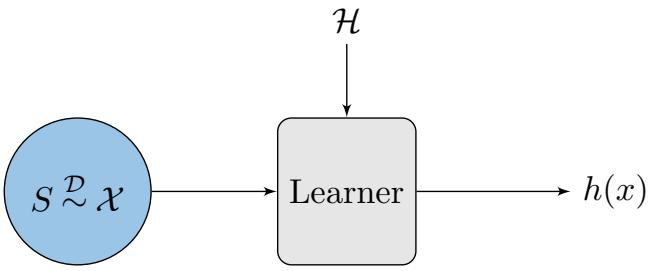


Figure 2.1: A diagram of the training phase of the standard PAC learning model. A training sample S is drawn according to a distribution $\mathcal{D}$ and presented to the learner. The learner uses this training sample S and its provided hypothesis space $\mathcal{H}$ to choose a hypothesis that best describes the data $h(x)$.

had the labels.

This learning model makes certain assumptions about the world to allow it to attain considerably strong requirements for its definition of successful learning. One of the most notable assumptions is that the data presented to the learner is *independent and identically distributed* (i.i.d.), meaning each example is drawn independently and from the same distribution. In other words, we make the assumption that the data we use for learning, regardless of whether it is synthetic or natural, has some underlying distribution that is not changing that determines the probability of any given example being chosen. Further, it is important in the PAC model that both the training set and testing set are both drawn according to the *same* distribution. That is, the probability of any given example being drawn and added to the training set must be exactly the same as it is for the testing set. These assumptions allow us to have a strict definition for success. Informally, we say that a class of concepts is PAC learnable if there exists an algorithm that can choose a good hypothesis in a reasonable amount of time with a reasonable amount of computation given a reasonable amount of examples. A good hypothesis is defined as one that has generalization error that is arbitrarily close to zero with arbitrarily high probability. We now go on to formalize these definitions.

In the PAC learning model, a classification problem is defined by a domain $\mathcal{X}$, a concept class $\mathcal{C}$, and a label space $\mathcal{Y}$. The domain is the space of all possible example instances that we might wish to be able to label. An example $x \in X$ is typically described as a vector of attributes $\langle x_1, x_2, \ldots, x_n \rangle$. The possible labels that can be attributed to each of the examples in $\mathcal{X}$ is the label set, $\mathcal{Y}$. For our purposes, we will consider $\mathcal{Y} = \{0, 1\}$, unless stated otherwise. This special case is known as binary classification, since there are only two possible classes, 0 and 1, with which we label examples. The concept class $\mathcal{C} \subseteq \{c : \mathcal{X} \to \mathcal{Y}\}$ is a set of functions that map examples to labels. $\mathcal{D}$ is a distribution over $\mathcal{X}$ from which a set $S = \{x_i \in \mathcal{X} | 1 \leq i \leq m\}$ of m training examples is drawn. The learner chooses a hypothesis h from the hypothesis class $\mathcal{H} \subseteq \{h : \mathcal{X} \to \mathcal{Y}\}$ that maps $\mathcal{X}$ to binary labels, just like the concept class $\mathcal{C}$. The goal is for h, with probability $1 - \delta$, to have at most ϵ expected error on testing data drawn from the distribution $\mathcal{D}$. We say a problem is PAC learnable if, for any $\epsilon, \delta > 0$, there is an algorithm $\mathcal{A}$ that finds such an h in time (and number of samples) polynomial

in $\frac{1}{\delta}$, $\frac{1}{\epsilon}$, and parameters capturing the size of the problem description.

2.2 Teaching

The concept of introducing a teacher into the standard learning framework is well studied. One way of categorizing different approaches in this area is by the power given to the teacher in the model. The most naïve approach, which makes the teacher maximally powerful, is to allow the teacher to generate samples with which to train the agent. Such a model is a complete departure from the PAC setting, since there is no longer a distribution over training examples. This model renders the learning problem trivial in many domains, since the teacher can directly convey exactly what the agent needs to know. That is, if the goal is to learn some sort of function representable by a DAG (as in the problems we consider), the teacher can generate $O(1)$ examples for each node in the DAG to show the agent the important bits and how they are related. For example, the teacher and agent can collude and agree before hand that the teacher will create examples in the following way. Assign an index for each possible operator. Pass the learner two examples at each round. The first is an example of all 0s except for two 1s in the position of the operand bits. The second is an example of all 1s except for a 0 on the bit representing the operator. The entire function can be taught with sample complexity $O(k)$ where k is the number of nodes in the function's DAG representation. This problem is not one that we are concerned with in our work, as the absence of a distribution on the domain means we will have no means to compare to learning in the standard PAC setting. More importantly, as the goal of this work is to create a model that works for different potentially real-world applicable learning problems, the reliance on the environment to provide the agent with experience harmonizes with many real-world applications, where hand crafting examples may be impractical.

A step in reducing the teacher's power is to force the teacher to draw examples from a distribution for training and testing the agent. Making samples distribution dependent brings the problem more in line with the standard PAC setting. An attempt at iterative learning with this type of teacher was proposed using a hierarchical learning algorithm presented by Rivest and Sloan [1994]. This work attempts to solve the problem of learning Boolean formulae by using a teacher who cannot create examples, but can still edit their labels before presenting them to the learner during the training rounds. They show how such a benevolent teacher can guide a learner to a hypothesis that is "reliable" (makes no incorrect classifications), and with high probability is "useful" (doesn't return "I don't know" too often). A diagram of their work is shown in Figure 2.2. Because the agent is allowed to refuse to classify some instances, this algorithm does not fit into the PAC framework in the strict sense. We will show later that we can still meaningfully compare their algorithm to our results, however.

The more troubling concern with this setup is the potency of the teacher, because the teacher is allowed to alter the training examples' labels during training. When this power is restricted to being used for this particular problem and within the context of the algorithm the authors provide,

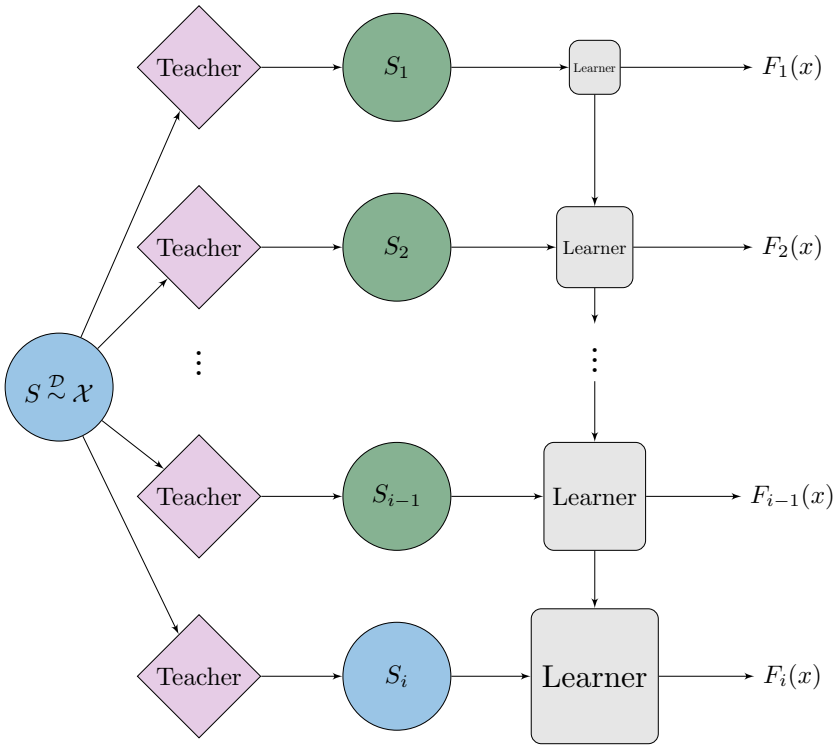


Figure 2.2: The above diagram shows the model used by Rivest and Sloan. In their model, the teacher (lavender diamond) takes examples from the world (blue circle). With these examples, the teacher relabels them to match the current subconcept being taught. This is represented by the change in the circle's color from blue to green. The relabeled examples are passed to the learner and used to learn each subconcept iteratively. By the end of the process, the learner is able to learn the full concept on the true labels.

there is no issue. However, since the goal of our work is to provide a learning framework applicable to a broad class of problems solved in this iterative way, we cannot allow the teacher to have so much power. Doing so would leave the model open to allowing collusion between the teacher and the learner leading to the design of brittle learning algorithms. Further, it may not always be the case that even the most helpful of teachers has the ability to alter the reality of the domain and relabel examples. Formal constraints for enforcing "collusion-free" interactions between teachers and learners have been addressed [Zilles et al., 2011]. We take a closer look at these constraints and their relation to our model in Section 3.3.3. Indeed, the setup of Rivest and Sloan [1994] does not qualify as a valid, collusion-free teacher–learner pairing, because it violates the rule that says a teacher must present a set of training samples that is consistent with the concept c being taught.

To combat this potential for collusion, we restrict the capabilities of the teacher. The restrictions we put in place, described in Section 3.3.3, bring our model in line with the criteria described by Zilles et al. [2011]. In the setting we propose, the teacher is still bound to drawing examples from the testing and training distribution. The teacher cannot alter any labels of the examples drawn from the domain distribution at training or testing time nor can the teacher and learner agree upon an ordering over the sample space. The only capability the teacher has is to disallow certain examples from being presented to the learner. The intuition here is that, if an example is too confusing or uninformative, the teacher can block the agent from seeing it. Because this example was drawn from the distribution and evaluated by the teacher, the framework will still count it towards the sample complexity calculation, even though the agent never gets the opportunity to learn from it. But, it also avoids the possibility of being confused by it.

This idea of limiting the data seen by the agent is used in other settings as well. For example, in knowledge distillation [Hinton et al., 2015], a subset of training data is used to train a distilled, smaller neural network after a more cumbersome model was already trained. In the knowledge-distillation setting, however, the authors are reusing the training examples to get a more efficient model, whereas our work seeks to limit training data to get a more efficient training algorithm. Additionally, the knowledge-distillation framework allows for relabeling examples, which we are disallowing here.

2.3 Connections to Privileged Information

Despite being concretely distinct from many of the ideas listed above, our work is very well situated in the work of learning with privileged information [Vapnik and Izmailov, 2015]. In this work, the authors develop an augmentation to the standard PAC model that is general and can be applied to many different learning problems. The idea is that, during training, there may be an opportunity to lend more information to the learner through additional or "privileged" information. This additional information can be extra observations made on the training data, crowdsourced label estimates, expert knowledge, or any other structured information that may be helpful in learning the target concept. The privileged information is presented to the learner as additional, special attributes

of the domain space. During training, the agent would get a set of augmented training examples $x =< x_1, x_2, \ldots, x_n, x'_1, x'_2, \ldots, x'_\eta >$. The additional η attributes appended to the end are bits of information that encode the privileged information. The learner can exploit this knowledge during training time to select a more appropriate hypothesis. However, because this privileged information will not be present at testing time, the agent can not use it in calculating a label for an instance. An example of this type of information given by Vapnik and Izmailov involves the problem of predicting the successful outcome of a surgery three weeks post surgery based on patient information leading up to the surgery. Privileged information in thie case might be information about the patient or complications that occured *during* the surgery. This information might be helpful in coming up with a rule to predict success before surgery, but will not be available at testing time for the algorithm. It is only available at training time because we already know both the outcome of the surgery three weeks after and how the surgery proceeded. Once training has completed, the rest of the learning process continues as it does in the standard PAC setting.

Our model, IMPACT, can be thought of as an instance of the privileged-information framework. As we will discuss in detail in Chapter 3, our model uses a series of rounds, presenting the learner with a different subset of the training data in each round. The teacher is the entity that chooses which examples to show to the learner in which rounds. Rather than proceed iteratively through this process, we could instead augment the training data with η extra bits, one for each round. Setting a privileged information bit to 1 would mean that example is presented to the learner in the corresponding round. Once the entire set is labeled with the privileged information in this way, the training data could be presented to the learner in a single batch, instead of iteratively. The learner could use the privileged information bits to recreate an identical learning process to ours, internally.

The work on privileged information is much more general and abstract than our particular use case, but it is an interesting lens with which we can take a new perspective on our work. In the paper, the authors encourage the reader to explore what types of privileged information would be useful in which settings and how a learner might take advantage of such information. These are research questions we find intriguing and we present IMPACT as a helpful test case. IMPACT learning (specifically cast as a privileged-information model) tells us that this "round inclusion" information we discussed can be shown to be an effective form of privileged information for the types of problem spaces we explore in our work. For a concrete definition of what that space of problems looks like, Chapter 3 discusses a set of sufficient criteria in our model.

In the privileged-information work, the authors describe the setting as an attempt to reduce the convergence time of known machine-learning algorithms. Similarly, our work can be seen as doing exactly that. In the coming chapters, we will tackle concepts that would take an exponential amount of time or resources to learn and show how they can be learned more easily. We accomplish this simplification by using an iterative method and show that it reduces the amount of time and resources it takes to find and compose a usable hypothesis.

Chapter 3

IMPACT Model

In this chapter, we give a formal definition of the model around which our work centers. We make connections back to the PAC model, discussed in Chapter 2, and show how our model is related. Additionally, we provide some insight into the process of creating such a model and making decisions about how the model is constructed and operates. Finally, we provide a general algorithm and proof structure that we will use repeatedly to prove the correctness and efficiency of our model for learning different concept classes.

3.1 Generalizing PAC for Teaching

At the highest level, the Iteratively Moderated PAC Teaching (IMPACT) model can be seen as a generalization of the standard PAC learning model discussed in Chapter 2. The two major differences are that we allow a teacher to filter the training data before presenting it to the learner, and we allow learning to occur over a series of predetermined rounds. The goal of each individual round is for the learner to conceptualize some new, slightly more complex concept that potentially builds on knowledge gained in previous rounds as well as establishes foundational knowledge that is essential for learning future rounds. In fact, any PAC problem can be viewed as a single-round IMPACT problem in which the teacher filters no examples from the training data.

Understanding IMPACT as a generalization in this way is important. Primarily, it establishes a strong similarity between PAC and IMPACT. This similarity is shown visually in Figure 3.1. A formal definition of each of these aspects of the model can be found in Section 3.2, but we present them here at a high level to lay a conceptual foundation for understanding how IMPACT works. Functionally, both models work by consuming a sample distribution and hypothesis space, and producing a hypothesis from within the given space. The goal in both models is to use a training set drawn from the sample distribution as efficiently and effectively as possible to produce a hypothesis that has the least error possible. In both models, we use the same metrics for evaluation—expected error on the sample distribution as well as an estimate of that expected error given by a testing set drawn from the sample distribution. Finally, we measure success in the same way in both models.

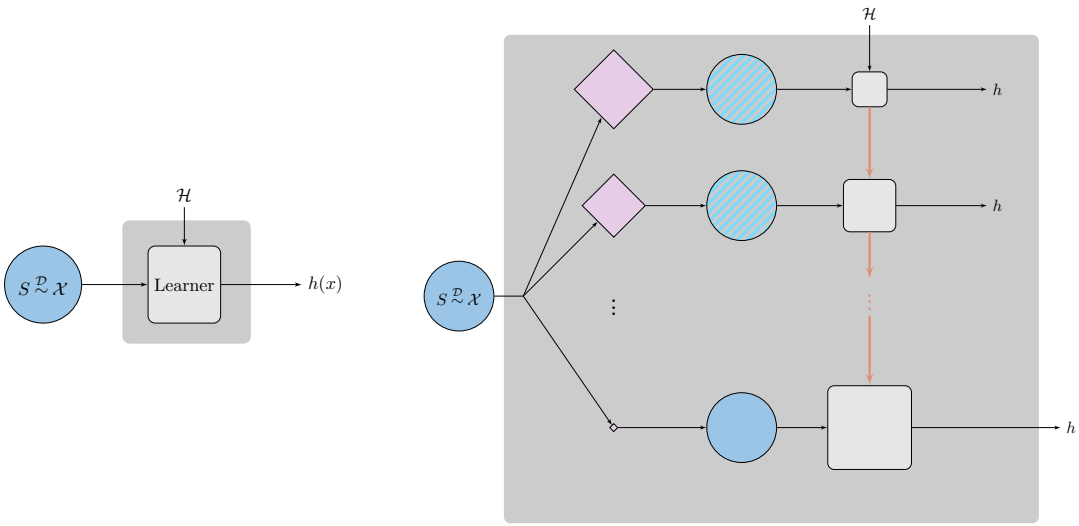


Figure 3.1: The diagram shows both PAC (left) and IMPACT (right) models. Outside of the grey box, the two models seem identical. They have the same functional inputs and outputs. However, within the box, there are components of IMPACT that make it inherently distinct from PAC.

We say that a concept class is learnable (or teachable in the case of IMPACT) when a learner is able to produce a low-error hypothesis with high probability given a limited size training sample and a limited amount of compute time.

These conceptual similarities between PAC and IMPACT allow us to compare the models in interesting, meaningful ways. However, it is essential to highlight the fact that PAC and IMPACT are inherently not identically the same. IMPACT carries with it some implicit assumptions that the PAC model doesn't require. For example, the IMPACT framework takes for granted the existence of a benevolent teacher that already knows the target concept as well as how to decompose that concept to make it teachable across many rounds of learning. In my work, I explored this concept for the different problems discussed in Chapter 4 and potential new problems. Further, for teaching to be possible in the IMPACT framework, the concept class of interest must be decomposable into more easily learned concepts. If a particular concept cannot be decomposed in this way, then we know of no means of teaching it in the IMPACT framework and we cannot make any conclusions about its teachability.

Understanding exactly how IMPACT and PAC relate to one another allows us to consider them together and evaluate our model, IMAPCT, with respect to the standard, PAC. This information helps illustrate the fact that IMPACT is not meant to be a replacement for the standard model, nor is it a universal approach that fits every problem. Instead, IMPACT is a separate tool that can be used to solve some new problems and to approach existing problems from a new direction. Additionally, understanding both PAC and IMPACT allows us to think about learnability and teachability of a concept class simultaneously. We will see that classes that are learnable in the PAC setting are always teachable in the IMPACT setting. For many cases, importantly those that motivated this

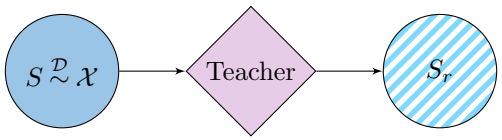


Figure 3.2: This diagram shows the role of the teacher in the IMPACT framework. Here, the teacher is shown as a lavender diamond. The solid circle on the left represents the training sample drawn from distribution $\mathcal{D}$. The teacher filters this sample set to create a subset training sample for round r. This partially colored circle at the teacher's output is used to emphasize that S_r is a subset of S.

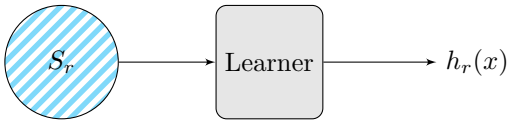


Figure 3.3: This diagram shows the role of the learner in the IMPACT framework. Here, the learner is shown as a grey square. The striped circle on the left represents the training sample presented by the teacher at round r. The teacher isn't pictured here, because it is of no direct consequence to and has no communication with the learner. The learner uses S_r to decide upon a hypothesis h_r to best classify examples.

work, the inverse is not true. There exists interesting classes of concepts that can be taught, but not independently learned. This realization helps motivate the creation of a new hardness class that is a superset of the set of learnable concepts. Understanding the boundary between the teachable and the unteachable could provide insight on what makes certain concepts unlearnable in the first place.

3.2 Model Definition

Since we propose IMPACT as a generalization of the standard PAC-learning model, we will define it with respect to PAC. In the PAC-learning model, a problem is defined by a domain $\mathcal{X}$ of possible instances, a concept class $\mathcal{C}$ that consists of functions mapping $x \in \mathcal{X}$ to labels $y \in \{0, 1\}$, and a distribution $\mathcal{D}$ over $\mathcal{X}$ from which a set $S = \{x_i \in \mathcal{X} | 1 \leq i \leq m\}$ of m training examples is drawn. The learner is tasked with finding a hypothesis h from the hypothesis class $\mathcal{H}$ (mapping $\mathcal{X}$ to binary labels much like the concept class $\mathcal{C}$). The goal is for h, with probability $1 - \delta$, to have at most ϵ expected error on testing data drawn from the distribution $\mathcal{D}$. We say a problem is PAC learnable if, for any $\epsilon, \delta > 0$, there is an algorithm $\mathcal{A}$ that finds such an h in time (and number of samples) polynomial in $\frac{1}{\delta}$, $\frac{1}{\epsilon}$, and parameters capturing the size of the problem description.

In our model, learning instead takes place in a number of rounds, R. In each round, the teacher selects $S_r \subseteq S$, a subset of the training sample, to show the learner. The teacher does this with the intention of having the learner learn some subconcept c_r, represented by the current target node in the CDAG. This process is shown in Figure 3.2. The learner uses these samples to construct some simple concept $h_r \in \mathcal{H}$, $1 \leq r \leq R$, as shown in Figure 3.3. We require that $\mathcal{H}$ be PAC learnable within the context of the round r. It is important to note it need not be the case that

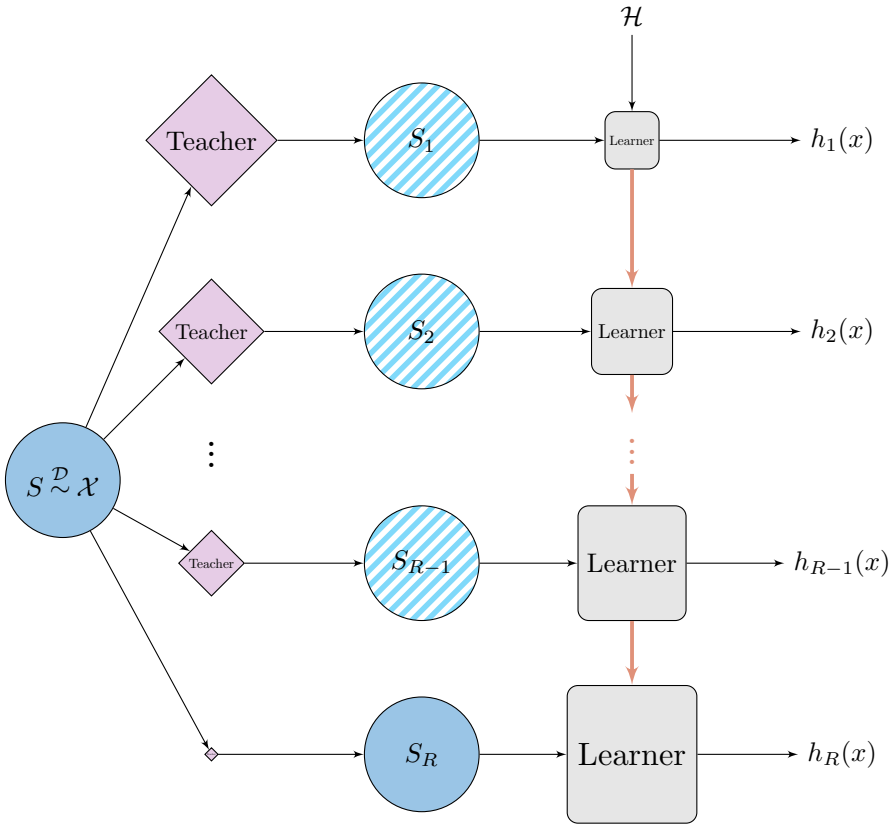


Figure 3.4: This diagram shows the flow of a standard IMPACT algorithm. Each round is depicted as a row on the right side of the image. The teacher draws a sample, S, from the world, $\mathcal{X}$ (left), and uses it to create subsamples for each round of learning. The striped fill on each S_r illustrates the idea that these sets are subsets of the original sample. The learner uses these subsamples to choose hypotheses h_r in each round. The vertical lines between the learner is meant to symbolize that the learner "remembers" information from previous rounds. The relative size of the teacher and learner across the rounds is to show that, with time, the teacher needs to filter fewer and fewer examples (and eventually none, as shown by the solid fill S_R in the final round) and the learner is increasing in ability to express more complex hypotheses.

$\mathcal{H} = \mathcal{C}$. In fact, if $\mathcal{H} = \mathcal{C}$, and if we set $R = 1$, then we revert to the standard PAC model. At the end of each round, the learner can choose to augment its representation of the attribute space in some way to carry information from the previous rounds into future rounds. Incorporating simple concepts from previous rounds allows the learner to gradually build upon knowledge for use in later rounds. This feature is the critical addition to the standard framework that allows the teacher to teach more effectively. The "simple" concept class $\mathcal{H}$ remains the same from round to round except that its domain changes to account for the information carried forward from previous rounds. A flow diagram explaining how a typical algorithm works and how each of these entities interacts is shown in Figure 3.4.

Additionally, we allow no communication between the learner and the teacher once the learning process has started. The teacher is not allowed to alter or order the examples in any way, except to filter out potentially irrelevant or confusing ones at each round. The learner only sees the set of examples the teacher has allowed, and has no other indication of how many or which examples were moderated. Similarly, the learner cannot communicate back to the teacher what its current hypothesis or augmented attribute space looks like. These constraints prevent the teacher from tailoring future example sets to the learner's progress.

3.3 Justifying Design Decisions

There are several aspects of the design of this learning framework that warrant additional discussion.

3.3.1 Sampling for IMPACT

One potentially unexpected aspect of the IMPACT framework is the way in which the training sample is handled. Specifically, one might expect that, because IMPACT is an iterative method, each round requires its own training sample to be drawn from the distribution anew and filtered to suit the round. In practice, this approach is perfectly reasonable. At each round of learning, we could simply draw a new sample, pass it to the teacher and allow the teacher to filter it, continuing on exactly as described by IMPACT. Even the diagram of the flow of the IMPACT framework would be very similar to that in Figure 3.4. Rather than one single solid blue circle representing the training sample on the far left, there would be R of them for each of the separate lines. Even theoretically, the model would remain sound. As discussed in Section 3.4.5, there is no reliance on $\mathcal{S}$ being identical, just $\mathcal{D}$.

We choose, however, to model the framework as reusing the same, single sample drawn at the beginning of learning throughout all the subsequent rounds. As we will see in Section 3.4.5, this decision is justifiable, since we can prove the effectiveness and efficiency of the model described in this way. The main reason for choosing this description over multiple, resampled training samples, then, is to maintain a close similarity to the standard PAC setting, and preserve the ability to easily compare the two, as illustrated in Figure 3.1.

3.3.2 The "Black-Box" Learner

Another salient feature of the IMPACT model is the "black-box" learner component. While the choice of what learning algorithm to use here is an important one, the flexibility provided by defining our model in this way gives IMPACT a lot of its power. Rather than predefining some type of learning algorithm, we instead establish a set of criteria to which this internal learning algorithm must adhere. Using these assumptions about the chosen learner, we are able to theoretically prove the correctness and efficiency of the IMPACT model without limiting its utility. The proofs for our model provided in Section 3.4.5 only rely on the fact that the subconcept at each round is PAC learnable. As long as we have a learner that conforms with these requirements and the criteria from Figure 3.5, we are guaranteed to have a working system.

Modeling the learner in this way also pushes the decision of what learner to use, and how to handle the information learned in each round, to the user. That is, rather than defining a rigid model that might be useful for only a small subset of problems, we leave the possibility for clever applications of IMPACT to solve interesting new problems. Depending on the user's chosen decomposition of the target concept, and the resulting curriculum established for the problem, the learner can change to suit the needs of the system, as discussed in Section 3.4.4.

3.3.3 Preventing Collusion

One concern that could come up whenever we discuss having multiple agents in a learning system work together, especially with one as knowledgeable as the teacher in IMPACT, is the issue of collusion. If it were possible for the teacher and learner to collude or use some sort of coding trick as a means to circumvent the intended method for learning, it would invalidate the IMPACT model. Our definition of the teacher, learner, and of the model itself were designed to disallow such coding tricks.

Learning in Collusion-Free Models

Because collusion is such a natural concern when it comes to teaching systems, it has been studied previously. However, formally defining what collusion entails as a means of prohibiting it can be difficult. At a high level, the concept is fairly plain. Any method of using pre-agreements or encoded messages via the sample set to the learner are prohibited. Formalizing that, though, is more difficult. Learning, itself, is nothing more than passing information to the learner via the training sample, after all. Instead, defining what qualifies as a collusion-free teacher–learner pair needs to be more clear. According to Zilles et al. [2011], a teacher and learner pair that is collusion free follows the following setup:

1. An adversary chooses some $c \in \mathcal{C}$ and presents it to the teacher.

2. The teacher, with knowledge of c, passes a set of labeled examples S to the adversary.

3. The adversary is then allowed to add any additional consistently labeled examples to S to create S_t and passes them to the learner.

4. The learner then learns h.

The idea here is that the teacher and learner can't collude by agreeing on some coding trick based on properties of the training sample. The teacher's, and more importantly the learner's, strategies have to be robust to any additional consistent information the adversary tries to inject. In other words, presenting the learner with additional consistent examples should not change the learner's output. Such a restriction prevents the teacher from cleverly encoding the concept in the number or selection of the examples given to the learner. IMPACT teacher–learner pairs have this property. A learner would never have reasonable motivation to choose a different hypothesis given additional information, as long as that information is still consistent with c_r, the target concept at the current round.

Limiting the Teacher and Learner

The choice of the limitations of the teacher and their interaction with the learner in IMPACT is motivated by more than just preventing collusion. We briefly discuss why other options were not viable in this section. One naïve approach could be to allow the teacher to create examples in each round to give the learner. Not only would this violate the criteria for collusion-free learning, but it would also trivialize the learning problem. If the teacher were allowed to create a sample set, the model would become completely divorced from the PAC setting. The focus of the problem would also shift from learning from data to how to create the most informative data to teach with.

Another option would be to force the teacher to draw the sample from a distribution, as we do in IMPACT and PAC, but then allow the teacher to relabel examples that might make it difficult for the learner to learn. This tack is how the problems presented in our work are handled in Rivest and Sloan [1994]. While this setup does lead to interesting results, we were interested in showing that similar results can be achieved with a less powerful teacher. Additionally, allowing the teacher to relabel examples does not address the potentially likely real-world situation in which a user may not be able to make such changes easily with real data. For this reason, we restrict our teacher to be only able to filter learning data-hiding examples that are too complex for the learner to understand without additional foundational knowledge. Even with enforcing this restriction, we are able to achieve similar results to Rivest and Sloan's work.

3.4 General Algorithm and Proofs

We begin by presenting a set of five criteria that we require of each of our algorithms for these problems. In addition, we present and make use of a general algorithm for these types of IMPACT problems. We will then prove that any concept class that has an appropriate algorithm that meets

I. The target concept must be efficiently representable as a computational directed acyclic graph (CDAG).

II. The sub-concepts at each internal node of the CDAG must be PAC learnable, given that the sub-concepts they require have been learned.

III. The output of each internal node of the DAG must be represented in the same way as the input attributes of the overall problem.

IV. In each round of learning, the filtering rule employed by the teacher must preserve at least some polynomial fraction of the original training sample.

V. Any error incurred at an individual node must compound at most additively as calculation continues through the CDAG structure.

Figure 3.5: The above are the five criteria that a learning problem and algorithm must satisfy to be valid for the IMPACT analysis described in our work.

these criteria is indeed IMPACT teachable. In doing so, we will analyze both the correctness and efficiency of this approach to teaching.

3.4.1 IMPACT Criteria

To discuss the general IMPACT teaching algorithm we use throughout our work, we must first describe the assumptions we make about the concept class and the components of the algorithm. The five criteria we require are listed in Figure 3.5. The first is a requirement of the problem space itself. We require that the problem can be efficiently represented as a computational directed acyclic graph (CDAG). A CDAG, here, is a specific type of DAG that we use to encode a function, namely the target concept. Each node in the CDAG is used to represent a step in the calculation of the function. The value at any given node in the CDAG can be calculated by doing some predefined operation on only the values provided by the children of the current node. More distant descendants influence the current node's value only indirectly. Finally, the output of the function the CDAG represents should be represented by the source node's value. It should be noted that, for convenience, we can consider calculation in the CDAG to proceed in the opposite direction (from source to sink). In this case, the previous definition still applies replacing children with parents, descendants with ancestors, and sources with sinks. When we say the CDAG should represent the target concept efficiently, we mean that the number of nodes of the resulting CDAG must be polynomial in the number of input bits of the problem. This criterion is important because we need to ensure that the target concept is not too complex or large to write down in reasonable time, let alone learn.

The remaining four criteria restrict aspects of the learning procedure itself with respect to the specific problem. We require that each of the learning subproblems represented by internal nodes of the CDAG are PAC learnable. This constraint is necessary to ensure that, within each round of learning in our model, the learner can make sufficient, meaningful progress on the problem without additional outside assistance from the teacher. These simpler, PAC learnable problems become the

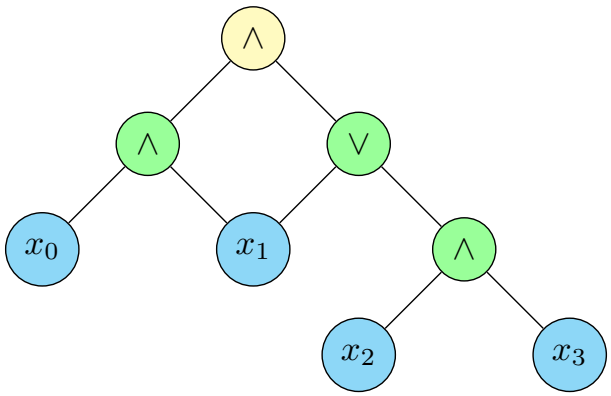


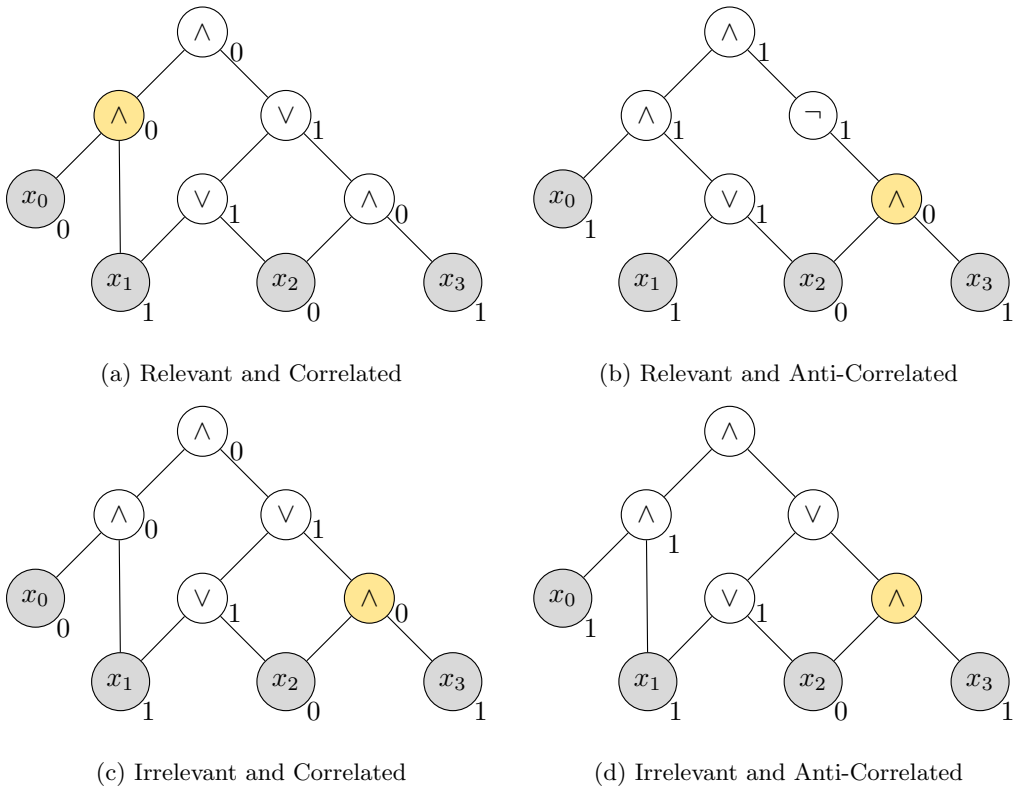
Figure 3.6: The three types of nodes in a CDAG are shown in the figure above. The base nodes are shown in blue. All internal nodes are shown in green, except for the root node, which is in yellow.

building blocks for the learning agent's understanding of the overall target problem. Further, we require that each internal node's "output signal" is represented the same way as any of its input attributes. An example of this idea is a learning problem that takes a bit string for input and has a single bit as the value at any internal node in the graph.

Next, we require that the teacher's filtering rule is not too strict. That is, it must retain at least some polynomial fraction of the original training sample in each round. If the filtering is too strict, the size of the original sample will have to be too large to allow for a sufficiently sized subsample for learning within the round. The final criterion is that any learning error incurred at individual nodes in the DAG is combined at most additively as calculation proceeds deeper into the structure. This property is important because, if small errors early on in the learning process were potentially blown up multiplicatively or worse at each layer of computation, the error itself could become exponentially large by the final output. It is important to note that while these criteria hold for all of the problems in this paper, as well as potentially others, we acknowledge the fact that there may be concept classes and algorithm combinations that violate these criteria, but are still feasible and valid IMPACT solutions. Our criteria are sufficient, but not necessary.

3.4.2 Types of Nodes in a CDAG

The CDAG is an integral part of the IMPACT framework. Each of the problems discussed will be translated into a CDAG to be solved. To support this discussion, we establish a vocabulary of terms related to the anatomy of the CDAG. Specifically, we will define three distinct types of nodes possible in a CDAG. The *root*, or *source* of the CDAG is the only node without any incoming edges. There is exactly one root in any CDAG. The value at the root represents the value output by the overall target concept for a given example. That is, if you represent a concept c^* as a CDAG, and calculate the value of a given example x using that CDAG, the root's value is the same as the value assigned to that example by the concept, $c^*(x)$.



(a) Relevant and Correlated

(b) Relevant and Anti-Correlated

(c) Irrelevant and Correlated

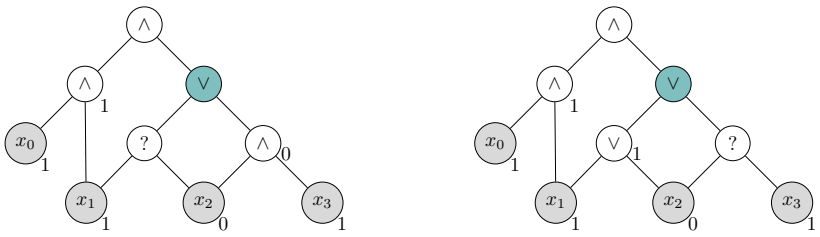
(d) Irrelevant and Anti-Correlated

Figure 3.7: This diagram illustrates the concepts of relevance and correlation in a CDAG. In each figure, the node that is being evaluated is in yellow and the example is written at the leaves. It should be noted that the structure of the CDAG in subfigure (b) differs from the others. A negation node was needed to show an anti-correlated, relevant example.

Complementary to the *root* are the *base* or *sink* nodes. These nodes represent the attributes defined by the domain X. As such, the base nodes are the only nodes in the CDAG that will not correspond to a round of learning. Finally, the last type of node is *derived* or *internal* nodes. These nodes are referred to as "derived" since their value can only be determined by using the values of their children. Additionally, when these nodes are learned, they are stored as new domain attributes by the learner for use in later rounds. While these nodes are treated the same in subsequent rounds as the pure attributes represented by base nodes, they are technically different and are the key feature that allows for the compositionality that enables the learner to understand complex concepts. An example of each of the nodes can be seen in Figure 3.6.

3.4.3 Relevance and Correlation

In addition to the structure of the CDAG itself, the relationship between various nodes and examples plays an important role in the teacher's decisions regarding filtering the training sets. As discussed in Section 3.4.2, the root node of the CDAG is the one whose value is that of the overall concept we are trying to teach. As such, when we refer to the "value at the root" in our discussion, we are implicitly



(a) Non-Blocking (b) Blocking

Figure 3.8: This diagram shows the difference between blocking and non-blocking nodes. In (a), from the perspective of the ? node, the teal highlighted node is non-blocking. It is an OR node where the other input is 0, so whatever the ? evaluates to will propagate up the tree. In (b), from the perspective of the ? node, the teal highlighted node is blocking. Because it is an OR node with the other input set to 1, regardless of the value of the ? node, the output of the teal node will be 1.

referring to the label given to a particular example in the classification problem. Therefore, we use these phrasings interchangeably where convenient in this section and throughout our work.

One way to intuitively understand a CDAG is to imagine that there is a signal that begins at the base nodes and propagates though the CDAG up to the root node. At each internal node, the signal from its children is combined to produce a new output signal. The rules for deriving this output signal from the inputs is determined by the function that the node itself represents. In this sense, the current example being evaluated does nothing more than determine the values of the base nodes that will propagate up the tree. With this idea of propagating values throughout the CDAG, we can classify internal nodes in one of two ways, *blocking* or *non-blocking*. Consider a base-to-root path p in a CDAG with a particular example x being evaluated. Each of the nodes along path p can be classified as blocking or non-blocking with respect to the values propagated along p. A blocking node on p is one that does not use the value being propagated along p to determine its output. For example, consider a node i along p in a Boolean formula CDAG. Assume i is an $\vee$ node. If the other input to i (the one that is not part of p) has a value of 1 as determined by x and the rest of the CDAG, then i is a blocking node. No matter what p's input value is at i, the output of i will always be 1. We call such a node "blocking" because we can imagine that the value along p is blocked by i and stops propagating through the CDAG. Similarly, a Boolean formula CDAG with an $\wedge$ node that has an input of 0 is also blocking. Unsurprisingly, a node that does use the value of the input from p is called a non-blocking node. In our previous example, a node in non-blocking when we have an $\vee$ node with another input of 0 or an $\wedge$ node with another input of 1. The value of the input on p will determine the outcome of the node and continue to propagate up through the CDAG.

Relevance is another classification of CDAG nodes. If whether a node is blocking or non-blocking is a local property, relevance is the global complement to that. It is a measure of the impact a particular node has on the overall output of CDAG given a particular example. If a node i is important in determining the output at the root of the CDAG for an example x, we say that i is *relevant* for x. If the value at node i is inconsequential for determining the value at the root of the

Input: set of initial attributes Z_0, learner's hypothesis space $\mathcal{H}$,
true concept $c \in \mathcal{C}$, examples S, number of rounds R
Output: learned concept $h_R \in \mathcal{H}$
for $r \leftarrow 1, \ldots, R$ **do**
    $S_r \leftarrow$ **ModerateSample**(S, r)
    $h_r \leftarrow$ **PACLearn**$(Z_{r-1}, S_r, \mathcal{H})$
    $Z_r \leftarrow$ **AugmentAttributes**(Z_{r-1}, h_r)
end for

Figure 3.9: Pseudocode for the general algorithm.

CDAG for an example x, then we say i is *irrelevant* for x. Alternatively, relevance is a measure of whether or not a node's signal propagates all the way to the root of the CDAG. So, we can define a relevant node as one whose node-to-root path is comprised of only non-blocking nodes for an example x. A simple test can be done to test the relevance of i for x. First, evaluate the CDAG for example x. Then, short-circuit the output of i and change its value. If the value at the root of the CDAG changes, then i is relevant for x, otherwise i is irrelevant for x. Relevance has an important role in the IMPACT setting. As previously explained, relevance is a property that holds for a particular example. However, we can consider how likely it is for a node to be relevant for an example drawn from our training or test distribution. If a node is irrelevant for many or most of the examples in the distribution of interest, knowing this fact can help us to bound the overall error of learning the CDAG in important ways. This approach will be explored in Section 3.4.5.

Similar to relevance, *correlation* is global property of a node in a CDAG with respect to a particular example. We use the term correlation to compare the value of the root to the value of a particular node for a given example. For a node i and an example x, we say a node is *correlated* if the value at i is the same as the value at the root. If the two values are not the same, we say that they are *anti-correlated*. The correlation at a particular node for an example is an important factor for the teacher's decisions about filtering the training set. Because the learner learns small, sub-concepts in isolated rounds, seeing both correlated and anti-correlated examples in the same data set would likely be confusing. For example, if the concept for the round is to learn $x_1 \wedge x_2$, correlated examples would be inherently labeled the same as $x_1 \wedge x_2$ and anti-correlated examples would be labeled according to $\neg(x_1 \wedge x_2)$. In general, we can make accommodations in our model to learn from either correlated or anti-correlated examples. However, having both in a single training set would be contradictory. At best, it would seem like the data set is noisy to the learner. At worst, it could be unlearnable or cause the learner to learn some other concept that is more consistent with the data than the intended target is.

3.4.4 Algorithm Description

Having established the required criteria for our method of teaching in the IMPACT setting, we will now define a general algorithm for solving the problems we discussed. To fully describe an algorithm in the IMPACT framework, we must define a variety of entities and procedures. Each

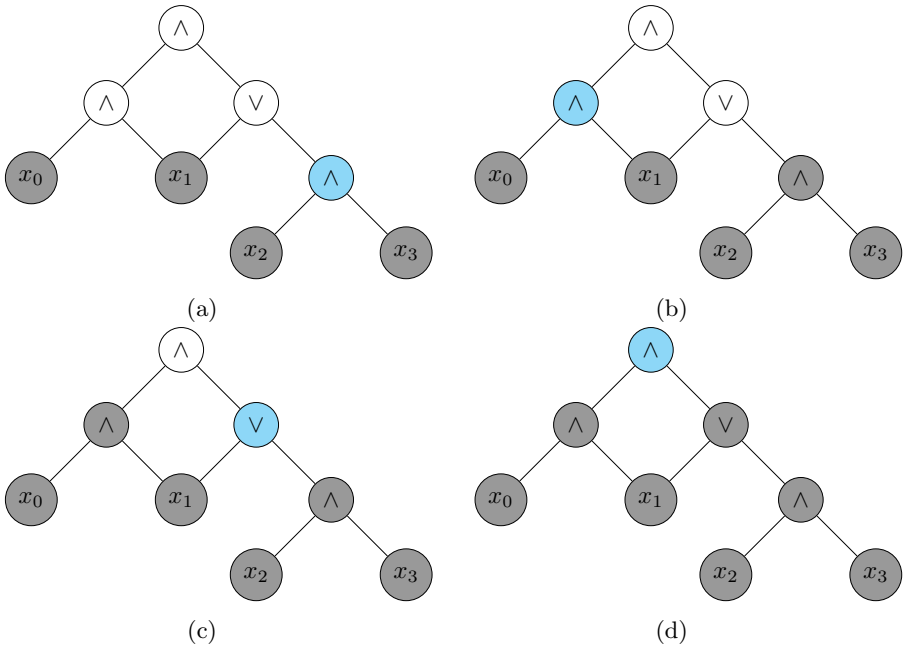


Figure 3.10: An example curriculum. Each subfigure represents a different round of learning (ordered alphabetically). Each white node in the images are concepts that the learner has not yet been taught and therefore does not know about. A grey node depicts a attribute (pure or derived) that is known to the learner. A blue node represents the target subconcept for the current learning round in each image. Importantly, in subfigure (d), the target node is also the root of the CDAG. In this round, the learner will incidentally learn the target concept of the overall problem.

of the components of the algorithm in Figure 3.9 must be concretely defined. For this section, we will take each of the symbols and subroutines from Figure 3.9 and discuss what must be defined to create an IMPACT algorithm. In Section 3.5, we will discuss how to make these decisions based on a particular target concept class.

Some parts of the algorithm are predefined by the problem, and do not need to be changed for using IMPACT. For example, the true concept c and set of training examples S are exactly what is provided by the learning problem itself. Obviously, c is still simply the target concept of the overall learning problem. Also, S is just the sample drawn from the distribution $\mathcal{D}$ on the attribute space $\mathcal{X}$ of the original problem—the same one you would use for learning in the PAC model.

The value Z_0 is somewhat of a unique case in defining an IMPACT algorithm. In many cases, just like S or $\mathcal{D}$, it is derived directly from the learning problem. Typically, that means that Z_0 is just the set of attributes given by $\mathcal{X}$. Sometimes, however, we need to give additional attributes to this initial set to ensure the learner can make progress even in the early rounds of learning. For example, in the IMPACT algorithm we will describe for Boolean formulae, $\mathcal{A}_{BF}$, we will define Z_0 as all of the attributes from the space $\mathcal{X}_{BF}$ as well as their negations.

The rest of the components used in designing an IMPACT algorithm are specific to IMPACT itself and are therefore not present in what would be an analogous PAC setup. The first of these is the number and structure of the R rounds of learning. This round structure is determined by the decomposition of the true concept. Each of the component subconcepts becomes a round of learning. Given that our problem already adheres to the criteria from Figure 3.5, the decomposition has already been handled and modeled as a CDAG. The CDAG also provides useful structural information that can inform the decision about how to order each of the rounds. These decisions are required for the definition of our algorithm because they contribute to the definition of the **ModerateSample**(S, r) routine. This routine executes a round-specific filtering rule to create S_r. Additionally, the learner's hypothesis space, $\mathcal{H}$, is in part defined by the CDAG. This hypothesis space must include all of the subconcepts represented in the CDAG that are necessary to learn efficiently at each round. The subroutine **PACLearn**$(Z_{r-1}, S_r, \mathcal{H})$ is the black box that contains the learning algorithm. The choice of learner for this piece of the algorithm is crucial to successfully teaching the target concept. Finally, we must decide upon how information is carried forward from one round of learning to the next. Typically, this step is done simply by augmenting the attribute space for future rounds with the output hypothesis from this round of learning. Sometimes, however, as is the case again with $\mathcal{A}_{BF}$, more clever applications of **AugmentAttributes**$(Z_{r-1}h_r)$ will need to be used.

In all of the examples in our work, we will follow the same basic algorithm structure we just described, given in Figure 3.9. For each example problem, all of the entities we discussed above will be explained fully in the detail related to the specific problem.

3.4.5 Proofs

Now that we have the desired set of problem properties and a suitable, albeit abstract, algorithm for working in the IMPACT setting, we will show that any problem and algorithm that satisfy the criteria in Figure 3.5 and conform to the general algorithm in Figure 3.9 are IMPACT teachable. The first step in showing that our algorithm is correct is ensuring that the teacher has a reasonable curriculum of rounds such that Criterion II is satisfied. We must be sure that all of the nodes that a node i uses in its computation are learned before trying to learn i. Lemma 1 accomplishes this goal.

Lemma 1 *In a CDAG where each node requires the value of other nodes on which it depends to be computed first, there exists some order of calculations that allows these requirements to be fulfilled.*

Proof In the case where the value of a node depends on those of its descendants, calculating the nodes in postfix order ensures that each descendant will be calculated before its parent. If the value of the node depends on its ancestors, prefix order will suffice. ■

Theorem 2 *A concept class $\mathcal{C}$ that is taught using teaching procedure $\mathcal{A}$ is IMPACT teachable given that the pair $\mathcal{A}, \mathcal{C}$ adhere to the criteria from Figure 3.5. That is, $\mathcal{A}$ can find a hypothesis $h \in \mathcal{H}$ such that h incurs at most ϵ error from c with probability at least $(1-\delta)$ using a sample S of data drawn according to an arbitrary distribution $\mathcal{D}$ on the input space $\mathcal{X}$ and R training rounds where $|S|, R = O(poly(\frac{1}{\epsilon}, \frac{1}{\delta}, lg|\mathcal{X}|))$.*

Proof Consider some concept class $\mathcal{C}$ and teaching procedure $\mathcal{A}$ that satisfy the criteria from Figure 3.5. Criterion I ensures that c, the target concept, is representable by a DAG. Therefore, $c(x)$ can be determined from some node in the DAG representation of c, $G(c)$. We will refer to this node as the "root". Further, since Criterion I guarantees an efficient representation, we know that the number of nodes R in $G(c)$ is $poly(n)$ where $n = lg\mathcal{X}$ is the size of the problem input.

Any individual internal node of G is calculated by combining information from other nodes within the DAG. By Criteria II and III, each of the sub-concepts at these internal nodes (including the root) must be PAC learnable given that the appropriate sub-concepts have been learned. By Lemma 1, we know that there is a curriculum of rounds that $\mathcal{A}$ can use to ensure each sub-concept is trained only when the requisite sub-concepts have been learned. Therefore, Criterion II guarantees that this current node can be learned with error at most ϵ'. Additionally, Criterion V assures us that any error contributed by dependent nodes will be at most additive. Therefore, the current node's overall test error is at worst the sum of its ϵ' and that of its contributing nodes. Since this argument holds for all internal nodes, it holds for the root. The error at the root is at most the sum of each ϵ' in the DAG. If we insist for all nodes that $\epsilon' \leq \frac{\epsilon}{R}$, then the total error of the final learned hypothesis h will be at most $R \cdot \frac{\epsilon}{R} = \epsilon$.[1]

[1] This bound can be met without prior knowledge of R by setting $\epsilon'_r = \frac{\epsilon}{2^r}$ [Rivest and Sloan, 1994].

Finally, Criterion IV ensures that the above steps are possible with a reasonable size sample S. For each subsample S_r (which can be as small as $\frac{|S|}{O(poly(n))}$) to be sufficiently large, S must be such that $|S| \geq O(poly(n))|S_r|$. Since Criterion IV guarantees $|S_r| = O(poly(n))$, then $|S| \geq O(poly(n))O(poly(n)) = O(poly(n))$, because the product of two polynomials is itself a polynomial.

Combining these arguments gives us that $\mathcal{C}$ is IMPACT teachable using $\mathcal{A}$ as a learning procedure. ■

Having shown that the general algorithm provides us with a proof that some concept class $\mathcal{C}$ is IMPACT teachable as long as the pair of $\mathcal{C}$ and the specific algorithm meets the criteria in Figure 3.5, we will continue to describe the particular algorithms for our problems of interest. By showing that the solutions we describe meet the criteria from Figure 3.5 and invoking Theorem 2, we can show that each is IMPACT teachable.

3.5 Designing an IMPACT Algorithm

This chapter has discussed what IMPACT is, a formal definition of the model and all of its components, what it means to create an IMPACT algorithm, and theoretical results for showing that, in the right conditions, the IMPACT model allows for teaching complex, difficult-to-learn concepts, in an efficient way. Just as there were special considerations made in the design of the IMPACT model, so too are there design considerations that need to be accounted for when creating an IMPACT algorithm to tackle a particular learning problem. In this section, we discuss these design decisions and give insights into how they were made with regards to the example problems in our work. Understanding how these decisions are made can help to clarify the reasons for certain aspects of the IMPACT model itself. Different features of the problem space or of the algorithm will rely on knowing that our instance of an IMPACT model conforms to the criteria from Figure 3.5.

Accordingly, the very first step of creating an IMPACT algorithm is to devise a way to decompose the target concept class into a class of CDAGs. We need to be sure that the concepts that we can target for teaching are representable as a CDAG that meets the criteria of an IMPACT problem. Because these criteria are fundamental parts of the IMPACT model, not meeting them would negate any other assumption or proof of correctness or efficiency that we have provided so far. Also, once the CDAG is derived, it can be used immediately to make many of the other design decisions regarding the algorithm.

For example, the number of rounds of an IMPACT algorithm is simply the cardinality of the vertex set of the CDAG. Ordering those rounds takes a bit more care, however. The rounds need to be ordered in such a way that whenever the algorithm is learning at a particular node, all requisite nodes are already learned. Criterion III, which says that the internal nodes of the CDAG must have their outputs represented in the same way as the attributes of the overall problem, allows us to view all of the rounds of learning as identical problems, with slightly different attribute sets. Ordering the rounds simply becomes a problem of choosing an ordering such that these attribute sets are

already established by the time we attempt to teach using them. The only real decision here then is whether or not to traverse the nodes in a top-down or bottom-up manner.

Having the rounds in order allows us to ensure that each node will be PAC learnable, given the proper information from previous rounds. Based on the decomposition of the concept into a CDAG and an understanding of the PAC learnability of the individual nodes, we can decide what the learner's hypothesis space, $\mathcal{H}$, is. The hypothesis space $\mathcal{H}$ needs to allow the learner to provably PAC-learn a subconcept that computes the correct signal to pass to the next node in the CDAG. The hypothesis space $\mathcal{H}$ can be a linear combination of the inputs, some decision rule based on the inputs, or any other PAC learnable space that makes sense for our decomposition.

Another consequence of having each node be PAC learnable is that the filtering routine referred to as **ModerateSample**(S, r) cannot be so limiting that it restricts the training sample to be too small to be usable for any given node. In other words, it cannot filter so many examples from S in creating S_r that we require S to be exponentially larger than S_r to have enough examples pass through the filter. The main reason we filter examples at each round is because the complexity of the overall concept makes it difficult to learn everything at once. So, some examples in S may either be correlated or anticorrelated. Showing both of these sets means that a large part of the data would contradict the subconcept trying to be taught. Typically, the filtering rule is something like "filter out any anticorrelated examples". However, maybe there is no way to guarantee that there will be enough correlated examples. Perhaps in those cases one might use a rule like "filter out the smaller of the correlated or anticorrelated examples".

Finally, the choice of how to pass information between rounds with **AugmentAttributes**(Z_{r-1}, h_r) is informed by most of the previous decisions. For instance, by Criterion III, it is easy to ensure that the form of the added attribute match the form of the other attributes. However, what happens if the learner is unaware of whether they were trained on correlated or anti-correlated data? Accomodations may need to be made in **AugmentAttributes**(Z_{r-1}, h_r) to account for these sorts of possibilities. A situation like this arises in Section 4 and we will use a clever choice of **AugmentAttributes**(Z_{r-1}, h_r) to make sure the true concept is still teachable.

Chapter 4

Classification

In this chapter, we explore three distinct classification problems. Following a formal definition of each, we will formulate and discuss the components of the specific IMPACT-style algorithms for solving them. Additionally, we provide sets of proofs that show how each problem and its associated algorithm conform to the five criteria from Section 3.2, Figure 3.5.

The problems selected for this section are of interest for a variety of reasons. Importantly, each is an ideal representation of the type of problem IMPACT was designed to solve. That is, in the standard PAC setting, these concept classes are unlearnable [Kearns and Valiant, 1994]. Further, because they were studied by Rivest and Sloan [1994], they make for a good comparison between the two methods. We are able to show, theoretically, that these concept classes can be learned in our IMPACT setting just as they are in Rivest and Sloan's work. However, we do so without enabling the teacher to have the unnecessary ability to edit example labels during training.

4.1 Boolean Formulae

Boolean Formulae is the problem in which the learner is tasked with learning to predict the output of a Boolean formula over n inputs. The domain is $\mathcal{X} = \{0,1\}^n$, n-bit strings, and the concept class, $\mathcal{C}_{BF}$, consists of all Boolean functions over n inputs that can be represented by Boolean circuits with $O(p(n))$ nodes, where $p(n)$ is some polynomial in n. We focus on polynomial-sized Boolean circuits since larger circuits would take too long for an algorithm to describe, let alone learn. Since we only require the algorithm to learn a Boolean formula, it need not learn the exact structure of the circuit representation as long as the concept it chooses is approximately functionally equivalent to the actual formula over the input distribution.

4.1.1 Algorithm

As discussed in Section 3.4.4, to fully define an algorithm for IMPACT, we need to define each of the components from the general algorithm in Figure 3.9. For reasons mentioned in Section 4.1.2, we

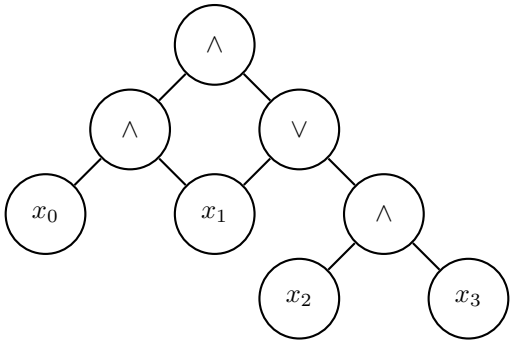


Figure 4.1: The graph above represents a Boolean formula. The base nodes are labeled with the propositions of the formula, and the internal nodes are labeled with the Boolean operators used to combine them. The specific formula represented by this graph, then, is $(x_0 \wedge x_1) \wedge (x_1 \vee (x_2 \wedge x_3))$.

will define Z_0 to be the set of all attributes from $\mathcal{X}$ as well as their negations. We will use the space of binary Boolean operations over the attributes to be $\mathcal{H}$. The **ModerateSample**(S, r) routine will return the larger of the two subsets of correlated and anti-correlated examples at node r. Because we are in the consistent realizable case, we can use simple ϵ-exhaustion [Mitchell, 1997] as our **PACLearn**$(Z_{r-1}, S_r, \mathcal{H})$ routine. Our teacher will train in the nodes using a postorder traversal of the CDAG. Finally, our **AugmentAttributes**(Z_{r-1}, h_r) will add both h_r and its negation to Z_{r-1} to create Z_r.

Now that we have specified the algorithm, we can give a high-level intuition for how and why it works. Because the learner is presented with a simple problem of combining two attributes by a single binary Boolean operation at each round, learning in each round is fairly straightforward. A key piece of the model is that **AugmentAttributes** includes these intermediate hypotheses as new attributes for future rounds of learning. As a result, the learner learns increasingly complex and deep Boolean expressions without having to learn them directly. The teacher is able to facilitate this learning by paring down the training sample so that these intermediate rounds can be learned without confusion from seemingly contradictory examples.

4.1.2 Handling Negation Nodes

One of the slightly strange aspects of the $\mathcal{C}_{BF}$ CDAG is that it can contain negation nodes. We take two separate approaches to handling such nodes in the theoretical and practical applications of our work. For our theoretical analysis, we remove the negation nodes from the CDAG by using DeMorgan's law in a preprocessing step to propagate all of the negation nodes down to the base nodes. In the practical application of our algorithm, however, this preprocessing step isn't necessary. We instead make a small change to the **AugmentAttributes** routine to account for potential negation nodes, and handle learning on possibly anti-correlated data.

In our analysis, as stated, we propagate all of the negation nodes down to the base nodes via DeMorgan's law as shown in Figure 4.2. By the end of the process, we can clear the CDAG of any

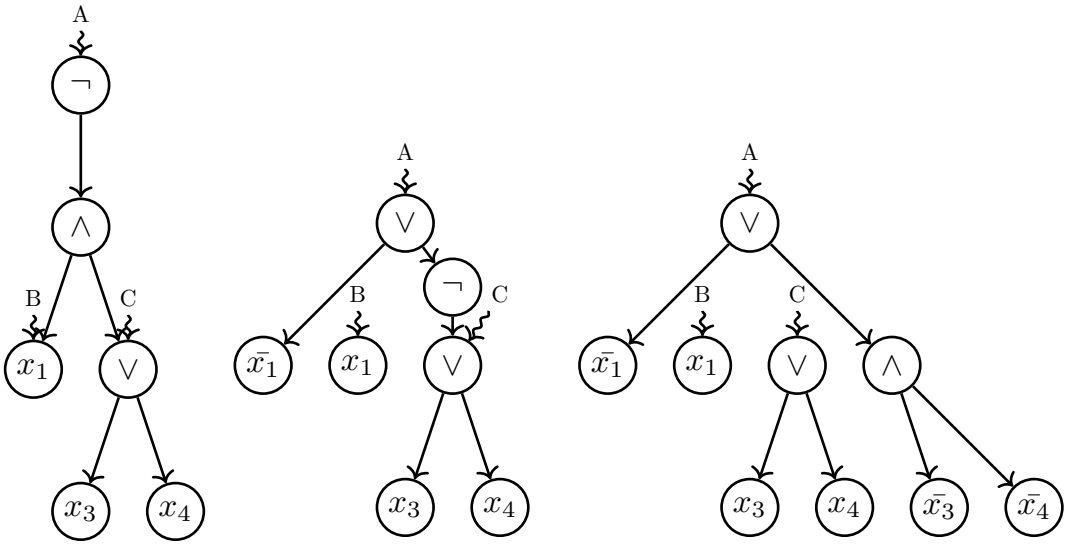


Figure 4.2: This figure shows the process of transforming a computation DAG representing a Boolean formula via DeMorgan's law. From left to right, the negation is propagated down one level in the DAG. The wavy arrows indicate incoming edges from other parts of the DAG not pictured (labeled by letter for consistency).

negation nodes by including the attributes as base nodes as well as a second set of base nodes with the negation of each attribute. We define this result as the Z_0 for $\mathcal{A}_{BF}$. One potential concern about doing this transformation is that the size of the new CDAG we create has become unwieldy. We show in Lemma 3, however, that this process only at most doubles the size of the CDAG. Once we do so, we can consider training on only correlated examples, because all relevant examples become correlated, since there are no negations in the tree. These simplifications, while unnecessary, help to make the analysis cleaner.

To train using the tree with negations existing, the teacher does not count the negation as a node to teach. Technically, no harm comes if the teacher does teach a negation as it just introduces some redundancy into the learning process. Instead, when we **AugmentAttributes** for our $\mathcal{A}_{BF}$, we include h_r as well as its negation $\bar{h_r}$, adding both to Z_r. This way, at every step of the algorithm, the learner is augmenting the attribute space with the subgraph that accounts for possibility that the target node was beneath a negation node. Additionally, since we can't guarantee that the teacher will pass only correlated examples in this case, augmenting Z_r in this way allows us to learn on anti-correlated data as well. Training on anti-correlated data causes the learner to learn $\bar{c_r}$ instead of c_r. Regardless of whether or not the learner is trained on correlated or anti-correlated data, both h_r and $\bar{h_r}$ are added to Z_0, so the proper hypothesis is added to the attribute space to use on subsequent rounds.

4.1.3 Analysis

To show, formally, that $\mathcal{C}_{BF}$ is IMPACT teachable, we need to show that the class and our algorithm for it, $\mathcal{A}_{BF}$, adheres to the criteria in Figure 3.5. That, combined with the proofs given in Section 3.4.5, brings us to our desired result.

Lemma 3 *Every $c \in \mathcal{C}_{BF}$ can be efficiently represented as a CDAG, G.*

Proof Consider an arbitrary $c \in \mathcal{C}_{BF}$. Since it is guaranteed to be polynomially sized, we can create one internal node for each operator in c and one base node for each input. Each node will have at most two children representing the inputs to that operator (negation nodes will only have one). We call this CDAG G'.

Now, using deMorgan's law, we can propogate all of the negation nodes to the base of the tree–just above the leaves. Each time we move a negation down a level, we may need to make two new nodes—one for each child of the changed operation. This process is illustrated in Figure 4.2. At most, this rewriting doubles the number of nodes in the CDAG, so the total number of nodes in this new CDAG, G, is still polynomial. ■

Lemma 4 *Let $\epsilon_{i,j}$ be the total weight of the examples that h_i misclassifies in the distribution induced by the teacher's filtering rule in round j $\mathcal{D}_j$. Given a node a and its parent node b,*

$$\epsilon_{a,b} \leq \epsilon_{a,a} \ .$$

Proof The examples in the distributions of $\mathcal{D}_a$ and $\mathcal{D}_b$ can be categorized into three classes. Examples that h_a correctly classifies, examples that h_a incorrectly classifies, and examples that have zero weight in $\mathcal{D}_a$, due to the teacher's filtering rule. Correct classifications from h_a do not contribute to error in h_b, by definition of being correct. Additionally, we know that examples that have zero weight in $\mathcal{D}_a$ were filtered. Since training in $\mathcal{A}_{BF}$ is done using relevant, correlated examples, we know these examples must be irrelevant at a. Therefore, they cannot contribute any additional error in $\mathcal{D}_b$. They may now have weight in the distribution, but the output at a has no effect on the label of these examples, by definition of being irrelevant. In order to prove the claim, then, we only need to show that the examples that are incorrectly classified by h_a, do not gain additional weight in $\mathcal{D}_b$. Again we reiterate that error can only come from relevant nodes. Because relevance is determined by whether or not a signal encounters a blocking node along its path to the root, the number of relevant examples at each node is monotonically increasing along any base-to-root path. As a result, the weight of a given relevant example at a parent node is at most as much as the weight of that example at the child. Therefore, examples which the child erred can contribute only at most as much error at the parent as they caused in the child. Combining these arguments shows us that for each type of example, the total weight of error at the parent is at most as much as the weight of the error at the child, so the claim holds. ■

Lemma 5 *Each subconcept in G is PAC learnable.*

Proof

First, we note there are two node types in the CDAG that can be the target for any given round. The first type, referred to henceforth as a *base node*, is a node whose children are only pure variables of the Boolean function, that is, not derived attributes. An interior node, then, is one that we define as having at least one child that is a base or interior node. The total number of interior and base nodes in the DAG representation will be denoted by N. As we mentioned previously, the DAG is polynomial in the size of the input, so $N = O(poly(n))$.

In the case of base nodes, we refer to classic PAC learning with a finite, realizable hypothesis class [Mohri et al., 2012] to argue that with high probability we can learn the relationship that governs the output of this node with sufficient accuracy, given sufficiently many relevant examples from the teacher. It is important to realize that since the learner is only attempting to choose two variables and their appropriate operand from potentially $O(N)$ variables, the size of the hypothesis space is bounded by

$$|\mathcal{H}| = O\left(\binom{N}{2}\right) = O(N^2) = O(poly(n)).$$

Here, we use N instead of n, since the learner may have access to some derived inputs at the time of learning the node, and it has the option of choosing from these variables as well.

Interior nodes are also PAC learnable, given that their children are PAC learnable. At all interior nodes (including the root), learning is slightly more complicated. At these nodes, due to the possibility of errors in previous rounds of learning (that is, at nodes within the subgraphs rooted at the current target node), our derived input values may be erroneous. To higher-level learning problems, these errors are experienced as corrupted versions of the derived inputs the algorithms need. So, to cleanly derive the bounds on learning error, we first introduce some notation. We denote the hypothesis returned by the learning algorithm in a particular round r as h_r, and the hypothesis induced at a particular node by the overall formula as h_r^*—the target hypothesis. By construction, $h_r^*, h_r \in \mathcal{H}$. Here, $h_r(x)$ and $h_r^*(x)$ are meant to be the value that the respective hypothesis has on data assuming all nodes have been learned perfectly. That is, both h_r and h_r^* provide a formula for combining some other two nodes in the CDAG thus far (either pure or derived), such as, $x_2 \wedge x_4$. Using that formula, we evaluate the hypothesis on the uncorrupted versions of the contributing nodes. Contrastingly, $h_r^*(\tilde{x})$ and $h_r(\tilde{x})$ have the formula of their respective hypotheses, but are evaluated on the possibly corrupted contributing nodes. It is crucial to our analysis that $h(x)$ and $h(\tilde{x})$ need not be equivalent if the contributing nodes are corrupted, for any $h \in \mathcal{H}$.

Because it is the case that these hypotheses might not have the same labels, we have a notation to express the difference between these hypotheses on similar inputs. We denote the difference between two (possibly the same) hypotheses on a distribution D of inputs (either corrupted or pure) as

$$||f(x) - g(\tilde{x})||_D = \mathbf{E}_{x\sim D}[f(x) \neq g(\tilde{x})].$$

This notation is understood to be the expected number of samples that will disagree when we evaluate f on uncorrupted data and g on corrupted data. Similarly, we denote the difference between two

functions on a sample S of inputs as

$$||f(x) - g(\tilde{x})||_S = \frac{1}{m}\sum_{i=1}^{m} 1\{f(x_i) \neq g(\tilde{x_i})\}.$$

This notation is understood in the same way, except, here, the expectation is replaced by an average over a sample, S, of inputs. The value tells us the fraction of samples on which the two disagree. Not surprisingly, these two values, when applied to $h_r(\tilde{x})$ and $h_r^*(x)$, correspond to our testing and training errors, respectively. They are a measure of the difference between the algorithm's hypothesis' classification of the potentially corrupted derived inputs it uses versus the true concept's classification of the uncorrupted input data.

To show a bound on the testing error of our chosen hypothesis, we argue that, at a particular node j, for any $h \in \mathcal{H}$, $||h(\tilde{x}) - h(x)||_D \leq \epsilon_{L,j} + \epsilon_{R,j}$, where ϵ_L, ϵ_R are the errors of the left and right children, respectively, evaluated for the distribution $\mathcal{D}_j$ induced by the teacher's filtering rule for round j. Intuitively, any single hypothesis can only classify the same input differently if a contributing child node has corrupted the signal. In other words, $h(x)$ and $h(\tilde{x})$ can only differ where x and $\tilde{x}$ differ. We know that $\tilde{x}$ only differs from x where some contributing node made an incorrect classification, and therefore is bounded by $\epsilon_{L,j} + \epsilon_{R,j}$.

Hoeffding's inequality [Mitzenmacher and Upfal, 2005] gives us a high probability upper bound on the testing error at a node:

$$\begin{aligned}\Pr\left(\left|||h_r(\tilde{x}) - h_r^*(x)||_{S_j} - ||h_r(\tilde{x}) - h_r^*(x)||_{D_j}\right| \geq \epsilon\right) &\leq 2e^{-2m\epsilon^2} \\ &\leq \delta.\end{aligned}$$

Rearranging the above equation gives us a bound of no more than $m \geq \frac{\ln(\delta/2)}{2\epsilon^2}$ samples to learn the node. Since the teacher is partitioning the original sample into at most 2 subsets as long as we have $2 \cdot \frac{\ln(\delta/2)}{2\epsilon^2}$ examples to start with, we'll always have sufficient training data at a node, and the overall sample will remain polynomial in size.

Further, the above implies with probability $1 - \delta$:

$$\begin{aligned}\epsilon_{j,j} &= ||h_r(\tilde{x}) - h_r^*(x)||_{D_j} \\ &\leq ||h_r(\tilde{x}) - h_r^*(x)||_{S_j} + \epsilon \\ &\leq ||h_r^*(\tilde{x}) - h_r^*(x)||_{S_j} + \epsilon \\ &\leq \epsilon_{L,j} + \epsilon_{R,j} + \epsilon \\ &\leq \epsilon_{L,L} + \epsilon_{R,R} + \epsilon.\end{aligned}$$

■

Lemma 6 *The output of each node in G is represented in the same way as the inputs.*

Proof Since each node represents a Boolean expression, its inputs are Boolean values and its output is a Boolean value. ■

Lemma 7 *The filtering rule used in $\mathcal{A}$ preserves some polynomial fraction of the original sample S.*

Proof As described previously, $\mathcal{A}$ has the teacher exclude the smaller of the two sets of correlated and anti-correlated examples at each round. The larger of these two sets is always guaranteed to be at least size $\frac{|S|}{2}$. In other words, the teacher is always preserving at least $\frac{1}{2}$ of the examples. ■

Lemma 8 *Error incurred at an individual node compounds at most additively as calculation progresses through G.*

Proof We showed in Lemma 5 that, with high probability, $||h_r(\tilde{x}) - h_r^*(x)||_D \leq \epsilon_L + \epsilon_R + \epsilon$. As a result, as we progress towards the root in the CDAG, the errors of child nodes are being combined additively with the learning error of the current node. ■

Theorem 9 *$\mathcal{A}_{BF}$ and $\mathcal{C}_{BF}$ meet the five criteria of an IMPACT teachable problem.*

Proof The proof follows directly from the application of Lemmas 3 through 8. ■

Now that we have shown our setup for $\mathcal{C}_{BF}$ fulfills the five criteria via Theorem 9, we have proven that $\mathcal{C}_{BF}$ is, in fact, IMPACT teachable by simply applying Theorem 2.

4.1.4 Simulation Results

To demonstrate the practical implications of our approach, we conducted a series of simulations with the class of polynomial-sized Boolean Formulae. The target function we taught was the parity function, which can be quite difficult to learn from examples. Formally, given an input of size n, we define the target function to be the parity of a subset of k input bits. We created a sample of m examples, drawn uniformly at random from the space $\{0,1\}^n$. Our IMPACT algorithm for Boolean Functions was then provided different subsets of this sample via the teacher's filtering rule. To compare with learning algorithms developed for the standard supervised learning model, we also gave the full sample to various learning algorithms. Finally, from the same distribution, we created a test sample of 1000 examples. We ran this experiment for 10 independent trials, and averaged over the results. We ran these experiments both on our algorithm and a set of algorithms selected from Weka's library [Hall et al., 2009]. The algorithms we chose, to represent a variety of techniques and performances, were RandomTree, LADTree [Holmes et al., 2001], BFTree [Shi, 2007, Friedman et al., 2000], J48 [Quinlan, 1993], SimpleCart [Breiman et al., 1984], AdaBoostM1 [Freund and Schapire, 1996], and DecisionTable [Kohavi, 1995]. With a training sample size of $m = 10$, all of the algorithms

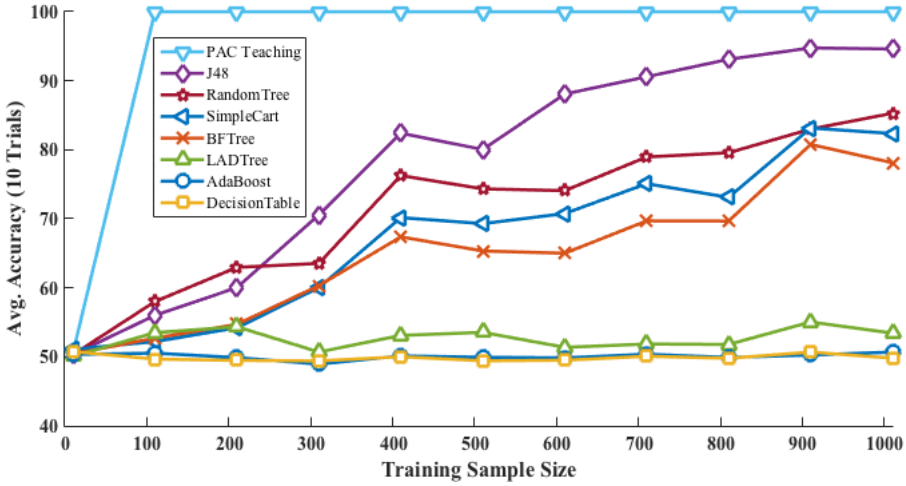


Figure 4.3: Comparison of average accuracy over 10 trials versus sample complexity for an IMPACT learner and standard PAC-framework supervised learners. The function used for this experiment was $f(x_0, \ldots, x_9) = x_1 \otimes x_6 \otimes x_8 \otimes x_9$ with an input size of $n = 10$.

are unable to classify better than random guessing. However, once we introduced additional training data, the IMPACT algorithm quickly outperforms the standard supervised learners. In fact, once the training sample has reached roughly 100 examples, our learner reliably achieved 100% accuracy, while the standard learners are still within 10% of random guessing. Figure 4.3 shows the accuracy of each of these algorithms on the test sample for various training sample sizes.

Additionally, we explored the effect of the size of the function to be learned, k, on our algorithm and the standard supervised algorithms. Each formula in this experiment was the parity of k out of the n input bits chosen uniformly at random. Figure 4.4 shows how well each of the algorithms performed given $m = 75$ training examples for problems of various sizes. The input was of size $n = 10$, and the value on the x-axis shows k, how many bits of the input contributed to the parity function. All of the algorithms were able to correctly learn when the output was a direct copy of a single bit of input. However, once the function was classifying the parity of two bits, 75 samples was insufficient for both AdaBoost and the DecisionTable. The rest of the standard algorithms reverted to having accuracy on par with random guessing by the time there were $k = 7$ contributing variables. Our algorithm, however, remained above 95% accuracy for all values we tested. Choosing a larger sample size would have allowed some of the other algorithms to survive a bit longer. However, anything much larger than $m = 75$ caused our algorithm to have 100% accuracy for problem sizes that were much larger than we were considering.

The purpose of this experiment is not to argue that the algorithms we evaluated are themselves inferior to PAC Teaching. Specifically, our algorithm requires that we have a teacher to help train the learner, so we are unable to make a true, direct comparison. However, the experiment demonstrates the larger point that the IMPACT framework itself makes it possible for appropriate algorithms to grapple with otherwise challenging learning problems. That is, given the teacher described above,

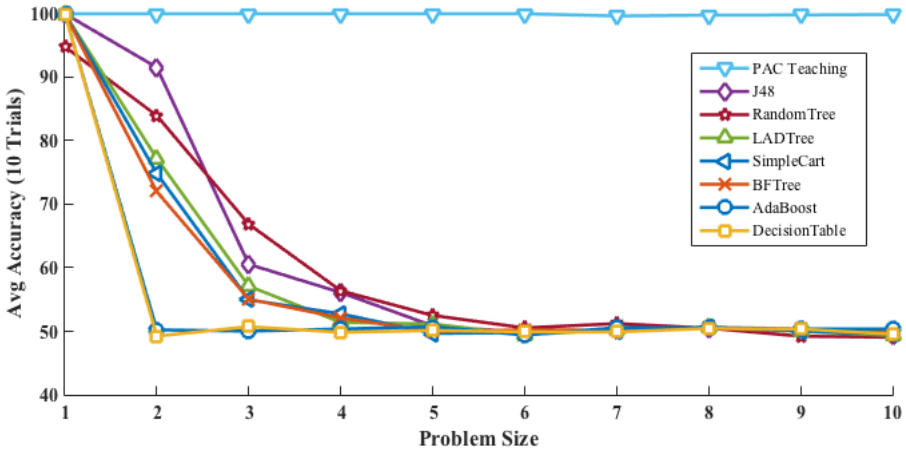


Figure 4.4: Comparison of average accuracy over 10 trials versus problem size, k, for our IMPACT learner and standard supervised learners. All input for this experiment was of size $n = 10$.

our approach is exceptionally effective.

4.2 Threshold Circuits

The *Threshold Circuits* problem has domain $\mathcal{X} = \{0,1\}^n$. The concept class $\mathcal{C}_{TC}$ consists of all constant depth circuits of polynomially many threshold gates in the number of input bits n. Each threshold gate consists of a set of k inputs (either input bits or outputs of other threshold gates in the circuit) and will output a 1 if and only if at least t of the k inputs is 1, and a 0 otherwise. This threshold value t is specific to the gate. The goal of the learner here is to model the circuit.

4.2.1 Algorithm

We will again describe the teaching algorithm for threshold circuits $\mathcal{A}_{TC}$ with regards to the general algorithm in Figure 3.9. The conversion from threshold circuit to CDAG is very straightforward and similar to that of $\mathcal{C}_{BF}$. Each gate becomes a node, and each node has an edge from itself to each of its inputs. Since there were polynomially many gates in the circuit, there will be polynomially many nodes in the CDAG. The learner will need to learn the specific threshold function on the correct inputs represented by each gate, so $\mathcal{H}$ will be the set of possible linear threshold functions at each round. For Z_0 we need to simply provide the learner with the attributes given by $\mathcal{X}$, the base nodes of the CDAG. Unlike $\mathcal{C}_{BF}$, each node represents a threshold gate, not a Boolean operator. Because the negation operation can be represented by the threshold gate itself, there are no explicit negation nodes. For this reason, we do not have to consider propagating negation nodes to the inputs of the tree as we did with $\mathcal{C}_{BF}$. Just as in the Boolean formula case, the teacher will arrange the rounds in a postfix order, so that each of the inputs is sure to be learned before any gates to which it contributes. We note that the input and output of this concept class is identical to $\mathcal{C}_{BF}$, and

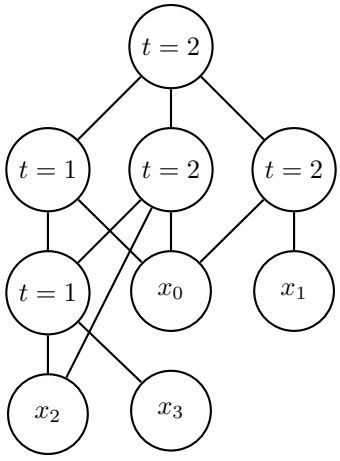


Figure 4.5: This is an example threshold circuit. Each of the base nodes here represent an input bit. The internal nodes of this circuit are labeled with the threshold value for that node. If at least t of the inputs to that node is 1, the output will be 1, otherwise, it will be 0.

will use the terms *correlated*, *anti-correlated*, *relevant*, and *irrelevant* analogously. Accordingly, in **ModerateSample** the teacher will again choose the larger of the correlated or anti-correlated sets as S_r. Because we may train on anti-correlated data, we again add both h_r and its compliment $\bar{h_r}$ to Z_{r-1} to create Z_r in **AugmentAttributes**.

In the case of C_{TC}, the learner will use the well-known perceptron algorithm [Rosenblatt, 1958] as the **PACLearn** routine to learn the behavior of each threshold circuit. The learner can choose a linear threshold function with weights over all inputs (some of which may be set to 0) that can fully capture the behavior of the gate. For example, say we have a threshold gate with k inputs and a threshold of t. This setting can be modeled by putting weight $\frac{t}{k}$ on the k relevant inputs, and 0 elsewhere.

4.2.2 Analysis

Now, to show that the class $\mathcal{C}_{TC}$ is IMPACT teachable using $\mathcal{A}_{TC}$, we just need to prove that this setup satisfies our five criteria.

Lemma 10 *Every $c \in \mathcal{C}_{TC}$ can be efficiently represented as a CDAG, G.*

Proof Consider an arbitrary $c \in \mathcal{C}_{TC}$. Since it is guaranteed to be polynomially sized, we can create one node for each gate in c. Each node in G will be connected to other nodes or inputs for the corresponding gates or inputs in c, respectively. Since c was polynomially sized, we know that G can be created efficiently and will also be polynomially sized. ■

Lemma 11 *Each subconcept in G is PAC learnable.*

Proof Each subconcept in G can be learned by learning an equivalent linear threshold function. These functions are PAC learnable [Blumer et al., 1989]. ■

Lemma 12 *The output of each node in G is represented in the same way as the inputs.*

Proof G is a CDAG that represents a threshold circuit. Each node in G represents a single threshold gate. The input of these gates is a set of Boolean values, and the output is a single Boolean value. Therefore, the inputs and outputs are represented identically. ■

Lemma 13 *The filtering rule used in $\mathcal{A}$ preserves some polynomial fraction of the original sample S.*

Proof Since the filtering rule is identical, this property follows from Lemma 7, directly. ■

Lemma 14 *Error incurred at an individual node compounds at most additively as calculation progresses through G.*

Proof We showed in Lemma 7 and Lemma 8 how the error from a contributing child node can be viewed as a noisy input for the current node being learned. Following the same logic as with $\mathcal{C}_{BF}$, at a given node j, for any $h \in \mathcal{H}$, $||h(\tilde{x}) - h(x)||_D \leq \sum_\ell \epsilon_{\ell,j}$. Rather than just right and left children, we have to account for all children of j, resulting in the term $\sum_\ell \epsilon_{\ell,j}$. Applying Hoeffding's inequality [Mitzenmacher and Upfal, 2005] again gives us a high probability upper bound on the testing error at a node:

$$\begin{aligned}\Pr\left(\left|||h_r(\tilde{x}) - h_r^*(x)||_{S_j} - ||h_r(\tilde{x}) - h_r^*(x)||_{D_j}\right| \geq \epsilon\right) &\leq 2e^{-2m\epsilon^2} \\ &\leq \delta.\end{aligned}$$

Rearranging the terms above implies with probability $1 - \delta$:

$$\begin{aligned}\epsilon_{j,j} &= ||h_r(\tilde{x}) - h_r^*(x)||_{D_j} \\ &\leq ||h_r(\tilde{x}) - h_r^*(x)||_{S_j} + \epsilon \\ &\leq ||h_r^*(\tilde{x}) - h_r^*(x)||_{S_j} + \epsilon \\ &\leq \sum_\ell \epsilon_{\ell,j} + \epsilon \\ &\leq \sum_\ell \epsilon_{\ell,\ell} + \epsilon.\end{aligned}$$

Each node incurs error either from the standard error inherent in learning from examples, ϵ, or as a result of one of the children providing corrupted input $\sum_{\ell=1}^{k} \epsilon_{\ell,j}$. As a result, just as before, as

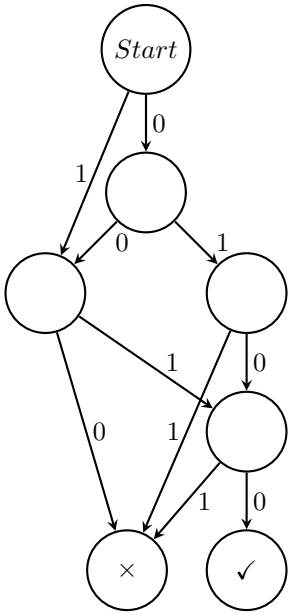


Figure 4.6: This DAG is a representation of an ADFSA. Each bit of the input string is consumed along an edge of the graph. The path that the input takes through the graph determines whether or not the string is accepted (✓) or rejected (×). For example, this automata would accept the string 0100, but would reject 0011.

we progress toward the root in the CDAG, the errors of child nodes are being combined additively with the learning error of the current node. ■

Theorem 15 *$\mathcal{A}_{TC}$ and $\mathcal{C}_{TC}$ meet the five criteria of an IMPACT teachable problem.*

Proof The proof follows directly from the application of Lemmas 10 through 14. ■

Once again, the straightforward application of Theorem 15 to Theorem 2 shows us that $\mathcal{C}_{TC}$ is IMPACT teachable using $\mathcal{A}_{TC}$. While all three problems we explore in this work are fundamentally related, $\mathcal{C}_{BF}$ and $\mathcal{C}_{TC}$ are more similar to each other than $\mathcal{C}_{ADFSA}$ is to the others.

4.3 Acyclic Deterministic Finite State Automata

Acyclic Deterministic Finite State Automata (ADFSA) problem involves learning concepts from the class $\mathcal{C}_{ADFSA}$ of acyclic deterministic finite automata that can recognize strings of length at most n. These automata have a single accept and reject state and have a total number of states polynomial in the input size, n. For simplicity, we consider $\mathcal{X} = \{0,1\}^n$, the set of all Boolean strings of at most length n, but other alphabets are handled similarly. Again, we only require that the learner identify

an ADFSA that is approximately functionally equivalent to the teacher's target ADFSA over the input distribution.

4.3.1 Filtering by Example Alignment

One of the major differences between teaching the previous problems and teaching ADFSA is how the input is handled as we traverse the CDAG. In the other cases, each input example was distributed bit by bit in parallel across a set of input nodes to the CDAG. This description does not hold in the case of the CDAG created by the ADFSA. In this situation, an example is processed one bit at a time as it traverses the CDAG. That means one particular node may be operating on bits in different positions of the examples depending on the length of the example and the path that was taken to arrive at the node. So, if a learner is trying to learn a particular node i, and is given some examples x_1 and x_2, there is no clear way to tell which bits in x_1 and x_2 should be used for evaluating each of the hypotheses when doing ϵ exhaustion. A more obvious example of this is the fact that there may be some nodes which never even evaluate a bit of a particular example, since it is not necessary that every example encounters every node on its path through the CDAG.

To remedy this issue, we have the teacher and the learner agree before learning begins that the learner will make the assumption that all of the examples for a given round have been β-aligned. A set, S, of examples is *β-aligned* for a node i if, for every example in S, it is the case that, in the target ADFSA, c^*, i consumes the β^{th}-to-last bit. Accordingly, the teacher will need to filter the training sample by passing the learner the largest correlated or anti-correlated β-aligned subset for each round. It is important to note that even examples that would not normally pass through a given node can be used in training as they can still be evaluated for a given β-alignment at a given node, just by starting evaluation at that node on on the β^{th}-to-last bit.

4.3.2 Algorithm

Similar to the situation in threshold circuits, the conversion of an ADFSA to a CDAG is uncomplicated. Each node in the ADFSA becomes a node in the CDAG. The edge set translates directly as well. Each node will represent a function that consumes the next bit of the input and returns a pair of next nodes to send the string to as well as the shortened input string to pass along. Initially, Z_0 contains only the terminal nodes. The learner's hypothesis space $\mathcal{H}$ is all of the ADFSA that can be created in the following way. Create a new initial node i. For each of $0, 1$, choose an ADFSA from Z_{i-1} to be the child node to traverse to depending on the β^{th}-to-the-last bit of the example. As nodes are learned, **AugmentAttributes**(Z_{r-1}, h_r) stores each as a small ADFSA. In doing so, the learner is able to gradually grow the full ADFSA, represented, as before, by the hypothesis chosen in the final round of learning. This learning process ensures that there is a single initial state and that the only terminal states are the accept and reject states. As is implied by this process, the teacher teaches nodes in a postfix order. A diagram showing this procedure can be seen in Figure 4.7.

At each round, the learner must create a new ADFSA node and choose where the two outgoing

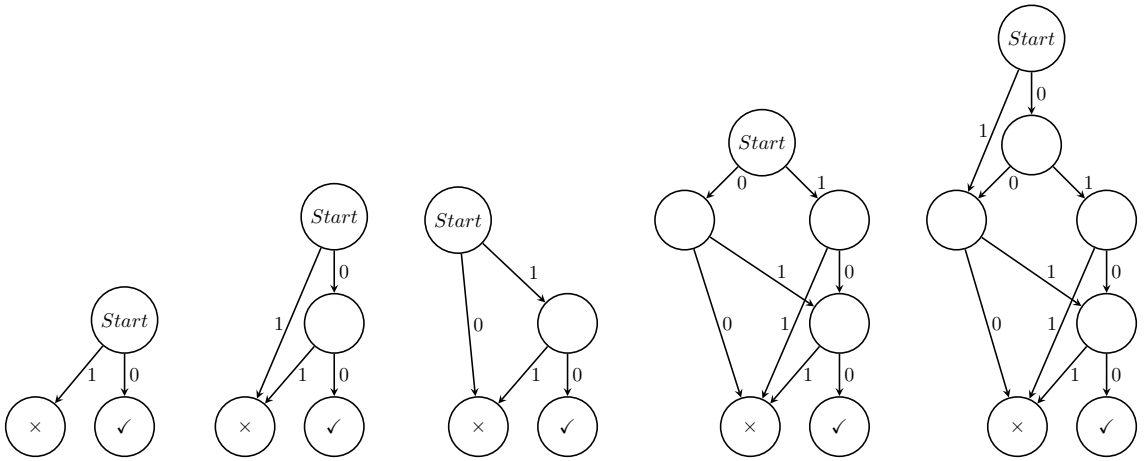


Figure 4.7: The above image shows the process of gradually learning an ADFSA. Each image from left to right shows the hypothesis learned by the learner in a single round. In each subsequent round the learner can direct edges out from a new start state to either the accepting state (✓), rejecting state (×), or a previously created state. The teacher trains in a postfix order, so the order of the second and third hypothesis is interchangeable.

edges get connected. The attribute space for the learner begins with just an accepting and a rejecting terminal node. After each round, the learner augments the attribute space with its current hypothesis and the complement of the current hypothesis. The complement has identical structure, but any terminal accepting or rejecting nodes are swapped. Again, the learner can use ϵ exhaustion as **PACLearn** to choose the best hypothesis for a given node. Because the learner knows to expect aligned examples, it can evaluate, for each possible offset of the input, all potential hypotheses, then choose the one that most closely matches the data.

4.3.3 Analysis

Using the learning procedure described above, referred to now as $\mathcal{A}_{ADFSA}$, C_{ADFSA} is IMPACT teachable.

Lemma 16 *Every $c \in \mathcal{C}_{ADFSA}$ can be efficiently represented as a CDAG, G.*

Proof Consider an arbitrary $c \in \mathcal{C}_{ADFSA}$. Since it is guaranteed to be polynomially sized, we can create one node in G for each node in c. Each node in G will be connected to the corresponding neighbors from c. Since c was polynomially sized, we know that G can be created efficiently and will also be polynomially sized. ■

Lemma 17 *Each subconcept in G is PAC learnable.*

Proof Each subconcept in G is learned via ϵ exhaustion. The hypothesis space for a given round consists of the two terminal states and any previously learned hypotheses and their complements,

all of which are evaluated at every possible β-alignment. There are at most $2 \cdot poly(n)$ previously learned hypotheses, since there are $poly(n)$ rounds total. Additionally, there are at most n β-alignments, since all inputs are at most size n. Therefore, the total size of the hypothesis space is $2n \cdot poly(n) = O(poly(n))$. We know that polynomaially-sized hypothesis spaces are PAC learnable via ϵ exhaustion. ■

Lemma 18 *The output of each node in G is represented in the same way as the inputs.*

Proof G is a CDAG that represents an ADFSA. Each node in the G takes a bitstring of length at most n and consumes one bit of it. The remainder of the bitstring is then output and sent to the next node. Therefore, both the input and output are represented as a bitstring of length at most n for all nodes in G. ■

Lemma 19 *The filtering rule used in $\mathcal{A}$ preserves some polynomial fraction of the original sample S.*

Proof The filtering rule has the teacher divide the sample into β-aligned subsets. For each example that touches i, there are at most n β-alignments. These subsets of β-aligned examples are further divided into correlated and anti-correlated subsets. Since there are $\frac{1}{2n}$ possible subsamples, at least one subset has to be of size at least $\frac{1}{2n}$, by the pigeon-hole principle. ■

Lemma 20 *Error incurred at an individual node compounds at most additively as calculation progresses through G.*

Proof Similar to the C_{BF} and C_{TC} cases, we can calculate the error of a given node i in G as the sum of two values. The first is the error ϵ_i that comes from the fact that i was learned from examples. The second is the errors of the nodes with edges leading to i. This error is similar to the error of the left and right children in the $\mathcal{C}_{BF}$ example. That is, errors in decisions made by earlier nodes could compound with each decision, or edge in the graph. The concern here is that i may be contributing to too many nodes for the error to compound additively. However, each node only has 2 outgoing edges. Therefore, the sum of the error at the terminal nodes will be $\Sigma_{i \in G} 2\epsilon_i$. ■

Theorem 21 *$\mathcal{A}_{ADFSA}$ and $\mathcal{C}_{ADFSA}$ meet the five criteria of an IMPACT teachable problem.*

Proof The proof follows directly from the application of Lemmas 16 through 20. ■

Since $\mathcal{A}_{ADFSA}$ and $\mathcal{C}_{ADFSA}$ meet the five criteria we designed, we have shown that $\mathcal{C}_{ADFSA}$ is also IMPACT teachable.

4.4 Teaching Classification Problems

The problems we have explored in this chapter have shown precisely what the power of the IMPACT model is. These concept classes, in their full generality, are too complex to be learned by a standard PAC learner. The concepts are a complicated composition of small subtle operations done over the domain's attributes in each setting. Boolean formulae are comprised of many nested binary operations that can interact with each other in interesting and confusing ways. Threshold circuits can involve complicated subsets of attributes and other operations that compose to make complex concepts. Finally, ADFSA can have intricate arrangements of mappings between states that makes them hard to learn in their full generality. However, we have seen that decomposing these complicated concepts and learning them in smaller, more manageable subconcepts is a sufficient change to allow the learner to be successful. Perhaps the hardness of these problems comes from the relationships between the subconcepts, rather than the concepts themselves. In the following section, we will investigate the ability of the IMPACT framework to solve yet another type of classification problem, separate from those studied by Rivest and Sloan.

Chapter 5

Polygon Teaching

In this section we discuss yet another hard learning problem and show how using the IMPACT model, we are able to teach a concept class that is provably unlearnable. That class is the class of polygons, $\mathcal{C}_P$. We will formally present the problem and give a proof of hardness in the PAC setting. Then we will present and analyse an IMPACT style algorithm for teaching $\mathcal{C}_P$.

5.1 Problem Description

A polygon is a plane figure whose boundary is composed of a set of line segments that enclose a space. We note that $\mathcal{C}_P$, for our work, is the class that contains all sets of non-overlapping polygons, convex or concave, as well as polygons with holes. We say that a polygon p is a polygon with a hole if there is at least one additional polygon ℓ that is completely inside of p. The space enclosed by ℓ is considered to be the hole and does not count as part of p. Figure 5.1 contains an examples of a polygon with holes. We can describe a polygon fully by writing it as a set of sets of n tuples. Each set represents one of the non-overlapping polygons. Each tuple within that set lists the vertices of a polygon in order, with one tuple for the outer polygon and each of its possible holes. Having concretely defined our concept class, we will now describe the rest of the learning problem. In this classification problem, our domain $\mathcal{X} = \mathbb{R}^2$ is the two dimensional plane. The label space is $\mathcal{Y} = \{-1, 1\}$, as before, since we are still doing binary classification. The goal of the learner in this problem is to learn a representation that allows them to accurately predict the label y of a given point x when y is determined by $c \in \mathcal{C}_P$. Each of the points on the boundary or interior of the polygon described by c are labeled with a 1, while the points on the exterior or in a hole are labeled as -1.

5.1.1 Learnability

We will now show that the class $\mathcal{C}_{PS}$ is unlearnable in the standard PAC setting. We do so by arguing that the VC dimension of $\mathcal{C}_P$ is infinite, since it is known that classes with infinite VC

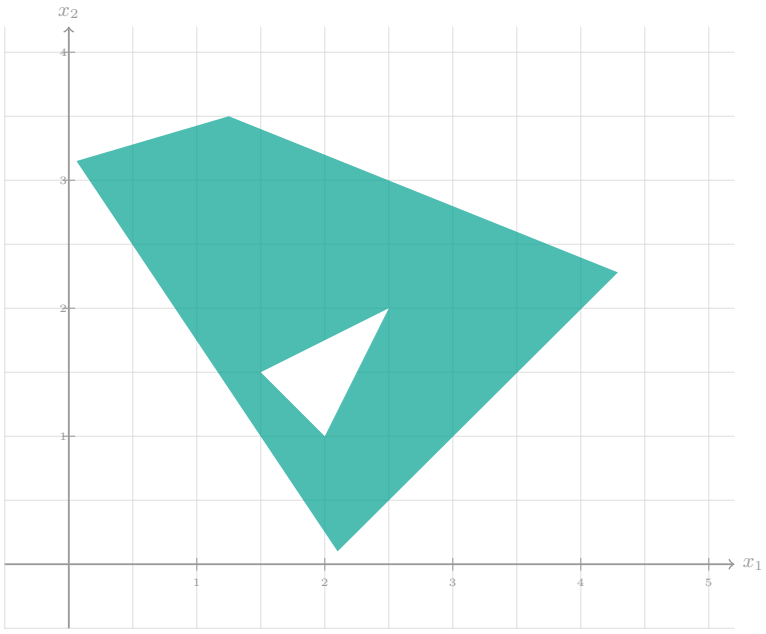


Figure 5.1: The above diagram gives an example of a polygon with a hole. For these polygons, only the colored areas are considered to be part of the polygon.

dimension are not PAC learnable.

Lemma 22 *The class $\mathcal{C}_P$ of sets of non-overlapping polygons has infinite VC dimension.*

Proof *In order to show that $\mathcal{C}_P$ is infinite, we must argue that it can shatter arbitrarily large sets of data. Consider S, a set of data points of an arbitrarily size, m. For each of the 2^m possible labelings of S, we will construct the $c \in \mathcal{C}_P$ that provides that labeling. For each data point labeled 1 in S, draw an infinitesimally small triangle around that point. Since we can construct this c for any labeling of S and for any size m, S will always be shattered by $\mathcal{C}_P$. Therefor, $\mathcal{C}_P$ has an infinite VC dimension.* ■

Lemma 22 is sufficient evidence to conclude that the class $\mathcal{C}_P$ is not PAC learnable. However, just as we demonstrated in Chapter 4, classes that are unlearnable in the PAC setting may still be teachable in the IMPACT setting. To show this, we will now present and analyse $\mathcal{A}_P$, an IMPACT algorithm that successfully teaches $\mathcal{C}_P$.

5.2 Algorithm

Just as we did with the algorithms discussed in Chapter 4, we have to define each of the separate parts of the general algorithm that are specific to learning $\mathcal{C}_P$. In this case, Z_0 will be defined as the two attributes of $\mathcal{X}$ only. At each individual round the hypothesis space will be the space of

all real valued weight vectors of length Z_r. We choose this as the hypothesis space because this algorithm will run the perceptron learning algorithm as its **PACLearn** subroutine. At each round, these perceptron classifiers will be added to the attribute space in **AugmentAttributes**. Similar to $\mathcal{C}_{BF}$, the teacher will filter out the smaller of the correlated or anti-correlated examples from the training sets in **ModerateSample**. The teacher will train the nodes in postorder, so that each of the requisite nodes are learned before they are needed.

The idea behind this algorithm is that we will use perceptrons to learn the polygons one edge at a time. Since these perceptrons will output labels that are in $\{-1, 1\}$, they can be treated just like the other attributes of the domain. In doing so, we allow perceptrons in subsequent rounds to consider not only combinations including the domain's attributes, but also the output of perceptrons from previous rounds. This allows for us to scale the complexity of the hypothesis space in each round as we have seen is a critical piece to making any IMPACT algorithm successful.

5.2.1 Analysis

Following the same format from Chapter 4, we will show that $\mathcal{C}_P$ and $\mathcal{A}_P$ both fulfill the criteria from Figure 3.5. Again, after doing so, we will argue that the proofs from Section 3.4.5 then are sufficient to prove that the class of sets of polygons with holes is IMPACT teachable.

Lemma 23 *Every $c \in \mathcal{C}_P$ can be efficiently represented as a CDAG, G.*

Proof Consider an arbitrary set of polygons with holes, $c \in \mathcal{C}_P$. We know c can be represented as a set of sets of tuples. Each set of tuples represents one polygon and its associated holes. If there are N total points in c, then there are a total of N edges that make up all of the polygons and holes in c, since we can label each line segment of the polygon or hole by the point at which it starts. For our CDAG, we will create a height 1 internal node for each of these edges. Each of these nodes will represent an inequality that contains the edge and the space the polygon occupies. For example, if a polygon has a bottom edge along $x = 1$, the node for that edge will be the inequality $x_0 \geq 1$. Now we can consider constructing polygons by doing Boolean operations on these inequalities. For example, if we use x_0 and x_1 to represent the horizontal and vertical axes, respectively, a 1-by-1 square that is at the origin might have the following formula:

$$((x_0 \geq 0) \wedge (x_1 \geq 0)) \wedge ((x_0 \leq 1) \wedge (x_1 \leq 1))$$

. Building the CDAG in this way will result in $2N - 1$ total internal nodes in the CDAG. We will have one leaf node for each attribute of the domain. Therefor the size of the CDAG will remain polynomial in the size of the description c. ■

Lemma 24 *Each subconcept in G is PAC learnable.*

Proof There are two types of nodes that we will learn over in G. The height 1 nodes that represent inequalities and the other internal nodes that represent boolean operations over those inequalities.

Perceptrons can be used to implement both the inequality and the boolean formula nodes. Further, perceptrons are PAC learnable. ■

Lemma 25 *The output of each node in G is represented in the same way as the inputs.*

Proof Since each node can be implemented as a perceptron, we know that the output at each node will be in $[-1, 1] \subset \mathbb{R}$. Additionally, each of the attributes of $\mathcal{X}$, and the inputs to the perceptron, are $x_i \in \mathbb{R}$. Therefor, the input and outputs of each node are represented identically. ■

Lemma 26 *The filtering rule used in $\mathcal{P}$ preserves some polynomial fraction of the original sample S.*

Proof As described previously, $\mathcal{P}$ has the teacher exclude the smaller of the two sets of correlated and anti-correlated examples at each round. The larger of these two sets is always guaranteed to be at least size $\frac{|S|}{2}$. In other words, the teacher is always preserving at least $\frac{1}{2}$ of the examples. ■

Lemma 27 *Error incurred at an individual node compounds at most additively as calculation progresses through G.*

Proof Error compounds when one of the children of the node has incurred error, passing inaccurate information to it's parent. All of the nodes which have learned nodes as their children are at a height of greater than 1 in our CDAG. Therefore, they represent Boolean operations. Since these nodes are functionally the same as those in $\mathcal{A}_{BF}$, we can use the proof of Lemma 8 to prove the same for $\mathcal{A}_P$. ■

Theorem 28 *$\mathcal{A}_P$ and $\mathcal{C}_P$ meet the five criteria of an IMPACT teachable problem.*

Proof The proof follows directly from the application of Lemmas 23 through 27. ■

The above proofs, based on the format from Section 3.4.5, show that the class $\mathcal{C}_P$ is IMPACT learnable. In doing so, we have demonstrated that the IMPACT learning model can be successfully used in teaching concepts, outside of those studied by Rivest and Sloan, that meet our five criteria. Now, we make a small shift in the focus of this dissertation. In theory, we have explored and discussed the efficacy and efficiency of our model. For the remaining chapters, we will explore what happens when we introduce a human component to the ideas we have established in the IMPACT model. We will first explore using a human as the teacher in a supervised learning setting in Chapter 6. Following that, we will have humans train IMPACT-like learning agents to act according to arbitrarily complex LTL defined tasks in Chapter 7.

Chapter 6

IMPACT with Human Teachers

While the IMPACT framework is theoretically sound, it does make some important assumptions that influence the usability of the method. Mainly, IMPACT requires the presence of a teacher who already knows the target concept well enough to successfully train the learner. As a consequence, in addition to knowing the concept *a priori*, the teacher must also be capable of decomposing the complex concept into simpler, learnable components for which she can also provide the correctly filtered samples for learning. In the standard approach to machine learning, professionals and researchers might be using machine-learning methods to discover and exploit unknown patterns in large data sets. In this case, while decomposing complex concepts into simpler ones may be reasonably expected of these experts, it is often the case that the true target concept is not already known. An alternative machine-learning scenario involves having non-expert end users interact and work with the learning systems. In this situation, it might be reasonable to assume that the end users have an idea of their target concepts. That is, they know some task, rule, or behavior that the system should be learning. However, it might not be reasonable to assume that without specialized training the average end users have the technical skills or knowledge to impart these tasks, rules, or behaviors to the system. In that case, the question of whether or not the end user can decompose and filter examples properly for learning to occur with an IMPACT system becomes an important one. We hypothesize that, for non-trivial problems, the average end user is able to successfully decompose and train a learner in the IMPACT framework. We created a set of studies to test this claim.

6.1 Participant Recruitment

The participants in each study were asked to complete the task of teaching a Boolean formula to an IMPACT agent. The participants were recruited using Amazon Mechanical Turk (AMT). The participants from AMT ("turkers") were filtered by several criteria before being allowed to participate in any study. Turkers needed to be at least 18 years old, located in the United States, have at least 1000 completed HITs, and have a 95% HIT approval rate. We recruited 100 unique

turkers in total. Compensation was given for completing our study, but there was no bonus reward given based on performance.

6.2 Boolean Formulae for Non-Experts

Since we did not want to require background knowledge in logic, we disguised the Boolean formula problem as a cleanup task. The decision to ground the abstract concept of Boolean formulae with physical, relatable objects was made to help the participants get a concrete understanding of the task, without requiring them to learn Boolean logic explicitly.

6.2.1 Modelling the Problem

We decided to model Boolean formulas as "rules" for deciding the ownership of children's toys. The thought was that non-experts could reason about the type of rules that we would want to encode as Boolean formula, but would not, however, have any knowledge of Boolean logic, formally. To ground the problem we came up with physical attributes to correspond to the boolean literals of the domain space $\mathcal{X} = \mathbb{R}^6$. Physical attributes included things like shape, size, color, fill, outline, and orientation. Each literal was made to represent some binary attribute of a toy. This allowed us to build small Boolean formula from three to four attributes and still have a few attributes left over as extraneous distractors. The example bit strings were represented by toys that have the attributes as described by their literals. For example, if x_1, x_2, x_3 represent the attributes of shape, color, and pattern, respectively, the string 011 might be a toy that is circular, blue, and striped. While each of these physical attributes is not necessarily binary or fully objective in the real world, we limited the space of objects we showed to participants in a way to make them binary. For example, for some participants the symbols they saw included only blue or orange toys.

6.3 Experiment Description

In our study, the participants completed a binary meta-classification task (classifying classified example toys). The two possible labels in the meta task are "include" or "omit" for each of the example toys. The concept defining the correct meta-labeling is the filtering function of the IMPACT framework discussed above. Because the goal of this research is to evaluate human participants' ability to complete this binary meta-classification task, we evaluated their performance using standard classification metrics of *precision* and *recall*. Precision is the ratio of true positives to the total number of true and false positives. High precision means that the participant included the correct examples and did not include too many examples erroneously. Recall is the ratio of true positives to the total number of true positives and false negatives. A high recall means that the participant did not omit too many examples that should have been included. Both of these metrics have range $[0, 1]$, with 1 being a perfect score.

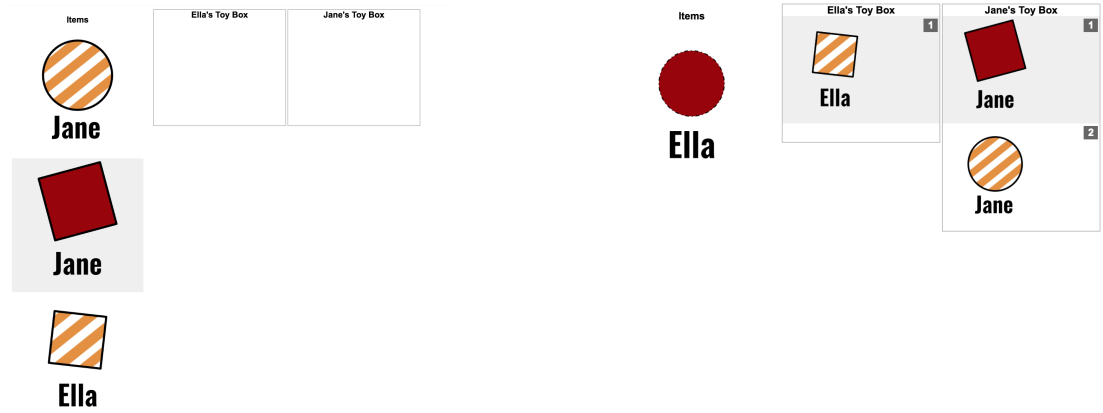


Figure 6.1: Two screenshots of participants choosing training examples. The left image shows the screen presented to turkers. On the right, the turker has successfully completed choosing toys to put in each box, leaving one unselected.

In this setting, while both high precision and high recall is ideal, recall is somewhat secondary to precision. Precision is more important in this setting because including a confusing example in the agent's training set (a false positive) is more detrimental to the learner's performance than omitting a few good examples (false negatives). It is easy to achieve high precision by selecting only very few obvious examples and nothing else. This too, is problematic for learning if taken to the extreme, which is why we still report recall in our results.

6.4 Tutorial

Participants were guided through a tutorial task where they were able to see an example of what was expected of them. First, they were presented with the target concept (called a "rule" to participants), "Jane gets toys that are BOTH violet or circular AND green or square-shaped, and Ella gets all of the other toys." Participants were told that the robot they were training could learn over time, but that he could only understand small bits of information at once, so he needed to slowly build his understanding of a large rule by composing smaller rules. Participants were shown multiple choice options of various potential curricula of smaller rules and asked to choose which seemed most reasonable for teaching their robot. Upon selecting any answer, they were given an explanation of why it was the correct or incorrect option. They were allowed to proceed with the study only after discovering the correct curriculum and reading the explanation.

Next, the participants were shown an example training-round problem (shown in Figure 6.1), which walked them through the logic of why each toy was or was not appropriate to include given the current target smaller rule. Again, only after participants read the explanation and successfully completed the task were they allowed to move on to the actual experimental part of the study.

This tutorial was the only training we gave to the participants before allowing them to attempt the task. We wanted to be careful not to teach participants what Boolean logic was or teach

Task	Jen gets toys that are...
CNF	hearts or striped AND are also yellow or has a shadow.
DNF	both hearts and striped OR toys that are yellow and have shadows.
MixedConj	both yellow and shadowed AND also are hearts or striped.
MixedDisj	striped or hearts or toys that are both yellow and shadowed.
XOR	either heart shaped or striped and are also not both heart shaped and striped.

Figure 6.2: Above are the full text descriptions given to the turkers for each rule, labeled by the corresponding tasks.

them a procedure for decomposing rules. However, feedback from pilot studies suggested that this much explanation was necessary to establish a common language with participants and create an understanding of what was expected of them.

6.5 Task Completion

In this part of the experiment, participants were randomly assigned to one of five study groups. Each group was then presented with a different rule. Each of these rules was based on a form of a Boolean function, including a conjunction of two disjunctions (CNF), a disjunction of two conjunctions (DNF), a conjunction of a conjunction and disjunction (MixedConj), a disjunction of a conjunction and a disjunction (MixedDisj), and a 2-bit exclusive or (XOR). The full text for each rule provided to the turkers is given in Figure 6.2. Participants were asked to write down the smaller subrules they would use in training the robot and provide the number of rounds they would need to train—one per subrule, and optionally a final round for the overall rule. Based on the answer to that question, the participants were given however many rounds they requested to train the robot for their rule. This section of the experiment closely mirrored the tutorial from Figure 6.1.

6.6 Data Acceptance Criteria

In terms of filtering the data to get our results, we took the most conservative approach that was reasonable. We did discard some data only when we were convinced that the participant genuinely only attempted to avoid attempting the task we presented. While these cases were very uncommon there were some instances of it occurring. These discarded data included participant reponses such as the participants entering nonsense text or copying and pasting the instructions into open ended response fields. In these situations, the actual sorting of toys was left completely unattempted. In cases like this, the response was deemed invalid and did not contribute to the data. From our processing of the data there were very few (less than 5%) of these invalid responses. However, we cannot calculate the exact number of these cases since the AMT system and our own survey software filters some of these invalid or incomplete responses automatically.

Other instances of of potentially bad data occurred when participants made errors that could have arguably disqualified them, but did not on a technicality. For example, some participants used

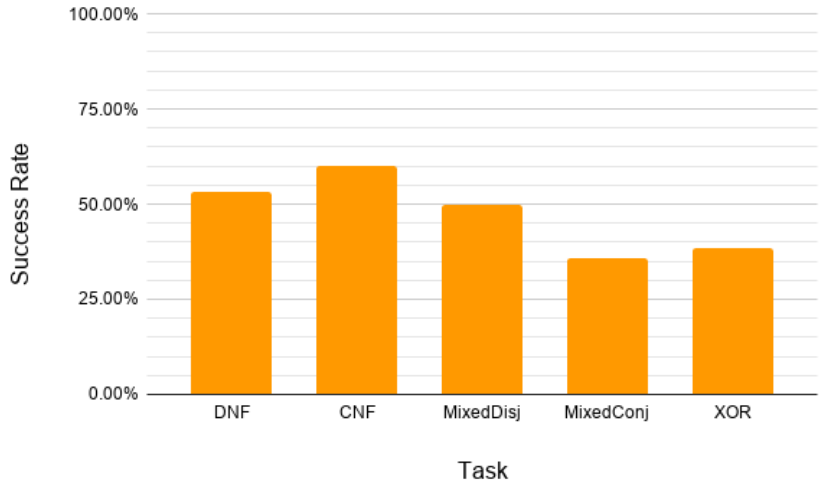


Figure 6.3: This chart shows the percentage of participants that were able to successfully decompose the target rule into a series of smaller rules to train the robot.

the overall target formula as the only round in their training plan. However, even though they likely copied and pasted this rule into their planned curriculum, if they made any sort of attempt at training the agent (rather than just clicking continue without selecting any examples), we counted this as a valid, but failed attempt. Without any means of verifying with the participant we cannot be certain that this was not an honest, but misguided, approach at solving our task.

6.7 Results

Overall, the results of the experiment suggests that end-users can be successful at decomposing and training formulae in the IMPACT setting. Because of the nature of AMT, some of the participant responses were ignored in the calculation of the decomposition success rate. For the evaluation of the other performance metrics, we limited their calculation to participants who had successfully decomposed the formula, since, if one didn't decompose the formula properly, there was no chance at training correctly. Additionally, when evaluating the decompositions, it became clear that some participants knew how to decompose the rule, but made a small mistake in recording what they thought the smaller rule was—using an "and" or an "or" incorrectly. It was apparent from their training rounds that they knew the correct subrules, but since they were not technically reported correctly, we counted them as incorrect. We used this strict interpretation of correctness to provide the most conservative results possible.

The percentage of participants with the ability to decompose a rule varies from the mid to high 30% range for the harder tasks (MixConj, XOR). It is unsurprising to see the participants struggle, because there is some ambiguity in the written English description of these rules. "Or" tends to be ambiguous (inclusive vs exclusive) and having a long string of "and" and "or" in the rule description

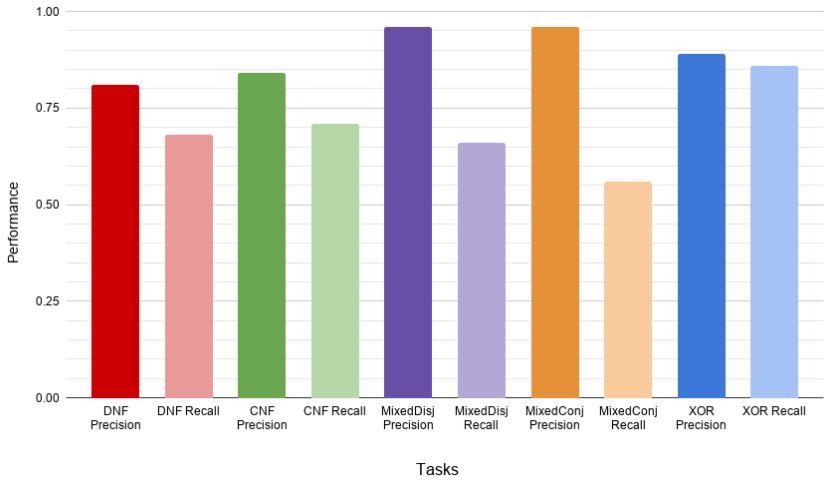


Figure 6.4: This chart shows the average precision and recall for each task. The average was computed across all rounds and all participants. For our work, precision is the primary measure of success.

only exacerbates the problem. Attempting to explain in more precise language with greater detail while not relying on domain knowledge of concepts like inclusivity, for example explicitly writing "A or B or both" or "A or B but not both" complicates the rule description further. The other formula structures (DNF, CNF, and MixedDisj) saw successful decomposition rates ranging from 50% to 60%. While decomposing these rules is not easy for any end user, it is possible for a majority of them in some cases.

In Figure 6.3, we see that participants who are capable of decomposing formulae properly can also choose examples to use in training well. In these measurements as well, we exercised a strict interpretation of correctness. In some cases, participants placed toys in the incorrect box—a toy labeled for Jen was put in Syd's box, or vice versa. In this case, both performance metrics can take a hit. The toy does not appear in its box when it should (a false negative), but it also shows up in a box where it should not (a false positive). When this happened, we counted this one example as both a false negative and a false positive, impacting both the precision and recall, since it was impossible to distinguish a small user error from an intentional incorrect response. Regardless, every formula structure type was trained with a precision of at least 0.80, with MixedConj and MixedDisj as high as 0.96. Recall, while less important for training, was still measured to be at least 0.56 for each structure and sometimes as high as 0.70–0.86. A chart displaying these results is shown in Figure 6.4.

This work has shown that there is evidence to suggest that non-expert end users are able to successfully train in our IMPACT setting. While the results of this user study were not overwhelmingly positive, they do show potential considering that our users had very little experience with our model and presumably no formal understanding of Boolean logic. The participants were working in

the space of English text descriptions which can be ambiguous even with the most careful experiment design. In the following chapter, we will turn the focus of our work away from the supervised learning setting and toward the reinforcement learning setting. Again we will study the feasibility of having human teachers train an IMPACT style agent, although we are technically departing from the IMPACT model in a strict sense of the definition.

Chapter 7

IMPACT for Reinforcement Learning

7.1 Reinforcement Learning

Reinforcement learning is a category of machine-learning problems in which an agent is motivated by rewards (and punishments) to learn a desired behavior. In this setup, the learner has agency to make decisions and take actions in the context of an environment. The environment provides feedback via rewards that signal to the learner the value of their actions taken. As the agent is able to explore more of the environment and discover the consequence of different actions, the objective is for it to learn how to act effectively in the world.

More formally, we define the agent's environment as a Markov decision process (MDP). An MDP, $M = \langle S, A, T, R \rangle$, is a set of state, actions, transitions, and rewards that fully describe the environment of a given agent. The state space $S = \{s_1, s_2, \ldots, s_i\}$ is the set of all possible configurations of properties of the world. The action set $A = \{a_1, a_2, \ldots, a_j\}$ is the set of all possible actions the agent can take at each time step. The transition function $T : S \times A \to S$ provides a new state given the current state and the action chosen by an agent. Finally, $R : S \to \mathbb{R}$ is the reward function, which takes a state and assigns a real-valued reward amount to that state. At each time step, the agent makes a decision as to which action to take. The transition function uses that action and the current state to determine the next state of the world. Then, the reward function calculates the reward of this new state. The reward is passed as feedback to the agent along with the new state. This process repeats itself until some terminal state is encountered that ends the process. A visualisation of this is given in Figure 7.1.

7.1.1 User-Generated Reinforcement

In this work, we investigate a reinforcement-learning problem in which a user generates the feedback signal. Thus, rather than having a predetermined reward function specifying the feedback to the

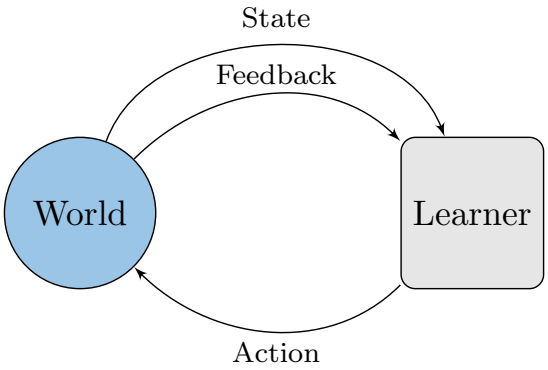


Figure 7.1: The above diagram shows the flow of information in a reinforcement learning setting. The agent makes a decision to take an action at the current time step. Then based on the current state and the action, the world uses the transition and reward functions to determine the next state and the reward. The next state is given to the agent as well as the reward feedback. Then, the process repeats from this next state until some terminal state is reached.

agent, a human decides the reward at each step in the process. Figure 7.2 shows how this setup works. Introducing a person into the reinforcement-learning model adds some complicating factors. Primarily, humans can be very unpredictable. We will no longer have a mathematically defined predetermined knowledge of what the reward signal will look like. In the sections to follow, we will discuss how an algorithm can handle such potentially unreliable reward signals and still be able to learn something reasonable.

7.2 Linear Temporal Logic (LTL)

The standard means of conveying a desired behavior to an RL agent is the reward function, R, discussed in Section 7.1. The reward function, while powerful, suffers from some serious drawbacks when it comes to practical use. First, designing a reward function can be a difficult task in and of itself. Even simple tasks like having an agent traverse an MDP to a goal state can have subtleties in the reward function that elicit unintended behavior. Translating behavior into the mathematical language of reward values can be challenging even for experts, let alone potential end users. Additionally, classic RL reward functions are simply incapable of representing a large set of behaviors that might be desirable. For instance, having an agent patrol between two states is not a task that can be represented using standard reward functions. This chapter addresses both of these issues by using a different means of representing tasks for RL.

Linear temporal logic (LTL) is a language for representing logical propositions whose truth values are evaluated over a length of time. LTL introduces new operators like next, until, eventually, and always to standard propositional logic. These temporal operators allow for the evaluation of the propositions not just at the current moment in time, but also into the future. Using these temporal

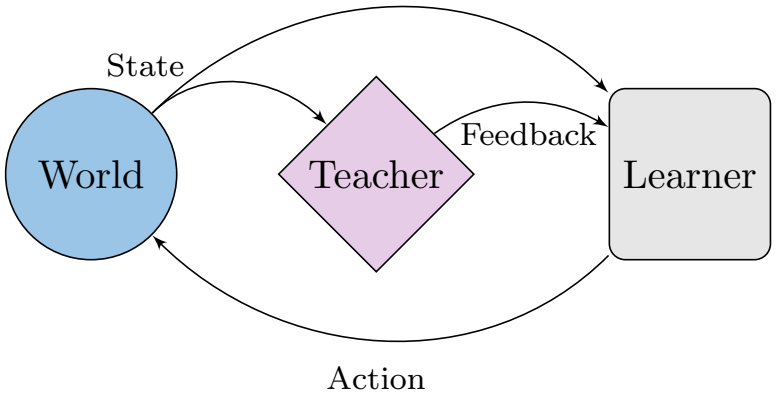


Figure 7.2: The flow of a system with a human providing reward feedback. At each time step, the human teacher can observe the new state generated by the world and determine how to give feedback to the learner.

operators alongside the standard negation, and, or, and implication operators allows for a very richly expressive task-specification language. We can now encourage the agent to have behavior that adheres to a certain LTL specification. For example, we might specify to the agent that it needs to make decisions in such a way that some proposition ϕ holds until some condition ψ is met. Or we might want to make sure that the agent performs some task while maintaining that some proposition ϕ is always holds. In these scenarios, ϕ and ψ could be defined as any relevant state information about the world in which the agent is operating.

While LTL (and Boolean logic, generally) also has its own subtleties in interpreting formulas as behavior, the syntax of LTL seems much more human-readable and end-user friendly than convoluted purely mathematical reward functions. This intuition provides some motivation for studying LTL in this context, ensuring it is theoretically sound, and verifying that it is usable for non-experts in some useful setting. For example, LTL allows us to provide goals that are explicitly task oriented. Imagine an agent acting in a real world home. A possible LTL goal might be something like "If it's dark outside, make sure you eventually turn the porch light on."

LTL is not without its own challenges, however. One such challenge is that the operators in LTL lack a sense of urgency that can be encoded in reward functions. That is because LTL is, by definition, evaluated over truth values that extend infinitely forward in time. For example, the eventually operation truly means that in the most literal sense. In our previous example, of turning on the porch light, even if the agent waited three weeks after it got dark to turn the light on, the goal would be satisfied. That is, as long as the proposition becomes true at some point in the future, no matter how distant, the eventually operator evaluates to True. This property of LTL is definitely not desirable when describing tasks to agents. Oftentimes we want tasks completed as soon as possible and almost never at some unspecified time in the distant future. For this reason, we employ the use of a variant of LTL known as geometric LTL (GLTL) Littman et al. [2017]. In

GLTL, at each timestep there is some probability γ that a GLTL operator will expire. At that point in time, evaluation stops for that operator specifically. So, for example, if an eventually operator expires and its condition has not yet been satisfied, then it evaluates to False, even if it becomes satisfied at some point in time beyond its expiration. The analogous arguments hold for until and eventually.

This geometric expiration timer on the operators of GLTL allows us to encourage urgency in the agents actions. The closer γ is to 1, the more likely it is that the operator will expire soon, and the faster the agent will need to satisfy it. As γ approaches 0, GLTL operators approach the behavior of standard LTL.

7.3 Learning LTL Formula

We define the problem of learning an LTL formula in the following way. Given a set of experiences $< s, a, s' >$ with an associated feedback label $\{-1, +1\}$, the learner must derive a formula from the space of all possible LTL formulas $\mathcal{C}_{LTL}$ over the state space domain S.The task of the learner is to choose a formula that coincides as closely as possible with the formula that generates the feedback signal. We define a concrete example of such a problem as well as a metric for measuring success in Section 7.4.

7.4 Teaching Arbitrarily Complex LTL Tasks

The goal of this work is to show that it is possible for a human end-user to successfully and efficiently train an agent to display behavior consistent with a task described by an arbitrarily complex GLTL formula. To do so, we will have the teacher break the complex formula into more digestible subformulas to teach over multiple rounds of learning, just as we did in Chapter 4.

In our work, the agent's environment will be an MDP of a GridWorld domain (discussed at length in Section 7.4.1) that is a formal representation of an example user's home's kitchen. Rather than have the typical reward functions described in Section 7.1, we use a GLTL expression from Section 7.2 to represent the agent's goal. These GLTL expressions will be referred to as *tasks*. One task, in particular, we will call a *mission*. The mission will be the task that represents the overall goal of the learning procedure—the complex formula that is the target of the entire process.

The teacher will be presented with a GLTL formula describing a mission. It will be their job to determine how best to divide the mission into smaller, trainable tasks. Within each round of learning, the teacher will provide either positive or negative feedback as the agent makes decisions in the world, to encourage the agent to exhibit behavior consistent with the current task being trained. Just like with IMPACT and classification, the agent will be able to store what it learned in each round to help learning progress in subsequent rounds. This process will iterate until the teacher decides to terminate the process.

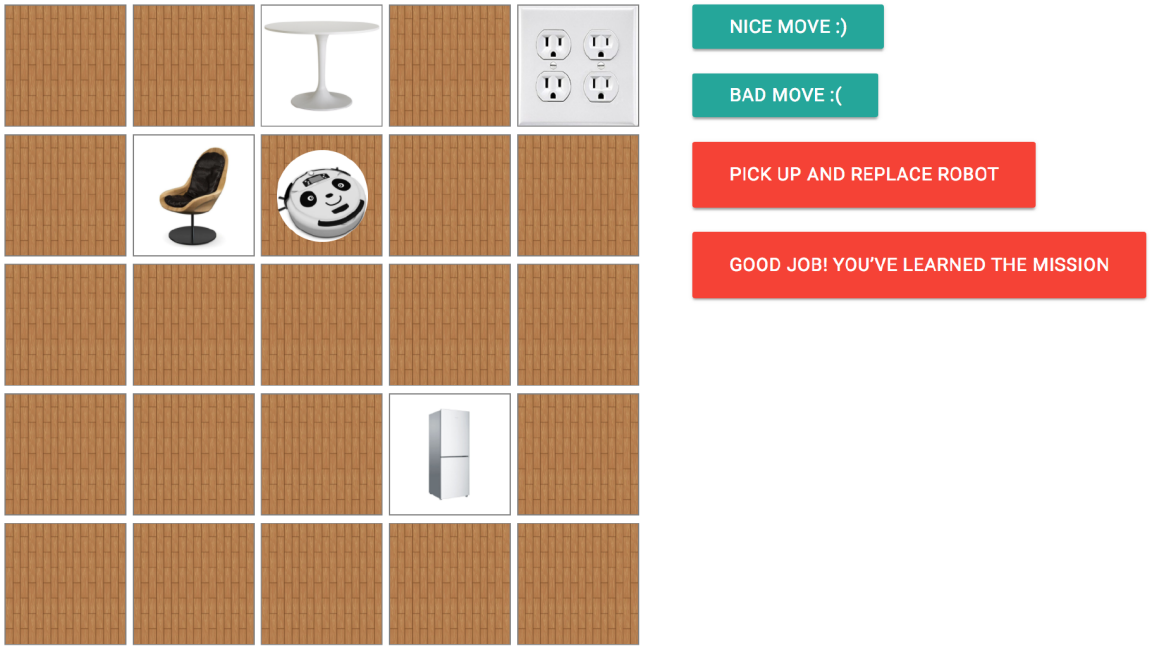


Figure 7.3: The GridWorld domain displayed to participants who trained an agent in our experiments. The small, round, robot is meant to indicate the agent. The buttons on the right side of the screen are used by the participant to interact with the agent and control the learning procedure.

7.4.1 Gridworld Domain

For our work, we used a GridWorld domain to represent a household robot in a home environment, specifically, a kitchen. A GridWorld is a state space comprised of an $n \times m$ grid of cells. Each cell represents a state of the world and can have its own properties. The agent takes actions that allow it to traverse the grid and interact with the world.

In this work, we have five distinct types of cells; a table, a chair, a fridge, a charger, and an empty floor. The agent is given four directional actions that allow it to move about the GridWorld. The empty floor space, unsurprisingly, represents a real-world robot traversing a part of a room. The other four spaces are meant to represent the agent interacting with their respective items. For example, the agent moving to a charger space could be representative of the agent returning to the charger and charging, while the fridge might be used for the agent fetching an item from inside. These cells are purely symbolic and used to ground the participants' understanding of the situation as our work focuses on the higher level goal of teaching an agent to understand the tasks defined by GLTL formulae.

7.4.2 Task-Learning Algorithm

At a high level, the algorithm we use for teaching in this environment is very similar to the IMPACT algorithms for classification. It uses composition and iteration to teach increasingly complex tasks to

	Known Expression		Derived Expression
	Atemporal transformations (X and Y are not necessarily temporal expressions)		
1.	X		$\Diamond X$
2.	X		$\Box X$
3.	X,	Y	$X \,\mathcal{U}\, Y$
	Temporal transformations (X, Y, are temporal expressions, x and y are not necessarily temporal expressions)		
4.	$X = \Diamond x$		$\Diamond\neg x$
5*a*.	$X = \Diamond x$,	$Y = \Diamond y$	$\Diamond(x \vee y)$
5*b*.			$\Diamond(x \wedge y)$
6.	$X = \Box x$		$\Box\neg x$
7*a*.	$X = \Box x$,	$Y = \Box y$	$\Box(x \vee y)$
7*b*.			$\Box(x \wedge y)$
8.	$X = \Box x$,	$Y = \Diamond y$	$x \,\mathcal{U}\, y$
9.	X		$\neg X$
10.	X,	Y	$X \wedge Y$
11.	X,	Y	$X \vee Y$
12.	X		$\Box\neg X$
13.	$X = \Diamond x$,	Y	$\Diamond(x \wedge Y)$
	Specialized transformations (X, Y, Z, x, y need not be temporal)		
14.	Z,	$X \,\mathcal{U}\, Y$	$X \,\mathcal{U}\, (Y \wedge Z)$
15.	$X = \Diamond x$		$\Diamond\Box x$
16.	$X = \Box x$		$\Box\Diamond x$
17.	$\Diamond(X \wedge \Diamond y)$		$\Diamond(X \wedge \Diamond\Box y)$

Table 7.1: Templates used in constructing temporal formulas.

the agent. To learn a mission, the agent first begins with learning some temporal transformation on the basic propositions of the space. For our work, the propositions were *table*, *chair*, *fridge*, *charger*. Each is set to true when the agent is on a cell occupied by the respective objects and false otherwise. The agent will construct a set of possible tasks using its current set of tasks (at the beginning, this collection is just the set of propositions) and applying transformations from Table 7.1. Based on feedback given by the trainer, the agent will eliminate inconsistent expressions until learning stops. When the round ends, the agent stores this task along with the other tasks and the entire process iterates until the mission has been deemed as learned by the teacher.

We will now propose formally Algorithm 7.4, the iterative algorithm that learns missions effectively and efficiently over a series of rounds, $i = 0, 1, \ldots, k$. Before the start of the first round, the agent is given K_0, a set of logical propositions in the domain. The set of initial hypotheses H_0 is generated by applying the transformations τ to K_0. After the training for round i, the agent identifies a set $L_i \subseteq H_i$ of learned tasks and sets $K_{i+1} = K_i \cup L_i$. In each subsequent round $i + 1$, the agent takes the set K_{i+1} and templates τ and generates $H_{i+1} = \tau(K_{i+1})$ from them to form new hypotheses for learning. This process continues until the trainer successfully conveys the mission Φ^*.

At round i, the trainer guides the agent to learn Φ_i by giving positive feedback when the agent's actions align with Φ_i and negative feedback otherwise. During training (LearnTask), the agent takes actions that elicit the most discriminative feedback possible from the trainer. Specifically, the agent takes an action such that it can rule out as close as possible to 50% of the policies currently under consideration. Given perfect feedback, whenever the agent sees feedback inconsistent with a given hypothesis, it can eliminate that hypothesis from consideration for the rest of the round. To be robust to occasional trainer errors, however, we instead have the agent allocate "strikes" to hypotheses that disagree with the feedback. When a round is ended, the agent is left with L, the set of hypotheses that had the fewest strikes against them. In our experiments, we direct the trainer to continue training if more than one hypothesis is in L. However, after 10 restarts, the algorithm simply returns two hypotheses selected at random from L.

7.4.3 Analysis

The algorithm we present is relatively natural, given the typical design of IMPACT-style algorithms presented so far. We learn to compose increasingly complex tasks all the while guided by a benevolent teacher who is attempting to iteratively teach understandable subtasks at each round. We prove now that this type of feedback can be used to learn the correct expression from a space of possible expressions.

Theorem 29 *Consider a set of tasks X, where every task $x \in X$ has a formula of length at most d and can be distinguished from the rest of the tasks in X using a trajectory of length m. Given that $|X|$ is polynomial in d and provided with evaluative feedback with at most n errors, a learning agent can successfully identify the target task with a number of interactions that is polynomial in m, n, and d.*

Proof Consider the following naïve, but sufficient, algorithm. For each hypothesis, the agent will conduct a test on that hypothesis by making decisions based on the policy induced by the hypothesis and observing feedback. During this test, whenever the trainer gives the agent negative feedback, the agent counts a strike against that hypothesis. Strikes are the result of either the hypothesis under testing being an incorrect hypothesis (causing it to make a bad decision) or the result of a trainer error in feedback.

For a hypothesis to be incorrect, the policy induced by that hypothesis must take at least one action that receives negative feedback. For the agent to be certain that the test hypothesis is not the target hypothesis, it must observe at least $n + 1$ strikes. After accumulating $n + 1$ strikes, only n strikes can be attributed to errors in the feedback signal, and at least one strike is a legitimate strike. So, the test hypothesis must be incorrect.

To be sure the agent has the opportunity to observe the $n + 1$ requisite strikes to disqualify the test hypothesis, the agent must run at most $2n + 1$ tests. The first n tests might not reveal the incorrect behavior because each of the negative feedback signals could be erroneous actions reported as positive. With $2n+1$ tests, the agent is sure to see at least $n+1$ strikes for an incorrect hypothesis.

Figure 7.4: Agent learning algorithm

Input: basic propositions K_0, templates τ
Initialize $H_0 \leftarrow \tau(K_0)$, $i \leftarrow 0$
while trainer has not finished mission **do**
    $L_i \leftarrow$ LearnTask(H_t)
    $K_{i+1} \leftarrow K_i \cup L_i$
    $H_{i+1} \leftarrow \tau(K_{i+1})$
    $i \leftarrow i + 1$
end while
return L_{i-1} as learned mission

function LearnTask(X)
    $t \leftarrow 0$
    restarts $\leftarrow 0$
    $L \leftarrow X$
    for $\phi \in X$ **do**
        $r_\phi \leftarrow 0$, initialize formula strike counter
    end for
    choose starting state s_t
    while task has not ended **do**
        observe current state s_t
        for all $a \in A$ **do**
            $c_a = \#$ formula in L with a optimal in s_t
        end for
        execute $a_t = \text{argmin}_a \, |c_a - |L|/2|$
        if trainer gives feedback f_t **then**
            for all $\phi \in X$ **do**
                if $f_t \neq f_{\phi,t}$ **then**
                    $r_\phi \leftarrow r_\phi + 1$
                end if
                $L = \{\phi \in X | r_\phi = \min_{\phi \in X} r_j\}$
            end for
        else if trainer attempts to end task **then**
            if $|L| = 1$ or restarts ≥ 10 **then**
                break
            else
                restarts $\leftarrow$ restarts $+ 1$
                choose starting state s_{t+1}
            end if
        end if
        $t \leftarrow t + 1$
    end while
    return L (at most 2, selected at random)
end function

By testing each of the $|X|$ hypotheses, the agent can be guaranteed to find the single correct hypothesis since it will be the only test with at most n strikes. Since each test requires $2n+1$ runs of the length m trajectory, each hypothesis' test can take at most $poly(m,n)$ interactions. We know that $|X| = poly(d)$, so the overall time to complete the algorithm and test all hypotheses is $poly(d,m,n)$. ■

Theorem 29 shows us that, if we know a trainer will make only at most n mistakes, we can ensure that we find the single correct hypothesis. It would be difficult, if even possible, to know the maximum number of mistakes a particular human trainer might make, so this assumption is a strong one. We can calculate a bound on the error rate of the trainer that the naïve algorithm in the proof will tolerate. Since we can err up to n times safely on each of the tests with $(n+1)m$ possible feedback signals, our acceptable error rate ρ is bounded by $\rho = \frac{n}{(2n+1)m} < \frac{1}{2m}$. Further, Algorithm 7.4 follows the same logic as the algorithm described in the above proof, except it more efficiently tests hypotheses by using feedback to gain information about multiple hypotheses at once. Requiring fewer feedback signals allows for the algorithm to be tolerant of higher rates of trainer error, in practice.

7.4.4 Building Missions from Tasks

We've shown that, given a space of tasks, our learning agent is able to correctly identify the target task with some tolerence for human error in the feedback. However, we still need to show that any arbitrarily complex LTL formula can be created using this process of repeatedly applying the transformations from Table 7.1.

To build missions in this way, we first introduce a data structure to help understand exactly how these temporal formulas work. Similar to standard Boolean logic, LTL formulas can be represented as syntax trees with nodes for the additional LTL operators. We partition the space of all LTL formulas into two classes, *temporal* and *atemporal*. We define a temporal LTL formula to be one in which, on every root-to-leaf path in the tree representation, there exists at least one temporal operator ($\Diamond$, $\Box$, $\mathcal{U}$). Conversely, an atemporal LTL formula is one in which there exists at least one root-to-node path in the syntax tree that does not contain a temporal operator. Any task that is defined by a temporal formula is referred to as a temporal task and any task defined by an atemporal formula is referred to as an atemporal task. Figure 7.1 lists the templates we use in our algorithm. The templates were chosen by inferring what kinds of transformations participants seemed to be expecting in the context of a pilot study, but they were then significantly modified to both ensure all temporal tasks could be constructed and to strike a balance between coverage and complexity.

The significance of temporal tasks is that they cannot be shown to be unsatisfied without at least a single temporal step, meaning that a trainer will have an opportunity to provide feedback. Therefore, to show that a mission can be trained by our multi-round curricular algorithm, it must be a temporal task *and* it must be able to be constructed via the templates we define where every

task en route to the mission is formulated as a temporal task.

Lemma 30 *Any temporal formula can be built up via application of the transformations from Table 7.1 starting from basic tasks that include simple temporal formulas of the domain's propositions. All of the intermediate formulas in the construction are themselves temporal.*

Remark 31 *The set of* $\{\wedge, \neg, \mathcal{U}, \Box, \Diamond\}$ *is functionally complete over the set we define as temporal formulas and are included in our basic templates via Templates* 1–3, 9–11. *So, to prove Lemma 30, we need to show that we can construct any given temporal tree via these transformations.*

Proof We can prove the claim using induction on the depth of the temporal formula tree. Since the basic templates cover all possible operators on propositions $\{\Diamond, \Box, \mathcal{U}\}$, the claim holds when the depth is 1. (We do not need to even consider $\wedge$, $\vee$, and $\neg$ since these operators used on propositions would create an atemporal function with a tree of depth 1.)

For induction, we assume that the claim holds for all depths $[1, k]$ for some arbitrary k. We need to show that the claim still holds for $k + 1$. We investigate two cases for a temporal formula tree of depth $k + 1$.

Case 1 *The root of the tree is a temporal operator.*

Since the root is temporal and, necessarily, on the path to every leaf, the resulting tree will always be temporal. The subtrees below the root are either temporal or atemporal and are of depth $\leq k$. Temporal subtrees can be constructed via the induction hypothesis on subtrees of depth k. Atemporal subtrees must be handled more carefully. We first build the subsubtrees by attaching a temporal operator to them. We use $\Box$ or $\Diamond$ to match the root of the overall tree. If the root of the tree has the form $a \,\mathcal{U}\, b$, we append $\Box$ if we are constructing subtrees of a or $\Diamond$ if we are constructing subtrees of b. These altered subsubtrees are of depth $k-1+1 = k$ after adding on the extra temporal operator as described. Therefore, they are temporal trees of depth k, and can now be constructed by the induction hypothesis. Once these altered subsubtrees are constructed, we can combine them to make our subtree by using them in Templates 4–8 or 13. After we have constructed the subtrees as stated above, we can construct the overall tree via Templates 1–3.

Case 2 *The root of the tree is an atemporal operator.*

In this case, we know that both subtrees must be temporal. If even one subtree were atemporal, it would include a path to a leaf that includes no temporal operator and our original claim that the function is temporal is contradicted. Since both of the subtrees are temporal, all leaves in the overall tree have a path from the root that traverses a temporal operator. Further, since the subtrees are of depth $k + 1 - 1 = k$, they can be constructed via the induction hypothesis. The overall tree can then be constructed via Templates 9–12. ■

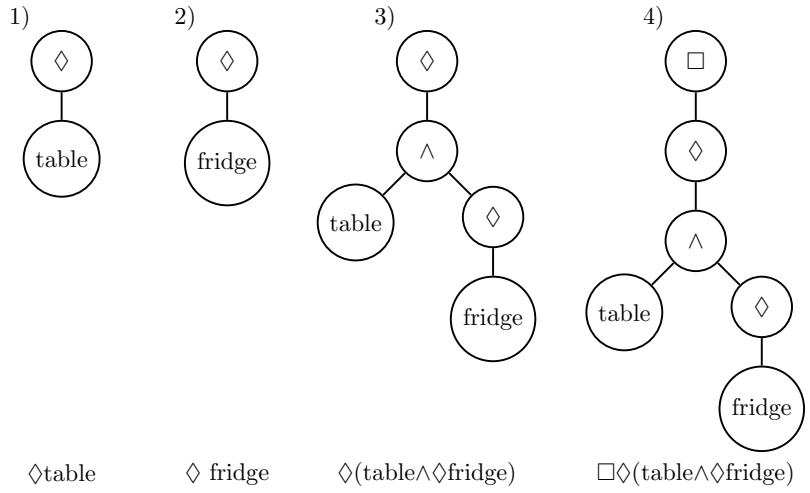


Figure 7.5: The training procedure for task $\Box\Diamond(\text{table} \wedge \Diamond\text{fridge})$. Here, table and fridge nodes stand for the propositions that exist in the environment denoting that the agent is on the space marked with the respective item. Steps 1) and 2) use Template 1 from Figure 7.1. Step 3) uses Template 13. The final step, 4), uses Template 2.

A less rigorous, but more intuitive description of the above proof is the following. Whenever we are training at a node that is an atemporal operator, we know, by definition, that both children must be temporal. So, we could have trained (bottom up) to this point without any issues via induction. When the current node is temporal, it is possible that one or both of its subtrees is atemporal. In that case, the procedure described in Case 1 shows us that we can propagate the temporal operator down to the subtrees. If we train according to this new tree structure, the atemporal subtrees become temporal and therefore trainable. This training strategy is illustrated in Figure 7.5.

7.5 Robustness for Human Teachers

So far, we have shown that our procedure is capable of building arbitrarily complex LTL expressions. Additionally, we showed that our learner is capable of learning such expressions given feedback of a reasonable quality. Assuming we have a person who is capable of creating a working decomposition and giving reasonable feedback, we are now prepared to show that our we can teach an agent to act in accordance with LTL-defined missions.

Theorem 32 *If a trainer can decompose any mission into tasks that satisfy Lemma 30 and they can provide evaluative feedback with low error rate, they can train the algorithm to learn any temporal LTL task in time polynomial in the size of the formula with high probability.*

Proof Successful training can be accomplished by using Algorithm 7.4 with rounds as described by Lemma 30. Since the transformations given by the procedure in the proof of Lemma 30 do not increase the size of the formula tree by more than a factor of 2, the bounds on the description of

the formula remains the same. Within each iteration of the algorithm, we increase the hypothesis set by $|\tau||L|$. Since we bound $|L|$ by a constant (we use 2) and $|\tau|$ is also constant, the overall size of the hypothesis space increases only polynomially over the course of the algorithm. As a result, training each round (even later ones) remains feasible and bounded, both in sample and computational complexity. Applying Theorem 29 to this approach to training proves our claim. ■

We have shown that our learning algorithm can be efficiently taught any temporal formula mission by an idealized trainer who does not make too many errors in feedback and who can decompose the mission formula appropriately into a curricular sequence of tasks.

Chapter 8

Conclusion

The work in this dissertation has helped to advance the understanding of machine teaching. In doing so, we are able to gain a deeper knowledge of what makes certain learning problems hard, and which tools we can use to tackle these problems with some help from a teacher. The closest related work to ours is that of Rivest and Sloan, so we will first present a comparison of their work and our model before outlining our specific contributions. Finally, we will discuss potential future research questions that are inspired by this work. We explore ideas that include both a new application of the IMPACT model as well as a novel problem that we can consider based on our model.

8.1 Discussion

In this dissertation, we were able to tackle hard learning problems by creating and developing a new learning model. Using our IMPACT model in classification and applying the same principles in reinforcement-learning settings, we were able to teach concepts that were otherwise impossible to learn. We began to discover the various types of problems for which IMPACT could provide the advantage we needed to make learning possible. Further, we were able to actually test the feasibility of real world systems that used IMPACT or IMPACT-inspired systems with actual non-expert end users. We saw success in doing this both in the supervised-learning and reinforcement-learning settings.

More concretely, we were able to create a novel learning model, IMPACT. We were able to show that, for a range of learning problems, the IMPACT model is provably effective and efficient. Understanding how to break down these problems and fit them into the IMPACT model helped to shed light on exactly what made them hard problems in the first place. While IMPACT may make some strong assumptions about the availability and competency of a teacher, we showed empirically that there are still relevant and useful applications.

At a higher level, creating and studying the IMPACT model served an important role in shaping the way we might approach new learning problems. IMPACT forces us to decompose difficult ideas into smaller more tractable ones. It helps us to focus on growing the agent's understanding and

capabilities over a period of time rather than expecting too much at once. While it surely is not the only approach that uses ideas like compositionality and iteration as tools to accomplish something difficult, it does highlight these ideas in a way that forces us to reconsider them. The IMPACT framework cannot work without careful consideration of how to decompose and allow the learner to recompose complex ideas iteratively over time. While these tools might not be the key that unlocks the solution for every problem, studying IMPACT gives us an example for how to use them to accomplish more difficult tasks using less data or computation.

8.1.1 Comparison to Rivest and Sloan

The work that was most closely related to our work from Chapter 2 was that of Rivest and Sloan. They used a similar setup to ours to tackle a set of problems that are provably unlearnable in the PAC setting. We showed that our model successfully handles these problems, as well.

Despite the similarities, our work is distinct from theirs in multiple regards.Primarily, the goal of our research was to devise a new learning model, suited not just specifically for the set of problems presented in Rivest and Sloan's work, but for an entire space of problems that are too difficult to learn. As such, we were concerned with modeling our new framework after the very common PAC model. We devised our definitions and theoretical bounds based around those of the PAC setting. In contrast, Rivest and Sloan departed from the standard PAC setting and measured their performance differently. They created an algorithm that is evaluated on its reliablity and usefulness, rather than the same accuracy metric used in PAC. We accomplish comparable results to their work, but do so with a teacher that is significantly less powerful. This restriction is necessary for our model to be collusion-free, as we discussed in Section 3.3.3.

8.1.2 Contributions

In this work, we presented the IMPACT teaching model and discussed our related projects. In Chapter 3, we have successfully invented and formalised that novel teaching model. Our model was shown to be able to teach concept classes that are provably hard to learn in the standard PAC setting. We presented a general algorithm and accompanying theory that can be used to show teachability of any problems that meet the criteria we provide. We have shown that, despite having some strong assumptions about the knowledge and competency of the teacher, our model does have relevant applications. For example, we provide another learning problem that IMPACT can provably efficiently teach in Chapter 5. In Chapter 6, we discussed how to bring IMPACT closer to real-world applications for supervised learning problems by introducing a non-expert human teacher into the model. Chapter 7 took inspiration from that work and expanded this idea into the reinforcement-learning setting. Here, we showed that people were able to successfully train our agents to learn behavior that satisfied arbitrarily complex LTL task specifications.

Our model, which in its own right furthers the understanding of the field of machine learning, also encourages thinking about other problems in potentially new ways. How can we use the tools

of compositionality or iterative data augmentation from the IMPACT setting to accomplish more work with less data or computation for other problems? We attempt to provide potential answers for this in the following section.

8.2 Future Work

The questions explored in our work potentially open the door to new avenues of research. In this section, we propose potential new research ideas based on the work in this dissertation. These ideas are either inspired by further refining the IMPACT framework we created or finding new applications of IMPACT or its ideas that may benefit the progress of solving other problems in machine learning.

8.2.1 Learning to Teach

One natural question is, "If IMPACT is teaching agents hard learning problems, can an agent learn or be taught how to be an IMPACT teacher for other agents?" On the surface this question seems to be a bit absurd. If an agent was able to learn how to teach other agents, then surely it could just learn to teach itself. In that case, the original concept was actually learnable and did not require the use of IMPACT in the first place. However, the problem of learning to teach a concept and learning a concept are separate in some important ways. The flow of learning a concept involves using examples to discover the concept. The problem of learning to be an IMPACT teacher involves using concepts to discover a filtering rule. This problem is new and could prove to be potentially more or less hard than the original.

8.2.2 Stable Teaching

Alternatively, we can look at applying the same tools we employ in the IMPACT setting to tackle other problems. One such potential problem is the *stable learning problem*. The stable learning problem arises when you attempt to have an agent learn a concept from examples. The attributes of this domain can be partitioned into stable and unstable attributes. The stable attributes are those that are used in calculating the example's true label. In contrast, the unstable attributes are attributes that may exhibit some strong correlations to the labels in the training distribution, but are not truly "causal" of the label. Typically, unstable attributes do not pose any significant issue for a problem in the PAC setting, because the training and testing distribution are assumed to be the same. As such, any correlations that exist in the training sample will also be present in the testing sample and the agent will suffer no loss of accuracy by exploiting those correlations. However, in the context of the stable learning problem, the testing and training distributions may differ. That means the correlations that exist in the training data need not be present in the testing data. A promising approach to this problem is to reweight the training sample intelligently so as to eliminate the spurious correlations [Kuang et al., 2018].

IMPACT too, in a sense, reweights the training samples of the learning problem by filtering. We

have shown that this reweighting does not affect learnability when the original training and testing distributions are identical. It would be interesting to study whether or not the same type of filtering of examples can be done by a teacher and, if so, if it is sufficient to learn stably.

8.2.3 Learning with Less Data

In our work thus far, we have explored the ways in which the IMPACT model can help reduce the sample and computational complexity of hard learning problems. The intention for the majority of this work had been to tackle problems that were unlearnable without the help of a teacher. However, even problems that are theoretically learnable in the PAC setting may stand to benefit from the improvements in complexity IMPACT provides. Even at a time where data is abundant, clean, usable, labeled data may be expensive to attain or maintain for some settings. In such cases, it would be beneficial to use the tools from our IMPACT model as a means to further reduce the complexity of already learnable classes. We have already shown that the IMPACT model accomplishes this goal for unlearnable classes. It would be interesting to see just how our model can be used to reduce data requirements for less hard problems.

Bibliography

Anselm Blumer, Andrzej Ehrenfeucht, David Haussler, and Manfred K Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965, 1989.

Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, California, 1984.

Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Thirteenth International Conference on Machine Learning*, pages 148–156, San Francisco, 1996. Morgan Kaufmann.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression : A statistical view of boosting. *Annals of statistics*, 28(2):337–407, 2000. ISSN 0090-5364.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Geoffrey Holmes, Bernhard Pfahringer, Richard Kirkby, Eibe Frank, and Mark Hall. Multiclass alternating decision trees. In *ECML*, pages 161–172. Springer, 2001.

Michael Kearns and Leslie Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

Ron Kohavi. The power of decision tables. In *8th European Conference on Machine Learning*, pages 174–189. Springer, 1995.

Kun Kuang, Ruoxuan Xiong, Peng Cui, Susan Athey, and Bo Li. Stable prediction across unknown environments, 2018.

Michael L. Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-independent task specifications via GLTL. arXiv preprint arXiv:1704.04341, 2017.

T M Mitchell. *Machine Learning*. McGraw-Hill international editions - computer science series. McGraw-Hill Education, 1997.

Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012.

Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

Ronald L Rivest and Robert Sloan. A formal model of hierarchical concept-learning. *Information and Computation*, 114(1):88–114, 1994.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Haijian Shi. Best-first decision tree learning. Master's thesis, University of Waikato, Hamilton, NZ, 2007. COMP594.

Vladimir Vapnik and Rauf Izmailov. Learning using privileged information: similarity control and knowledge transfer. *Journal of Machine Learning Research*, 16:2023–2049, 2015.

Lev Semenovich Vygotsky. *Mind in society: The development of higher psychological processes*. Harvard university press, 1980.

Sandra Zilles, Steffen Lange, Robert Holte, and Martin Zinkevich. Models of cooperative teaching and learning. *Journal of Machine Learning Research*, 12(Feb):349–384, 2011.