Optimally Teaching Learning Agents under Various Constraints

By

Shubham Kumar Bharti

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2025

Date of final oral examination: December 1st, 2025

The dissertation is approved by the following members of the Final Oral Committee:

Xiaojin Zhu (Advisor), Professor, Computer Sciences
Josiah Hanna, Assistant Professor, Computer Sciences
Stephen Wright, Professor, Computer Sciences
Qiaomin Xie, Assistant Professor, Industrial and Systems Engineering

To my parents, Vijay Kumar Bharti and Soni Bharti.

Acknowledgments

I would like to express my sincere gratitude to my advisor, Prof. Xiaojin (Jerry) Zhu, for his invaluable guidance and support throughout my PhD journey. His intellectual curiosity, profound technical insights and the rigor with which he approaches problems have been a guiding light during my doctoral career. He has been a phenomenal mentor, helping me build strong fundamentals early on, while giving me the freedom to identify and pursue the research directions I found most rewarding in the later stages of my PhD.

My sincere thanks go to my committee members, Prof. Stephen (Steve) Wright, Prof. Qiaomin Xie, and Prof. Josiah Hannah, for their insightful feedback from the qualifying stages through to the dissertation. A special acknowledgement is due to Steve, who became both a mentor and a collaborator. I am deeply appreciative of his generosity with his time, despite his demanding schedule as department chair. His dedication to research, while maintaining a vibrant outlook on life - exemplified by his passion for sailing and fitness - remains a source of inspiration for me.

I extend my thanks to my collaborator, Prof. Adish Singla, whose mentorship shaped my understanding of various aspects of machine teaching. I am also grateful for his hospitality at MPI-SWS, which offered a unique window into the research and cultural landscape in Germany. I would also like to thank Prof. Kangwook Lee for the various research discussions and collaborations I had with him on multiple occasions. His insights into problems and his ability to connect various topics have greatly influenced my own thinking. I would like to thank my labmates and close collaborators Xuezhou Zhang, Yuzhe Ma, Yiding Chen, Jeremy McMahan, Deep Patel, Jihyun Rho, Ziqian Lin with whom I had the pleasure of working on some interesting problems and learning from them throughout these years.

I would also like to thank my internship mentors at Amazon, Mukul Prasad and Vidya Sivakumar for providing an awesome learning experience during my internship and for broadening my perspective on LLM agents in industrial research, to Gaurush Hiranandani,

Suyash Gupta and Vassilis Ioannidis for helping me gain valuable experience in RL and recommendation systems, and develop strong leadership skills early on in my career.

This journey would have been incomplete without the constant support of my friends. I am very grateful to Vishnu Lokhande for being a fantastic running and adventure buddy, and for the long hours spent debating everything from research to life; to Prince Kumar for the late-night philosophical discussions that kept me grounded during the pandemic; to Sourav Pal for being a great friend and sharing the highs and lows of this journey and to Ashwin Maran - who was also an amazing roommate for four years - for the time spent brainstorming challenging TCS problems. I also cherished the friendship of Mrugank Bhatt, for being an excellent ski partner during the cold Madison winters; Jeremy Goddard, for being an amazing travel companion who was always generous with rides and invites to fun events; and Tejas Patel, for the long drives we made in search of good food. I also want to thank my friends in the Bay Area, Daniel Paul-Pena, Utkarsh Gupta, Amrit Singhal, and Diwanshu Jain. Thank you for being incredible adventure buddies and for turning demanding internship summers into memories filled with fun and exploration.

Finally, none of this would have been possible without the constant support of my parents - my mom Soni Bharti, my dad Vijay Kumar Bharti who always stood by me in all my decisions and have made innumerable sacrifices in their lives to support me throughout my academic journey. I am also thankful to my siblings, Satyam Kumar Bharti and Richa Gyana, for being my constant cheerleaders and the most amazing siblings I could ever ask for. Last but not least, I would also like to offer my deepest gratitude to my grandfather, Sachidanand Modi, for having planted the seed of pursuing the highest pursuits in life early on in my childhood.

Contents

List of Tables

List of Figures

Abstract

Machine teaching studies the fundamental problem of how a knowledgeable teacher can design an optimal demonstration to efficiently teach a target concept to learning agents. Prior works have established the *Teaching Dimension* as a measure of sample complexity for various types of learners. However, these works predominantly assume idealized conditions: an omnipotent teacher capable of constructing arbitrary datasets, single-stage learners, and an unlimited resource budget. In contrast, teachers in the real world are often subject to strict resource and transition constraints of the environment. This thesis bridges the gap by developing machine teaching algorithms for various real-world settings, specifically addressing: teaching a family of linear learners, teaching under budget constraints, and teaching under environment transition constraints.

First, we tackle the challenge of teaching a family of consistent linear learners (e.g., Support Vector Machines, Logistic Regression) using a single dataset. We show that this problem reduces to teaching the most unbiased learner in the family, a Linear Version Space learner, and present a provable approximation algorithm based on the extreme ray structure of the target cone to solve the problem efficiently.

Second, we study teaching behavior cloning learners where the teacher cannot create an arbitrary dataset and is subject to transition constraints of a Markov Decision Process (MDP) environment. We formulate this problem as *Stochastic Set Cover Problem* (SSCP) to find a teaching policy that minimizes the coverage time of a universe set. We solve this problem by reducing it to a well-studied Stochastic Shortest Path Problem, providing value/policy iteration algorithms with convergence guarantees.

Third, we extend the notion of optimal teaching to the Reinforcement Learning (RL) setting. We formulate and study *Minimum Expected Teaching Length (METaL)* for Q-learning agents under four levels of control by the teacher. We propose efficient algorithms to solve the problem exactly in the first two levels and achieve a tight matching bound on the

worst-case teaching complexity when the teacher is further subject to transition constraints of the MDP environment in the latter two levels.

Finally, we introduce the framework of "Nurture-then-Nature" to study budget-constrained teaching of supervised learning agents. When the teaching budget is insufficient to fully teach the target concept, the teacher aims to create a dataset that optimizes the learner's efficiency in the subsequent nature learning phase. We study this problem in various settings and propose exact and approximation algorithms to solve it efficiently.

Introduction

Machine teaching studies a fundamental problem: *how should a knowledgeable teacher design an optimal dataset or experience to teach a target concept to a learning agent optimally?* The pioneering work in this direction was done by Goldman et. al. [29] who formalized the idea of optimal teaching through the *Teaching Dimension* (TD), the size of the smallest dataset that is required to uniquely teach the target concept to a given learner. They studied teaching of classical *version space* learners, in which a teacher provides examples that shrink the set of hypotheses consistent with the dataset until only the target remains. More recent works have extended this notion to more complex supervised learners and have studied their teaching dimension. In most of these settings, success often comes from exploiting *learner's algorithmic bias* to teach them efficiently. For instance, the max-margin bias of Support Vector Machines (SVM) [54] allows them to be taught with just a few data points.

Despite clear insights, much of the prior theory adopts idealized conditions: the teacher can construct *arbitrary* dataset, the learner is typically a *single supervised algorithm*, the environment has no *transition dynamics* that the teacher has to obey, and the teacher's *budget* is large enough to meet TD exactly. These assumptions are frequently violated in many practical scenarios, making them less appealing. For example, when teaching a student in real world sequential decision-making environment, the teacher is subject to the state transition constraints of the environment and thus it cannot generate arbitrary dataset. Or in curriculum learning setting, the teacher can only teach a small fixed number of lessons to the students before they graduate and so teaching the target concept completely to them may not be possible. This thesis develops machine teaching principles and algorithms to capture these real world scenarios specifically, (i) teaching families of linear learners in a classroom setting, (ii) optimal teaching under transition constraints of the environment, (iii) teaching sequential decision making learners whose actions shape their experience they receive, and finally (iv) teaching under strict budget constraints on the teacher.

Novel Challenges in Realistic Teaching Settings

We develop the traditional notion of teaching in two major orthogonal dimensions based on the type of agent/learner being taught and the level of control given to the teacher:

From Single Stage Supervised Agents to Multi Stage RL Agents

The machine learning literature is replete with a wide spectrum of learners, from supervised classifiers like SVMs, logistic regression, neural networks etc., to sequential decision making agents in *reinforcement learning* (RL) like Q-learning, policy-gradient methods etc. For each such learner, one may ask how a knowledgeable teacher, with some control over the learner's experience, can teach a target concept or policy to them in an optimal way.

Most existing works in machine teaching focus on *individual supervised learners*, such as finite-hypothesis version space learners, SVMs, logistic regression, perceptrons, etc., and typically require the teacher to teach a target concept to the individual learners using a minimum-sized dataset. We address two gaps in this direction. First, we extend optimal teaching from teaching individual supervised linear learners (e.g., SVM, perceptron), to teaching an entire family of consistent linear learners. Importantly, instead of solving optimal teaching problem for each of the learner the family, the teacher has to construct a dataset that can succeed in teaching all the learners in the family simultaneously.

Second, we move beyond single-stage supervised agents to multi-stage RL agents. We develop a teaching framework for RL agents like Q-learning, which interact with a Markov Decision Process(MDP) environment over a long time horizon to learn a policy that maximizes their cumulative return. In this setting, the teacher's task shifts from constructing a static dataset to designing a sequence of experiences that steers the learner's learning trajectory toward a desired policy. Due to the online nature of the environment, the goal of the teacher shifts from finding a minimal-size teaching dataset to finding a teaching policy that can teach the target policy to the learner in *minimum expected time*.

From Omnipotent to Budget and Transition Constrained Teaching:

A second dimension of realism from the *level of control* the teacher has over the learning experience of the learner. In classical supervised settings, the teacher is often modeled as omnipotent: it can construct an arbitrary dataset in the dataset space tailored to optimally teach the target concept to the learner. In practice, however, teachers are frequently constrained in both *quantity* and *form* of experience they can provide to the learners.

The most natural form of constraints comes from the finiteness of time and resources; for example, in curriculum learning settings, an instructor may only have a fixed number (budget) of lectures that they can teach to the student before they graduate. The number

Agent ↓ \ Constraints →	No constraints	Budget constraints	MDP constraints
Supervised Learning	① Chapter 1	② Chapter 4	③ Chapter 2
Reinforcement Learning	④ Chapter 3	⑤ Future Work	⑥ Chapter 3

Overview of teaching settings studied in this thesis, arranged along two key dimensions.

of lessons may not be sufficient to fully teach the target concept to the students. However, the students themselves may keep learning about the target concept by interacting with the environment after graduation. We study this problem of budget constrained teaching called 'Nurture-then-Nature' teaching and consider two different teaching objectives for the teacher: 1.) minimize the risk of the learner at the end of the environment learning phase, or 2.) minimize the time taken to learn the target concept in the environment learning phase.

Moving to reinforcement learning further enriches the control spectrum: a teacher may influence one or more of the learner's state, its action choices, and the rewards it receives, but often only in restricted ways and subject to the MDP's transition dynamics. This leads to several interesting challenges and solutions from teachers perspective. We note that while budget constraints can be applied together with transition constraints, we defer this setting to a future work to address.

We study several regimes along this constraints axis, ranging from powerful constructive teachers that can teach arbitrary datasets, to teachers that can provide only a budgeted teaching dataset, down to highly constrained RL teachers that must obey the environment dynamics and may only shape learner trajectory through reward manipulation. We present an overview of the individual teaching problems investigated in this thesis, accompanied by an outline of the associated chapters. Notably, each of these work incorporate one or more of the characteristics outlined above.

(A) Teaching A Family of Linear Learners using a Single Dataset.

Consider a setting where a teacher has to teach a classroom of students. Each student has their own learning mechanism: one uses max-margin SVMs, another logistic ERM, a third a version space learner, and so on. However, all of them satisfy a consistency property, i.e., they eventually learn one or a subset of hypotheses that is consistent with the dataset. The instructor wants to create a *single* optimal dataset so that all students, despite differing learning mechanisms, learn the same target policy.

The teaching problem essentially boils down to teaching the most unbiased student in the class, the linear version space learner. We study optimal teaching of this learner and connect it to optimal teaching of the entire consistent linear family. For a consistent presentation, we adopt the perspective of a linear behavior cloning (BC) agent in RL setting, which is equivalent to linear version space (LVS) learner in a supervised learning setting.

Problem setting: We consider an environment with finite state space $\mathcal{S}$, finite action space $\mathcal{A}$ and a feature map $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ that induces a linear policy under a weight vector $w \in \mathbb{R}^d$ as follows,

$$\pi_w(s) = \arg\max_{a \in \mathcal{A}} w^\top \phi(s, a).$$

Given demonstrations $D \subseteq \mathcal{S} \times \mathcal{A}$, any *consistent* linear BC learner(e.g., linear SVM, linear logistic ERM, or linear version space learner) must return a policy or a subset of it agreeing with D. The hypothesis set consistent with dataset D form an *open polyhedral cone*,

$$\mathcal{V}(D) = \Big\{ w : \; w^\top \psi_{sab} > 0 \;\; \forall (s, a) \in D, \, \forall b \neq a \Big\}, \quad \text{where} \quad \psi_{sab} = \phi(s, a) - \phi(s, b).$$

The teacher can create an arbitrary dataset and is challenged with the task to choose a single dataset D so that $\mathcal{V}(D)$ contains only those parameters that induce the *unique* target policy π^* on *all* consistent learners $\mathfrak{C}$,

$$\begin{aligned} (\textbf{Teach-}\mathfrak{C}) \qquad & D^* \leftarrow \min_{D \subseteq \mathcal{S} \times \mathcal{A}} \quad |D| \\ & \text{s.t.} \quad \forall \mathcal{L} \in \mathfrak{C}, \quad \mathcal{L} \text{ learns } \pi^* \text{ uniquely.} \end{aligned} \tag{1}$$

Chapter Outline: **Chapter 1** formalizes a family of consistent linear behavior-cloning learners with features $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ and weights w, where demonstrations induce an open

polyhedral cone of feasible parameters. We show the equivalence between teaching the entire family and teaching the hardest-to-teach member in the family - the LVS learner. We show that teaching such a learner naively requires solving an impractical **infinite set-cover** problem in w-space. We then characterize optimal teaching in terms of extreme rays of the target cone and reduce the teaching problem to a set cover problem over the finite **extreme ray** space induced by the target policy π^*. We then prove that the smallest dataset that covers all the extreme rays ends up teaching the target policy to the LVS learner. We also prove **NP-hardness** of finding an exact optimal teaching set and provide an efficient and provable **approximation algorithm** that first computes the extreme rays using a sequence of linear programs and then solves a set cover over those extreme rays using a greedy mechanism to find the minimum teaching set. Finally, we present experimental results on toy and real-world environments like "Pick the Right Diamond"; and "Visual Block Programming" to validate the effectiveness of our algorithm and compare them to baselines.

(B) Teacher Constrained by Environment Dynamics.

In the previous setting, the teacher was allowed to create any dataset to teach the behavior cloning learners. However, in real-world settings, the teacher may not have the power/-control to create or put the learner in arbitrary states. For example, consider a teaching setting where a driving instructor wants to teach a learner how to drive. The teacher cannot place the car at each critical intersection to demonstrate the right-of-way rule. Instead, the instructor must *drive* to those intersections by following the dynamics of the environment (laws of the physical world, stochastic traffic, and diversions), and optimize for the time spent to reach critical spots required to teach the target concept to the learner.

Problem setting: We model this problem as optimal teaching of finite or linear BC learners under state transition constraints of the environment imposed on the teacher. The teacher has to act in the environment to efficiently collect a demonstration dataset that can be demonstrated to the BC learners. Due to the stochastic nature of the environment, some states may take much longer to reach than others and hence the size of dataset |D| no longer remains a relevant cost function; rather, we minimize the *expected time* required by the teacher to find a valid teaching dataset (one that eliminates all inconsistent policies

from the policy class) from stochastic trajectory roll-outs and is given as follows:

$$\textbf{Online-Teach}(\mathcal{S}, \mathcal{A}, P, \Pi, \pi^*): \\ \pi^\dagger(s) \leftarrow \min_{\pi} \mathbb{E}_{\tau \sim \pi} \left[\min\{t \geqslant 0 : \cup_{t \leqslant t'} W_{s_{t'}} = \Pi \backslash \pi^* | s_0 = s \right]$$

We refer the readers to the chapter 2 for a complete description. Finding a valid teaching dataset in both finite and linear settings requires covering a finite universe of elements using subsets induced by individual states, except now they have to be covered by following the transition dynamics of the environment. In essence, the goal of the teacher is to find an *optimal navigation policy* that covers a set of universe elements through subsets induced by individual states that are connected through stochastic MDP transitions.

Outline: In **Chapter 2**, we study this problem and cast optimal teaching as a **Stochastic Set Cover Problem (SSCP)** with a goal to **minimize expected cover time** objective. We transform this problem to a well-studied problem in sequential decision-making literature called **Stochastic Shortest-Path Problem (SSPP)** in a meta MDP. We characterize the expected and optimal cover times using Bellman equations. Under standard assumptions (known transitions, reachability), we present a **pseudo-polynomial algorithm** to solve SSCP via SSPP using value and policy iteration, thereby establishing the correctness and complexity guarantees. We discuss the hardness of this problem, specifically, the problem captures two NP-hard problems: 1.) the set cover problem and 2.) the traveling salesman problem making it a much harder NP-hard problem. We conjecture that this problem is inapproximable even in a deterministic transition setting.

(C) Teaching an Online RL Agent.

In this work, we shift our attention to fully sequential decision-making framework, wherein agents learn to maximize long-term rewards by interacting with stochastic environments. To appreciate this setting, consider a robot running an ε-greedy Q-learning on-device to learn to solve a robotic manipulation task. The robot interacts with the environment over multiple steps of time, and its behavior directly influences the future that it gets to experience. Unlike a supervised learner, the robot cannot be taught by a one-shot batch dataset; instead, it has to be taught through a sequence of experiences.

A powerful teacher can place the robot in a specific position, guide its action, and provide different rewards to guide the robot to the target policy much more efficiently than letting it learn from trial-and-error interaction with the environment. However, such a level of control may not always be possible in real-world scenarios, for example, in a more realistic scenario, the teacher cannot move the robot or directly manipulate its action, but it can still influence its behavior through reward. To accommodate these scenarios, we consider teachers with different levels of control on the agent's experience and study their corresponding optimal teaching complexity to "teach" a target policy $\pi^\dagger$ to the agent very *quickly*.

Problem setting: We study optimal teaching of Q-learning algorithm that learns using a Q-table Q_t, which is updated through experience tuples $e_t = (s_t, a_t, r_t, s_{t+1})$ obtained from environment or the teacher. The teacher's control ranges from being able to fully control (s, a, r) to a *reward-only* control regime where it has to obey environment state transition and actions taken by the agent. We measure teaching complexity via the *Minimum Expected Teaching Length* defined as follows:

$$\text{METaL}(\mathcal{M}, L, Q_0, \pi^\dagger) = \min_{\nu} \mathbb{E}_\nu[\min\{t : \pi_{Q_t} = \pi^\dagger\}],$$

and study the class optimal teaching dimension across different control regimes. We refer the readers to chapter 3 for a complete description. To design an effective teaching strategy, the teacher needs to reason about the *online update* mechanism of the learner and design interventions that shape the evolution of the learner's Q-value table over time towards the induced target policy.

Outline: In **Chapter 3**, we introduce a teaching protocol for ε-greedy Q-learning with initial table Q_0, defines the **Minimum Expected Teaching Length** METaL for instances and an RL **teaching dimension** (worst-case METaL) over instance families, and propose four control regimes. In the **unconstrained** setting: Level 1 with full control over (s, a, r), our teacher constructs a minimal teaching trajectory that achieves exact METaL. In Level 2 teaching with state-and-reward control, we design a teacher that constructs an optimal state/reward experience, thereby driving the Q-updates to the target with near-optimal time guarantees. In Level 3, we introduce the more challenging notion of control where the teacher can only create states supported by P, μ and propose a novel navigation-guided

teaching algorithm called **Nav-Teach** to teach the learner effectively. We provide a worst-case lower bound on METaL objective and show that Nav-Teach achieves a matching **tight upper bound**. Finally, in Level 4, we study a teacher with the strictest level of control, where it has to completely obey the environment transition dynamics and can only influence the behavior of the learner through **reward-based** control. We prove that the Nav-Teach algorithm also achieves a nearly tight bound on the **worst-case TDim** in this setting.

(D) Nurture–then–Nature Teaching under Budget Constraints.

In this chapter, we shift our attention to a budget-constrained teacher in a supervised learning setting. Consider a semester curriculum, where an instructor has at most B lectures to shape the understanding of the students/learners before they graduate and head to work in the real world, where they still keep learning from real-world data. Given a limited budget and the vastness of the syllabus, perfect mastery in class is unrealistic; so the goal shifts to optimally position students so that they can learn rapidly and effectively in the real world.

Problem setting: Consider a two-phase learning process called Nurture-then-Nature where in the first phase called the nurture (teaching) phase, the learner learns under the guidance of a budget constrained teacher that can provide a dataset D_T of size at most B, leaving the learner with a surviving version space $\mathcal{V}_1 = \mathcal{V}(D_T; \mathcal{H})$ at the end of teaching. After teaching, the learner moves to the second phase called the nature (environment learning) phase and keeps learning from the independent and identically distributed (i.i.d.) data $D_E \sim P$ received from the environment until a satisfactory mastery is achieved. When $B < TD$, mastery in the nurture phase is impossible; so we study two novel objectives: (i) minimizing risk at the end of the nature phase, given as,

$$D_T^* \leftarrow \underset{D_T : |D_T| \leqslant B}{\arg\min}\ \mathbb{E}_{D_E \sim P^n}\left[R_P(\mathcal{A}(D_T \cup D_E)\right],$$

or (ii) minimize expected *time-to-mastery* in the nature phase, given as,

$$D_T^* \leftarrow \min_{D_T : |D_T| \leqslant B} E_{D_E \sim P^\infty}[T(D_E, V(D_T \backslash \{h^*\}))].$$

The teacher must *shape* the version space $\mathcal{V}_1$ that survives at the end of nurture phase (e.g., its VC dimension, or the hardest hypothesis in it) to optimize downstream learning performance of the learner in the nature phase under sample from P.

Outline: In **Chapter 4**, we first formalize the two-phase learning framework called Nurture-then-Nature, where learning under a teacher (nurture) produces a surviving version space $\mathcal{V}_1$ and subsequent i.i.d. learning under the environment (nature) shrinks it to $\mathcal{V}_2$. We study the optimal teaching problem in two different settings: 1.) the **instance-agnostic** setting (unknown P), and 2.) the **instance-aware** setting. In instance-agnostic setting, we reduce the problem to optimizing the VC complexity of the version space and provide efficient and provably near **optimal algorithms** that **minimize the VC dimension of** $\mathcal{V}_1$ under budget B for various hypothesis classes like (i) finite binary hypothesis classes, (ii) axis-aligned rectangles on a grid, (iii) linear separators, and (iv) polynomial classifiers etc. In the **instance-aware** setting (known P), we first study a finite hypothesis class and present an efficient algorithm that achieves the **bicriteria approximation guarantee** for minimizing expected nature-phase learning time. Then, we study teaching using function approximation, and propose methods that utilize risk estimators like **linear and neural datamodels** to approximate NtN risk and produce practical teaching sets to reduce NtN risk for any ERM learners under suitable assumptions. Finally, we present experimental results to demonstrate the efficacy of our algorithm in both settings and compare them to baselines.

Chapter 1

Optimal Teaching for Linear BC Agents

1.1 Motivation

Behavior Cloning (BC) [9, 22, 84] is an important paradigm of learning in Reinforcement Learning (RL), that has been applied extensively to solve real-world problems like teaching machines to drive autonomous vehicles [69, 70], fly planes [75], perform robotic manipulations [50] etc. These real-world environments have large state space where the ability to generalize using linear or neural hypothesis class becomes essential for effective learning.

However, naively teaching an optimal policy to a BC learner using i.i.d. sample often demands a dataset that scales with the horizon length, the complexity of learner's hypothesis class and desired error [4, 74]. In many scenarios, like teaching to drive cars, an expert teacher may know a (near)-optimal policy and can leverage this knowledge to construct a small, non-i.i.d. dataset to teach the target policy to the BC learner far more efficiently. This problem is known as Machine Teaching and the size of smallest teaching set so produced is called Teaching Dimension (TD) [29, 96].

Several existing works [54, 55, 65] have studied optimal teaching in linear settings, primarily targeting individual surrogate learners, such as linear support vector machines (SVM). These surrogate learners often exhibit optimization biases, arguably making it easier to teach them individually. Consequently, the teaching set for a specific learner is often highly tailored to their biases, limiting its effectiveness for others. In contrast, in many real-world scenarios, such as teaching a classroom of students [98], the teacher must teach the entire class of students with a single lesson, even though each student may have unique

biases. In this work, we focus on the task of optimally teaching a family of linear BC learners that satisfy the consistency property, meaning each learner in this family produce a (subset of) hypotheses consistent with a demonstration dataset. We seek to answer the following question:

What is the smallest dataset required to teach a policy to a family of consistent linear BC learners?

To demonstrate the effectiveness of optimally teaching the linear BC family with a single dataset, we consider the following example.

Example 1.1 (Pick the Right Diamond)**.** *The game is shown in Figure 1.1(a). There is a board with* $n = 6$ *slots where each slot can have one of 4 different types of diamonds or can be empty. The game rule says that one must pick the most expensive diamond i.e. one with the highest number of edges, first; and if there are ties one must pick the rightmost one. The game continues until the board is empty. The teacher wants to find a minimal demonstration set to convey this rule to the agent.*

There are $5^n - 1$ *number of states with* $\mathcal{A} = [n]$*. Consider the family of consistent linear BC learners with a two-dimensional feature space denoting slot index and the number of edges in the slot. A naive teacher would demonstrate target action in all* $5^n - 1$ *states which grows exponentially with* n*. However, a clever teacher succeeds by just demonstrating two states(refer to Section 1.5.1 for complete results), thereby significantly saving the teaching cost from* $O(5^n)$ *to 2.*

Towards our goal of optimal teaching, we make the following contributions:

1. We formulate the problem of optimally teaching a family of linear BC learners and show that this problem is equivalent to teaching the hardest member in the family, i.e., a linear version space learner (Lemma 1.4).

2. We characterize optimal teaching in terms of covering extreme rays of primal cone and design a novel algorithm called 'TIE' 1 to optimally teach the family (Theorem 1.9).

3. However, as shown in Theorem 1.10, solving this problem is NP-hard and we propose an efficient algorithm with an approximation ratio of $\log(|\mathcal{A}| - 1)$ on TD (Theorem 1.11).

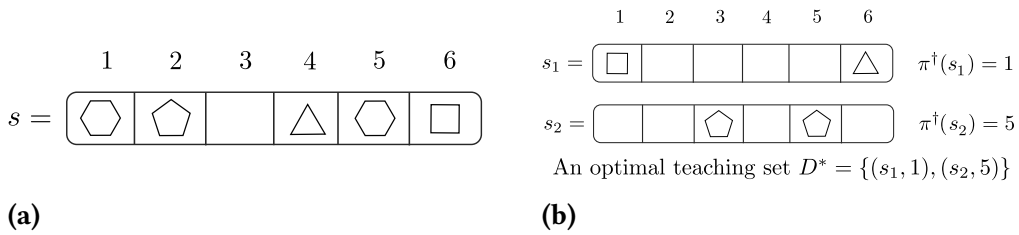


Figure 1.1: a) A board in a "Pick the Right Diamond" game. In this example 1.1, the target policy says to pick the diamond with the highest edge breaking the tie in favor of the rightmost slot if any. There are a total of $5^n - 1$ candidate teaching state and action pairs. We ask what is the minimum set of demonstrations of such boards would allow the teacher to teach the target policy to consistent linear learners. b) Only two carefully chosen demonstrations are sufficient to teach.

4. Through a set of experiments on real-world environments, we demonstrate the effectiveness of our TIE algorithm compared to other baselines (Section 1.5).

1.2 Related Work

Several prior works have studied optimal teaching of version space learners but mostly in finite or countable infinite version space settings [29, 44]. Some works like [98] have studied teaching multiple learners simultaneously but in an unsupervised learning setting of mean teaching. Instead, we study teaching a family of consistent behavior-cloning learners in a linear hypothesis space setting.

Comparatively, studies on optimal teaching of different linear learners are highly relevant to our work. For example, [54, 65] examined teaching linear learners like SVM, perceptron and logistic regression which can be seen as individual instances of consistent linear BC learners. These works focus on teaching individual learners, where teachers could exploit the strong biases of these learners to teach them relatively easily. On the other hand, we aim to teach the entire family of consistent linear BC learners where the teacher cannot base their teaching on the bias of individual learners. Additionally, [55] delved into the optimal teaching of iterative learners like gradient descent which is also biased [81]. Further, [48, 71] have explored the teaching dimension of kernel learners for teaching a linear/non-linear boundary in $\mathbb{R}^d$ space. Furthermore, these studies typically assume a more powerful teacher capable of constructing arbitrary covariate and label pairs, whereas our teacher is restricted

to selecting states from a fixed state space and aims to teach the learner to generalize to other states using feature covariates induced by the feature function.

Another significant line of research involves teaching-by-demonstration in an RL setting. Relevant studies by [14, 17] have focused on teaching linear IRL learners [2, 64] which reward based imitation learners that learn primarily in reward space and require planning access to the environment to eventually learn an optimal policy. Unlike them, our linear BC learners learn directly in the policy space by only using teaching demonstrations and do not require access to the MDP environment.

1.3 Problem Formulation

Consider a Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, P, \gamma, \mu)$ where $\mathcal{S}$ is a state space, $\mathcal{A}$ a finite action space, $R : \mathcal{S} \times \mathcal{A} \to [0,1]$ is reward function, $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is transition function, γ is the discount factor and μ is the initial state distribution. For simplicity, we assume $\mathcal{S}$ is finite, however, our analysis also extends to infinite case under reasonable assumptions. Let $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ be a feature function that defines a structured linear policy class. Given a fixed $w \in \mathbb{R}^d$, it induces a set of policies Π_w defined as follows:

$$\forall s \in \mathcal{S}, \Pi_w(s) = \Delta\left(\arg\max_{a \in \mathcal{A}} w^\top \phi(s,a)\right).$$

Consider a linear hypothesis class $\mathcal{H} = \mathbb{R}^d$ and let $\Pi = \cup_{w \in \mathcal{H}} \Pi_w$, $\Pi_{\text{Det}} = \{\Pi_w \in \Pi : \Pi_w \in \mathcal{A}^{\mathcal{S}}\}$ be the set of all stochastic and deterministic policies induced by $\mathcal{H}$ respectively. The value of a policy $\pi \in \Pi$ in MDP $\mathcal{M}$ is given by $V_\mu^\pi = \mathbb{E}_{\pi,P}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$. Furthermore, a class optimal policy $\pi^* \in \Pi$ is the one that maximizes value among all policies in the linear class, i.e., $\pi^* = \arg\max_{\pi \in \Pi} V_\mu^\pi$.

1.3.1 The Learner Family

We consider a Behavior Cloning (BC) learner $\mathcal{L} : D \to 2^{\mathcal{H}}$ that learns using a linear policy class $\mathcal{H} = \mathbb{R}^d$. On receiving a dataset $D = \{(s_i, a_i) : i \in [n]\} \subseteq \mathcal{S} \times \mathcal{A}$, it aims to learn a 'good' policy by imitating the dataset using a supervised learning algorithm [1, 74].

Given a dataset D, the learner maintains a set $\mathcal{L}_l(D)$ of empirical risk minimizing (ERM)

hypotheses defined by a loss function $\ell : \mathcal{A} \times \mathcal{A} \to \mathbb{R}^+$, i.e.,

$$\mathcal{L}_\ell(D) \leftarrow \arg \min_{\pi \in \Pi_w, w \in \mathcal{H}} \sum_{(s,a) \in D} \mathbb{E}_{a' \sim \pi(s)}[\ell(a', a)].$$

During deployment, the learner first arbitrarily selects a $w \in \mathcal{L}_\ell(D)$ and a $\pi \in \Pi_w$ and then uses π to execute all its actions. Correspondingly, it suffers a worst-case value risk of $R(D; \mathcal{L}) = V_\mu^{\pi^*} - \min_{\pi \in \Pi_w, w \in \mathcal{L}_\ell(D)} V_\mu^\pi$ in the MDP environment. We remark that a BC learner is nothing but a supervised learner applied to RL setting.

Consistent Linear BC Learners: We consider teaching a family of linear BC learners that have the consistency property and denote the family by $\mathfrak{C}$. The consistency property is as follows: given any realizable dataset D, i.e., a dataset generated by any policy $\pi \in \Pi_{\text{Det}}$, $\mathcal{L}$ always maintains a non-empty subset of hypotheses consistent with D, i.e.,

$$\forall \pi \in \Pi_{\text{Det}}, D \sim \Delta(\cup_{s \in \mathcal{S}}\{(s, \pi(s))\}), \text{ we have that, } \forall w \in \mathcal{L}_\ell(D), \Pi_w(s) = \pi(s), \forall (s, \pi(s)) \in D.$$

In linear settings, many well-known learners, such as the linear support vector machine(SVM), linear perceptron are consistent learners. We remark that each consistent learner may have their own bias to prefer certain consistent hypotheses over others which is directly influenced by their surrogate loss function or update methods [23, 81]. For example, an SVM learner always prefers a max-margin hypothesis over other hypotheses.

However, this family also contains arguably the most simplest linear BC learner, one that maintains the entire version space of consistent hypotheses and does not have any bias to prefer one consistent hypothesis over the other. We call it a linear version space(LVS) learner.

Linear Version Space (LVS) Learner: An LVS learner maintains the entire version space of hypothesis $\mathcal{V}(D)$ consistent with input dataset D, i.e.,

$$\mathcal{V}(D) = \{w \in \mathbb{R}^d : w^\top(\phi(s, a) - \phi(s, b)) > 0, \forall (s, a) \in D, a \neq b\}. \tag{1.1}$$

Equivalently, it does empirical risk minimization with respect to zero-one loss, i.e., $\mathcal{V}(D) = \mathcal{L}_{0\text{-}1}(D)$. Note that for a realizable dataset D, $\mathcal{V}(D) \supsetneq \{\}$ is an open polyhedral cone in $\mathbb{R}^d$.

Remark 1.2. *We introduce the following notation: let* $\psi_{sab} := \phi(s,a) - \phi(s,b)$ *be the feature difference vector for preferring action* a *over* b *in state* s*,* $\Psi(D)$ *be the set of all feature difference vectors induced by dataset* D*, i.e.,* $\Psi(D) = \{\psi_{sab} : (s,a) \in D, b \in \mathcal{A}, b \neq a\}$*. We define the primal cone of* $\Psi(D)$ *as* $\text{cone}(\Psi(D)) := \{\sum_{\psi \in \Psi(D)} \lambda_\psi \psi : \lambda_\psi \geqslant 0, \lambda \neq 0\}$*, and its dual as* $\text{cone}^*(\Psi(D)) := \{w \in \mathbb{R}^d : \langle w, \psi \rangle > 0, \forall \psi \in \Psi(D)\}$*. Note that the version space is the dual cone of* $\Psi(D)$*, i.e.,* $\mathcal{V}(D) = \text{cone}^*(\Psi(D))$*. We refer to Example 1.5 for an illustration.*

1.3.2 The Teacher

In our setup, there is a helpful teacher who controls the dataset $D \subseteq \mathcal{S} \times \mathcal{A}$ provided to the learner. The teacher knows an optimal deterministic policy $\pi^* : \mathcal{S} \to \mathcal{A}$ induced by a $w^* \in \mathcal{H}, \text{i.e.}, \pi^* = \pi_{w^*}$ and has the following teaching objective:

It wants to unambiguously teach the target policy π^* *to the entire family of consistent linear BC learners* $\mathfrak{C}$ *using as few demonstrations as possible.*

We remark that our framework can handle teaching any deterministic policy in Π_{Det} to the learner. But for simplicity, we will consider teaching an optimal deterministic policy π^*. Formally, given a teaching instance $(\mathcal{M}, \phi, \pi^*)$, the optimal teaching problem of the teacher is defined by the following optimization problem:

$$(\textbf{Teach-}\mathfrak{C}) \qquad \begin{aligned} D^* \leftarrow \min_{D \subseteq \mathcal{S} \times \mathcal{A}} \quad & |D| \\ \text{s.t.} \quad \forall \mathcal{L} \in \mathfrak{C}, \quad & \mathcal{L} \text{ learns } \pi^* \text{ uniquely.} \end{aligned} \tag{1.2}$$

This formulation models a classroom teaching setting, where the teacher is required to teach π^* to all learners in $\mathfrak{C}$ using a single dataset which is more challenging than teaching individual biased learners studied in prior works [48, 54, 65]. The size of the optimal teaching set $\text{TD}(\pi^*; \mathfrak{C}) = |D^*|$ is called the teaching dimension(TD) of the family $\mathfrak{C}$.

Remark 1.3. *Teaching the entire family* $\mathfrak{C}$ *has its drawback; if the teacher knows the learning bias of a specific learner, it may be able to possibly teach them with a smaller dataset. For example, to optimally teach a linear SVM in* $\mathbb{R}^d$ *just requires two examples [54]. However, such individual learner-specific teaching sets may not even be a valid teaching set for other learners in the* $\mathfrak{C}$ *like version space learners, hence useless for teaching the entire family. See Figure 1.3(a) for an example.*

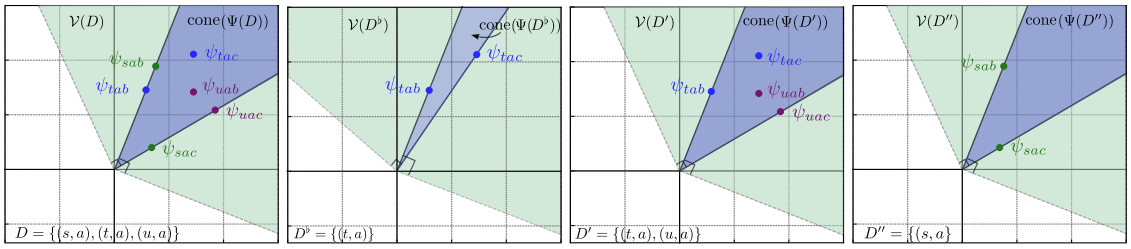


Figure 1.2: A simple illustration on importance of extreme rays. D, D', D'' succeed in teaching but $D^\flat$ fails depending on if they cover the extreme rays of cone($\Psi(D)$).

In a finite state setting, a naive teacher could succeed in teaching by demonstrating a full dataset $D_{\mathcal{S}} = \{(s, \pi^*(s)) : s \in \mathcal{S}\}$ to the learner. However, teaching on entire state space can be suboptimal and prohibitively expensive for large state space environments. A clever teacher who knows π^* can utilize the linear feature function of the learner family to teach π^* to them using a much smaller dataset. As shown in Example 1.5, demonstrating π^* on only one state is sufficient for teaching on the entire state space.

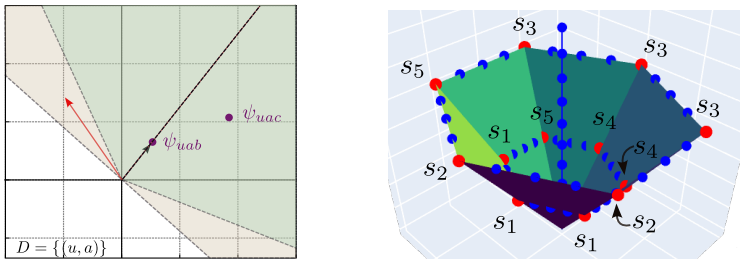
We recall that our problem 1.2 requires the teacher to teach π^* to all learners in $\mathfrak{C}$, which includes a large and diverse set of learners. In fact, enumerating all consistent learners may not even be practical. To address this issue, our next lemma shows that it is sufficient to focus on teaching the most challenging member of the family, i.e., the linear version space learner. The proof can be found in the Appendix.

Lemma 1.4. *Optimally teaching the family of consistent linear BC learners is equivalent to optimally teaching the linear version space BC learner.*

Hence, the teacher can achieve its objective by just focusing on optimally teaching the LVS learner. From now on, we will focus on optimally teaching π^* to an LVS learner given by the following optimization problem:

$$\begin{aligned} (\textbf{Teach-LVS}) \qquad & D^* \leftarrow \min_{D \subseteq \mathcal{S} \times \mathcal{A}} \quad |D| \\ \text{s.t.} \quad & \forall w \in \mathcal{V}(D), \pi \in \Pi_w, \quad \pi(s) = \pi^*(s), \forall s \in \mathcal{S}. \end{aligned} \tag{1.3}$$

This requires finding a minimal data D^* that induces π^* uniquely as the version space under D^*.



(a) **(b)**

Figure 1.3: a.) Optimal teaching set D of a (biased) consistent learner like SVM induce a larger space of weights w some of which (shown in yellow region) are inconsistent wrt π^* and so they cannot succeed in teaching LVS learner and the entire family of consistent learners. b.) Optimal teaching example in higher dimension $d \geqslant 3$ can have a large number of extreme rays to be covered using a subset of states making it an NP-hard problem 1.10.

Previous works have studied the problem of optimal teaching of version space learners, but have mostly been limited to either a finite hypothesis setting [7, 29] or highly structured hypothesis classes like axis-aligned rectangles [19, 29] which is very different from our structured linear setting. Before delving into the algorithm, we present an illustrative example in $\mathbb{R}^2$.

Example 1.5 (An instance of teaching linear version space BC learner in $\mathbb{R}^2$)**.** *Let* $S = \{s, t, u\}$, $\mathcal{A} = \{a, b, c\}$, *and* $\pi^*(s) = a, \forall s \in S$. *Consider the full demonstration set* $D = \{(s, a), (t, a), (u, a)\}$ *that induce* $\Psi(D) = \{\psi_{sab}, \psi_{sac}, \psi_{tab}, \psi_{tac}, \psi_{uab}, \psi_{uac}\}$ *as indicated by dots in Figure 1.2(a). The primal cone* $\text{cone}(\Psi(D))$ *is shown in blue, and the version space* $\mathcal{V}(D)$ *is in green. We note that the primal cone is supported by two extreme rays.*

The subset $D^\flat$ *is not valid/feasible teaching set as its version space* $\mathcal{V}(D^\flat)$ *(shown in green in Figure 1.2(b)) is wider than* $\mathcal{V}(D)$ *and contains some* w*'s that do not induce* π^* *in all states, thus violating the feasibility condition in equation 1.3. On the other hand, both* D' *and* D'' *induce the correct version space* $\mathcal{V}(D_S)$ *(as shown in green in Figures 1.2(c) and 1.2(d)) on the learner and succeeds in teaching* π^* *to it. Furthermore,* D'' *which consists of teaching on only one state is the optimal set. The problem becomes challenging as we move to higher dimensions where we can have a large number of extreme rays as shown in Figure 1.3(b).*

1.4 Teaching Algorithm and Analysis

We first describe a naive teaching algorithm that frames optimal teaching as an infinite set covering problem in the hypothesis space. This approach underscores the challenge of addressing our problem using the greedy inconsistent hypothesis elimination algorithm proposed in prior works [29].

1.4.1 Optimal Teaching as an Infinite Set Cover Problem in w Space

We observe that demonstrating $\pi^*(s)$ on a state s induces $|\mathcal{A}|-1$ feature difference vectors $\Psi_{s\pi^*(s)} = \{\psi_{s\pi^*(s)b} : b \in \mathcal{A}, b \neq \pi^*(s)\}$ in the primal (feature) space and correspondingly a version space $\text{cone}^*(\Psi_{s\pi^*(s)}) = \{w \in \mathbb{R}^d : w^\top \psi > 0, \forall \psi \in \Psi_{s\pi^*(s)}\}$ in the dual (weight) space of the LVS learner. Each such inequality, $w^\top \psi_{s\pi^*(s)b} > 0$, eliminates a halfspace $W_{sb} := \{w : w^\top \psi_{s\pi^*(s)b} \leqslant 0\} \subset \mathbb{R}^d$. Therefore, the effect of demonstrating $(s, \pi^*(s))$ is to eliminate the set of weights $W_s := \cup_{b \neq \pi^*(s)} W_{sb} = (\text{cone}^*(\Psi_{s\pi^*(s)}))^C$. The full demonstration set $D_{\mathcal{S}} = \cup_{s \in \mathcal{S}} \{(s, \pi^*(s))\}$ over all states eliminates the union $\cup_{s \in \mathcal{S}} W_s$, such that only the consistent version space $\mathcal{V}(D_{\mathcal{S}}) = \{w \in \mathbb{R}^d : w^\top \psi_{s\pi^*(s)b} > 0, \forall s \in \mathcal{S}, b \in \mathcal{A}, b \neq a\}$ survives.

The optimal teaching problem requires finding the smallest demonstration set that produces $\mathcal{V}(D_{\mathcal{S}})$ in the dual space which is equivalent to covering/eliminating the infinite set of inconsistent weights $\mathcal{V}(D_{\mathcal{S}})^C$ by a smallest finite collection of infinite subsets $\{W_s\}_{s \in \mathcal{S}}$. This is an infinite set cover problem in the weight space given as follows:

$$\min_{T \subseteq \mathcal{S}} |T| \quad \text{s.t.} \quad \mathcal{V}(D_{\mathcal{S}})^C = \cup_{t \in T} W_t.$$

At first glance, solving this problem may seem daunting. Certainly, since the inconsistent hypotheses set is uncountably infinite, we cannot keep track of inconsistent weights that have been eliminated so far and perform a greedy hypotheses elimination by greedily selecting the state that eliminates the maximal number of inconsistent hypotheses, as proposed by prior works [29, 44].

However, we note that $\mathcal{V}(D_{\mathcal{S}}) = \text{cone}^*(\Psi(D_{\mathcal{S}}))$ has a nice polyhedral cone structure that can be utilized further to simplify our problem as we show in the next section.

1.4.2 Teaching Via Covering of Extreme Rays of Primal Cone

To overcome the challenge mentioned above, we characterize the target version space cone $\mathcal{V}(D_{\mathcal{S}})$ in terms of extreme rays of primal cone$(\Psi(D_{\mathcal{S}}))$ and devise an optimal teaching algorithm based on this insight. Before doing that, we introduce some definitions below.

Definition 1.6 (Extreme Ray and its Cover)**.** *A ray $\mathcal{R}$ induced by a vector $v \in \mathbb{R}^d \backslash \{0\}$ is the set $\mathcal{R} = \{cv : c > 0\}$. Any vector in $\mathcal{R}$ serves as a representative of $\mathcal{R}$. A ray $\mathcal{R}$ is called an extreme ray of a cone $K \subseteq \mathbb{R}^d$ if for any $x, y \in K$, $x + y \in \mathcal{R} \implies x, y \in \mathcal{R}$. We say that a state $s \in \mathcal{S}$ covers a ray $\mathcal{R}$ if $\exists b \neq \pi^*(s) : \psi_{s\pi^*(s)b} \in \mathcal{R}$. Similarly, $T \subseteq \mathcal{S}$ is said to cover $\mathcal{R}$ if $\exists s \in T$ that covers $\mathcal{R}$.*

Recall that demonstrating π^* on a state s induces the feature difference set $\Psi_{s\pi^*(s)}$ in the primal space. Collectively teaching π^* on entire $\mathcal{S}$ induces feature difference set $\Psi(D_{\mathcal{S}})$ in primal and correspondingly version space cone$^*(\Psi(D_{\mathcal{S}}))$ in the dual space. By definition, a $w \in \mathbb{R}^d$ induces π^* if and only if $w \in$ cone$^*(\Psi(D_{\mathcal{S}}))$. Thus, for successful teaching 1.2, the teacher needs to exactly induce the version space cone$^*(\Psi(D_{\mathcal{S}}))$ in the dual space of the learner. This is equivalent to covering all the extreme rays of the primal cone$(\Psi(D_{\mathcal{S}}))$ as shown by the next lemma. We defer the proof to the appendix.

Lemma 1.7 (Necessary and Sufficient Condition for Teaching)**.** *A subset $T \subseteq \mathcal{S}$ is a valid teaching set if and only if it induces a representative vector on each extreme ray of the primal* cone$(\Psi(D_{\mathcal{S}}))$.

We denote the extreme ray set of primal cone$(\Psi(D_{\mathcal{S}}))$ by Ψ^*. Note that demonstrating $D_{\mathcal{S}}$ trivially induces Ψ^*, however, doing so may not be optimal. Instead, as suggested by the above lemma, it is sufficient to find a minimal subset of states that covers all rays in Ψ^*.

At a high level, our algorithm TIE 1 utilizes this insight to solve the optimal teaching problem in two stages. It first finds the extreme ray set Ψ^* of primal cone$(\Psi(D_{\mathcal{S}}))$. Next, it solves a set cover problem to find a minimal set of states that covers Ψ^*.

Stage 1: Finding extreme rays of primal cone$(\Psi(D_{\mathcal{S}}))$**:** Given a set of vectors $\mathcal{X}$, we propose an iterative algorithm to find all the extreme rays, i.e., a representative for each extreme rays of the primal cone$(\mathcal{X})$. The algorithm solves a sequence of linear program $LP(x, \mathcal{X})$, where at each step it tests whether a candidate $x \in \mathcal{X}$ is a unique representative of an extreme ray of cone$(\mathcal{X})$. If not, it removes x from $\mathcal{X}$ and moves to the next candidate as

shown in **MinimalExtreme** procedure 1. Otherwise, it has to keep x to cover all extreme rays. The proof can be found in the appendix.

Lemma 1.8 (Extreme Ray Test)**.** *Given a set of vectors $\mathcal{X} \in \mathbb{R}^d$, a candidate $x \in \mathcal{X}$ is a unique representative of an extreme ray of primal* $\text{cone}(\mathcal{X})$*, i.e.,* $x \notin \text{cone}(\mathcal{X} \backslash \{x\})$ *if and only if* $\text{LP}(x, \mathcal{X}) = -\infty$*, where,*

$$LP(x, \mathcal{X}): \qquad \min_w \quad \langle w, x \rangle \text{ s.t. } \langle w, x' \rangle \geqslant 1 \quad \forall x' \in \mathcal{X} \backslash \{x\}.$$

We remark that x is not a unique representative if and only if $\text{LP}(x, \mathcal{X}) > 0$ and in that case we can safely remove x. Employing this test iteratively on each element of $\mathcal{X}$ produces a unique representative for each extreme ray of primal $\text{cone}(\mathcal{X})$. We apply this process to $\mathcal{X} = \Psi(D_{\mathcal{S}})$ to obtain an extreme ray set $\Psi^* \subseteq \Psi(D_{\mathcal{S}})$ that contains exactly one representative for each extreme ray of $\text{cone}(\Psi(D_{\mathcal{S}}))$.

Stage 2: Finding minimal subset of states that cover the extreme rays Ψ^*: Once we have the extreme ray set Ψ^*, Lemma 1.7 requires a valid teaching set to cover all the rays in Ψ^*. To do that optimally using the smallest dataset, the teacher has to solve the following set covering problem on extreme rays space:

$$\min_{T \subseteq \mathcal{S}} |T| \text{ s.t. } \cup_{s \in T} V_s = U.$$

where universe $U = \Psi^*$ and each state $s \in \mathcal{S}$ covers a subset of extreme rays $V_s \subseteq \Psi^*$. Note that, unlike the infinite set cover problem over hypotheses space (1.4.1), this is a finite set cover problem over an extreme ray set. An optimal solution to this subproblem produces the optimal teaching set for teaching LVS learners which, by Lemma 1.4, is also an optimal teaching set for teaching the entire family of consistent linear BC learners.

1.4.3 Theoretical Results

We provide a complete pseudocode of our teaching algorithm'TIE' in Algorithm 1. 'TIE' achieves the following guarantee on the Teaching Dimension.

Theorem 1.9 (Optimal Teaching in Finite State Setting)**.** *Given an optimal teaching problem instance $(\mathcal{M}, \phi, \pi^*)$ 1.2, our teaching algorithm TIE 1 correctly finds the optimal teaching set D^* and achieves the Teaching Dimension $\text{TD}(\pi^*; \mathfrak{C})$.*

Computational Complexity: We note that stage 1 of 'TIE' is efficiently solvable. However, stage 2 involves solving a finite set cover problem where each subset V_s can cover as many as $|\mathcal{A}|-1$ elements. Note that for $|\mathcal{A}|=2$, each subset is singular, and the set cover problem can be efficiently computed. Hence, 'TIE' efficiently computes the optimal teaching set for instances with $|\mathcal{A}| \leqslant 2$.

Algorithm 1: Teach using Iterative Elimination (TIE)

def MinimalExtreme($\mathcal{X}$):

1: **for** each $x_j \in \mathcal{X}$ **do**
2: Solve LP$(x_j, \mathcal{X}/\{x\})$ defined by 1.8
3: **if** $v_j > 0$ **then**
4: $\mathcal{X} \leftarrow \mathcal{X} \backslash x_j$ ▷ eliminate x_j if not necessary
5: return $\mathcal{X}$ ▷ extreme vectors

def OptimalTeach($\mathcal{S}, \mathcal{A}, \pi^*, \phi$):

1: let $\Psi(D_{\mathcal{S}}) = \{\psi_{s\pi^*(s)b} \in \mathbb{R}^d : s \in \mathcal{S}, b \in \mathcal{A}, b \neq \pi^*(s)\}$ ▷ compute feature differences
2: $\Psi^* \leftarrow$ **MinimalExtreme**$(\Psi(D_{\mathcal{S}}))$
3: **for** $s \in \mathcal{S}$ **do**
4: $V_s \leftarrow \left\{\psi \in \Psi^* : \exists \psi_{s\pi^*(s)b} \in \Psi(D_{\mathcal{S}}), \hat{\psi}_{s\pi^*(s)b} = \hat{\psi}\right\}$ ▷extreme rays covered by s
5: $\{V_s : s \in T^* \subseteq \mathcal{S}\} \leftarrow \text{SetCover}(\Psi^*, \{V_s\}|_{s\in\mathcal{S}})$ ▷T^* is smallest cover of all extreme rays
6: teach $D^* = \{(t, \pi^*(t)) : t \in T^*\}$ to the agent ▷D^* is the minimum demonstration set

However, for $|\mathcal{A}| > 2$, stage 2 requires solving a general set cover problem which is NP-hard to solve. We show that no teacher can avoid this hardness by giving a poly-time reduction from a finite set cover problem to our optimal teaching problem. We defer the proof to the appendix.

Theorem 1.10 (Hardness of Optimal Teaching)**.** *Finding an optimal teaching set for teaching a linear version space BC learner is NP-hard in general for instances with action space size* $|\mathcal{A}| > 2$.

Although the set cover subproblem is NP-hard to solve, we can obtain an approximate solution efficiently using a greedy covering strategy. Applying this approach to solve our set cover problem in line 5 of Algorithm 1 yields an efficient, approximately optimal algorithm, called '*Greedy-TIE*' with the following guarantee:

Corollary 1.11 (Approximately Optimal Teaching)**.** *Our algorithm Greedy-TIE 1 efficiently teaches a family of consistent linear BC learners,* $\mathfrak{C}$*, and finds an approximately optimal teaching set* $\tilde{D}$ *such that* $|\tilde{D}| \leqslant \log(|\mathcal{A}|-1)|D^*|$.

The $\log(|\mathcal{A}|-1)$ approximation ratio of Greedy-TIE comes from the approximating set cover problem [90]. Furthermore, '*Greedy-TIE*' runs in poly-time $O((|\mathcal{S}||\mathcal{A}|)^3)$.

So far, we have assumed that the state space is finite. This assumption can be relaxed to infinite state setting under mild assumption as stated next. The proof can be found in the appendix.

Corollary 1.12 (Optimal Teaching for Infinite State Setting)**.** *Consider our optimal teaching problem with infinite state space* $\mathcal{S}$*. Under the assumption that* $\text{cone}(\Psi(D_{\mathcal{S}}))$ *is a closed and convex with finite extreme rays and the teacher knows the extreme rays to state mapping, our algorithm Greedy-TIE 1 correctly finds an approximately optimal teaching set.*

In the general case of an infinite state space, the induced version space may contain an (uncountable) infinite number of extreme rays. Consequently, covering this infinite set of extreme rays with a finite teaching set becomes impossible, which renders optimal teaching impractical.

Now, we turn to the issue of distribution shift which has been a pertinent issue in behavior cloning in RL [74]. For a BC learner imitating teacher's policy π^* using a learnt policy π, the error amplification in value is given by $|V_\mu^\pi - V_\mu^{\pi^*}| \leqslant \frac{1}{1-\gamma} \cdot \mathbb{E}_{s\sim d^{\pi^*}} \left[||\pi(s) - \pi^*(s)||_1\right]$.

However, in our teaching setting, this issue is resolved as the optimal teacher ensures the learner precisely learns π^*, leading to the following corollary.

Corollary 1.13 (Optimal Value Guarantee)**.** *Under teaching by our algorithm TIE, the entire family of linear learners* $\mathfrak{C}$ *achieve a zero approximate value risk, i.e.,* $\forall \mathcal{L} \in \mathfrak{C}$, $R(D^*; \mathcal{L}) = 0$.

Furthermore, it can be argued that BC learners are the most natural choice for a learner when a supportive teacher is available to demonstrate the target behavior. Unlike other learners like inverse RL [3, 14], BC operates directly in policy space, eliminating the need for planning. On the downside, since they maintain consistent hypotheses, they are limited to teaching only deterministic policies.

1.5 Experiments

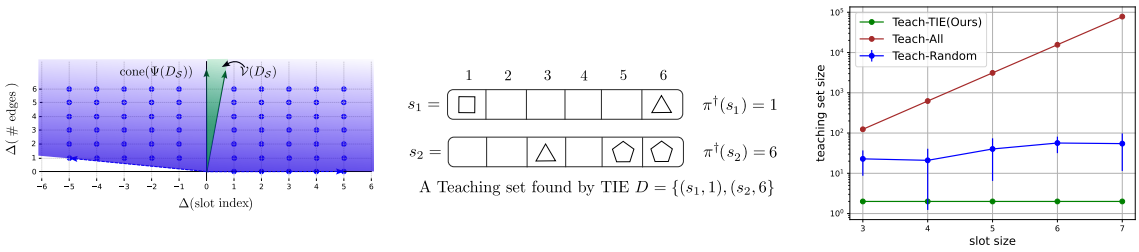
We evaluate our teaching algorithm *Greedy-TIE* on three environments: 1) *Pick the Right Diamond*, 2) *Visual Programming in Maze with Repeat Loops* and 3) *Polygon Tower environment* (provided in the appendix). Through these experiments, we aim to demonstrate the following: a) Our algorithm *Greedy-TIE* finds an optimal or near-optimal teaching set in all these environments. b) The optimal teaching dataset so produced is competitive with a learner-specific optimal teaching set and can teach any consistent linear BC learners, and c) *Greedy-TIE* performs significantly better than competitive baselines like *Teach-Random* and *Teach-All* that we define below.

Baselines: We consider two baselines. 1) *Teach-All*: This teacher simply teacher the target action in all states to the learner, 2) *Teach-Random*: This teacher draws states uniformly at random $s \sim \mathsf{U}(\mathcal{S})$ and adds it to a collection until the collection becomes a valid teaching set, i.e., it induces the target cone $\mathcal{V}(D_{\mathcal{S}})$. We note that the teaching set produced by prior works [54, 65] are specialized to individual learners and do not yield a feasible set for teaching the entire family of consistent linear learners. Furthermore, their teacher directly constructs covariate vectors (features) in $\mathbb{R}^d$ and is not able to choose individual states, thus, not directly applicable to our setting.

1.5.1 Pick the Right Diamond

Recall the game from Example 1.1. A state in $\mathcal{S} = \{\hexagon, \pentagon, \square, \triangle, o\}^n / \{o\}^n$ consists of a n dimensional board with one of four types of diamond or be empty(o). Each action in action space $\mathcal{A} = [n]$ represents picking an object in one of the cells. The complete description of the MDP environment can be found in the appendix.

Feature representation & optimal policy: The learner uses a natural feature function in $\mathbb{R}^2$ given as follows, $\phi(s, a) = [a, \#\text{edges of diamond at } a]$, where [#edges of diamond at a] is 0 if the slot is empty. The optimal policy is to collect the diamonds in order of decreasing value i.e. from a large to a small number of edges. In the case of ties, the learner should choose the rightmost diamond. This policy is feasible under the above featurization, for example, $w^* = [1, 10]$ uniquely induces π^*. For a board of size $n = 6$, there are a total of $5^6 - 1$ states, and their feature difference vectors $\Psi(D_{\mathcal{S}})$ are shown as blue dots in Figure 1.4(a).



(a) **(b)** **(c)**

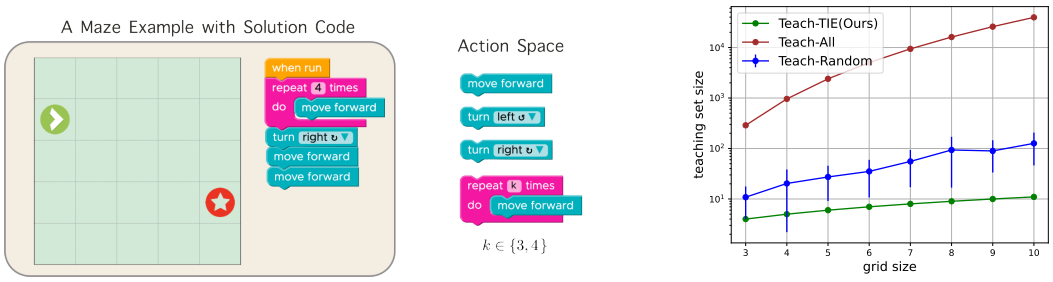
Figure 1.4: Optimal teaching in "Pick the right diamond" with $n = 6$ slots. a) Feature difference vectors $\Psi(D_S)$ induced by target policy is shown as blue dots, primal cone $\text{cone}(\Psi(D_S))$ as blue area, and dual version space $\mathcal{V}(D_S)$ as green area. b) A teaching set produced by *Greedy-TIE* on board of size 6. c) Comparison of our *Greedy-TIE* algorithm with other baselines.

The primal cone $\text{cone}(\Psi(D_S))$ is the blue-shaded area. It contains two extreme rays, both need to be covered for successful teaching. The version space is denoted in green.

Optimal teaching set: We note that any set that covers the two extreme rays is a valid teaching set. On a board instance of size $n = 6$, our algorithm *Greedy-TIE* produces a teaching set of size two as illustrated in Figure 1.4(b). This is an instance optimal teaching set and shows a dramatic improvement over teaching all $5^6 - 1$ states. We performed experiments on boards of different sizes and found that *Greedy-TIE* significantly outperforms the other two baselines as shown in Figure 1.4(c).

1.5.2 Visual Block Programming in Maze with Repeat Loop

We consider a real-world visual programming platform used for teaching kids/learners to write code to complete visual tasks in a maze environment [5, 16, 18, 24]. Further, we choose a domain that aims to teach learners to use repeat code blocks to write succinct code to complete a navigation-based task in maze environments of different sizes. The environment state consists of a $n \times n$ maze with a turtle (shown in green in Figure 1.5(a)) facing one of four directions, a goal cell (shown by a red star), and a (partial) piece of code that can be executed to move the turtle in the maze. The learners' objective is to assemble code blocks in sequence to write a piece of code that can solve the given maze task.



(a) **(b)**

Figure 1.5: a) An example of a programming task in 5×5 with solution code. The maze contains a turtle facing one of four directions (shown by a green arrow) and a goal cell (shown by a red star). The optimal (smallest) solution code to lead the turtle to the goal is shown on the side. The action space consisting of 5 basic code blocks is shown on the right. b) Performance of *Greedy-TIE* compared to baselines on this domain with different maze sizes.

The action space $\mathcal{A}$ consists of n actions (each representing a basic code block) available to the learner to write code and is given as follows: *Turn-Left (TL)*: turns turtle to its left, *Turn-Right (TR)*: turns turtle to its right, *Move-Forward (MV)*: moves turtle forward by one cell, *Repeat-k-Times-Move (R_k-MV)* is a complex block with repeat loop that moves the turtle forward by k cells in a single command where $k \in \{3, \cdots, n-1\}$ The task is to teach the agent to write most succint piece of code that can be executed to make the turtle reach the goal cell. This is captured by a reward function that gives a reward of -1 to the first three code blocks (*TL/TR/MV*) and a reward of -2 to repeat blocks R_k-*MV*. [1] The complete description of the MDP defining this problem can be found in the appendix.

Feature representation & optimal policy: We consider an execution-guided feature representation [18] that takes an initial board with a partial piece of code and constructs a feature vector by first executing the partial code to get an intermediate state and extracting features from that state. We use a natural feature representation $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$ that encodes the relative orientation and distance of the goal cell from the turtle cell; refer to the appendix for more details. The optimal policy is realizable by a linear policy under this

[1] We note that repeats are complex code blocks that have two components and should be used only when they provide an advantage, i.e., they can substitute more than two basic blocks.

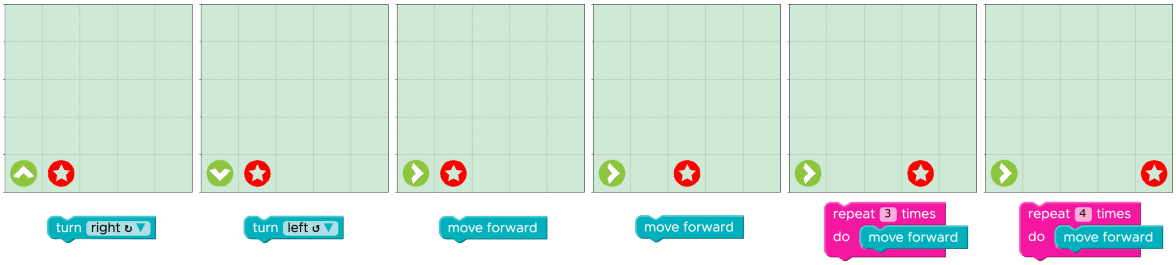


Figure 1.6: Optimal Teaching Set produced by *Greedy-TIE* on a goal-reaching coding task with 5×5 maze. The demonstration consists of states with an initial board without any partial code. The optimal action demonstrated to the learner is shown below each state.

representation. The teacher knows ϕ and can construct a dataset D of (state and optimal action) tuples and provide it to the learners. Its goal is to teach the target optimal policy of writing a succinct code to the entire family of learners $\mathfrak{C}$.

Optimal teaching set: We run our algorithm *Greedy-TIE* on environments with different sizes of maze and observe that it is able to find an optimal teaching set for each of the environments; refer to Figure 1.6 for an example on 5×5 maze. This optimal teaching set demonstrates each action exactly once on a suitable maze state where that action is an optimal one. Our algorithm performs significantly better than the other two baselines: *Teach-Random* and *Teach-All* when run on a maze of different sizes as shown by Figure 1.5(b). We also trained other candidate consistent learners like linear SVM, linear perception, and linear logistic regression on teaching set obtained by *Greedy-TIE* and verified that all of them achieve a risk of zero as claimed by our Theorem 1.9.

Chapter 2

Teaching BC Agents under MDP Constraints

2.1 Motivation

In the previous chapter, we studied a very powerful teacher that had the power to construct any dataset that it wanted to teach the target policy to the behavior cloning agent. However, in many real-world sequential decision-making settings, the teacher cannot put the learner in arbitrary states and instead has to first lead the learners to individual states (obeying the state transition constraints of the environment) and then teach them there. For example, consider a teacher helping a student learner to learn how to drive. The teacher cannot arbitrarily place the learner at challenging spots on the roads to teach them an important lesson; rather, it has to first navigate them to those spots and then teach the right action there. This introduces an additional aspect of optimal navigation to satisfy the teaching criteria characterized by the objective of minimizing the expected time required to reach a valid teaching set of states.

In this chapter, we consider teaching a finite or linear behavior cloning agent under an online MDP transition constraint on the teacher. In this setting, the teacher cannot generate an arbitrary dataset but rather has to sample a teaching sequence by acting in the environment. We consider a simpler online batch teacher where the teacher has to first collect a subset of "valid" teaching states by acting in a reward-free environment MDP $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathrm{P}\}$ using some navigation policy $\pi^{\dagger} : (\mathcal{S} \times \mathcal{A})^* \to \Delta(\mathcal{A})$ and once it has done that,

it can teach the target action on a subset of those states to the learner with a goal to induce the target policy in them.

The notion of what constitutes a valid teaching set is defined by the learning algorithm, for example, in linear behavior cloning setting a valid teaching set consists of a subset of states that covers all the extreme rays of the induced polyhedral cone while for finite tabular version space learner, a valid teaching set consists of subset of states that eliminates all the inconsistent hypothesis $\mathcal{H} \backslash \mathrm{h}^*$ from the version space. Note that both of these problems are characterized by an underlying finite set cover structure, which is NP-hard to solve. Moreover, in the online RL setting, the teacher no longer has the power to construct arbitrary states and has to obey the environment's transition dynamics to find a "valid" teaching set in minimum expected time. Towards that end, we study the following question in this setting:

Question: What is the optimal strategy to teach π^* to a BC learner in minimum time under online MDP constraints?

and make the following contribution: 1.) We formalize the problem of teaching BC agents under online MDP constraints. 2.) We show that this problem is equivalent to solving a novel problem Stochastic Set Cover Problem (SSCP) problem and provide an efficient reduction from SSCP to Stochastic Shortest Path problem (SSPP) which has been well studied in literature. 3.) We present planning algorithm and relevant convergence guarantee of value and policy iteration for solving SSCP through SSPP in Meta-MDP. 4.) Finally, we comment on the hardness of this problem and future directions.

For simplicity, we present this chapter from the viewpoint of a finite BC agent which can be easily generalized to linear BC by using relevant set cover structure.

2.2 Problem Formulation

This setting contains three important components, 1.) The MDP Environment, 2.) The Behavior Cloning (BC) learner, and 3.) The Teacher.

The MDP Environment

We consider a finite rewardless MDP environment defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathrm{P})$, where,

1. $\mathcal{S}$, $\mathcal{A}$ are finite state and action spaces.
2. $P : \mathcal{S} \times \mathcal{A} \to \Delta(A)$ is a Markovian state transition.

As we will see, the MDP transition constraints the teacher to only provide the demonstrations on the states that are visited by the teacher in a trajectory rollout.

The Learner

We consider a behavior cloning agent that essentially does version space learning using a finite tabular hypothesis class $\Pi \subseteq \mathcal{A}^{\mathcal{S}}$. On receiving a dataset $D = \{s_i, \pi^*(s_i)\}|_{i=1}^n$, the agent maintains a version space,

$$\mathcal{V}(D) = \{\pi \in \Pi : \pi(s_i) = \pi^*(s_i), \forall (s_i, \pi^*(s_i)) \in D\},$$

at the end of the training phase. In the test phase, it chooses one of the surviving policies in the version space and acts according to it.

The Teacher

The teacher wants to teach the target policy $\pi^* \in \mathcal{S} \to \mathcal{A}$ to the agent. However, unlike classical teaching setting, the teacher cannot construct arbitrary teaching dataset $D_T \subseteq (\mathcal{S} \times \mathcal{A})^*$. Rather, it has to use a navigation policy $\pi^\dagger$ to role out a trajectory,

$$\tau = (s_0, a_0, s_1, a_1, \cdots, s_t, a_t, \cdots) \sim P^{\pi^\dagger}$$

from the environment MDP until a "valid" teaching set of states have been collected. It then provides the target action demonstration to the student on a subset of those states to teach the target policy.

Remark 2.1. *Note that we allow the teacher to teach any action to the student once a state has been visited. This can be further restricted to only allow the teacher to teach the action that it has taken in the trajectory by defining set cover over action edges rather than states.*

Teaching a target policy without transition constraints requires solving a set cover problem, formally stated as,

$$D_T^* \leftarrow\equiv \min_{D_T \subseteq \{(s,\pi^*(s)):s\in\mathcal{S}\}} |D_T| \quad \text{s.t.} \quad \bigcup_s W_s = \Pi \setminus \pi^*$$

where $W_s = \{\pi \in \Pi\backslash\pi^* : \pi(s) \neq \pi^*(s)\}$ is the subset of inconsistent hypotheses eliminated by teaching $\pi^*(s)$ on state s, and $\mathcal{U} = \Pi\backslash\pi^*$ is the universe of all inconsistent hypotheses that has to be eliminated to succeed in teaching.

Definition 2.2 (A Valid Teaching Set)**.** *A subset $S \subseteq \mathcal{S}$ is called a valid teaching set if teaching π^* on them induces target policy in the learner, i.e., $\cup_{s\in S} W_s = \Pi\backslash\{\pi^*\}$.*

The Teaching Objective

The goal of the teacher is to find an optimal policy $\pi^\dagger : (\mathcal{S}\times\mathcal{A})^* \to \Delta(\mathcal{A})$ that minimizes the expected time required to teach $\pi^\dagger$ to the agent stated as follows:

$$\begin{aligned}\textbf{Online-Teach}(\mathcal{S},\mathcal{A},P,\Pi,\pi^*):\\ \pi^\dagger(s) \leftarrow &\arg\min_{\pi:(\mathcal{S}\times\mathcal{A})^*\to\Delta(\mathcal{A})} \mathbb{E}_{\tau\sim\pi}[\min\{t \geqslant 0 : \text{ teacher can teach } \pi^* \text{ using states } s_{0:t}]\\ \equiv &\arg\min_{\pi:(\mathcal{S}\times\mathcal{A})^*\to\Delta(\mathcal{A})} \mathbb{E}_{\tau\sim\pi}\left[\min\{t \geqslant 0 : \cup_{t'\leqslant t} W_{s_{t'}} = \Pi\backslash\pi^* | s_0 = s\right] \qquad (2.1)\end{aligned}$$

where the universe $\mathcal{U}$ of all inconsistent hypotheses has to be covered by the subcollection of $\{W_{s_1}, W_{s_2}, \cdots\}$ drawn in a trajectory rollout. This problem is a stochastic version of set cover problem called Stochastic Set Cover Problem that we study next.

Remark 2.3. *The optimal teaching policy need not be Markovian in $\mathcal{S}$. Unlike unconstrained teaching, covering a teaching set with the smallest size might not be a good strategy as it can be very difficult to reach them under MDP transition constraints. Refer to example 2.1.*

2.3 Stochastic Set Cover Problem

The Stochastic Set Cover Problem (SSCP) is a set cover problem embedded in an MDP and is defined by the tuple $(\mathcal{S},\mathcal{A},P,\mathcal{U},\{W_s\}_{s\in\mathcal{S}})$, where,

1. $\mathcal{S}, \mathcal{A}$ are finite state and action spaces.
2. $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{A})$ is a Markovian transition function.
3. U is a finite set of universe.
4. $\{W_s\}|_{s \in \mathcal{S}}$ a collection of subset of universe indexed by states.

The agent (that aims to solve SSCP) starts with an initial state $s_0 = s$ and takes action based on its navigation policy $\pi^\dagger$ that stochastically leads it to the next state as governed by the transition P. Whenever the agent reaches a state s, it gets to cover the associated subset $W_s \subseteq U$. The eventual objective is to find an optimal policy that takes the minimum expected time to cover the universe starting from any initial state, given as follows,

$$\textbf{SSCP}(\mathcal{S}, \mathcal{A}, P, U, \{W_s\}) :$$
$$\pi^\dagger(s) \leftarrow \min_{\pi:(\mathcal{S}\times\mathcal{A})^* \to \Delta(\mathcal{A})} \mathbb{E}_{\tau\sim\pi}\left[\min\{t \geqslant 0 : \cup_{t \leqslant t'} W_{s_{t'}} = U | s_0 = s\right], \forall s \in \mathcal{S}. \tag{2.2}$$

We take a look at a simple example to appreciate this problem.

Example 2.4. *Consider the example shown in Figure 2.1 of teaching a finite BC agent under MDP constraints. In this example, there are three states $\{s_1, s_2, s_3\}$, two actions $\{a_1, a_2\}$ and a finite policy class with three policies $\Pi = \{\pi_1, \pi_2, \pi_3\}$ as denoted in Figure 2.1(a). The teacher wants to teach the target policy $\pi_3(s) = a_1, \forall s \in \mathcal{S}$ to the learner.*

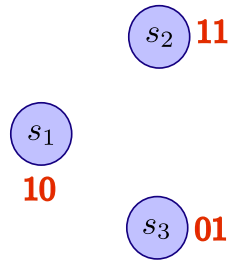
In classical teaching (without MDP constraints), as depicted in Figure 2.1(b), the universe set $\{\pi_1, \pi_2\}$ has to be covered/eliminated to succeed in teaching. There are two "valid" teaching sets $\{s_2\}$ or $\{s_1, s_3\}$. Without transition constraints, the teacher can simply choose to teach (s_2, a_1) leading to $TD = 1$. However, with transitions (as shown in Figure 2.1(c)), the teacher starts with initial state s_1 and has to rollout a trajectory (as shown in Figure 2.1(d)) to reach one of the two "valid" teaching sets. The goal of the teacher is to find a policy that covers any one of the valid teaching set in minimum time.

Lemma 2.5 (SSCP is NP-hard)**.** *SSCP contains two NP-hard problem, the set cover problem and the Asymmetric Traveling Salesman Problem (ATSP) making it NP-hard.*

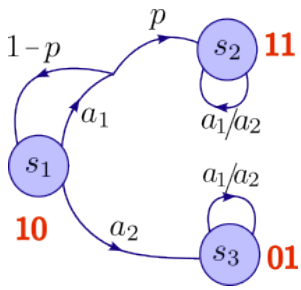
1. *A deterministic SSCP instance with a complete MDP graph is also an instance of a standard finite set cover problem.*

	s_1	s_2	s_3
π_1	a_2	a_2	a_1
π_2	a_1	a_2	a_2
π_3	a_1	a_1	a_1

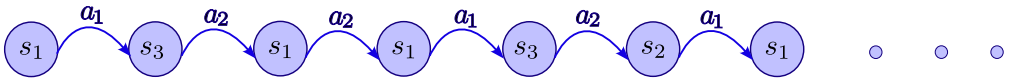
(a) A finite hypothesis class



(b) Set cover structure



(c) Set cover with transitions



(d) Random trajectory rolled out using navigation policy $\pi^\dagger$

Figure 2.1: From classical constructive teaching to online teaching under MDP transition constraints. The bits in red denote which of the two inconsistent policies $\{\pi_1, \pi_2\}$ is eliminated by teaching in that state. The smallest teaching set is $D_1 = \{s_2, \pi^*(s_2)\}$. The expected time to cover $\{s_2\}$ can be arbitrarily bad than covering another larger teaching set $\{s_1, s3\}$.

2. *A deterministic SSCP instance with a single optimal set cover is an instance of an subset ATSP problem - find the shortest path that covers the given subset of states.*

Hence, SSCP is NP-hard as well.

Limitations of Classical Algorithms for Optimal Teaching

Efficiently covering the universe of elements in minimum time presents a significant challenge. Using the example in Figure 2.1, we demonstrate that two intuitive methods for solving the SSCP can perform arbitrarily poorly compared to the optimal solution:

Method 1: Identify the smallest cover first, then find a policy to reach them.

Method 2: Utilize the target policy itself to collect teaching states for the learner.

Consider two navigation policies that the teacher uses for teaching the target π_3 to the BC agent: 1.) the target policy π_3 itself, and 2.) an alternative policy π' defined as $\pi'(s) = a_2, \forall s$.

The expected time required to cover the universe $\{\pi_1, \pi_2\}$ using these policies is:

$$\mathbb{E}[T|\pi_3] = 1 + \frac{1}{p}, \quad \mathbb{E}[T|\pi'] = 2.$$

We observe that π_3 covers the minimal teaching set and would therefore be the solution selected by both the methods. However, as $p \to 0$, the performance of π_3 degrades arbitrarily compared to the contender policy π', which is, in fact, time-optimal.

Remark 2.6. *We remark that SSCP is an infinite-horizon problem and cannot be modeled by a discounted MDP.*

Next, we propose a planning algorithm that aims to solve SSCP by reducing it to a well known optimal navigation problem in MDP called the Stochastic Shortest Path Problem (SSPP). We first formalize the SSPP problem and then provide a reduction of SSCP to SSPP.

2.4 Stochastic Shortest Path Problem

The stochastic shortest path problem (SSPP) [10, 12] is a stochastic version of the shortest path problem on a directed graph. Given a reward-free MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P)$ with finite state and action space and a target state $s^\dagger \in \mathcal{S}$, the goal is to find a policy that reaches the target state in minimum expected time from any starting state $s \in \mathcal{S}$. The target state is an absorbing state $s^\dagger$ satisfying $P(s^\dagger|s^\dagger, a) = 1, \forall a \in \mathcal{A}$.

Since the transitions are stochastic, the interaction of an agent with the environment using a policy generates a stochastic trajectory rollout $\tau = (s_0, a_0, s_1, a_1, \cdots)$ which may or may not terminate at the target state. The objective of the agent is formalized as finding a policy that can reach the target absorbing state $s^\dagger$ from any initial state $s \in \mathcal{S}$ in the minimum expected time,

$$\pi^\dagger(s) \leftarrow \underset{\pi:(\mathcal{S}\times\mathcal{A})^* \to \Delta(\mathcal{A})}{\arg\min} \; \mathbb{E}_{\tau\sim\pi}\left[\min\{t \geqslant 0 : s^\dagger \in \tau_t\} | s_0 = s\right].$$

where $\tau_t = (s_0, a_0, s_1, \cdots, s_{t-1}, a_{t-1}, s_t)$ is the partial trajectory experienced by agent till time step t and we denote expected time to reach target state under policy π as H^π which

can be restated in form of undiscounted cost as follows:

$$H^{\pi}(s) = \lim_{T\to\infty} \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=1}^{T} c(s_t, a_t) \mid s_0 = s\right] \qquad \text{where the cost } c(s_t, a_t) = \mathbb{1}[s_t \neq s^{\dagger}]$$

The goal of SSPP is reformulated as,

$$H^{*}(s) = \min_{\pi:(\mathcal{S}\times\mathcal{A})^{*}\to\Delta(\mathcal{A})} H^{\pi}(s)$$

We denote the optimal policy $\pi^{\dagger}(s) = \arg\min_{\pi:(\mathcal{S}\times\mathcal{A})^{*}\to\Delta(\mathcal{A})} H^{\pi}(s), \forall s \in \mathcal{S}$. Similar to the connectivity and non-negative cost cycle requirements in the deterministic shortest path problem, SSPP requires the existence of a proper policy to ensure the target is reachable from any initial state.

Definition 2.7 (Proper Policy)**.** *A proper policy is one that reaches the target state* $s^{\dagger}$ *from any initial state* $s \in \mathcal{S}$ *with probability one. A policy that is not proper is called improper.*

Assumption 2.8. *We assume that* $\mathcal{M}$ *admits a proper policy and for all improper policies* $\exists s \in \mathcal{S}$ *s.t.* $T^{\pi}(s) = \infty$*.*

We have that for a proper policy, $T^{\pi}(s) < \infty, \forall s \in \mathcal{S}$. Under these two assumptions, prior works [11, 12] have shown that SSPP can be efficiently solved by a value/policy iteration algorithms.

2.5 Solving SSCP using SSPP

We first provide an efficient reduction of SSCP to SSPP and then utilize the policy/value iteration algorithms for SSPP to solve the SSCP problem.

2.5.1 Construction of Meta MDP.

The construction of meta–MDP $\bar{\mathcal{M}} = (\bar{\mathcal{S}}, \bar{\mathcal{A}}, \bar{P}, \bar{c})$ from stochastic set cover MDP $\mathcal{M}$ is given as follows:

- **Meta State Space:** $\bar{\mathcal{S}} = \mathcal{S} \times 2^{\mathcal{U}}$. We write $\bar{s} = (s, u)$ meaning the agent is at environment state s and has already covered universe elements in $u \subseteq \mathcal{U}$.

- **Meta Action Space:** $\bar{\mathcal{A}}(\bar{s}) = \mathcal{A}(s) = \mathcal{A}$.
- **Goal/absorbing set:** $\bar{G} \triangleq \{(s, U) : s \in \mathcal{S}\}$ is the meta absorbing state.
- **Meta Transitions:** The transition for taking action $a \in \mathcal{A}$ in state (s, u) is given as,

$$\bar{P}\big((s', u') \mid (s, u), a\big) \;=\; P(s' \mid s, a) \;\cdot\; \mathbf{1}\{u' = u \cup W_{s'}\}$$
$$\bar{P}(\bar{G} | \bar{G}, a) = 1, \forall a \in \mathcal{A}.$$

- **Cost (time):** Unit cost per step until $\mathcal{U}$ is covered, $\bar{c}\big((s, u), a\big) = \mathbf{1}\{u \neq U\}$.

We note that goal state is constructed by joining all meta states where universe U has already been covered and making it self absorbing. SSPP is a well studied problem in stochastic control literature. Next, we adapt the Bellman value and optimality equations for SSPP problem in meta MDP $\bar{M}$.

Solving SSPP in a meta MDP requires connectivity and non-negative cycle cost assumption similar to that in deterministic shortest path problem. The is stated through the existence of a proper policy in meta MDP as we state next.

Assumption 2.9 (Meta Properness)**.** *There exists a stationary policy on $\bar{\mathcal{M}}$ that reaches $\bar{G}$ almost surely from every $\bar{s} = (s, u)$; any policy that fails to reach $\bar{G}$ has infinite expected cost from at least one $\bar{s}$.*

Our next lemma proves the equivalence between finding optimal policy for solving SSCP to that of SSCP.

Lemma 2.10 (Reduction Correctness)**.** *For any initial $s_0 \in \mathcal{S}$, and policy $\pi : (\mathcal{S} \times \mathcal{A})^* \to \mathcal{A}$,*

$$\mathbb{E}_\pi \left[\min\{t \geqslant 0 : \cup_{t' \leqslant t} W_{s_{t'}} = \Pi \backslash \pi^* | s_0 = s\right] = \mathbb{E}_\pi \left[\min\{t \geqslant 0 : \cup_{t' \leqslant t} \bar{s_t}' \supseteq \bar{G} | \bar{s}_0 = (s, \varnothing)\right]$$

Proof. The proof follows directly from the equivalence between the SSCP trajectory and its corresponding SSPP trajectory. □

2.5.2 Bellman Equations for Expected Time

Let $H^*(s,u)$ be the optimal expected remaining time to cover all of U from meta state (s,u), and let H^π denote the time under a fixed stationary meta–policy π. The Bellman optimality and value equations in the meta MDP is given as follows:

Bellman Optimality equations.

$$\boxed{\begin{aligned} &H^*(s,U) = 0, \qquad \forall s \in S, \\ &H^*(s,u) = 1 + \min_{a \in \mathcal{A}(s)} \sum_{s' \in S} P(s' \mid s,a)\, H^*(s', u \cup W_{s'}) \quad (u \subsetneq U). \end{aligned}}$$

Bellman operators. Define the *minimum–time* Bellman operator T and policy operator T^π on functions $H : \bar{S} \to \mathbb{R}_{\geqslant 0}$ by

$$(TH)(s,U) = 0, \quad (TH)(s,u) = 1 + \min_{a \in \mathcal{A}(s)} \sum_{s'} P(s'|s,a)\, H(s', u \cup W_{s'}) \ (u \subsetneq U),$$

$$(T^\pi H)(s,U) = 0, \quad (T^\pi H)(s,u) = 1 + \sum_{s'} P(s'|s,\pi(s,u))\, H(s', u \cup W_{s'}) \ (u \subsetneq U).$$

Then H^* is the unique fixed point of T on the admissible set, and for every stationary π, H^π is the unique fixed point of T^π.

Theorem 2.11 (Well Posedness)**.** *Under meta–properness assumption, the Bellman system with operator T has a unique finite solution H^* with boundary $H^*(\cdot, U) = 0$, and there exists an optimal stationary deterministic meta–policy π^* attaining H^*.*

2.6 Algorithms and Results

Next, we present the value iteration method that can find the optimal policy for the stochastic shortest path problem in meta MDP, thereby equivalently solving the stochastic set cover problem in the original MDP.

Meta–Value Iteration (VI). Initialize $H_0 \equiv 0$ and iterate

$$H_{k+1} \leftarrow TH_k, \qquad k = 0, 1, 2, \ldots$$

with greedy extraction

$$\pi_{k+1}(s, u) \in \arg\min_{a \in \mathcal{A}(s)} \sum_{s'} P(s'|s, a)\, H_k(s', u \cup W_{s'}).$$

Meta–Policy Iteration (PI). Start from any proper π_0 on $\bar{\mathcal{M}}$. For $i = 0, 1, 2, \ldots$

(Evaluation)

$$H^{\pi_i} \leftarrow \text{solve } H = T^{\pi_i} H, \tag{2.3}$$

(Improvement)

$$\pi_{i+1}(s, u) \in \arg\min_{a} \sum_{s'} P(s'|s, a)\, H^{\pi_i}(s', u \cup W_{s'}). \tag{2.4}$$

Stop when $\pi_{i+1} = \pi_i$.

The value and policy iteration algorithms stated above comes with following convergence guarantees which follow directly from the convergence guarantee of VI/PI iteration in SSPP [11, 12].

Theorem 2.12 (VI Convergence under T). *With $H_0 \equiv 0$, the VI sequence $H_{k+1} = TH_k$ is monotone increasing and converges pointwise to H^*. Moreover, there exist weights $w(\bar{s}) \geqslant 1$ and $\beta \in (0, 1)$ such that*

$$\|TH - TH'\|_w \leqslant \beta\, \|H - H'\|_w \quad \textit{for all } H, H',$$

hence $\|H_k - H^*\|_w \leqslant \beta^k \|H^*\|_w$.

Theorem 2.13 (PI Convergence under T). *If π_0 is proper, all iterates are proper, the values satisfy $H^{\pi_{i+1}} \leqslant H^{\pi_i}$ componentwise, and with consistent tie-breaking PI terminates in finitely many improvements at an optimal π^* with value H^*.*

Remark 2.14 (Dependence on Universe Size). *We note that the meta state size grows exponentially in $|\mathcal{U}|$, leading to an efficient solution only when $\mathcal{U}$ is constant or small. This happens*

in many realistic settings of linear behavior cloning learners wherethe extreme rays set can be very small even though the number of states is huge.

2.7 Discussion and Open Problems

Conjecture: inapproximability even with deterministic transitions. We conjecture that the SSCP, *minimizing expected time to cover the universe* $\mathcal{U}$ *via meta–states* $\bar{s} = (s, u)$ *with unit cost until* $u = \mathcal{U}$, is *inapproximable* even in the special case of *deterministic* transition systems (i.e., a directed graph with actions determining a unique next state). In particular, we conjecture that no polynomial-time algorithm can achieve PTAS unless $P = NP$.

Scope and limitations. Our algorithms (meta–VI with the Bellman operator T) *compute optimal solutions* on the meta–MDP but have worst-case state growth $|\bar{\mathcal{S}}| = |\mathcal{S}|\, 2^{|\mathcal{U}|}$. Thus, the per-iteration work is exponential in $|\mathcal{U}|$ in the absence of exploitable structure. The conjectured inapproximability hints that this exponential dependence may be unavoidable in general.

Future directions. We leave a thorough hardness and approximability analysis to future work, and highlight several promising avenues:

- **Structural assumptions for efficient approximation.** Identify graph/MDP classes where coverage exhibits additional structure (e.g., bounded width or DAG topology, small *covering dimension*, submodular gain with metric travel, bounded diameter), and design algorithms with provable factors (e.g., $O(\log|\mathcal{U}|)$ via greedy+reachability or bicriteria time/coverage trade-offs).
- **Learning under generative access/simulators.** When P is unknown but a simulator exists, investigate optimistic planning with coverage-aware bonuses; characterize sample complexity in terms of the optimal time-to-cover $\mathcal{U}$.
- **Bicriteria and robust objectives.** Allow small coverage slack (e.g., cover $(1-\alpha)$ fraction of $\mathcal{U}$) or introduce budgets/risk constraints, aiming for scalable approximations with guarantees under uncertainty.

Chapter 3

Optimal Teaching of RL Agent

3.1 Introduction

In recent years, reinforcement learning (RL) has seen applications in a wide variety of domains, such as games [61, 80], robotics control [8, 46] and healthcare [47, 79]. One of the fundamental questions in RL is to understand the sample complexity of learning, i.e. the amount of training needed for an agent to learn to perform a task. In the most prevalent RL setting, an agent learns through continuous interaction with the environment and learns the optimal policy from natural reward signals. For standard algorithms such as Q-learning, naive interaction with MDP suffers exp complexity [52]. In contrast, many real-world RL scenarios involve a knowledgable (or even omniscient) teacher who aims at guiding the agent to learn the policy faster. For example, in the educational domain, a human student can be modeled as an RL agent, and a teacher will design a minimal curriculum to convey knowledge (policy) to the student (agent) [21].

In the context of reinforcement learning, teaching has traditionally been studied extensively under the scheme of *teaching-by-demonstration (TbD)*, where the teacher provides demonstrations of state/action trajectories under a good policy, and the agent aims to mimic the teacher as closely as possible [36]. However, in many applications, it is inconvenient for the teacher to demonstrate because the action space of the teacher is distinct from the action space of the learner. In contrast, it is usually easier for the teacher to *teach by reinforcements (TbR)*, i.e. with rewards and punishments. For example, in dog training, the trainer can't always demonstrate the task to be learned, e.g. fetch the ball with its mouth, but instead

would let the dog know whether it performs well by giving treats strategically [21]; In personalizing virtual assistants, it's easier for the user to tell the assistant whether it has done a good job than to demonstrate how a task should be performed. Despite its many applications, TbR has not been studied systematically.

In this chapter, we close this gap by presenting to our knowledge the first results on TbR. Specifically, we focus on a family of RL algorithms called Q-learning. Our main contributions are:

1. We formulate the optimal teaching problem in TbR.
2. We characterize the worst-case sample complexity of teaching, termed as "teaching dimension" (TDim), for Q-learning under four different teachers, distinguished by their power (or rather constraints) in constructing a teaching sequence. See Table 3.1 for a summary of results.
3. For each level, we design an efficient teaching algorithm which matches the TDim.
4. We draw connections between our results and classic results on the sample complexity of RL and of TbD.

Table 3.1: Our Main Results on Teaching Dimension of Q-Learning

Teacher	**Level 1**	**Level 2**	**Level 3**	**Level 4**
Constraints	none	respect agent's a_t	$s_{t+1} : P(s_{t+1}\|s_t, a_t) > 0$	$s_{t+1} \sim P(\cdot\|s_t, a_t)$
TDim	S	$S(A-1)$	$O\left(SAH\left(\frac{1}{1-\varepsilon}\right)^D\right)$	$O\left(SAH\left(\frac{1}{(1-\varepsilon)p_{\min}}\right)^D\right)$

3.2 Related Work

Classic Machine Teaching Since computational teaching was first proposed in [29, 78], the teaching dimension has been studied in various learning settings. The vast majority focused on batch supervised learning. See [99] for a recent survey. Of particular interest to us though is teaching online learners such as Online Gradient Descent (OGD) [51, 55], active learners [31, 67], and sequential teaching for learners with internal learning state [20, 35, 58]. In contrast to OGD where the model update is fully determined given the teacher's data, the RL setting differs in that the teacher may not have full control over the agent's behavior (e.g. action selection) and the environment's evolution (e.g. state transition), making efficient

teaching more challenging. Several recent work also study data poisoning attacks against sequential learners [39, 56, 57, 73, 91, 93, 95]. The goal of data poisoning is to force the agent into learning some attacker-specified target policy, which is mathematically similar to teaching.

Teaching by Demonstration Several recent works studied teaching by demonstrations, particularly focusing on inverse reinforcement learning agents (IRL) [14, 17, 33, 40, 86, 89]. IRL is a sub-field of RL where the learners aim at recovering the reward function from a set of teacher demonstrations to infer a near-optimal policy. Teaching in IRL boils down to designing the most informative demonstrations to convey a target reward function to the agent. Their main difference to our work lies in the teaching paradigm. IRL belongs to TbD where the teacher can directly demonstrate the desired action in each state. The problem of exploration virtually disappears, because the optimal policy will naturally visit all important states. On the other hand, as we will see next, in the TbR paradigm, the teacher must strategically design the reward signal to *navigate* the learner to each state before it can be taught. In other words, the challenge of exploration remains in reinforcement-based teaching, making it much more challenging than demonstration-based teaching. It is worth mentioning that the NP-hardness in finding the optimal teaching strategy, similar to what we establish in this chapter (see Appendix), has also been found under the TbD paradigm [89].

Empirical Study of Teaching-by-Reinforcement Empirically, teaching in RL has been studied in various settings, such as reward shaping [63], where teacher speeds up learning by designing the reward function, and action advising [6, 85], where the teacher can suggest better actions to the learner during interaction with the environment. Little theoretical understanding is available in how much these frameworks accelerate learning. As we will see later, our teaching framework generalizes both approaches, by defining various levels of teacher's control power, and we provide order-optimal teaching strategies for each setting.

3.3 Problem Definitions

The machine teaching problem in RL is defined on a system with three entities: the underlying MDP environment, the RL agent (student), and the teacher. The teaching process is defined in algorithm 2. Whenever the boldface word "**may**" appears in the protocol, it

depends on the level of the teacher and will be discussed later. In this chapter, we assume that there is a clear separation between a training phase and a test phase, similar to the best policy identification (BPI) framework [26] in classic RL. In the training phase, the agent interacts with the MDP for a finite number of episodes and outputs a policy in the end. In the test phase, the output policy is fixed and evaluated. In our teaching framework, the teacher can decide when the training phase terminates, and so teaching is regarded as completed as soon as the target policy is learned. Specifically, in the case of Q-learning, we do not require that the estimated Q function converges to the true Q function w.r.t. the deployed policy, which is similarly not required in the BPI or PAC-RL frameworks, but only require that the deployed policy matches the target policy exactly.

Algorithm 2: Machine Teaching Protocol on Q-learning

Entities: MDP environment, learning agent with initial Q-table Q_0, teacher with target policy $\pi^\dagger$.

1: **while** $\pi_t \neq \pi^\dagger$ **do**
2: MDP draws $s_0 \sim \mu_0$ after each episode reset. But the teacher **may** override s_0.
3: **for** $t = 0, \dots H-1$ **do**
4: The agent picks an action $a_t = \pi_t(s_t)$ with its current behavior policy π_t. But the teacher **may** override a_t with a teacher-chosen action.
5: The MDP evolves from (s_t, a_t) to produce immediate reward r_t and the next state s_{t+1}. But the teacher **may** override r_t or move the system to a different next state s_{t+1}.
6: The agent updates $Q_{t+1} = f(Q_t, e_t)$ from experience $e_t = (s_t, a_t, r_t, s_{t+1})$.
7: Once the agent learns $\pi^\dagger$, the teacher ends the teaching phase, and the learned policy is fixed and deployed.

Environment M: We assume that the environment is an episodic Markov Decision Process (MDP) parameterized by $M = (\mathcal{S}, \mathcal{A}, R, P, \mu_0, H)$ where $\mathcal{S}$ is the state space of size S, $\mathcal{A}$ is the action space of size A, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition probability, $\mu_0 : \mathcal{S} \to \mathbb{R}$ is the initial state distribution, and H is the episode length. Next, we define two quantities of interest of an MDP that we will use in our analysis.

Definition 3.1. *Let the* ***minimum transition probability*** $p_{\min}$ *of an MDP be defined as* $p_{\min} = \min_{s,s' \in \mathcal{S}, a \in \mathcal{A}, P(s'|s,a)>0} P(s'|s,a)$.

Definition 3.2. *Let the **diameter** D of an MDP be defined as the minimum path length to reach the hardest-to-get-to state in the underlying directed transition graph of the MDP. Specifically,*

$$\begin{aligned} D = \max_{s \in S} \quad & \min_{T,(s_0,a_0,s_1,a_1,\ldots,s_T=s)} T \\ \textit{s.t.} \quad & \mu_0(s_0) > 0, P(s_{t+1}|s_t,a_t) > 0, \forall t \end{aligned} \tag{3.1}$$

RL agent L**:** We focus on a family of Q-learning agents $L \in \mathcal{L}$ with the following properties:

1. **Behavior policy**: The agent behaves according to the ε-greedy policy for some $\varepsilon \in [0,1]$, i.e.
$$\pi_t(s) := \begin{cases} a^* \leftarrow \arg\max_a Q_t(s,a) & \text{w.p. } 1-\varepsilon \\ \text{Unif}(\mathcal{A} \backslash a^*), & \text{w.p. } \varepsilon. \end{cases}$$
Note this definition is slightly different but equivalent to standard ε-greedy exploration, where we merged the probability of choosing $\arg\max_a Q_t(s,a)$ in the second branch into the first. This simplifies our notation later.
2. **Learning Update**: Given experience $e_t = (s_t, a_t, r_t, s_{t+1})$ at time step t, the learning update $Q_{t+1} = f(Q_t, e_t)$ only modifies the (s_t, a_t) entry of the Q-table. Furthermore, the Q-table is "controllable": for any s_t, a_t, s_{t+1}, there exists a reward r such that the ranking of a_t within $Q_{t+1}(s_t, \cdot)$ can be made first, last or unchanged, respectively.

This family includes common Q-learning algorithms such as the standard ε-greedy Q-learning, as well as provably efficent variants like UCB-H and UCB-B [38].

Teacher: In this chapter, we study four levels of teachers from the strongest to the weakest:

1. **Level 1**: The teacher can generate arbitrary tuples $(s_t, r_t, s_{t+1}) \in S \times \mathbb{R} \times S$, and override the agent chosen action a_t. None of these needs to obey the MDP (specifically μ_0, R, P).
2. **Level 2**: The teacher can still generate arbitrary state s_t, reward r_t and next state s_{t+1}, but cannot override the agent's action a_t. The agent has "free will" in choosing its action.
3. **Level 3**: The teacher can still generate arbitrary reward r_t but can only generate MDP-supported initial state and next state, i.e. $\mu_0(s_0) > 0$, and $P(s_{t+1}|s_t,a_t) > 0$. However, it does not matter what the actual nonzero MDP probabilities are.
4. **Level 4**: The teacher can still generate arbitrary reward r_t but the initial state and next state must be sampled from the MDPs dynamics, i.e. $s_0 \sim \mu_0$ and $s_{t+1} \sim P(\cdot|s_t,a_t)$.

In all levels, the teacher observes the current Q-table Q_t and knows the learning algorithm $Q_{t+1} = f(Q_t, e_t)$.

In this work, we are interested in analyzing the **teaching dimension**, a quantity of interest in the learning theory literature. We define an RL teaching problem instance by the MDP environment M, the student L with initial Q-table Q_0, and the teacher's target policy $\pi^\dagger$. We remark that the target policy $\pi^\dagger$ need not coincide with the optimal policy π^* for M. In any case, the teacher wants to control the experience sequence so that the student arrives at $\pi^\dagger$ quickly. Specifically,

Definition 3.3. *Given an RL teaching problem instance* $(M, L, Q_0, \pi^\dagger)$*, the* ***minimum expected teaching length*** *is* $\mathrm{METaL}(M, L, Q_0, \pi^\dagger) = \min_{T,(s_t,a_t,r_t,s_{t+1})_{0:T-1}} \mathbb{E}[T]$*, s.t.* $\pi_T = \pi^\dagger$*, where the expectation is taken over the randomness in the MDP (transition dynamics) and the learner (stochastic behavior policy).*

METal depends on nuisance parameters of the RL teaching problem instance. For example, if Q_0 is an initial Q-table that already induces the target policy $\pi^\dagger$, then trivially METal=0. Following the classic definition of teaching dimension for supervised learning, we define TDim by the hardest problem instance in an appropriate family of RL teaching problems:

Definition 3.4. *The* ***teaching dimension*** *of an RL learner* L *w.r.t. a family of MDPs* $\mathcal{M}$ *is defined as the worst-case METal:* $\mathtt{TDim} = \max_{\pi^\dagger \in \{\pi: \mathcal{S} \to \mathcal{A}\}, Q_0 \in \mathbb{R}^{\mathcal{S} \times \mathcal{A}}, M \in \mathcal{M}} \mathrm{METaL}(M, L, \pi^\dagger)$.

Remark 3.5. *In the previous chapter, we observed that minimizing instance dependent expected time even for behavior cloning learner is a very challenging problem. So, here we instead consider a 'worst case' notion of teaching dimension characterized by the worst instance in the instance family and study algorithms to solve instead.*

3.4 Teaching without MDP Constraints

We start our discussion with the strongest teachers. These teachers have the power of producing arbitrary state transition experiences that do not need to obey the transition dynamics of the underlying MDP. While the assumption on the teaching power may be unrealistic in some cases, the analysis that we present here provides theoretical insights

that will facilitate our analysis of the more realistic/less powerful teaching settings in the next section.

3.4.1 Level 1: Teacher with Full Control

The level 1 teacher is the most powerful teacher we consider. In this setting, the teacher can generate arbitrary experience e_t. The learner effectively becomes a "puppet" learner - one who passively accepts any experiences handed down by the teacher.

Theorem 3.6. *For a Level 1 Teacher, any learner* $L \in \mathcal{L}$*, and an MDP family* $\mathcal{M}$ *with* $|\mathcal{S}| = S$ *and a finite action space, the teaching dimension is* $\mathtt{TDim} = S$.

It is useful to illustrate the theorem with the standard Q-learning algorithm, which is a member of $\mathcal{L}$. The worst case happens when $\arg\max_a Q_0(s,a) \neq \pi^\dagger(s), \forall s$. The teacher can simply choose one un-taught s at each step, and construct the experience $(s_t = s, a_t = \pi^\dagger(s), r_t, s_{t+1} = s')$ where s' is another un-taught state (the end case is handled in the algorithm in appendix). Importantly, the teacher chooses

$$r_t \in \left\{ \frac{\max Q_t(s_t,\cdot) + \theta - (1-\alpha)Q_t(s_t,a_t)}{\alpha} - \gamma \max Q_t(s',\cdot) : \theta > 0 \right\},$$

knowing that the standard Q-learning update rule f is $Q_{t+1}(s_t,a_t) = (1-\alpha)Q_t(s_t,a_t) + \alpha(r_t + \gamma \max_{a\in A} Q_t(s',a))$. This ensures that $Q_{t+1}(s,\pi^\dagger(s)) = \max_{a\neq\pi^\dagger(s)} Q_0(s,a) + \theta > \max_{a\neq\pi^\dagger(s)} Q_0(s,a)$, and thus the target policy is realized at state s. Subsequent teaching steps will not change the action ranking at state s. The same teaching principle applies to other learners in $\mathcal{L}$.

3.4.2 Level 2: Teacher with State and Reward Control

At level 2 the teacher can still generate arbitrary reward r_t and next state s_{t+1}, but now it cannot override the action a_t chosen by the learner. This immediately implies that the teacher can no longer teach the desired action $\pi^\dagger(s)$ in a single visit to s: for example, Q_0 may be such that $Q_0(s,\pi^\dagger(s))$ is ranked last among all actions. If the learner is always greedy with $\varepsilon = 0$ in (1), the teacher will need to visit s for $(A-1)$ times, each time generating a punishing r_t to convince the learner that the top non-target action is worse than $\pi^\dagger(s)$. However, for a learner who randomly explores with $\varepsilon > 0$ it may perform $\pi^\dagger(s)$ just by

chance, and the teacher can immediately generate an overwhelmingly large reward to promote this target action to complete teaching at s; it is also possible that the learner performs a non-target action that has already been demoted and thus wasting the step. Despite the randomness, interestingly our next lemma shows that for any ε it still takes in expectation $A-1$ visits to a state s to teach a desired action in the worst case.

Lemma 3.7. *For a Level 2 Teacher, any learner in $\mathcal{L}$, and an MDP family $\mathcal{M}$ with action space size A, it takes at most $A-1$ visits in expectation to a state s to teach the desired action $\pi^\dagger(s)$ on s.*

Proof Sketch: Let us consider teaching the target action $\pi^\dagger(s)$ for a particular state s. Consider a general case where there are $A-c$ actions above $\pi^\dagger(s)$ in the current ordering $Q_t(s,\cdot)$. In the worst case $c=1$. We define the function $T(x)$ as the expected number of visits to s to teach the target action $\pi^\dagger(s)$ to the learner when there are x higher-ranked actions. For any learner in $\mathcal{L}$, the teacher can always provide a suitable reward to either move the action selected by the learner to the top of the ordering or the bottom. Using dynamic programming we can recursively express $T(A-c)$ as

$$T(A-c) = 1 + (c-1)\frac{\varepsilon}{A-1}T(A-c) + \\ (1-\varepsilon+(A-c-1)\frac{\varepsilon}{A-1})T(A-c-1).$$

Solving it gives $T(A-c) = \frac{A-c}{(1-(c-1)\frac{\varepsilon}{A-1})}$, which implies $\max_c T(A-c) = T(A-1) = A-1$. □Lemma 3.7 suggests that the agent now needs to visit each state at most $(A-1)$ times to learn the target action, and thus teaching the target action on all states needs at most $S(A-1)$ steps:

Theorem 3.8. *For a Level 2 Teacher, any learner in $\mathcal{L}$, and an MDP family $\mathcal{M}$ with state space size S and action space size A, the teaching dimension is $\mathrm{TDim} = S(A-1)$.*

We present a concrete level-2 teaching algorithm in the appendix. For both Level 1 and Level 2 teachers, we can calculate the exact teaching dimension due to a lack of constraints from the MDP. The next levels are more challenging, and we will be content with big O notation.

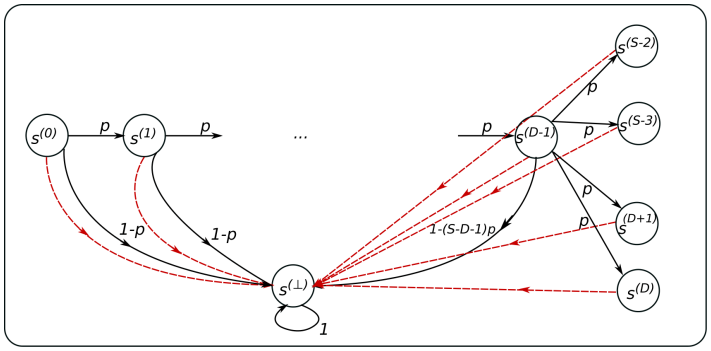


Figure 3.1: The “peacock" MDP for establishing lower bound on the Teaching Dimension.

3.5 Teaching subject to MDP Constraints

In this section, we study the TDim of RL under the more realistic setting where the teacher must obey some notion of MDP transitions. In practice, such constraints may be unavoidable. For example, if the transition dynamics represent physical rules in the real world, the teacher may be physically unable to generate arbitrary s_{t+1} given s_t, a_t (e.g. cannot teleport).

3.5.1 Level 3: Teacher with Control on Reward and Support States

In Level 3, the teacher can only generate a state transition to s_{t+1} which is in the support of the appropriate MDP transition probability, i.e. $s_{t+1} \in \{s : P(s \mid s_t, a_t) > 0\}$. However, the teacher can freely choose s_{t+1} within this set regardless of how small $P(s_{t+1} \mid s_t, a_t)$ is, as long as it is nonzero. Different from the previous result for Level 1 and Level 2 teacher, in this case, we are no longer able to compute the exact TDim of RL. Instead, we provide matching lower and upper-bounds on TDim.

Theorem 3.9. *For Level 3 Teacher, any learner in $\mathcal{L}$ with ε probability of choosing non-greedy actions at random, an MDP family $\mathcal{M}$ with episode length H and diameter $D \leqslant H$, the teaching dimension is lower-bounded by*

$$\mathrm{TDim} \geqslant \Omega\left((S-D)AH\left(\frac{1}{1-\varepsilon}\right)^{D}\right). \tag{3.2}$$

proof. The proof uses a particularly hard RL teaching problem instance called the “peacock MDP” in Figure 3.1 to produce a tight lower bound. The MDP has S states where

the first D states form a linear chain (the "neck"), the next $S-D-1$ states form a star (the "tail"), and the last state $s^{(\perp)}$ is a special absorbing state. The absorbing state can only be escaped when the agent resets after episode length H. The agent starts at $s^{(0)}$ after reset. It is easy to verify that the peacock MDP has a diameter D. Each state has A actions. For states along the neck, the a_1 action (in black) has probability $p > 0$ of moving right, and probability $1-p$ to go to the absorbing state $s^{(\perp)}$; all other actions (in red) have probability 1 of going to $s^{(\perp)}$. The a_1 action of $s^{(D-1)}$ has probability p to transit to each of the tail states. In the tail states, however, all actions lead to the absorbing state with probability 1. We consider a target policy $\pi^\dagger$ where $\pi^\dagger(s)$ is a red action a_2 for all the tail states s. It does not matter what $\pi^\dagger$ specifies on other states. We define Q_0 such that a_2 is $\arg\min_a Q_0(s,a)$ for all the tail states.

The proof idea has three steps: (1) By Lemma 3.7 the agent must visit each tail node s for $A-1$ times to teach the target action a_2, which was initially at the bottom of $Q_0(s,\cdot)$. (2) But the only way that the agent can visit a tail state s is to traverse the neck every time. (3) The neck is difficult to traverse as any ε-exploration sends the agent to $s^{(\perp)}$ where it has to wait for the episode to end.

We show that the expected number of steps to traverse the neck once is $H(\frac{1}{1-\varepsilon})^D$ even in the best case, where the agent's behavior policy (1) prefers a_1 at all neck states. In this best case, the agent will choose a_1 with probability $1-\varepsilon$ at each neck state s. If a_1 is indeed chosen by the agent, by construction the support of MDP transition $P(\cdot \mid s, a_1)$ contains the state to the right of s or the desired tail state (via the transition with probability $p > 0$). This enables the level 3 teacher to generate such a transition regardless of how small p is (which is irrelevant to a level 3 teacher). In other words, in the best case, the agent can move to the right once with probability $1-\varepsilon$. A successful traversal requires moving right D times consecutively, which has probability $(1-\varepsilon)^D$. The expected number of trials (to traverse) until success is $(\frac{1}{1-\varepsilon})^D$. A trial fails if any time during a traversal the agent picked an exploration action a other than a_1. Then the support of $P(\cdot \mid s, a)$ only contains the absorbing state $s^{(\perp)}$, so the teacher has no choice but to send the agent to $s^{(\perp)}$. There the agent must wait for the episode to complete until resetting back to $s^{(0)}$. Therefore, any failed trial incurs exactly H steps of wasted teaching. Putting things together, the expected number of teaching steps until a successful neck traversal is done is at least $H(\frac{1}{1-\varepsilon})^D$.

There are $S-D-1$ tail states. Each needs an expected $A-1$ neck traversals to teach. This leads to the lower bound $(S-D-1)(A-1)H(\frac{1}{1-\varepsilon})^D = \Omega\left((S-D)AH\left(\frac{1}{1-\varepsilon}\right)^D\right)$. □

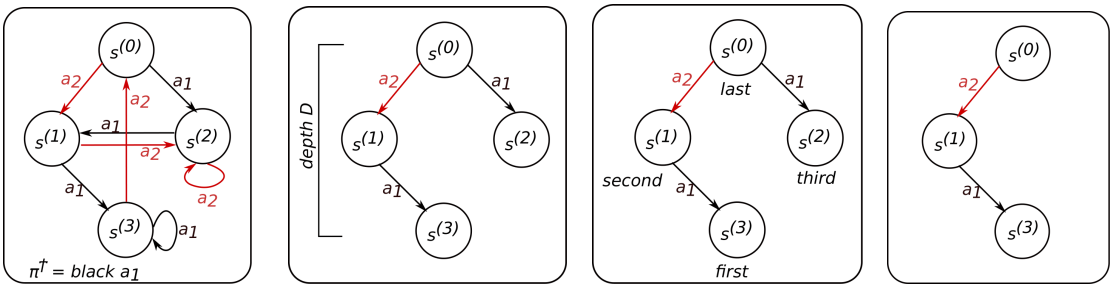
Our next result shows that this lower bound is nearly tight, by constructing a level-3 teaching algorithm that can teach any MDP with almost the same sample complexity as above.

Theorem 3.10. *Under the same conditions of Theorem 3.9, the level-3 teaching dimension is upper-bounded by*

$$\mathsf{TDim} \leqslant O\left(SAH\left(\frac{1}{1-\varepsilon}\right)^{D}\right). \tag{3.3}$$

proof. We analyze a level-3 teaching algorithm NavTeach (Navigation-then-Teach) which, like any teaching algorithm, provides an upper bound on TDim. The complete NavTeach algorithm is given in the appendix; we walk through the main steps on an example MDP in Figure 3.2(a). For the clarity of illustration the example MDP has only two actions a_1, a_2 and deterministic transitions (black and red for the two actions respectively), though NavTeach can handle fully general MDPs. The initial state is $s^{(0)}$.

Let us say NavTeach needs to teach the "always take action a_1" target policy: $\forall s, \pi^\dagger(s) = a_1$. In our example, these black transition edges happen to form a tour over all states, but the path length is 3 while one can verify the diameter of the MDP is only $D = 2$. In general, though, a target policy $\pi^\dagger$ will not be a tour. It can be impossible or inefficient for the teacher to directly teach $\pi^\dagger$. Instead, NavTeach splits the teaching of $\pi^\dagger$ into subtasks for one "target state" s at a time over the state space in a carefully chosen order. Importantly, before teaching each $\pi^\dagger(s)$ NavTeach will teach a different navigation policy π^{nav} for that s. The navigation policy π^{nav} is a partial policy that creates a directed path from $s^{(0)}$ to s, which is similar to the neck in the earlier peacock example. The goal of π^{nav} is to quickly bring the agent to s often enough so that the *target* policy $\pi^\dagger(s) = a_1$ can be taught at s. That completes the subtask at s. Critically, NavTeach can maintain this target policy at s forever, while moving on to teach the next target state s'. This is nontrivial because NavTeach may need to establish a different navigation policy for s': the old navigation policy may be partially reused, or demolished. Furthermore, all these need to be done in a small number of steps. We now go through NavTeach on Figure 3.2(a). The first thing NavTeach does is to carefully plan the subtasks. The key is to make sure that (i) each navigation path is at most D long; (ii) once a target state s has been taught: $\pi^\dagger(s) = a_1$, it does not interfere with later navigation. To do so, NavTeach first constructs a directed graph where the vertices are the MDP states, and the edges are non-zero probability transitions of all actions. This is



(a) MDP Instance **(b)** Breadth-First Tree **(c)** Depth-First Traversal **(d)** Navigation $\pi^\dagger$

Figure 3.2: NavTeach algorithm demo on a simple example.

the directed graph of Figure 3.2(a), disregarding color. NavTeach then constructs a breadth-first-tree over the graph, rooted at $s^{(0)}$. This is shown in Figure 3.2(b). Breadth-first search ensures that all states are at most depth D away from the root. Note that this tree may uses edges that correspond to non-target actions, for example the red a_2 edge from $s^{(0)}$ to $s^{(1)}$. The ancestral paths from the root in the tree will form the navigation policy π^{nav} for each corresponding node s. Next, NavTeach orders the states to form subtasks. This is done with a depth-first traversal on the tree: a depth-first search is performed, and the nodes are ranked by the last time they are visited. This produces the order in Figure 3.2(c). The order ensures that later navigation is "above" any nodes on which we already taught the target policy, thus avoiding interference.

Now NavTeach starts the first subtask of teaching $\pi^\dagger(s^{(3)}) = a_1$, i.e. the black self-loop at $s^{(3)}$. As mentioned before, NavTech begins by teaching the navigation policy π^{nav} for this subtask, which is the ancestral path of $s^{(3)}$ shown in Figure 3.2(d). How many teaching steps does it take to establish this π^{nav}? Let us look at the nodes along the ancestral path. By Lemma 3.7 the agent needs to be at the root $s^{(0)}$ $A-1$ times in expectation in order for the teacher to teach $\pi^{nav}(s^{(0)}) = a_2$; this is under the worst case scenario where the initial agent state Q_0 places a_2 at the bottom in state $s^{(0)}$. We will assume that after a visit to $s^{(0)}$, the remaining episode is simply wasted. [1] Therefore it takes at most $H(A-1)$ teaching steps to establish $\pi^{nav}(s^{(0)}) = a_2$. After that, it takes at most $H(A-1)(\frac{1}{1-\varepsilon})$ expected

[1] It is important to note that the teacher always has a choice of r_t so that the teaching experience does not change the agent's Q_t state. For example, if the agent's learning algorithm f is a standard Q-update, then there is an r_t that keeps the Q-table unchanged. So while in wasted steps the agent may be traversing the MDP randomly, the teacher can make these steps "no-op" to ensure that they do not damage any already taught subtasks or the current navigation policy.

number of teaching steps to teach $\pi^{nav}(s^{(1)}) = a_1$. This is the same argument we used in Theorem 3.9: the teacher needs to make the agent traverse the partially-constructed ancestral path ("neck") to arrive at $s^{(1)}$. The worst case is if the agent performs a random exploration action anywhere along the neck; it falls off the neck and wastes the full episode. In general to establish a nagivation policy π^{nav} with path length d, NavTeach needs to teach each navigation edge at depth $i = 1 \dots d$ with at most $H(A-1)(\frac{1}{1-\varepsilon})^{i-1}$ teaching steps, respectively. After establishing this π^{nav} for $s^{(3)}$, NavTeach needs to go down the neck frequently to ensure that it visits $s^{(3)}$ $(A-1)$ times and actually teach the target policy $\pi^{\dagger}(s^{(3)}) = a_1$. This takes an additional at most $H(A-1)(\frac{1}{1-\varepsilon})^{d}$ teaching steps.

When the $s^{(3)}$ subtask is done, according to our ordering in Figure 3.2(c) NavTeach will tackle the subtask of teaching $\pi^{\dagger}$ at $s^{(1)}$. Our example is lucky because this new subtask is already done as part of the previous navigation policy. The third subtask is for $s^{(2)}$, where NavTeach will have to establish a new navigation policy, namely $\pi^{nav}(s^{(0)}) = a_1$. And so on. How many total teaching steps are needed? **A key insight is NavTeach only needs to teach any navigation edge in the breadth-first tree exactly once.** This is a direct consequence of the depth-first ordering: there can be a lot of sharing among navigation policies; a new navigation policy can often re-use most of the ancestral path from the previous navigation policy. Because there are exactly $S-1$ edges in the breadth-first tree of S nodes, the total teaching steps spent on building navigation policies is the sum of $S-1$ terms of the form $H(A-1)(\frac{1}{1-\varepsilon})^{i-1}$ where i is the depth of those navigation edges. We can upperbound the sum simply as $(S-1)H(A-1)(\frac{1}{1-\varepsilon})^{D}$. On the other hand, the total teaching steps spent on building the target policy $\pi^{\dagger}$ at all target states is the sum of S terms of the form $H(A-1)(\frac{1}{1-\varepsilon})^{d}$ where d is the depth of the target state. We can upperbound the sum similarly as $SH(A-1)(\frac{1}{1-\varepsilon})^{D}$. Putting navigation teaching and target policy teaching together, we need at most $(2S-1)H(A-1)(\frac{1}{1-\varepsilon})^{D} = O\left(SAH\left(\frac{1}{1-\varepsilon}\right)^{D}\right)$ teaching steps. □

We remark that more careful analysis can in fact provide matching lower and upper bounds up to a constant factor, in the form of $\Theta\left((S-D)AH(1-\varepsilon)^{-D} + H\frac{1-\varepsilon}{\varepsilon}[(1-\varepsilon)^{-D}-1]\right)$. We omit this analysis for the sake of a cleaner presentation. However, the matching bounds imply that a deterministic learner, with $\varepsilon = 0$ in the ε-greedy behavior policy, has the smallest teaching dimension. This observation aligns with the common knowledge in the standard RL setting that algorithms exploring with stochastic behavior policies are provably sample-inefficient [52].

Corollary 3.11. *For Level 3 Teacher, any learner in $\mathcal{L}$ with $\varepsilon = 0$, and any MDP M within the MDP family $\mathcal{M}$ with $|\mathcal{S}| = S$, $|\mathcal{A}| = A$, episode length H and diameter $D \leqslant H$, we have* $\mathtt{TDim} = \Theta(SAH)$.

3.5.2 Level 4: Teacher with Only Reward Control

In Level 4, the teacher no longer has control over state transitions. The next state will be sampled according to the transition dynamics of the underlying MDP, i.e. $s_{t+1} \sim P(\cdot|s_t, a_t)$. As a result, the only control power left for the teacher is the control of reward, coinciding with the reward shaping framework. Therefore, our results below can be viewed as a sample complexity analysis of RL under *optimal reward shaping*. Similar to Level 3, we provide near-matching lower and upper-bounds on TDim.

Theorem 3.12. *For Level 4 Teacher, and any learner in $\mathcal{L}$, and an MDP family $\mathcal{M}$ with $|\mathcal{S}| = S$, $|\mathcal{A}| = A \geqslant 2$, episode length H, diameter $D \leqslant H$ and minimum transition probability $p_{\min}$, the teaching dimension is lower-bounded by* $\mathtt{TDim} \geqslant \Omega\left((S-D)AH\left(\frac{1}{p_{\min}(1-\varepsilon)}\right)^D\right)$.

Theorem 3.13. *For Level 4 Teacher, any learner in $\mathcal{L}$, and any MDP M within the MDP family $\mathcal{M}$ with $|\mathcal{S}| = S$, $|\mathcal{A}| = A$, episode length H, diameter $D \leqslant H$ and minimum transition probability $p_{\min}$, the Nav-Teach algorithm in the appendix can teach any target policy $\pi^\dagger$ in a expected number of steps at most* $\mathtt{TDim} \leqslant O\left(SAH\left(\frac{1}{p_{\min}(1-\varepsilon)}\right)^D\right)$.

The proofs for Theorem 3.12 and 3.13 are similar to those for Theorem 3.9 and 3.10, with the only difference that under a level 4 teacher the expected time to traverse a length D path is at most $H(1/p_{\min}(1-\varepsilon))^D$ in the worst case. The $p_{\min}$ factor accounts for sampling from $P(\cdot \mid s_t, a_t)$. Similar to Level 3 teaching, we observe that a deterministic learner incurs the smallest TDim, but due to the stochastic transition, an exponential dependency on D is unavoidable in the worst case.

Corollary 3.14. *For Level 4 Teacher, any learner in $\mathcal{A}$ with $\varepsilon = 0$, and any MDP M within the MDP family $\mathcal{M}$ with $|\mathcal{S}| = S$, $|\mathcal{A}| = A$, episode length H, diameter $D \leqslant H$ and minimum transition probability $p_{\min}$, we have* $\mathtt{TDim} \leqslant O\left(SAH\left(\frac{1}{p_{\min}}\right)^D\right)$.

3.6 Sample efficiencies of standard RL, TbD and TbR

In the standard RL setting, some learners in the learner family $\mathcal{L}$, such as UCB-B, are provably efficient and can learn a δ-optimal policy in $O(H^3SA/\delta^2)$ iterations [38], where δ-optimal means that the cumulative rewards achieved by the output policy is only δ-worse than the optimal policy, i.e. $V^*(\mu_0) - V^\pi(\mu_0) \leqslant \delta$. One direct implication of such a measure is that the remote states that are unreachable also hardly affect the policy's performance, so quantities like the diameter of the MDP does not appear in the bound.

In contrast, in our TbR work, we aim at learning the *exact* optimal policy, and will thus suffer exponentially if some states are nearly unreachable. However, if we assume that all states have reasonable visiting probabilities, then even the weakest teacher (Level 3 and 4) can teach the optimal policy in $O(HSA)$ iterations, which is of H^2 factor better than the best achievable rate without a teacher. More interestingly, even the learners with a not as good learning algorithm, e.g. standard greedy Q-learning, which can never learn the optimal policy on their own, can now learn just as efficiently under the guidance of an optimal teacher.

Teaching-by-demonstration is the most sample efficient paradigm among the three, because the teacher can directly demonstrate the optimal behavior $\pi^\dagger(s)$ on any state s, and effectively eliminate the need for exploration and navigation. If the teacher can generate arbitrary (s, a) pairs, then he can teach any target policy with only S iterations, similar to our Level 1 teacher. If he is also constrained to obey the MDP, then it has been shown that he can teach a δ-optimal policy in $O(SH^2/\delta)$ iterations [72, 82], which completely drops the dependency on the action space size A compared to both RL and TbR paradigms. Intuitively, this is due to the teacher being able to directly demonstrate the optimal action, whereas, in both RL and TbR paradigms, the learner must try all actions before knowing which one is better. In summary, in terms of sample complexity, we have

$$\text{RL} > \text{TbR} > \text{TbD}. \tag{3.4}$$

Contribution Statement: This work has been done jointly with Xuezhou Zhang as leading author. Shubham formulated the problem statement, worked on theoretical results for level 1 and 2 teachers while Xuezhou led the efforts for level 3 and 4 teachers. This work was published at AAAI'21.

Chapter 4

Nurture-then-Nature Teaching

4.1 Introduction

The problem of designing an optimal dataset to teach a target concept $h^* : \mathcal{X} \to \mathcal{Y}$ to a learner, also known as Machine Teaching, has been a long-studied problem [30, 53, 94]. Prior works on Optimal Teaching have mainly focused on a single-phase learning setting where the student learner solely learns under the guidance of the teacher, who has an unlimited teaching budget [30, 49, 53, 94]. However, in many practical scenarios, the teacher may only have a limited budget of teaching lessons that it can provide to the learner. For example, consider a university curriculum setting where a teacher has to teach a concept, say how to identify a disease from an MRI scan to a student(see Figure 4.1) but it can only teach a limited number of lessons (dataset $D_T = \{(x_i, y_i)\}|_{i=1}^{n}$ s.t. $n \leqslant B$) to them before they graduate from the program. This first phase of learning, which takes place under the guidance of the teacher, is called the "Nurture" phase. Since *"Teaching Dimension"*(TD) [30] is the smallest possible dataset to teach a concept, the teacher will not be able to teach completely if the budget is less than *TD* [30].

However, from the student's perspective, learning does not stop after graduating from the university. Rather, they transition to a "Nature" learning phase and continue to learn about the target concept by receiving an i.i.d. dataset D_E drawn from the joint distribution of the nature/environment $P \in \Delta(\mathcal{X} \times \mathcal{Y})$. For example, in Figure 4.1, the student keeps learning about disease identification from i.i.d. MRI scans drawn from a digital library with the hope of mastering the concept over time. We call this two-phase learning setting

*generated by DALL-E for illustration.

Nurture Learning: under a Teacher. Nature Learning: from i.i.d. dataset.

Figure 4.1: An illustrative example of Nurture-then-Nature learning in the medical teaching domain. In the 'nurture' phase, the student learns to identify a brain disease from MRI scans under the guidance of a teacher. This is followed by the 'nature' phase, where the student continues learning using an i.i.d. sample from a digital database.

"Nurture-then-Nature"(NtN) learning. The goal of a good teacher is to design an "optimal" dataset D_T^* to minimize students' error at the end of the nature phase in NtN learning. To study this further, we ask the following question:

What is an optimal teaching demonstration to minimize the error in the NtN setting?

We study this problem and make the following contributions: 1.) We propose a novel mathematical framework of "Nurture-then-Nature" learning for studying budget-constrained teaching where the goal of the teacher is to minimize the final error of the student. 2.) We study the problem under two levels of knowledge by the teacher and propose teaching algorithms for each of them:

1. In **Instance Agnostic** setting, we consider a teacher who does not know the environment distribution P and is required to teach instances with any P. Our efficient teacher constructs an optimal teaching set to simplify the complexity of the learner's version space at the end of the 'Nature' phase, thereby making it easy for them to learn from i.i.d. sample in 'Nature' phase.

2. In **Instance Aware** setting, the teacher knows P and is required to be competitive with the instance optimal teaching set. We propose an algorithm using datamodels [37] to exactly solve this problem under the linear assumption. Unlike (a), this method works

for any learner that satisfies the linear risk assumption and extends to non-linear datamodels as well.

We present both theoretical and experimental results to validate the effectiveness of our algorithms in both settings and compare their performance to a simulated baseline algorithm.

4.2 Related Works

Machine Teaching has been a well-studied problem in the literature [30, 78, 97]. Past works have studied optimal teaching in various learning settings ranging from supervised learning [13, 30, 49, 53] to online/active learning [68, 94] to sequential decision making and reinforcement learning [15, 87, 92]. However, most of these works have focused on unconstrained teaching setting where the teacher is free to design and teach a dataset of any arbitrary size which may not be possible under real-world constraints. Our work studies budget-constrained teaching in a two-phase supervised learning setting where the teacher can only provide a dataset up to a fixed size.

Some recent works have considered other forms of constraints that are distinct from our budget constraints, like time constraint [27], preference constraint [87]. The most relevant work to ours is the budget-constrained teaching problem examined by [45]. However, the authors have only considered a single-phase teaching setting of [30] where the goal is to minimize the learner's error at the end of the teaching phase. Moreover, their algorithm and analysis is very specialized to distribution-independent teaching of a class of monomials [41]. On the other hand, our framework is much more general with a clearly different goal. Furthermore, our teaching algorithms can handle infinite hypothesis classes like linear/polynomial classifiers through VC reduction and linear datamodel connections.

Optimal teaching has been shown to be a hard bilevel optimization problem [30, 100], which limits its practical utility. The main difficulty often lies in estimating the risk of the learner as a function of the dataset. Naive methods require simulating a learner to estimate the risk on different independent datasets, making it a challenging task. However, recent works like linear datamodels [37] have taken a function approximation approach and have shown that risk can be well approximated by a linear function of the dataset in many real-world problems. Prior works have utilized this connection to detect backdoor attacks [42],

forget training data using machine unlearning [28], and to select good datasets for training large models [25], which aligns closely with the objective of a single-phase machine teaching setting, which is clearly different from our budget-constrained Nature-then-Nature teaching.

4.3 Problem Formulation

4.3.1 The Learner and The Environment

Consider a predictive modeling task from an input space $\mathcal{X}$ to an output space $\mathcal{Y}$ defined by a joint distribution P over $\mathcal{X} \times \mathcal{Y}$. During learning, a learner receives a dataset $D = \{(x_i, y_i)\}|_{i\in[n]} \subseteq (\mathcal{X} \times \mathcal{Y})^n$ and tries to learn a good predictive model that does well on future data from P.

An Empirical Risk Minimization (ERM) [62, 76] learner/student $\mathcal{A}$ starts with a hypothesis class $\mathcal{H}$ and minimizes empirical risk with respect to a loss $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geqslant 0}$ on the training dataset D,

$$\mathcal{A}(D;\mathcal{H}) = \arg\min_{h\in\mathcal{H}} \frac{1}{n} \sum_{(x_i,y_i)\in D} \ell(h(x_i), y_i) \tag{4.1}$$

where ℓ is the loss function. It eventually aims to learn a hypothesis with the smallest risk $R_P(h) = \mathbb{E}_{(x,y)\sim P}[\ell(h(x), y)]$. We make the following simplifying assumption on the realizability of the environment, which has been well used in literature [30, 53].

Definition 4.1 (Realizability & Version Space Learner)**.** *An environment is said to be realizable if* $\exists h^* \in \mathcal{H}$ *such that* $P = P_X \cdot P_{Y|X}$ *and* $P_{Y|X}(Y = h^*(X)) = 1$. *Under realizability, an ERM learner that minimizes the risk w.r.t.* $0-1$ *loss and maintains the entire subset of ERM hypotheses is called a version space(VS) learner.*

Remark 4.2. *We note that the output of learning* $\mathcal{A}(D;\mathcal{H})$ *can be a single hypothesis or a subset of them, depending on the learner.*

4.3.2 The Teacher

There is a helpful teacher who is required to teach a target hypothesis $h^* \in \mathcal{H}$ to the learner. The teacher knows h^* but can only provide a dataset of size up to budget $B \in \mathbb{Z}^+$ to the

learner before they graduate. The teacher will not be able to teach h^* completely if B is less than the TD. However, after graduating, the learner keeps learning about h^* using an i.i.d. sample from the environment P.

We consider two teaching settings based on different levels of knowledge of the teacher:

1. **Instance agnostic setting**: In this setting, the teacher does not know the underlying P_X and has to teach a learner in instance instance-agnostic way, i.e., the teaching should work for any P_X.

2. **Instance aware setting**: In this setting, the teacher knows the underlying distribution of the environment P_X and has to be competitive wrt to instance optimal solution.

Next, we define the interaction of the learner with the teacher and the environment.

4.3.3 The Nurture-then-Nature Setting

The learning process of the version space learner in the NtN setting is split into two phases:

Phase I - The Nurture Phase: In this phase, the learner learns under the guidance of the teacher. It receives a dataset D_T from the teacher and learns a version space of hypothesis $\mathcal{V}(D_T;\mathcal{H})$ consistent with D_T given as,

$$\mathcal{V}_1 := \mathcal{V}(D_T;\mathcal{H}) = \{h \in \mathcal{H} \mid h(x_i) = y_i, \forall (x_i, y_i) \in D_T\}. \tag{4.2}$$

Phase II - The Nature Phase: The nurture phase is followed by the *nature/i.i.d. learning phase* where the learner starts with the surviving version space $\mathcal{V}_1$ from previous phase and continues to learn about h^* by receiving a i.i.d. dataset $D_E \sim P^n$ of size n from the environment distribution P. It then learns a version space $\mathcal{V}(D_E;\mathcal{V}_1)$ consistent with D_E on $\mathcal{V}_1$, i.e.,

$$\mathcal{V}_2 := \mathcal{V}(D_E;\mathcal{V}_1) = \{h \in \mathcal{V}_1 : h(x_i) = y_i, \forall (x_i, y_i) \in D_E\}.$$

At the end of this phase, the learner hopes to have learned h^* with as small a risk as possible.

Remark 4.3. *We make the following remarks on the two phases:*

1. *With budget $B \geqslant TD$, this phase captures the standard unconstrained teaching problem [30]. However, with a budget $B < TD$, the teacher can only teach h^* partially to the learner, leading to a very different teaching problem.*

2. *We note that in the Nature phase, the learner learns using a hypothesis class $\mathcal{V}_1$ that has been simplified from $\mathcal{H}$ by the teaching set D_T provided by the teacher. In effect, the teacher controls the complexity of learning from Nature by simplifying $\mathcal{V}_1$ using D_T.*

4.4 Instance Agnostic Teaching Setting

In an instance agnostic setting, the teacher does not know the environment distribution P_X and aims to minimize a high probability instance agnostic objective defined as follows:

Teaching objective: Given an instance $(\mathcal{X}, \mathcal{Y}, P_X, h^*, \mathcal{H}, \delta, n, B)$, the instance agnostic teaching objective is defined as,

$$D_T^* \leftarrow \underset{\varepsilon, D_T : |D_T| \leqslant B}{\arg\min} \ \varepsilon$$
$$\text{s.t. } \forall P, \quad \mathbb{P}_{D_E \sim P^n} \left(\max_{h \in \mathcal{V}(D_E; \mathcal{V}(D_T; \mathcal{H}))} R_P(h) \leqslant \varepsilon \right) \geqslant 1 - \delta \tag{4.3}$$

Remark 4.4. *We make the following remarks: 1.) The teacher does not know P_X, they have to ensure thatthe learner succeeds in any P. 2.) The teacher influences learners' performance through a budget-constrained dataset $D_T : |D_T| \leqslant B$ that reduces $\mathcal{H}$ to $\mathcal{V}(D_T; \mathcal{H})$.*

Note that the feasibility constraint in 4.3 requires satisfying (n, δ) PAC guarantee for any P. Prior works in PAC-learning [34, 88] provide the following instance agnostic bound on error of the version space learner that helps to simplify the problem objective.

$$\textbf{(PAC-guarantee)} : \forall P, \forall n \in \mathbb{N}, \delta \in (0,1), \quad \text{if } D \overset{iid}{\sim} P^n,$$
$$w.p. \geqslant 1 - \delta, \quad \max_{h \in R(\mathcal{V}(D; \mathcal{H}))} R(h) \leqslant \varepsilon(n, \delta, \mathcal{H}) \tag{4.4}$$

where, $R(h)$ is the risk with respect to the $0-1$ loss. For a hypothesis class $\mathcal{H}$ and a fixed (n, δ), PAC-guarantee satisfies 4.4 with $\varepsilon(n, \delta, \mathcal{H}) = O(\frac{1}{n} \cdot (d(\mathcal{H}) + \log(\frac{1}{\delta})))$ where $d(\mathcal{H})$ is the VC dimension of $\mathcal{H}$. Later, [32] also proved that this guarantee is optimal with respect to $d(\mathcal{V}(\mathcal{H}))$.

Note that the feasibility constraints of Equation 4.3 are nothing but a PAC-guarantee with surviving version space $\mathcal{V}_1 = \mathcal{V}(D_T, \mathcal{H})$ as the hypothesis space. This reduces our

teaching objective in 4.3 to:

$$D_T^* \leftarrow \underset{D_T:|D_T|\leqslant B}{\arg\min} \frac{1}{n}\left(d(\mathcal{V}(D_T;\mathcal{H}))+\log\left(\frac{1}{\delta}\right)\right). \tag{4.5}$$

Since (n,δ) are fixed, we essentially need to minimize the VC of the version space $\mathcal{V}(D_T;\mathcal{H})$ maximally under the budget constraint B, leading to the following theorem on the teaching algorithm.

Theorem 4.5. *A teaching algorithm that optimally reduces the VC dimension of the version space $\mathcal{V}(D_T;\mathcal{H})$ surviving at the end of the Nurture phase solves the instance-agnostic NtN teaching problem optimally.*

Computing VC is tractable for hypothesis classes like axis-aligned rectangles, linear classifiers, and polynomial classifiers [62, 76]; however, in general, this is an NP-hard problem [59, 60, 77]. Since optimally reducing VC is at least as hard as computing it, we cannot hope to reduce the VC of general hypothesis classes efficiently. Instead, we focus on optimally reducing the VC dimension for tractable hypothesis classes under fa inite teaching budget.

We begin with one of the simplest hypothesis classes, a finite binary hypothesis class [30], and then extend our analysis to several other hypothesis classes.

4.4.1 Finite Binary Hypothesis Class.

A finite binary hypothesis class consists of a set of hypotheses, each mapping a finite input space $\mathcal{X}$ to binary labels $\{0,1\}$, i.e., $\mathcal{H}\subseteq 2^{\mathcal{X}}$. We know that computing VC of $\mathcal{H}$ takes $\Theta(n^{\log(n)})$ time [60, 66] and is NP-hard. This eventually makes optimizing for VC an NP-hard problem as well. Hence, we further upper-bound VC by the size of the hypothesis class and aim to minimize that instead.

Given a budget B, the teacher aims to find teaching set $D_T\subseteq\mathcal{X}, |D_T|\leqslant B$, that reduces the size of version space $\mathcal{V}(D_T;\mathcal{H})$ maximally as follows:

$$D_T^* \leftarrow \arg\min_{D_T:|D_T|\leqslant B} |\mathcal{V}(D_T;\mathcal{H})|. \tag{4.6}$$

It turns out that even this problem is NP-hard since it's equivalent to another NP-hard problem called *Budgeted Maximum Coverage Problem* [43]. However, there exists an efficient algorithm to solve this problem approximately, leading to the following theorem.

Theorem 4.6. *There exists an efficient algorithm that reduces the version space size of a finite hypothesis class up to an approximation ratio of* $1 - \frac{1}{e}$.

The algorithm and the proof of the theorem can be found in the appendix. Next, we study another classic hypothesis class considered in literature, the axis-aligned rectangle hypothesis class [30].

4.4.2 Axis-aligned Rectangles on $\mathbb{Z}^2$ grid

This class consists of all axis-aligned rectangles in $\mathbb{Z}^2$ space. A hypothesis $h \in \mathcal{H}$ is defined by the two opposite corners $(x_{min}, y_{min}), (x_{max}, y_{max}) \in \mathbb{Z}^2$ and it produces the following classifier:

$$h((x,y)) = 2 \cdot \mathbb{1}[x_{min} \leqslant x \leqslant x_{max} \wedge y_{min} \leqslant y \leqslant y_{max}] - 1.$$

We recall that VC of this class $d_{VC}(\mathcal{H}) = 4$ [62], and, the TD for teaching any $h \in \mathcal{H}$ is 6 [30]. For our NtN setting, we focus on non-trivial cases with $B < TD$.

Theorem 4.7 (Optimal VC reduction for axis-aligned rectangles.)**.** *The VC dimension of axis-aligned rectangles in* $\mathbb{Z}^2$ *can be optimally reduced as follows:*

Budget B	1	2	3	4	5	$\geqslant 6$
min VC	4	3	2	2	1	0

Table 4.1: Minimum VC achievable by B-budgeted teaching on axis-aligned rectangle class in $\mathbb{Z}^2$.

We refer the readers to the appendix for a complete proof. Next, we consider two popular hypothesis classes, a homogeneous linear and a polynomial class.

4.4.3 Linear Hypothesis Classifiers in $\mathbb{R}^d$

Consider teaching a family of homogeneous linear binary classifiers in $\mathcal{H} = \mathbb{R}^d$. Given a $w \in \mathbb{R}^d$, it induces a linear classifier of form,

$$h_w(x) = 2 \cdot \mathbb{1}[w^\top x \geqslant 0] - 1.$$

Prior works have studied optimal teaching of linear decision boundaries in an unconstrained setting and have shown that $TD = d + 1$ for the perceptron learner [49] and $TD = 2$ for the max-margin learner [53]. We also know that the VC-dimension of linear class $\mathcal{H}$ is d [62] and address the following question:

"How to optimally reduce the VC-dimension $\mathcal{V}(D_T; \mathcal{H})$ using a constrained teaching set $|D_T| \leqslant B$?".

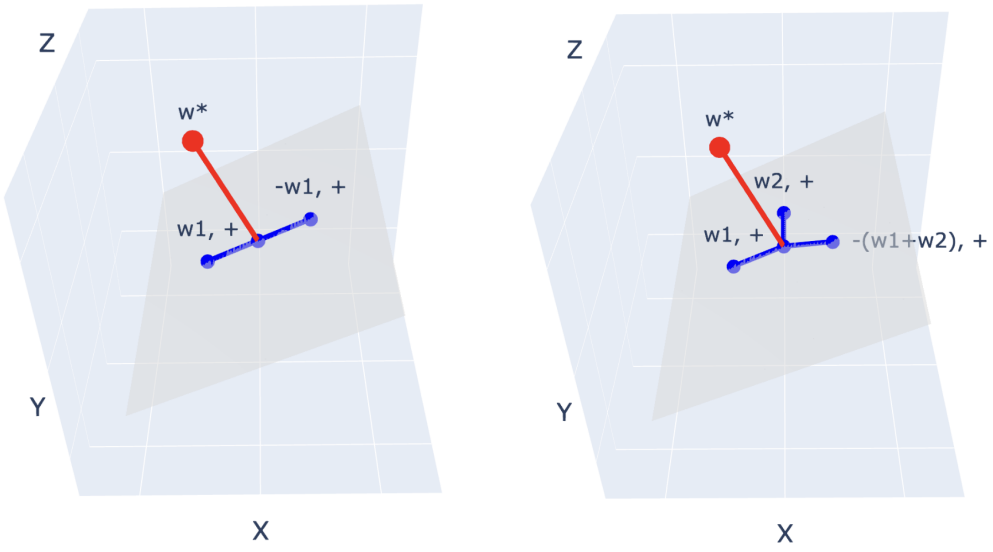


Figure 4.2: Teaching dataset produced by our optimal VC reduction algorithm for teaching $w^* \in \mathbb{R}^3$ with $B = 2$ and $B = 3$ kills a one and two-dimensional subspace of $(w^*)^\perp$, respectively.

To do so, we characterize the version space in terms of a polyhedral cone and prove that minimizing VC eventually requires reducing the ambient dimensionality of the version space as stated in Lemma C.7,C.8 of the appendix. We then provide an algorithm to compute the optimal dataset that essentially works by killing the $B - 1$ orthogonal subspace of w^* on a budget B, leading to the theorem stated next.

Theorem 4.8. *There exists an algorithm that* $\forall B \leqslant d+1$*, optimally reduces VC of the linear class to* $d-B+1$ *and optimal teaching set is given as* $D_T^B = \{(v_1,+1),\ldots,(v_{B-1},+1),(-\sum_{i\in[B-1]} v_i,+1)\}$*, where,* $\{v_1,\ldots,v_{B-1}\}$ *is a* B*-basis of* $w^{*\perp}$ *subspace.*

Our analysis involves a novel way of characterizing VC of linear version spaces, and we utilize it to provide guarantees on optimally reducing VC using a teaching dataset. The complete proof is deferred to the appendix. Next, we consider a polynomial hypothesis class in $\mathbb{R}^d$.

4.4.4 Polynomial Hypothesis Classifiers in $\mathbb{R}^d$

Let $\mathcal{H}$ be hypothesis class of k-degree polynomial classifiers in $\mathbb{R}^d$, given by,

$$\mathcal{H} = \{h \mid \ h(\mathbf{x}) = \mathbb{1}[\sum_{|\alpha|\leqslant k} w_\alpha \mathbf{x}^\alpha \geqslant 0],\ \alpha \in \mathbb{N}^d\}$$

and let $\phi : \mathbb{R}^d \to \mathbb{R}^l$ denote the feature mapping for the corresponding Kernel Hilbert space. We know that $l = \binom{d+k-1}{k}$ and the bases feature functions is given by,

$$\mathcal{B} = \left\{ \mathbf{x}^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \ldots x_d^{\alpha_d} \;\middle|\; |\alpha| \leqslant k, \sum_j \alpha_j = k,\ \alpha \in \mathbb{N}^d \right\}.$$

Furthermore, any hypothesis $h \in \mathcal{H}$ can be represented by a parameter $w^* \in \mathbb{R}^l$ in the basis of the Hilbert space.

The polynomial classifier is linear in the ϕ feature space, and the teacher aims to minimize the VC of the corresponding version space. However, unlike the linear model, each of the teaching input vectors in feature space must be realizable under the feature function ϕ on some $x \in \mathbb{R}^d$.

Assuming the feature function is rich, i.e., the preimages of feature vectors exist, the optimal reduction in VC of polynomial classifiers is given by the following theorem,

Theorem 4.9. *For any target polynomial* $h^* \in \mathcal{H}$*, the optimal teaching set that reduces the VC dimension of the polynomial version space by* $B-1$ *is given as* $D_T^B = \{(x_i,+1) : \phi(x_i) \perp w^*, i \in [B-1], \forall i \neq j, \phi(x_i) \perp \phi(x_j)\} \cup \{(\phi(x_B) = -\sum_{i=1}^{B-1} \phi(x_i),+1)\}$.

We defer the proof to the appendix. Note that the above algorithm relies on computing preimage feature vectors, and we also propose an algorithm to compute them in the appendix.

4.5 Instance Aware Teaching Setting

In this section, we will study the instance-aware setting where the teacher knows the environment distribution $P \in \Delta(\mathcal{X} \times \mathcal{Y})$ and is required to construct a teaching set competitive w.r.t. to the instance optimal solution.

We first study this problem on classical finite version space learners and provide a provably efficient approximation algorithm and a mixed integer non-linear program optimization formulation to solve this problem.

4.5.1 Teaching Finite Hypothesis Class

In a finite hypothesis setting, we have a domain $\mathcal{X} = [m]$, a hypothesis class $\mathcal{H} = \subseteq 2^{\mathcal{X}}$ s.t. $|\mathcal{H}| = n$ and a target hypothesis $h^* \in \mathcal{H}$. The environment distribution is defined by a join distribution $P_{\mathcal{X},\mathcal{Y}} = P_{\mathcal{X}} \cdot P_{\mathcal{Y}|\mathcal{X}}$.

Assumption 4.10 (Realizability)**.** *There exists a realizable distribution* $P_{\mathcal{X},\mathcal{Y}}$*, s.t.,*

$$P_{\mathcal{X},\mathcal{Y}}(x, y) = P_{\mathcal{X}}(x) \cdot \mathbb{1}[y = h^*(x)]$$

We assume that the environment is realizable w.r.t. h^* *and denote the joint distribution as* $P_{\mathcal{X},h^*}$*.*

The P_X induces a subset of hypotheses that is equivalent to h^* in terms of risk,

$$\{h^*\}_{P_{\mathcal{X}}} := \{h \in \mathcal{H} : R_{P_{\mathcal{X},h^*}}(h) = 0\} = \{h \in \mathcal{H} : \sum_{x:h(x)\neq h^*(x)} p_x = 0\}. \tag{4.7}$$

From the learner's point of view, learning is complete if it learns any non-empty subset of $\{h^*\}_{P_{\mathcal{X}}}$.

Teaching Objective: The goal of the teacher is to design an optimal teaching dataset that minimizes the expected nature time of the learner, stated as follows:

$$\begin{aligned} D_T^* \leftarrow \min_{D_T} \quad & \mathbb{E}_{D_E \sim P_{\mathcal{X},h^*}^{\infty}}[\min\{t \geqslant 0 : \phi \subsetneq \mathcal{V}(D_T \cup D_E^{(t)}) \subseteq \{h^*\}_{P_{\mathcal{X}}}\}] \\ \text{subject to} \quad & |D_T| \leqslant B \end{aligned} \tag{4.8}$$

where $D_E = \{(x_j, y_j)\}|_{j=1}^{\infty}$ denotes a infinite sequence of sample drawn from $P_{\mathcal{X},h^*}$ in the nature phase and $D_E^{(t)} = \{(x_j, y_j)\}|_{j=1}^{t}$.

Recall that teaching $(x, h^*(x))$ to the learner eliminates/covers a subset of hypotheses,

$$W_x = \{h \in \mathcal{H} : h(x) \neq h^*(x)\} \subseteq \mathcal{H}.$$

We denote the collection of these subsets as $W = \{W_x : x \in \mathcal{X}\}$. To completely learn h^* (up to an equivalence class of zero risk), the learner should be able to eliminate the universe of all inconsistent hypotheses, $U = \mathcal{H} \backslash \{h^*\}_{P_{\mathcal{X}}}$ by the end of the nature phase.

Given a nurture teaching dataset D_T, the set of (bad) inconsistent hypothesis surviving after being taught by D_T at the end of nurture phase is given as,

$$J(D_T) := \mathcal{V}(D_T) \backslash \{h^*\}_{P_{\mathcal{X}}} = \{h \in \mathcal{H} \backslash \{h^*\}_{P_{\mathcal{X}}} : \forall (x, h^*(x)) \in D_T, h(x) = h^*(x)\}.$$

$J(D_T)$ is also the hypothesis set that has to be eliminated in the nature phase using an i.i.d. dataset D_E so that the learner succeeds in learning h^*. The corresponding elimination time by a nature dataset D_E is expressed as,

$$T(D_E, J(D_T)) = \min\{t \geqslant 0 : \cup_{j \leqslant t} W_{x_j} \supseteq J(D_T)\}.$$

Expected Time Objective: The objective of minimizing the expected nature time through teaching in the nurture phase is restated as follows:

$$\begin{aligned} D_T(A^*; B) \leftarrow \min_{D_T \subseteq (\mathcal{X} \times \mathcal{Y})^*} & \underbrace{\mathbb{E}_{D_E \sim P_{\mathcal{X},h^*}^{\infty}}[T(D_E, J(D_T))]}_{f(D_T)} \\ \text{subject to } & |D_T| \leqslant B \end{aligned} \tag{4.9}$$

We want an algorithm A^* that minimizes this exact expected time. Next, we characterize this expected time metric and prove that it is EXP-TIME to evaluate it.

Lemma 4.11 (Expected Time to Cover Inconsistent Version Space)**.** *The expected time to cover* $J(D_T) = \mathcal{V}(D_T) \backslash \{h^*\}_{P_X}$ *using a nature sample, is given as,*

$$f(D_T) = \sum_{\emptyset \neq I \subseteq J(D_T)} (-1)^{|I|-1} \frac{1}{q_I},$$

where q_I *is the probability that a single random draw* $W_x \sim W$ *contains at least one element in the set* I*, meaning* $W_x \cap I \neq \emptyset$*, and* $q_I = \sum_{x : W_x \cap I \neq \emptyset} p_x$.

Theorem 4.12 (Hardness of Expected Time)**.** *Computing and hence optimizing expected time in the nature phase for a teaching dataset* D_T *takes exponential time and hence is inefficient.*

We note that one needs to enumerate over all possible subsets of inconsistent hypotheses in $J(D_T)$, thereby requiring an exponential time. Trying to optimize this objective would hence be an even harder problem.

To avoid this bottleneck, we bound the expected time by a closely matching lower and upper bound as below, and then minimize the upper bound to find a 'good' teaching dataset.

Lemma 4.13 (Bound on Expected Time)**.** *The expected time objective* $f(D_T)$ *is bounded as,*

$$\frac{1}{\min_{h \in J(D_T)} p_h} \leqslant f(D_T) \leqslant \frac{\ln(|J(D_T)|) + 1}{\min_{h \in J(D_T)} p_h}.$$

4.5.1.1 Optimizing Relaxed Upper Bound

For simplicity, we will first relax the upper bound further and provide an efficient algorithm to optimize that relaxed upper bound that achieves a bi-criteria approximation guarantee on the exact expected time objective. The relaxed upper bound in given as,

$$f(D_T) \leqslant \frac{\ln(|J(D_T)|) + 1}{\min_{h \in J(D_T)} p_h} \leqslant \frac{\ln(|\mathcal{H}|) + 1}{\min_{h \in J(D_T)} p_h}. \tag{4.10}$$

Since $\mathcal{H}$ in Equation 4.10 is fixed, the optimization problem with relaxed upper bound metric is stated as follows:

$$\begin{aligned} D_T^{(RU)} &\leftarrow \underset{D_T:|D_T|\leqslant B}{\arg\min} \frac{\ln(|\mathcal{H}|)}{\min_{h\in J(D_T)} p_h} \\ &\equiv \underset{D_T:|D_T|\leqslant B}{\arg\max} \min_{h\in J(D_T)} p_h \end{aligned} \tag{4.11}$$

Remark 4.14. *1.) For a fixed nurture dataset D_T, the objective value $\min_{h\in J(D_T)} p_h$ is the smallest probability of a "bad" hypothesis in the surviving version space $V(D_T)$. 2.) By equation 4.10, this probability approximately characterizes the nature time to cover the entire "bad" hypothesis set $J(D_T)$. 3.) In other words, to optimize nature cover time, the teacher needs to cover as many low probability hypotheses as possible under the budget B.*

The optimization problem in 4.11 can be reduced to a version of the maximal coverage problem [43], but with a fixed ordering defined over the universe elements, and an objective to cover as long a suffix as possible under a finite budget B as described next.

Ordered Maximal Suffix Coverage (OMC) Problem The ordered maximal suffix coverage problem is defined by a tuple (U, W) where,

1. $U = \{1, 2, \ldots, n\}$ is a finite ordered universe set.
2. $W = \{W_i \subseteq U : i \in [m]\}$ is a finite collection of subsets of $\mathcal{U}$.

Any sub-collection of $D \subseteq W$ covers a subset of U and correspondingly a suffix in U. The objective of OMC is to select a sub-collection $D \subseteq W : |D| \leqslant B$, that maximizes the length of the covered suffix. Formally, the optimization objective is given as:

$$D^* = \underset{D\subseteq W, |D|\leqslant B}{\arg\max} \; p(D) \tag{4.12}$$

where $p(D)$ is the suffix length function for any sub-collection D defined as:

$$p(D) = \max \left\{ k \geqslant 0 \mid \{u_{n-k+1}, \cdots, u_n\} \subseteq \bigcup_{W_i \in D} W_i \right\}.$$

This problem is NP-hard, as it contains the set cover problem as a sub-problem. Furthermore, due to ordering, the objective is non-submodular.

Reduction to OMC Problem To reduce the optimization problem in equation 4.11 to an OMC problem instance, we compute $p_h, \forall h \in \mathcal{H} \backslash \{h^*\}_{P_X}$ and reorder the universe of hypotheses based on it as,

$$U = \{h_1, h_2, \cdots, h_{|\mathcal{H}\backslash\{h^*\}_{P_X}|}\} \qquad \text{s.t.} \qquad \forall i, p_{h_i} \geqslant p_{h_{i+1}},$$

and the cover subsets are defined by $W = \{W_x : x \in \mathcal{X}\}$ where $W_x = \{h \in \mathcal{H} : h(x) \neq h^*(x)\}$. The equivalence of the objectives follows from the fact that maximizing the smallest probability of an uncovered h is the same as maximizing the covered h suffix.

We first take a look at an example below, and then propose an algorithm to solve this problem that achieves a bi-criteria approximation guarantee.

Example 4.15 (An example instance)**.** *We explain the effect of the teaching dataset on the inconsistent hypothesis universe* $U = \mathcal{H}\backslash\{h^*\}_{P_X}$ *and its corresponding impact on the expected time in the nature phase through the following illustrative example:*

Let's say there are 15 inconsistent hypotheses in U *and we order them in increasing order* ($\forall i, p_{h_i} \geqslant p_{h_{i+1}}$) *of expected time required to cover them from i.i.d. samples* $E[T_h] = \frac{1}{p_h}$ *as defined above. The smaller the* p_h*, the harder it is for it to eliminate in the nature learning phase by equation* (4.10).

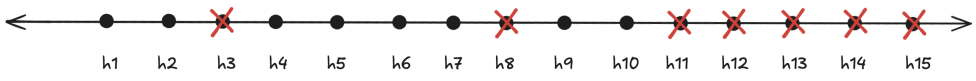


Figure 4.3: An example of the ordered set cover problem for near-optimal NtN teaching of a finite hypothesis class.

Consider a teaching dataset that eliminates 7 of the hypotheses so that there are 8 hypotheses remaining in $J(D_T)$ *that have to be eliminated/covered in the i.i.d. nature phase. The expected time to cover* J *is dominated by the* $\frac{1}{p_{h_{10}}}$*, which corresponds to the lowest probability hypothesis in* J*. An optimal teaching dataset under budget* B *can cover as many small probability*

hypotheses, i.e. largest suffix, as possible, leading to the following ordered maximal coverage problem.

Guarantee with Ordered Maximal Coverage(OMC) Oracle Suppose that we can solve the ordered maximal coverage problem exactly. This gives the following bound on the exact expected time NtN objective,

Theorem 4.16. *The algorithm A_O with OMC oracle achieves the following approximation bound on the exact expected time NtN objective, i.e.,*

$$f(D_T(A_O; B)) \leqslant \log(|\mathcal{H}|) \cdot f(D_T(A^*; B)),$$

where $D_T(A; B)$ represent the dataset constructed by algorithm A on budget B.

Proof. Let $D_T(A^*; B)$ and $D_T(A_O; B)$ be the teaching dataset produces by the algorithm A^* (that minimizes equation 4.9) and the algorithm A_O (that minimizes the relaxed upper bound in equation 4.11) using budget B respectively.

We have that,

$$f(D_T(A_O; B)) \overset{(1)}{\leqslant} \frac{\log(|\mathcal{H}|)}{p_{min}(J(D_T(A_O; B)))} \overset{(2)}{\leqslant} \frac{\log(|\mathcal{H}|)}{p_{min}(J(D_T(A^*; B)))} \overset{(3)}{\leqslant} \log(|\mathcal{H}|) \cdot f(D_T(A^*; B))$$

where $p_{min}(J(D_T)) = \min_{h \in J(D_T)} p_j$, (1) and (3) follows from upper bound definition, (2) follows from optimality of $D_T(A_O; B)$ on loose upper bound objective, i.e.,

$$p_{min}(J(D_T(A_O; B))) \geqslant p_{min}(J(D_T(A^*; B))).$$

□

Solving the OMC problem is NP-hard as it contains the set cover. So, we propose an iterative greedy algorithm that achieves a bi-criteria approximation guarantee on the objective.

Iterative Greedy Set Cover Algorithm We propose a greedy algorithm to solve the ordered maximal coverage problem, which yields an approximate bi-criteria optimality.

To achieve the objective of covering as long a suffix as possible under budget B, we enumerate over the order hypothesis suffix $h_{j:}$ from $j = M, \cdots, 1$ and try to cover them by using the greedy set cover algorithm. We enumerate over all j until we can no longer cover all $h_{j:}$ under budget B. At that point, we return the previous cover set as the optimal teaching set, refer to algorithm 3 for more details.

Algorithm 3: Greedy Algorithm for Optimizing Relaxed Upper Bound

Input: Input space $\mathcal{X}$, hypothesis class $\mathcal{H}$, target hypothesis h^*, and budget B
Output: A teaching set of size B

Compute set cover $U = \mathcal{H} \backslash \{h^*\}$, $W = \{W_x : x \in \mathcal{X}\}$.
Sort $\mathcal{H} \backslash \{h^*\}_{P_{\mathcal{X}}}$ in decreasing order of p_h as $U = \{h_1, h_2, \cdots, h_n\}$.
for k in $\{1, \cdots, |\mathcal{H} \backslash \{h^*\}_{P_{\mathcal{X}}}|\}$ **do**
    # construct a set cover $\mathcal{C}_k$ for the suffix universe $U_k = \{h_{n-k+1}, \ldots, h_n\}$.
    Initialize the set of chosen subsets: $\mathcal{C}_k = \emptyset$.
    Initialize the set of uncovered elements: $U_{unc} = U_k$.
    while $U_{unc} \neq \emptyset$ **do**
        Select $W^* = \arg\max_{W \in \mathcal{W}} |W \cap U_{unc}|$ with maximum uncovered elements and it to the solution: $\mathcal{C}_k = \mathcal{C}_k \cup \{W^*\}$.
        Update the uncovered elements: $U_{unc} = U_{unc} \setminus W^*$.
    if $|\mathcal{C}_k| > B$ **then**
        break
return $\{(x, h^*(x)) : W_x \in \mathcal{C}_{k-1}\}$.

Theorem 4.17. *Our iterative greedy algorithm achieves the following bi-criteria approximation bound on the NtN objective,*

$$f(D_T(A_G; B)) \leqslant \log(|\mathcal{H}|) \cdot f(D_T(A^*; B/\log(|\mathcal{H}|))).$$

The front $\log(|\mathcal{H}|)$ *factor is due to minimizing the upper bound on expected nature time, and the* $B/\log(|\mathcal{H}|)$ *is due to using the greedy algorithm for solving the OMC subproblem.*

Proof. Consider the maximum suffix s^* covered by the OMC oracle A_O under budget B, i.e., the number of subsets required to cover s^*, $n^*(s^*) = B$.

To cover the same suffix using greedy algorithm A_G, one would need at most

$$n_{A_G} \leqslant \log(|s^*|) \cdot n^*(s^*) \leqslant \log(|\mathcal{H}|) \cdot n^*(s^*) = \log(|\mathcal{H}|) \cdot B$$

subsets, i.e. if we have a budget of $\log(|\mathcal{H}|) \cdot B$, we should be able to obtain the optimal solution of OMC problem on an instance with budget B by running A_G with budget $B \cdot \log(|\mathcal{H}|)$. This would achieve the corresponding optimal expected time guarantee for B for the optimal NtN problem, given as,

$$f(D_T(A_G; B \cdot \log(|\mathcal{H}|))) \leqslant \log(|\mathcal{H}|) \cdot f(D_T(A^*; B)).$$

Substituting B with $\frac{B}{\log(|H|)}$, we obtain the result.

□

Remark 4.18. *We note that we can only guarantee approximation competitiveness with respect to a weaker budget* $B/\log(|\mathcal{H}|)$. *We further conjecture that due non-submodularity of the OMC objective, finding an efficient approximation algorithm that is competitive w.r.t. budget* B *is likely an NP-hard problem.*

4.5.1.2 Optimizing the Tight Upper Bound

In this section, we present a Mixed Integer Non Linear Program that directly optimizes the tight upper bound on the expected time objective, thereby achieving a better teaching guarantee. We start with a generic formulation of the Budgeted Cover Ratio problem that directly models the upper bound objective and provides the MINLP formulation.

Budgeted Cover Ratio Minimization We consider the following budgeted coverage problem. Let $U = \{u_j \mid j = 1, 2, \ldots, n\}$ denote a finite *universe* of items. Each item u_j has an associated nonnegative value $p_j \in \mathbb{R}_{\geqslant 0}$. We are given a family of subsets,

$$\mathcal{W} = \{W_i \subseteq U \mid i = 1, 2, \ldots, m\},$$

and a budget $B \in \mathbb{N}$ that limits the number of subsets we may select: we must choose a subcollection $W \subseteq \mathcal{W}$ satisfying $|W| \leqslant B$.

Given a chosen subcollection $W \subseteq \mathcal{W}$, we denote by $\bigcup_{W_i \in W} W_i$ the set of covered items and by $\{u_j \in U : u_j \notin \cup_{W_i \in W} W_i\}$ the uncovered items. Our objective is to minimize the ratio

$$\min_{W : |W| \leqslant B} \frac{\log\left(n - |\cup_{W_i \in W} W_i|\right)}{\min\limits_{j : u_j \notin \cup_{W_i \in W} W_i} p_j}, \tag{4.13}$$

where we assume that there is at least one uncovered item (so the numerator is well-defined) and that the corresponding min in the denominator exists and is strictly positive.

Remark 4.19. *Note that choosing $\mathcal{U} = \mathcal{H} \backslash h^*$ and $\mathcal{W} = \{W_x : x \in \mathcal{X}\}$ leads to the tight upper bound optimization problem.*

Theorem 4.20. *An algorithm that solves 4.13 exactly achieves the following bound on the NtN objective,*

$$f(D_T(A;B)) \leqslant \log(|\mathcal{V}(D_T(A^*))|) \cdot f(D_T(A^*;B)).$$

Proof. The proof follows the same structure as the proof of algorithm 4.16. □

Remark 4.21. *Compared to the OMC method for optimizing relaxed upper bound, this algorithm can perform significantly better when $\mathcal{V}(D_T(A^*))$ is very small compared to $\mathcal{H}$. However, solving this problem is very challenging as it entails a complex non-submodular objective. We propose a practical MINLP program to practically solve it without any theoretical guarantee.*

Mixed-Integer Nonlinear Reformulation We now derive a mixed-integer nonlinear program (MINLP) that encodes the objective (4.13). For convenience, we introduce the indicators $\phi_{ij} \in \{0,1\}$ for $i \in [m]$, $j \in [n]$, and binary decision variables $b_i \in \{0,1\}$ for $i \in [m]$:

$$\phi_{ij} = \begin{cases} 1, & \text{if } u_j \in W_i, \\ 0, & \text{otherwise,} \end{cases} \qquad b_i = \begin{cases} 1, & \text{if } W_i \in W, \\ 0, & \text{otherwise.} \end{cases}$$

We introduce coverage variables $c_j \in [0,1]$, $j = 1, \ldots, n$, indicating whether item u_j is covered by at least one selected subset. The variables c_j are linked to the selection variables b_i through the standard "OR" linearization:

$$c_j \leqslant \sum_{i=1}^{m} \phi_{ij} b_i, \quad \forall j \in [n], \quad c_j \geqslant \phi_{ij} b_i, \quad \forall i \in [m],\, j \in [n].$$

Thus $c_j = 1$ if and only if at least one chosen subset W_i with $b_i = 1$ contains u_j; otherwise

$c_j = 0$. The budget constraint on the number of selected subsets is

$$\sum_{i=1}^{m} b_i \leqslant B, \text{ and define } e = \log\Big(n - \sum_{j=1}^{n} c_j\Big).$$

so that $n - \sum_j c_j$ is the number of uncovered items and e is its logarithm. We enforce this relationship and its domain explicitly by $n - \sum_{j=1}^{n} c_j > 0$. Finally, for a candidate ratio θ, we require that θ be at least e/p_j for every uncovered item u_j (those with $c_j = 0$). This can be encoded compactly as

$$(1 - c_j)\, e \leqslant \theta\, p_j, \qquad \forall j \in [n],$$

since the left-hand side equals e when $c_j = 0$ and 0 when $c_j = 1$. This motivates the following feasibility problem for a fixed parameter θ:

MINLP(θ)

$$\min_{b,c,e} \quad 0 \tag{4.14a}$$

$$\text{s.t.} \quad b_i \in \{0,1\}, \qquad \forall i \in [m], \tag{4.14b}$$

$$c_j \in [0,1], \qquad \forall j \in [n], \tag{4.14c}$$

$$c_j \leqslant \sum_{i=1}^{m} \phi_{ij}\, b_i, \qquad \forall j \in [n], \tag{4.14d}$$

$$c_j \geqslant \phi_{ij}\, b_i, \qquad \forall i \in [m],\ j \in [n], \tag{4.14e}$$

$$\sum_{i=1}^{m} b_i \leqslant B, \tag{4.14f}$$

$$e = \log\Big(n - \sum_{j=1}^{n} c_j\Big), \tag{4.14g}$$

$$n - \sum_{j=1}^{n} c_j > 0, \tag{4.14h}$$

$$(1 - c_j)\, e \leqslant \theta\, p_j, \qquad \forall j \in [n]. \tag{4.14i}$$

This is a mixed-integer nonlinear program (MINLP) in the variables (b, c, e), parametrized by θ. We denote this feasibility problem by MINLP(θ).

Proposition 4.22. *Let $\theta^\star$ denote the optimal value of* (4.13)*. Then:*

- *If $\theta \geqslant \theta^\star$, the feasibility problem MINLP(θ) admits at least one feasible solution.*
- *If $\theta < \theta^\star$, the feasibility problem MINLP(θ) is infeasible.*

Thus, $\theta^\star$ is characterized as the smallest value of θ for which MINLP(θ) is feasible.

We can exploit the monotonicity of feasibility in θ to compute $\theta^\star$ using a bisection (binary search) scheme. We assume we are given initial lower and upper bounds $0 \leqslant \theta_L < \theta_U$ such that MINLP(θ_U) is feasible and MINLP(θ_L) is infeasible. One natural choice is to take $\theta_L = 0$ and construct a conservative upper bound θ_U from problem data (e.g., using worst-case coverage patterns).

At each iteration, we solve the feasibility problem MINLP(θ) for the midpoint $\theta = (\theta_L + \theta_U)/2$. If MINLP$(\theta)$ is feasible, we can safely decrease the upper bound to $\theta_U \leftarrow \theta$; otherwise, we increase the lower bound to $\theta_L \leftarrow \theta$. This process is repeated until the interval $[\theta_L, \theta_U]$ is smaller than a prescribed tolerance $\varepsilon > 0$. The corresponding algorithm is summarized in Algorithm 4.

In practice, any off-the-shelf MINLP solver can be used in the inner loop to solve MINLP(θ) as a pure feasibility problem. Once $\hat{\theta}$ has been computed to the desired tolerance, the corresponding selection W can be deployed as the solution to the original problem (4.13).

4.5.2 NtN Teaching Through Function Approximation of Risk

Unlike the version space learner on a finite hypothesis setting, characterizing the expected time or the risk of training a complex learner on a dataset D_T, and further optimizing it over the nurture dataset D_T is a very challenging problem. To avoid this issue, we take a function approximation approach where we first approximate the risk of a learner using datamodels and then use the datamodel approximator to find the optimal teaching dataset for optimal NtN teaching.

The beauty of this algorithm is that one can utilize it to teach any learner as long as a good approximator of its risk is available. In this section, we consider the risk-based NtN objective - teaching of learners to minimize their expected risk at the end of the nature phase as stated below.

Algorithm 4: Bisection Algorithm for Budgeted Coverage Ratio Minimization

Require: Universal set U, subsets $\mathcal{W}$, values p_j, budget B, initial bounds $\theta_L < \theta_U$, tolerance $\varepsilon > 0$.

Ensure: Approximate optimal ratio $\hat{\theta}$ and selection W.

Set $k \leftarrow 0$.
while $\theta_U - \theta_L > \varepsilon$ **do**
  $\theta \leftarrow (\theta_L + \theta_U)/2$.
  Solve the feasibility problem MINLP(θ) in variables (b, c, e), given by constraints (4.14b)–(4.14i).
  if MINLP(θ) is feasible **then**
    $\theta_U \leftarrow \theta$.
    Store the corresponding solution $(b^{(k)}, c^{(k)}, e^{(k)})$.
  else
    $\theta_L \leftarrow \theta$.
  $k \leftarrow k + 1$.
Set $\hat{\theta} \leftarrow \theta_U$.
Recover the final selection $W \leftarrow \{W_i \in \mathcal{W} : b_i^{(k)} = 1\}$ from the last feasible solution.
return $(\hat{\theta}, W)$.

4.5.2.1 Problem Formulation

The Learner: We consider any predictive modeling algorithm that does ERM w.r.t. a hypothesis class $\mathcal{H}$, i.e., on receiving a dataset $D = \{(x_i, y_i)\}|_{i=1}^{K}$, it minimizes the empirical risk defined by a loss function as follows,

$$\hat{h} \leftarrow \arg\min \frac{1}{K} \sum_{i=1}^{K} \ell(h(x_i), y_i) + \lambda \cdot R_{reg}(h).$$

The empirical test risk of the learned on a seperate test set $\hat{D} = \{(x_j, y_j)\}|_{j=1}^{m}$ is computed as $\hat{R}(A(D)) = \frac{1}{m} \sum_{i=1}^{m} \ell(\hat{h}(x_j), y_j)$.

The Environment and the Teacher: The environment is specified by a realizable distribution P_X, h^*. In the nature phase, the learner receives a dataset of size n from the environment. The teacher can construct any dataset of size up to B and is required to minimize the expected risk of the learner at the end of the nature phase.

Expected Risk Objective: Given an instance $(\mathcal{X}, \mathcal{Y}, P, h^*, \mathcal{H}, n, B)$, the expected risk

objective of NtN is defined as follows:

$$D_T^* \leftarrow \underset{D_T : |D_T| \leqslant B}{\arg\min}\ \mathbb{E}_{D_E \sim P^n}\left[R(\mathcal{A}(D_T \cup D_E)\right]. \tag{4.15}$$

Remark 4.23. *Unlike the instance-agnostic setting 4.3, the objective 4.15 requires the teacher to produce a teaching set that is competitive w.r.t. the instance-specific* P.

As alluded to before, computing the closed form of the risk of training an algorithm is a challenging problem. To handle this, we take a function approximation approach by first approximating the risk using a datamodel [37] and then using this risk approximator to solve NtN. For simplicity, we first start with a linear approximator called a linear datamodel, which allows us to obtain a closed-form solution to NtN teaching.

4.5.2.2 Teaching using Linear Datamodel

Linear datamodel proposed by [37] aims to approximate the risk $R(\mathcal{A}(D))$ of training an algorithm $\mathcal{A}$ on a dataset D as a linear function of the dataset.

More formally, given a pool of input universe $\mathcal{X}$, the test risk of training an algorithm $\mathcal{A}$ on dataset D is modeled as a linear function in the indicator feature representation of dataset D, i.e., $\mathbb{1}_D \in \{0,1\}^{\mathcal{X}}$ as,

$$\hat{R}(\mathcal{A}(D)) \approx w_P^\top \mathbb{1}_D. \tag{4.16}$$

The parameter w_P is estimated directly by solving a meta-learning problem on meta-dataset $\mathcal{D} = \{\mathbb{1}_{D_i}, R(\mathcal{A}(D_i))\}_{i=1}^m$ sampled from a distribution defined over data subsets $P_{2^{\mathcal{X}}}$,

$$w_P \leftarrow \arg\min_w \frac{1}{m} \sum_{i=1}^{m} \ell_2(w^\top \mathbb{1}_{D_i}, \hat{R}(\mathcal{A}(D_i)) + \lambda \|w\|_1. \tag{4.17}$$

Remark 4.24. *Note that this method can be applied for any learner* $\mathcal{A}$ *and hypothesis class* $\mathcal{H}$*, as long as one can efficiently train the base learner* $\mathcal{A}$ *on a collection of datasets* $\mathcal{D}$.

Once we have w_P, the function $w_P^\top \mathbb{1}_D$ serves as a surrogate for test risk, which is then used to solve the original NtN problem under the following realizability assumption.

Assumption 4.25 (Realizability of Linear Datamodel)**.** *The risk function of learning algorithm $\mathcal{A}$ is realizable under linear datamodel iff* $R(\mathcal{A}(D)) = w_P^\top \mathbb{1}_D, \forall D$.

Algorithm using Linear Datamodel: Using linear datamodel under assumption 4.25, the risk in 4.15 can be expressed as $R(\mathcal{A}(D \cup D_T)) = w_P^\top \cdot \mathbb{1}_{D \cup D_T}$, which simplifies NtN objective to:

$$D_T^* \leftarrow \min_{D_T : |D_T| \leqslant B} w_P^\top \cdot \mathbb{E}_D[\mathbb{1}_{D \cup D_T}]. \tag{4.18}$$

Expanding $E_D\left[\mathbb{1}_{D \cup D_T}\right]_x = (1-(1-P_x)^n) + \mathbb{1}_{x \in D_T} \cdot (1-P_x)^n$, we note that the first term is independent of D_T and thus can be ignored. This reduces equation 4.18 to the following,

$$D_T^* \leftarrow \arg \min_{D_T : |D_T| \leqslant B} \sum_{x \in \mathcal{X}} \mathbb{1}_{x \in D_T} \cdot w_{P,x}(1-P_x)^n. \tag{4.19}$$

This is a *Unit Profit Knapsack problem* where every item x has a unit cost and weight $w_{P,x}(1-P_x)^n$. It is efficiently solvable by choosing B items with the smallest weight, leading to the following theorem.

Theorem 4.26. *Under assumption 4.25, the instance-aware NtN problem is efficiently solvable, and the optimal solution is given by,*

$$D_T^* \leftarrow \arg\min_B \{w_{P,x}(1-P_x)^n : x \in \mathcal{X}\}. \tag{4.20}$$

Remark 4.27. *We remark that in contrast to the solution of [25], which can be interpreted as single-phase teaching, our algorithm also utilizes* P *to be instance-aware in the nature phase.*

4.5.2.3 Teaching using Neural Datamodel

The linearity assumption in the linear datamodel can be very strict, even for simpler models, risk is non-linear and non-additive as a function of the dataset. So, we propose to use more complex models like neural networks to estimate the risk and use it for the purpose of teaching. We call such a neural network trained to estimate the risk of a learning algorithm a neural datamodel.

Similar to a linear datamodel, a neural datamodel can be trained through backpropagation on a neural network architecture using the meta-dataset defined above; however, the challenge is that we have to account for risk in the nature phase of NtN while training it.

More formally, given a pool of input universe $\mathcal{X}$, the NtN risk of training an algorithm $\mathcal{A}$ on dataset D is modeled as a neural function in the indicator feature representation of dataset D, i.e., $\mathbb{1}_D \in \{0,1\}^{\mathcal{X}}$,

$$\mathbb{E}_{D_E \sim P^n}[R(\mathcal{A}(D \cup D_E))] \approx f_{\theta_P}(\mathbb{1}_D). \tag{4.21}$$

As before, the parameter θ_P is estimated by solving meta-learning problem on meta-dataset $\mathcal{D} = \{\mathbb{1}_{D_i}, \frac{1}{k}\sum_{k \in K} R(\mathcal{A}(D_i \cup D_k))\}|_{i=1}^{m}$ sampled from a distribution defined over possible data subsets $P_{2^{\mathcal{X}}}$, and k i.i.d. samples from environment distribution $P_{\mathcal{X}}$

$$\theta_P \leftarrow \arg\min_{w} \frac{1}{m}\sum_{i=1}^{m} \ell_2(f_\theta(\mathbb{1}_{D_i}), \mathbb{E}_{D_E}[R(\mathcal{A}(D \cup D_E))] + \lambda\|w\|_1. \tag{4.22}$$

Once we have θ_P, the function $f_{\theta_P}(1_D)$ serves as a surrogate for true NtN risk, which is utilized to solve the original NtN problem using projected gradient descent on the input space using the algorithm below.

Algorithm 5: Regularized Projected Gradient Descent

Input: model family: $f_\theta : x \to \mathbb{R}$, target: $y^* = 0$, sparsity: B, learning rate: η
Output: $x^* \in \{0,1\}^d$ s.t. $\|x\|_1 \leqslant B$

1: initialize $x \sim \{0,1\}^d$ randomly.
2: **while** until convergence **do**
3: gradient step: $x = x - \eta \nabla_x l_\theta(x)$
4: projection step: $x \leftarrow \text{Proj}_B(x)$ {set B top entries of x to 1, others to 0.}
5: return x

While neural datamodels offer superior representation of the NtN risk compared to linear baselines, they incur higher training costs. Additionally, although the projected gradient descent solver is efficient to implement, it lacks theoretical guarantees for global convergence. We leave the resolution of these trade-offs and the exploration of more robust optimization strategies to future work.

4.6 Experiments

We evaluate our teaching algorithms for teaching learners with diverse hypothesis classes (in both NtN settings) and compare their performance against relevant baselines, as outlined below:

1. **Performance of the algorithm**: 1.) *Teach vs no-teach*: Given a fixed teaching budget, do our teaching algorithms produce significant gains compared to no teaching? 2.) *Role of budget*: Does higher budget lead to better teaching performance?

2. **Comparison with simulated teaching**: In *simulated teaching* (Sim-Teach), the teacher simulates K learners(we choose a moderate K for fair comparison based on computation cost), each with a random teaching set of size B, and then selects the best-performing teaching set among them for final teaching.

Our experiments, designed for conceptual clarity, serve as clear proof of concept to corroborate our theory. A natural extension of our work would involve more complex benchmarks and datamodels. This represents a promising direction for future work, and we outline a path for it in the appendix.

4.6.1 Instance Agnostic Teaching by Optimal VC Reduction

We apply our optimal VC reduction algorithm(OPT-VC) to a version space learner with a linear and an axis-aligned rectangle class to demonstrate its effectiveness in an instance-agnostic setting.

4.6.1.1 Homogeneous Linear Classifiers

We consider teaching a $w^* \in \mathbb{R}^4$ to a homogeneous linear version space classifier. The nature's P is a uniform distribution over the sphere $\mathbb{S}^4$. We discretize the weight space and do exact version space learning, as specified in equation [4.2]. The error is computed by evaluating the worst classifier in the version space $\mathcal{V}(D)$ on a held-out test set.

Our results: We tested our algorithm 4.4 for teaching this learner on various budget $B \in \{0, \ldots, d+1\}$ and plot its NtN performance $\hat{R}(\mathcal{A}(D_T \cup D_E))$ as a function of n_{iid} as shown in Figure 4.4.

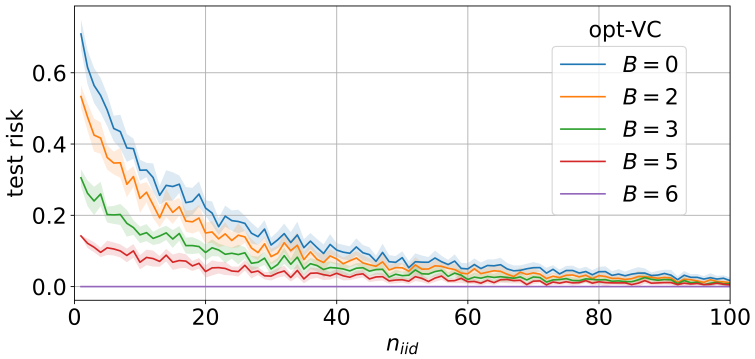


Figure 4.4: Performance of our OPT-VC algorithm on a linear learner.

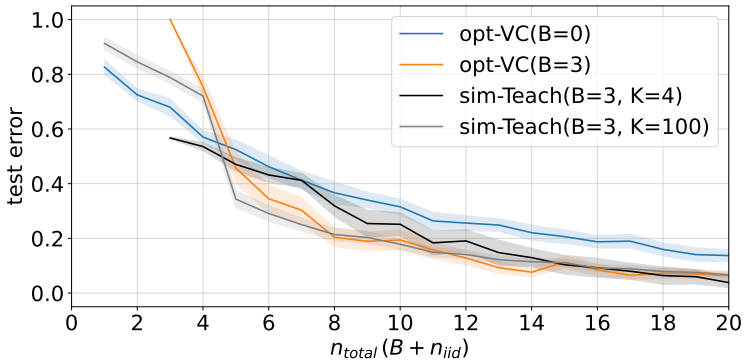


Figure 4.5: Comparing our optimal VC reduction algorithm to a simulated baseline for teaching a linear learner.

The blue curve $B = 0$ denotes no-teaching, i.e., the learner only learns from n_{iid} points, while other curves represent teaching with respective budget B. Figure 4.4, we observe that OPT-VC outperforms no-teaching for all B. Moreover, a higher B consistently leads to lower test risk, reaching to zero for all n_{iid} once $B \geqslant TD = 5$. We also compare our algorithm against the Sim-Teach with $B = 3$ and $K = 4,100$ simulations as shown in Figure 4.5. We observe that Sim-Teach with $K = 4$ performs a bit better than i.i.d. teaching but is still outperformed by OPT-VC. Even with $K = 100$, which is computationally expensive, Sim-Teach could barely compete with our OPT-VC algorithm.

4.6.1.2 Axis Aligned Rectangle Class

We consider axis-aligned rectangle class defined on space $\mathcal{X} = \{-n, \cdots, n\}^2$ and choose a target rectangle h^* and $P_{\mathcal{X}} = U(\mathcal{X})$. As before, the version space learner maintains a version

space $\mathcal{V}(\mathrm{D};\mathcal{H})$ and is evaluated by the worst hypothesis in $\mathcal{V}(\mathrm{D})$.

Our results: Figure 4.6 shows the NtN performance of our OPT-VC algorithm on various budget sizes 4.1 as a function of $\mathfrak{n}_{\mathrm{iid}}$ on the x-axis.

As before, the blue curve corresponds to no teaching, while others represent budgeted teaching with various B. We see from Figure 4.6 that our OPT-VC consistently outperforms no-teaching and leads to lower error with a higher budget. Again, once $\mathrm{B} \geqslant \mathrm{TD} = 6$, nurture alone leads to a zero risk.

We also compare OPT-VC to Sim-Teach and show the results in Figure 4.7. Sim-Teach simulates $\mathrm{K} = 4,100$ learners with a random $\mathrm{B} = 3$ teaching set and picks the best one it finds. Unlike the linear case, Sim-Teach significantly underperforms w.r.t. our OPT-VC algorithm on both K's.

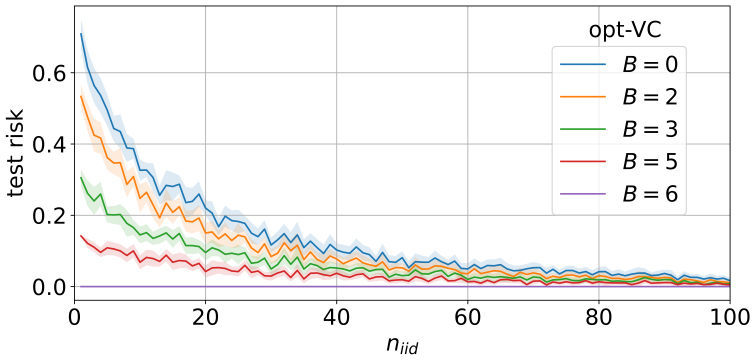


Figure 4.6: Performance of our optimal VC reduction algorithm on the learner.

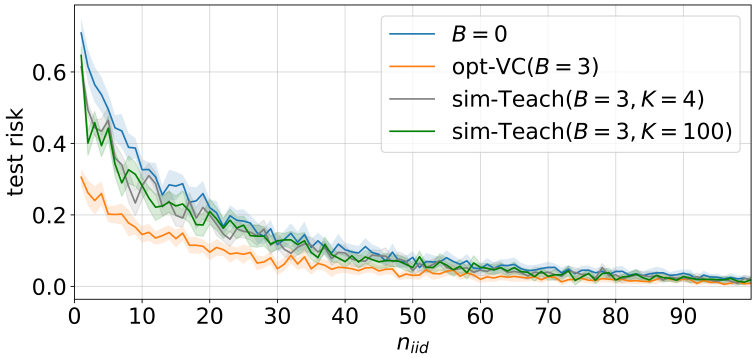


Figure 4.7: Comparing our optimal VC reduction algorithm to the simulated baseline in teaching axis-aligned rectangle class.

4.6.2 Instance Aware Teaching through Linear Datamodel

In this section, we demonstrate the effectiveness of our linear datamodel algorithm to teach a linear perceptron learner in the instance-aware setting of section [4.5].

For simplicity, we choose a finite-size universe, $\mathcal{X} \subset \mathbb{R}^2$, consisting of equally-spaced points on the unit circle, a $w^* \in \mathbb{R}^2$ and $P_{\mathcal{X}} = U(\mathcal{X})$ as shown in Figure 4.9. We first train a linear datamodel w_P that represents the risk of a linear perceptron(refer to the appendix for more details). Once we obtain $\hat{w}_P$, we select the bottom B input $\mathcal{X}$'s as the teaching dataset based on the value $\hat{w}_{P,x}(1-P_x)^n$.

Our results: We evaluate the NtN risk of perceptron on the teaching dataset produced by the datamodel method(OPT-DM) and report it in Figure 4.8. We observe that OPT-DM with budget $B = 2,3$ significantly outperforms no-teaching $(B = 0)$. It is also worth noting that D_T^* generated by OPT-DM differs somewhat from those produced by OPT-VC, as illustrated in Figure 4.9. Nevertheless, both approaches reduce the learner's risk compared to just using an i.i.d. dataset, as shown in Figure 4.8.

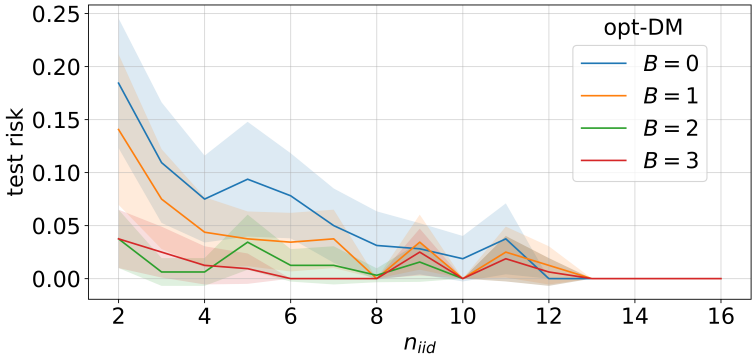


Figure 4.8: Comparing the teaching set (constructed with linear datamodel) with the case of no teaching.

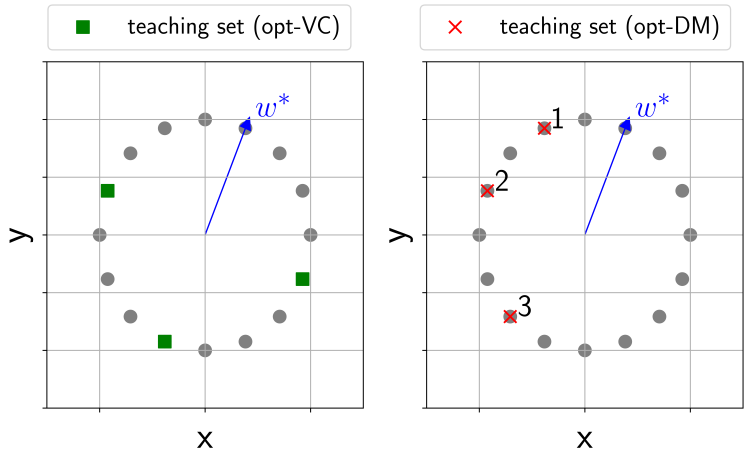


Figure 4.9: Teaching sets as constructed by linear datamodel method on a perceptron learner in $\mathbb{R}^2$.

Bibliography

[1] *High Performance Outdoor Navigation from Overhead Data using Imitation Learning*, pages 262–269. 2009.

[2] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[3] Stephen Adams, Tyler Cody, and Peter A Beling. A survey of inverse reinforcement learning. *Artificial Intelligence Review*, 55(6):4307–4346, 2022.

[4] Alekh Agarwal, Nan Jiang, Sham M. Kakade, and Wen Sun. *Reinforcement Learning: Theory and Algorithms*. 2021. `https://rltheorybook.github.io/`.

[5] Umair Ahmed, Maria Christakis, Aleksandr Efremov, Nigel Fernandez, Ahana Ghosh, Abhik Roychoudhury, and Adish Singla. Synthesizing tasks for block-based programming. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 22349–22360. Curran Associates, Inc., 2020.

[6] Ofra Amir, Ece Kamar, Andrey Kolobov, and Barbara J. Grosz. Interactive teaching strategies for agent training. In *IJCAI*, pages 804–811, 2016.

[7] Martin Anthony, Graham Brightwell, and John Shawe-Taylor. On specifying boolean functions by labelled examples. *Discrete Applied Mathematics*, 61(1):1–25, 1995.

[8] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[9] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129, 1995.

[10] Frank Balbach. *Models for Algorithmic Teaching*. PhD thesis, Universität Duisburg-Essen, 2008.

[11] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II*. Athena Scientific, Belmont, MA, 4th edition, 2012. Comprehensive textbook treatment of SSPP algorithms and infinite horizon planning.

[12] Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Mathematics of Operations Research*, 16(3):580–595, 1991. Foundational paper establishing convergence of Value Iteration and Policy Iteration for SSPPs under proper policy assumptions.

[13] Shubham Kumar Bharti, Stephen Wright, Adish Singla, and Jerry Zhu. On the complexity of teaching a family of linear behavior cloning learners. In *Proceedings of the Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[14] Daniel S Brown and Scott Niekum. Machine teaching for inverse reinforcement learning: Algorithms and applications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7749–7758, 2019.

[15] Daniel S Brown and Scott Niekum. Machine teaching for inverse reinforcement learning: Algorithms and applications. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4829–4837, 2019.

[16] Rudy Bunel, Matthew J. Hausknecht, Jacob Devlin, Rishabh Singh, and Pushmeet Kohli. Leveraging grammar and reinforcement learning for neural program synthesis. *CoRR*, abs/1805.04276, 2018.

[17] Maya Cakmak and Manuel Lopes. Algorithmic and human teaching of sequential decision tasks. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[18] Xinyun Chen, Chang Liu, and Dawn Song. Execution-guided neural program synthesis. In *International Conference on Learning Representations*, 2019.

[19] Yuxin Chen, Adish Singla, Oisin Mac Aodha, Pietro Perona, and Yisong Yue. Understanding the role of adaptivity in machine teaching: The case of version space learners. *Advances in Neural Information Processing Systems*, 31, 2018.

[20] Yuxin Chen, Adish Singla, Oisin Mac Aodha, Pietro Perona, and Yisong Yue. Understanding the role of adaptivity in machine teaching: The case of version space learners. In *Advances in Neural Information Processing Systems*, pages 1483–1493, 2018.

[21] Yun-Shiuan Chuang, Xuezhou Zhang, Yuzhe Ma, Mark K Ho, Joseph L Austerweil, and Xiaojin Zhu. Using machine teaching to investigate human assumptions when teaching reinforcement learners. *arXiv preprint arXiv:2009.02476*, 2020.

[22] Felipe Codevilla, Eder Santana, Antonio M. Lopez, and Adrien Gaidon. Exploring the limitations of behavior cloning for autonomous driving. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[23] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20:273–297, 1995.

[24] Aleksandr Efremov, Ahana Ghosh, and Adish Kumar Singla. Zero-shot learning of hint policy via reinforcement learning and program synthesis. In *Educational Data Mining*, 2020.

[25] Logan Engstrom, Axel Feldmann, and Aleksander Madry. Dsdm: Model-aware dataset selection with datamodels. *arXiv preprint arXiv:2401.12926*, 2024.

[26] Claude-Nicolas Fiechter. Efficient reinforcement learning. In *Proceedings of the seventh annual conference on Computational learning theory*, pages 88–97, 1994.

[27] Sergio Filho, Eduardo Laber, Pedro Lazera, and Marco Molinaro. Time-constrained learning. *Pattern Recognition*, 142:109672, 2023.

[28] Kristian Georgiev, Roy Rinberg, Sung Min Park, Shivam Garg, Andrew Ilyas, Aleksander Madry, and Seth Neel. Attribute-to-delete: Machine unlearning via datamodel matching. *arXiv preprint arXiv:2410.23232*, 2024.

[29] Sally A Goldman and Michael J Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50(1):20–31, 1995.

[30] Sally A Goldman and Michael J Kearns. On the complexity of teaching. *Journal of Computer and System Sciences*, 50(1):20–31, 1995.

[31] Steve Hanneke. Teaching dimension and the complexity of active learning. In *International Conference on Computational Learning Theory*, pages 66–81. Springer, 2007.

[32] Steve Hanneke. The optimal sample complexity of pac learning. *Journal of Machine Learning Research*, 17(38):1–57, 2016.

[33] Luis Haug, Sebastian Tschiatschek, and Adish Singla. Teaching inverse reinforcement learners via features and demonstrations. In *Advances in Neural Information Processing Systems*, pages 8464–8473, 2018.

[34] David Haussler, Nick Littlestone, and Manfred K Warmuth. Predicting $\{0,1\}$-functions on randomly drawn points. *Information and Computation*, 115(2):248–292, 1994.

[35] Anette Hunziker, Yuxin Chen, Oisin Mac Aodha, Manuel Gomez Rodriguez, Andreas Krause, Pietro Perona, Yisong Yue, and Adish Singla. Teaching multiple concepts to a forgetful learner. In *Advances in Neural Information Processing Systems*, 2019.

[36] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. *ACM Computing Surveys (CSUR)*, 50(2):1–35, 2017.

[37] Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting predictions from training data. In *Proceedings of the 39th International Conference on Machine Learning*, pages 9473–9502. PMLR, 2022.

[38] Chi Jin, Zeyuan Allen-Zhu, Sebastien Bubeck, and Michael I Jordan. Is q-learning provably efficient? In *Advances in Neural Information Processing Systems*, pages 4863–4873, 2018.

[39] Kwang-Sung Jun, Lihong Li, Yuzhe Ma, and Jerry Zhu. Adversarial attacks on stochastic bandits. In *Advances in Neural Information Processing Systems*, pages 3640–3649, 2018.

[40] Parameswaran Kamalaruban, Rati Devidze, Volkan Cevher, and Adish Singla. Interactive teaching algorithms for inverse reinforcement learning. In *IJCAI*, pages 2692–2700, 2019.

[41] Michael Kearns, Ming Li, and Leslie G. Valiant. Learning boolean formulas. *Journal of the ACM*, 41(6):1298–1328, 1994.

[42] Alaa Khaddaj, Guillaume Leclerc, Aleksandar Makelov, Kristian Georgiev, Hadi Salman, Andrew Ilyas, and Aleksander Madry. Rethinking backdoor attacks. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 16216–16236. PMLR, 2023.

[43] Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999.

[44] Hayato Kobayashi and Ayumi Shinohara. Complexity of teaching by a restricted number of examples. 01 2009.

[45] Hayato Kobayashi and Ayumi Shinohara. Complexity of teaching by a restricted number of examples. In *Proceedings of the 22nd Annual Conference on Learning Theory (COLT)*, pages 294–308, 2009.

[46] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[47] Matthieu Komorowski, Leo A Celi, Omar Badawi, Anthony C Gordon, and A Aldo Faisal. The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care. *Nature medicine*, 24(11):1716–1720, 2018.

[48] Akash Kumar, Hanqi Zhang, Adish Singla, and Yuxin Chen. The teaching dimension of kernel perceptron. In *International Conference on Artificial Intelligence and Statistics*, pages 2071–2079. PMLR, 2021.

[49] Akash Kumar, Hanqi Zhang, Adish Singla, and Yuxin Chen. The teaching dimension of kernel perceptron. In *Proceedings of the 24th International Conference on Artificial Intelligence and Statistics*, pages 2071–2079. PMLR, 2021.

[50] Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by watching: extracting reusable task knowledge from visual observation of human performance. *IEEE Transactions on Robotics and Automation*, 10(6):799–822, 1994.

[51] Laurent Lessard, Xuezhou Zhang, and Xiaojin Zhu. An optimal control approach to sequential machine teaching. *arXiv preprint arXiv:1810.06175*, 2018.

[52] Lihong Li. Sample complexity bounds of exploration. In *Reinforcement Learning*, pages 175–204. Springer, 2012.

[53] Ji Liu and Xiaojin Zhu. The teaching dimension of linear learners. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 117–126. PMLR, 2016.

[54] Ji Liu, Xiaojin Zhu, and Hrag Ohannessian. The teaching dimension of linear learners. In *International Conference on Machine Learning*, pages 117–126. PMLR, 2016.

[55] Weiyang Liu, Bo Dai, Ahmad Humayun, Charlene Tay, Chen Yu, Linda B. Smith, James M. Rehg, and Le Song. Iterative machine teaching, 2017.

[56] Yuzhe Ma, Kwang-Sung Jun, Lihong Li, and Xiaojin Zhu. Data poisoning attacks in contextual bandits. In *International Conference on Decision and Game Theory for Security*, pages 186–204. Springer, 2018.

[57] Yuzhe Ma, Xuezhou Zhang, Wen Sun, and Jerry Zhu. Policy poisoning in batch reinforcement learning and control. *Advances in Neural Information Processing Systems*, 32, 2019.

[58] Farnam Mansouri, Yuxin Chen, Ara Vartanian, Jerry Zhu, and Adish Singla. Preference-based batch and sequential teaching: Towards a unified view of models. *Advances in neural information processing systems*, 32, 2019.

[59] Pasin Manurangsi. Improved inapproximability of vc dimension and littlestone's dimension via (unbalanced) biclique. *arXiv preprint arXiv:2211.01443*, 2022.

[60] Pasin Manurangsi and Aviad Rubinstein. Inapproximability of vc dimension and littlestone's dimension. In *Proceedings of the 30th Conference on Learning Theory*, pages 902–932. PMLR, 2017.

[61] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[62] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, 2nd edition, 2018.

[63] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[64] Andrew Y Ng, Stuart Russell, et al. Algorithms for inverse reinforcement learning. In *Icml*, volume 1, page 2, 2000.

[65] Xuezhou Zhang Hrag Gorune Ohannessian, Ayon Sen, Scott Alfeld, and Xiaojin Zhu. Optimal teaching for online perceptrons. *University of Wisconsin-Madison*.

[66] Christos H Papadimitriou and Mihalis Yannakakis. On limited nondeterminism and the complexity of the v-c dimension. *Journal of Computer and System Sciences*, 53(2):161–170, 1996.

[67] Tomi Peltola, Mustafa Mert Çelikok, Pedram Daee, and Samuel Kaski. Machine teaching of active sequential learners. In *Advances in Neural Information Processing Systems*, pages 11202–11213, 2019.

[68] Tomi Peltola, Mustafa Mert Celikok, Pedram Daee, and Samuel Kaski. Machine teaching of active sequential learners. In *Advances in Neural Information Processing Systems*, volume 32, pages 1–14, 2019.

[69] Dean A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988.

[70] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991.

[71] Hong Qian, Xu-Hui Liu, Chen-Xi Su, Aimin Zhou, and Yang Yu. The teaching dimension of regularized kernel learners. In *International Conference on Machine Learning*, pages 17984–18002. PMLR, 2022.

[72] Nived Rajaraman, Lin F Yang, Jiantao Jiao, and Kannan Ramachandran. Toward the fundamental limits of imitation learning. *arXiv preprint arXiv:2009.05990*, 2020.

[73] Amin Rakhsha, Goran Radanovic, Rati Devidze, Xiaojin Zhu, and Adish Singla. Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning. In *International Conference on Machine Learning*, pages 7974–7984. PMLR, 2020.

[74] Stephane Ross and Drew Bagnell. Efficient reductions for imitation learning. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 661–668, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[75] Claude Sammut, Scott Hurst, Dana Kedzier, and Donald Michie. Learning to fly. 02 1992.

[76] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

[77] Ayumi Shinohara. Complexity of computing vapnik-chervonenkis dimension and some generalized dimensions. *Theoretical Computer Science*, 137(1):129–144, 1995.

[78] Ayumi Shinohara and Satoru Miyano. Teachability in computational learning. *New Generation Computing*, 8(4):337–347, 1991.

[79] Susan M Shortreed, Eric Laber, Daniel J Lizotte, T Scott Stroup, Joelle Pineau, and Susan A Murphy. Informing sequential clinical decision-making through reinforcement learning: an empirical study. *Machine learning*, 84(1-2):109–136, 2011.

[80] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[81] Daniel Soudry, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. The implicit bias of gradient descent on separable data. *Journal of Machine Learning Research*, 19(70):1–57, 2018.

[82] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *International Conference on Machine Learning*, pages 3309–3318. PMLR, 2017.

[83] Ola Svensson, Jakub Tarnawski, and László A. Végh. A constant-factor approximation algorithm for the asymmetric traveling salesman problem. *CoRR*, abs/1708.04215, 2017.

[84] Faraz Torabi, Garrett Warnell, and Peter Stone. Recent advances in imitation learning from observation. *CoRR*, abs/1905.13566, 2019.

[85] Lisa Torrey and Matthew Taylor. Teaching on a budget: Agents advising agents in reinforcement learning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1053–1060, 2013.

[86] Sebastian Tschiatschek, Ahana Ghosh, Luis Haug, Rati Devidze, and Adish Singla. Learner-aware teaching: Inverse reinforcement learning with preferences and constraints. In *Advances in Neural Information Processing Systems*, 2019.

[87] Sebastian Tschiatschek, Ahana Ghosh, Luis Haug, Rati Devidze, and Adish Singla. Learner-aware teaching: Inverse reinforcement learning with preferences and constraints. *arXiv preprint arXiv:1906.00429*, 2019.

[88] V. Vapnik. Principles of risk minimization for learning theory. In J. Moody, S. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4, pages 831–838. Morgan Kaufmann, 1992.

[89] Thomas J Walsh and Sergiu Goschin. Dynamic teaching in sequential decision making environments. *arXiv preprint arXiv:1210.4918*, 2012.

[90] Valika K. Wan and Khanh Do Ba. Approximating set cover, 2005. Accessed: 2023-10-15.

[91] Yizhen Wang and Kamalika Chaudhuri. Data poisoning attacks against online learning. *arXiv preprint arXiv:1808.08994*, 2018.

[92] Xuezhou Zhang, Shubham Kumar Bharti, Yuzhe Ma, Adish Singla, and Xiaojin Zhu. The sample complexity of teaching-by-reinforcement on q-learning. *arXiv preprint arXiv:2006.09324*, 2021.

[93] Xuezhou Zhang, Yuzhe Ma, Adish Singla, and Xiaojin Zhu. Adaptive reward-poisoning attacks against reinforcement learning. *arXiv preprint arXiv:2003.12613*, 2020.

[94] Xuezhou Zhang, Hrag Gorune Ohannessian, Ayon Sen, Scott Alfeld, and Xiaojin Zhu. Optimal teaching for online perceptrons. In *Advances in Neural Information Processing Systems*, volume 29, pages 1–9, 2016.

[95] Xuezhou Zhang, Xiaojin Zhu, and Laurent Lessard. Online data poisoning attack. *arXiv preprint arXiv:1903.01666*, 2019.

[96] Xiaojin Zhu. Machine teaching: An inverse problem to machine learning and an approach toward optimal education. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[97] Xiaojin Zhu. Machine teaching: An inverse problem to machine learning and an approach toward optimal education. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, pages 4083–4087, 2015.

[98] Xiaojin Zhu, Ji Liu, and Manuel Lopes. No learner left behind: On the complexity of teaching multiple learners simultaneously. pages 3588–3594, 08 2017.

[99] Xiaojin Zhu, Adish Singla, Sandra Zilles, and Anna N. Rafferty. An overview of machine teaching. *CoRR*, abs/1801.05927, 2018.

[100] Xiaojin Zhu, Adish Singla, Sandra Zilles, and Anna N Rafferty. An overview of machine teaching. *arXiv preprint arXiv:1801.05927*, 2018.

Appendix A

Optimal Teaching for Linear BC Agents

A.1 Proofs of Theorems and Lemmas

Lemma A.1 (Proof of Lemma 1)**.** *Optimally teaching the family of consistent linear BC learners is equivalent to optimally teaching the linear version space BC learner.*

Proof. Consider any learner $\mathcal{L} \in \mathfrak{C}$. Given a consistent dataset D, the learner maintains a subset of consistent hypotheses $\mathcal{L}_\ell(D) \subseteq \mathcal{V}(D)$. We note that since $\mathcal{L}_\ell(D) \subseteq \mathcal{V}(D)$, any teaching set for the linear version space(LVS) learner is also a valid teaching set for other learners in the family. Hence, it is sufficient to teach the LVS learner. Secondly, since the LVS learner is also a consistent learner, it is necessary to teach LVS as well. Hence, teaching LVS is both necessary and sufficient to teach the entire family. □

We introduce some definitions before proceeding further. Given a finite set of vectors $\mathcal{X} = \{x_i \in \mathbb{R}^d : i \in [n]\}$, we define the primal cone generated by this set as

$$\text{cone}(\mathcal{X}) = \left\{ \sum_{i \in \mathcal{X}} \lambda_i x_i, \;\; \lambda_i \geqslant 0, \;\; \forall i \in \mathcal{X} \right\}. \tag{A.1}$$

Given any set U, we define the dual cone as

$$\text{cone}^*(U) := \{y \,|\, y^T x > 0, \; \forall x \in U, x \neq 0\}. \tag{A.2}$$

In particular, if the finite set $\mathcal{X}$ has $x_i \neq 0$ for all $i \in [n]$, we have

$$\text{cone}^*(\mathcal{X}) := \{y \,|\, y^T x_i > 0, i = 1, 2, \ldots, n\}. \tag{A.3}$$

We prove some basic properties about cones in $\mathbb{R}^d$.

Proposition A.2. *For any finite sets U, V s.t. $U \subseteq V \subset \mathbb{R}^d$, we have that,*

1. $\text{cone}^*(U) = \text{cone}^*(\text{cone}(U))$.

2. $\text{cone}^*(U) \supseteq \text{cone}^*(V)$.

3. $\text{cone}(U) = \text{cone}(V) \implies \text{cone}^*(U) = \text{cone}^*(V)$.

Proof. 1 For any $w \in \text{cone}^*(U), \langle w, u_i \rangle > 0, \forall u_i \in U \implies \forall i, \lambda_i \geqslant 0, \sum_i \lambda_i u_i \neq 0, \langle w, \sum_i \lambda_i u_i \rangle > 0 \implies w \in \text{cone}^*(\text{cone}(U))$. For the opposite direction, let $\forall \lambda_i \geqslant 0, \sum_i \lambda_i u_i \neq 0, \langle w, \sum_i \lambda_i u_i \rangle > 0$. For a fixed i, choose $\lambda_i = 1$ and $\lambda_j = 0, \forall j \neq i$. Then, we have $\langle w, u_i \rangle > 0, \forall u_i \in U$, thus, $w \in \text{cone}^*(U)$.

2 Now, for second part of the proposition, let $x \in \text{cone}^*(V)$ i.e. $\langle x, v \rangle > 0, \forall v \in V$. Since, $U \subseteq V$, this implies $\langle w, u_i \rangle > 0, \forall u_i \in U$. Thus, $x \in \text{cone}^*(U)$, thus proving the statement.

3 Finally, for the third part, we have that $\text{cone}^*(U) = \text{cone}^*(\text{cone}(U)) = \text{cone}^*(\text{cone}(V)) = \text{cone}^*(V)$, where the first and third equality follows from part 1 of this proposition and second equality follows from the premise. □

A.1.1 Finding extreme rays of primal cone

In the remainder, we assume that the finite set $\mathcal{X} = \{x_i \in \mathbb{R}^d : i \in [n]\}$ contains all nonzero vectors such that $\text{cone}^*(\mathcal{X})$ is nonempty. Our problem is to find a set $\mathcal{X}^* \subset \mathcal{X}$ of minimum cardinality such that $\text{cone}^*(\mathcal{X}^*) = \text{cone}^*(\mathcal{X})$.

Recall that $\text{cone}^*(\mathcal{X}) = \{y \,|\, y^T x_i > 0, i \in [n]\}$. We can define $\text{cone}^*(\mathcal{X})$ alternatively as follows:

$$\begin{aligned} \text{cone}^*(\mathcal{X}) &= \{\alpha z \,|\, \alpha > 0, z \in P(\mathcal{X})\} \\ \text{where } \; P(\mathcal{X}) &:= \{z \,|\, z^T x_i \geqslant 1, i \in \mathcal{X}\}. \end{aligned} \tag{A.4}$$

Proof. Any αz satisfying (A.4) clearly has $z^T x_i > 0$ for all $i \in \mathcal{X}$, so $z \in \text{cone}^*(\mathcal{X})$. Conversely, given any y with $y^T x_i > 0$ for all $i \in [n]$, we set $\alpha = \min_{i \in [n]} y^T x_i > 0$ and $z = y/\alpha$ to get α and z satisfying (A.4). □

The key element of the algorithm is an LP of the following form, for some $x \in \mathcal{X}$:

$$\begin{aligned} \text{LP}(x, \mathcal{X}/\{x\}): \qquad & \min_{w} w^T x \\ \text{subject to } & w^T x_i \geqslant 1 \ \forall i \in \mathcal{X}/\{x\}. \end{aligned} \tag{A.5}$$

Note that this problem can be written alternatively, using the notation of (A.4), as

$$\min_{w} w^T x \text{ subject to } w \in P(\mathcal{X}/\{x\}). \tag{A.6}$$

The dual of (A.5) will also be useful in motivating and understanding the approach:

$$\begin{aligned} \text{LP-Dual}(x, \mathcal{X}/\{x\}): \qquad & \max_{\{\lambda_i : x_i \in \mathcal{X}/\{x\}\}} \sum_{x_i \in \mathcal{X}/\{x\}} \lambda_i \\ \text{s.t.} \quad & \sum_{x_i \in \mathcal{X}/\{x\}} \lambda_i x_i = x, \ \ \lambda_i \geqslant 0 \text{ for all } x_i \in \mathcal{X}/\{x\}. \end{aligned} \tag{A.7}$$

We prove a lemma with several observations.

Lemma A.3 (Proof of Lemma 3)**.** *For a non-empty set $\mathcal{X}$ and an $x \in \mathcal{X}$, we have the following results.*

(i) When (A.5) *is unbounded,* (A.7) *is infeasible, so $x \notin \text{cone}(\mathcal{X}/\{x\})$. Furthermore, $\exists w \in \mathbb{R}^d$ s.t. $w \in \text{cone}^*(\mathcal{X}/\{x\})$ but $w \notin \text{cone}^*(\mathcal{X})$.*

(ii) if (A.5) *has a solution, the optimal objective value must be positive.*

(iii) When (A.5) *has a solution with a positive optimal objective, then $x \in \text{cone}(\mathcal{X}/\{x\})$.*

Furthermore, we can iterate over all candidate elements in $\mathcal{X}$ to find a unique representative set $\mathcal{X}^*$ which contains on vector for each supporting halfspace of $\text{cone}^*(\mathcal{X})$ *and equivalently each extreme ray of* $\text{cone}(\mathcal{X})$.

Proof. (i) From LP duality, when (A.5) is unbounded, then (A.7) is infeasible, giving the first part of the result. For the second part, we note by the feasibility condition of

A.5 that the optimal solution $w^* \in \text{cone}^*(\mathcal{X}/\{x\})$ but since solution is unbounded, i.e., ${w^*}^T x \to -\infty < 0$, we have that, $w^* \notin \text{cone}^*(\mathcal{X})$. Such an x represents a supporting halfspace of $\text{cone}^*(\mathcal{X})$ or, equivalently, an extreme ray of $\text{cone}(\mathcal{X})$.

(ii) If (A.5) were to have a solution with optimal objective 0, then by LP duality, the optimal objective of (A.7) would also be zero, so the only possible value for λ is $\lambda_i = 0$ for all $x_i \in \mathcal{X}/\{x\}$. The constraint of (A.7) then implies that $x = 0$, which cannot be the case, since we assume that all vectors in $\mathcal{X}$ are nonzero.

Note that (A.5) cannot have a solution with a finite *negative* optimal objective value, because by LP duality, (A.7) would also have a solution with negative objective value. However, the value of the objective for (A.7) is non-negative at all feasible points, leading to a contradiction.

(iii) When (A.5) has a solution with positive optimal objective, then LP duality implies that (A.7) has a solution with the same objective. Thus, there are nonnegative λ_i, $x_i \in \mathcal{X}/\{x\}$, not all 0, such that the constraint in (A.7) is satisfied, giving the result.

□

The above lemma completely characterizes the set of supporting halfspaces that define $\text{cone}^*(\mathcal{X})$. To find all the supporting halfspaces we iterate over all $x \in \mathcal{X}$ and remove the ones that are not necessary, i.e., $\text{cone}^*(\mathcal{X}\backslash\{x\}) = \text{cone}^*(\mathcal{X})$, and keep the ones that are necessary to preserve $\text{cone}^*(\mathcal{X})$, i.e., $\text{cone}^*(\mathcal{X}\backslash\{x\}) \subsetneq \text{cone}^*(\mathcal{X})$.

By the lemma, the former happens when $\text{LP}(x, \mathcal{X}\backslash\{x\}) > 0$ while the later happens when $\text{LP}(x, \mathcal{X}\backslash\{x\}) \to -\infty$. This iterative procedure outputs a set $\mathcal{X}^*$ whose vectors uniquely represent a collection of supporting halfspaces of $\text{cone}^*(\mathcal{X})$ in dual space and equivalently a collection of extreme rays of $\text{cone}(\mathcal{X})$ in primal space.

Lemma A.4. *Let* U *and* V *be finite sets with* $U \subseteq V \subseteq \mathbb{R}^d$ *and* $\text{cone}^*(V)$ *is non-empty. Then* $\text{cone}(U) = \text{cone}(V)$ *and* $\text{cone}^*(U) = \text{cone}^*(V)$ *if and only if* U *contains at least one vector on each of the extreme rays of* $\text{cone}(V)$.

Proof. For the sufficiency direction, we note that for a set $U \subseteq V$, if U contains at least one vector on each of the extreme rays of $\text{cone}(V)$ then $\text{cone}(U) = \text{cone}(V)$ (by definition all vectors in a cone can be expressed as a conic combination of all extreme vectors). Furthermore, by Proposition 3, we have $\text{cone}^*(U) = \text{cone}^*(V)$.

For necessary direction, assume that U does not contain a vector $x \in V$ that uniquely represents an extreme ray $cx \in \text{cone}(V)$, then by Lemma A.3 point (i), $\text{cone}(U) \subsetneq \text{cone}(V)$ and correspondingly $\text{cone}^*(U) \supsetneq \text{cone}^*(V)$.

□

Lemma A.5 (Proof of Lemma 2)**.** *A subset $T \subseteq S$ is a valid teaching set if and only if it induces a representative vector on each extreme ray of the primal* $\text{cone}(\Psi(D_S))$.

Proof Sketch. Recall that teaching is successful if the teacher can induce a non-empty subset of consistent version space $\mathcal{V}(D_S)$ on the learner using a subset of states $T \subseteq S$, i.e., $\{\} \neq \mathcal{V}(D_T) \subseteq \mathcal{V}(D_S)$. Since $D_T \subseteq D_S$, we have that $\mathcal{V}(D_T) \supseteq \mathcal{V}(D_S)$ 1.1. Hence, for successful teaching, the teacher has to induce complete $\mathcal{V}(D_S)$ on the learner. Using Lemma A.4 with $V = \Psi(D_S)$, we observe that teaching is successful, i.e., $\text{cone}^*(\Psi(D_T)) = \text{cone}^*(\Psi(D_S))$ if and only if $\Psi(D_T)$ cover each extreme ray of $\text{cone}(\Psi(D_S))$. □

Theorem A.6 (Proof of Theorem 4)**.** *Given an optimal teaching problem instance $(\mathcal{M}, \phi, \pi^*)$ 1.2, our teaching algorithm TIE 1 correctly finds the optimal teaching set D^* and achieves the Teaching Dimension $TD(\pi^*; \mathfrak{C})$.*

Proof. Lemma A.5 tells us that for valid teaching, the teacher must induce at least one feature difference vector on each of the extreme rays of $\text{cone}(\Psi(D_S))$. Since $S, \mathcal{A}$ is finite, there are only a finite number of extreme rays possible. The iterative elimination procedure in **MinimalExtreme** in Algorithm 1 first finds unique representatives for each extreme ray of $\text{cone}(\Psi(D_S))$. This follows from Lemma A.3. Let $\mathcal{X}$ be the surviving set of vectors at the start of an iteration where x is considered. We have that if $x \in \text{cone}(\mathcal{X}/\{x\})$, it will get eliminated by the extreme ray test 1.8. On the other hand, if x is a unique representative for an extreme ray in $\mathcal{X}$, we have $x \notin \text{cone}(\mathcal{X}/\{x\})$, and thus x will not get eliminated. At every iteration, we either eliminate a vector in $\Psi(D_S)$ or that vector is a unique representative for an extreme ray of $\text{cone}(\Psi(D_S))$ and cannot be eliminated. Thus, at the end of the iterative elimination procedure, we recover a set of unique representative vectors Ψ^* for each extreme ray of $\text{cone}(\Psi(D_S))$.

The next step involves finding the smallest subset of states $T \subseteq S$ that can cover all the extreme rays. This is done by a set cover problem defined on lines 4-7 of the **OptimalTeach** procedure in Algorithm 1. The set of unique representatives of extreme rays forms the universe to be covered, and each state defines a subset of representatives for extreme rays

that it can cover. The minimum number of subsets that can cover the entire universe is the minimum number of states that covers all the extreme rays, giving us $T^* \subseteq \mathcal{S}$ as an optimal solution for the teaching problem. For instance, with $|\mathcal{A}| = 2$, every state can induce at most one extreme ray, so picking one state for each extreme ray gives the optimal teaching set. □

Theorem A.7 (Proof of Theorem 5)**.** *Finding an optimal teaching set for teaching a linear version space BC learner is NP-hard in general for instances with action space size* $|\mathcal{A}| > 2$.

Proof. We provide a poly-time reduction from the set cover problem to the optimal teaching version space BC learner problem 1.3. Since the set cover is an NP-hard problem, this implies that optimal teaching is NP-hard to solve as well. Let $P = (U, \{V_i\}_{i \in [n]})$ be an instance of set cover problem where U is the universe and $\{V_i\}_{i \in [n]}$ is a collection of subsets of U. We transform P into an instance of optimal teaching problem $Q = (\mathcal{M}, \phi, \pi^*)$.

Construction: For each subset V_i of P, we create a state s_i of Q. For each element k in the universe U of P, we create an extreme ray vector ψ_k of feature difference vectors in Q. The complete construction is given as follows :

1. $\mathcal{S} = [n], \mathcal{A} = [A]$ where $A = \max_{i \in [n]} |V_i| + 1$.
2. The target policy is $\pi^*(s) = A, \forall s \in \mathcal{S}$.
3. $\Psi = \{\psi_k = (\cos(\frac{2\pi k}{n}), \sin(\frac{2\pi k}{n}), 10) : k \in [|U|]\}$.
4. for each $s \in \mathcal{S}$ we construct feature vectors $\{\phi(s, a) : a \in \mathcal{A}\}$ such that the feature differences map to extreme rays ψ's. Enumerating over element of $V_s := \{V_{s1}, \cdots, V_{s|V_s|}\}$, we define the induced feature difference vectors as,

$$\psi_{sAb} = \psi_{V_{sb}}, \quad \forall b < |V_s| - 1 \tag{A.8}$$

$$\psi_{sAb} = \psi_{V_{s|V_s|}}, \forall |V_s| - 1 \leqslant b \leqslant A - 1 \tag{A.9}$$

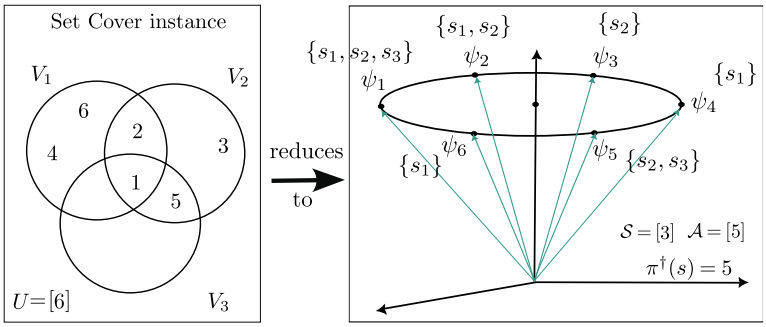


Figure A.1: A reduction example from a set cover problem to othe ptimal teaching LBC problem

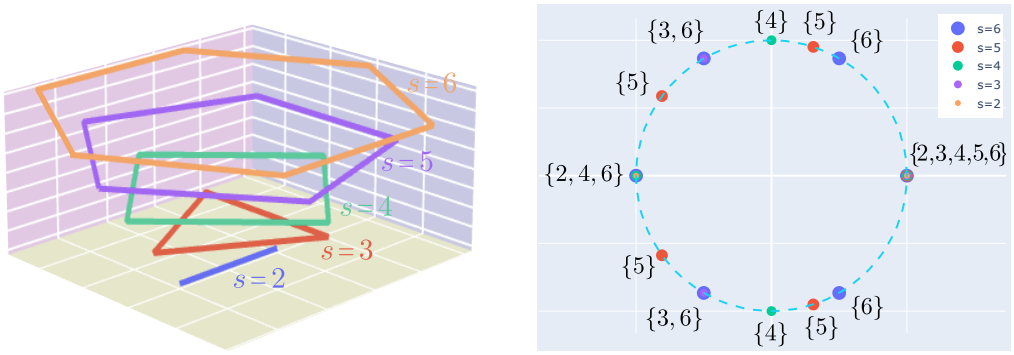
Claim A.8. *A solution of optimal teaching LBC instance* $(\mathcal{S}, \mathcal{A}, \pi^*, \phi)$ *gives a solution to the set cover problem* $(U, \{V_i\}_{i\in[n]})$ *and vice versa.*

Finding a collection of subsets $\{V_i\}_{i\in[n]}$ of smallest size that covers all elements in the universe U is equivalent to selecting a subset of states $\mathcal{S}$ of smallest size that covers all the extreme rays defined by Ψ.

For a solution $\{V_j\}_{j\in T^*}$ s.t. $T^* \subseteq [n]$ to the set cover instance $(U, \{V_i\}_{i\in[n]})$, the set of states indexed by $T^* \subseteq \mathcal{S}$ is a solution to the optimal teaching instance $(\mathcal{S}, \mathcal{A}, \pi^*, \phi)$ and vice versa. The argument follows from a direct translation between two instances. See Figure A.1. □

Theorem A.9 (Proof of Corollary 7)**.** *Consider our optimal teaching problem with infinite state space* $\mathcal{S}$*. Under the assumption that* $\text{cone}(\Psi(D_{\mathcal{S}}))$ *is a closed and convex set with finite extreme rays and the teacher knows the extreme rays to state mapping, our algorithm Greedy-TIE 1 correctly finds an approximately optimal teaching set.*

Proof. Since the convex cone $\text{cone}(\Psi(D_{\mathcal{S}}))$ is closed, we know that extreme rays must be contained in it. Furthermore, an extreme ray must be induced by one of the states. The teacher knows the states that induce each extreme ray or a subset of it, and can construct a set cover problem over a finite extreme ray set to solve the optimal teaching problem. □



(a) **(b)**

Figure A.2: Polygon Tower. a) All feature difference vectors for $n = 6$. b) Top-down view of the extreme vectors of the primal cone for $n = 6$.

A.2 More Experimental Results

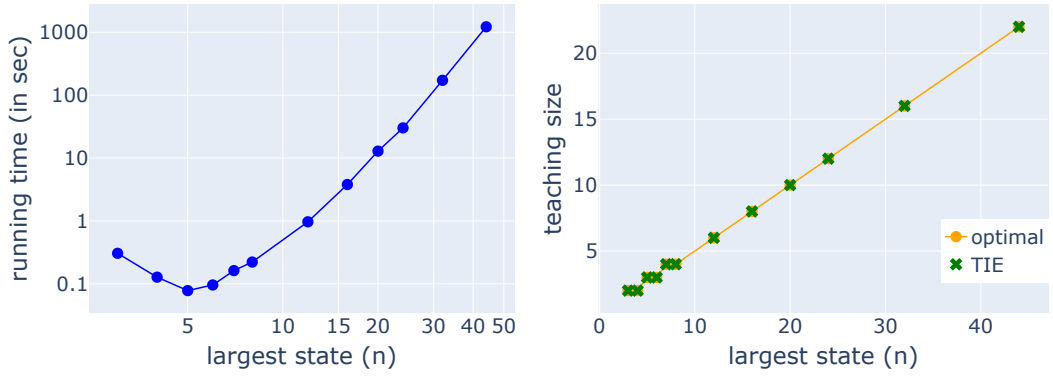
A.2.1 Polygon Tower

Let the state space be $\mathcal{S} = \{2, \ldots, n\}$, the action space be $\mathcal{A} = [n+1]$, the feature function be $\phi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^3$ given by

$$\phi(s, a) = \begin{cases} [0, 0, s] & \text{if } a = n+1 \\ \left[-s \cdot \cos(\frac{2\pi a}{s}), -s \cdot \sin(\frac{2\pi a}{s}), 0\right] & \text{otherwise} \end{cases} \tag{A.10}$$

We note that for a fixed state s, the feature vectors for actions $1 \ldots n$ lie on a polygon of radius s centered around the origin on the xy plane.

Target Policy The teacher wants to teach the target policy $\pi^\dagger$ where $\forall s \in \mathcal{S}, \pi^\dagger(s) = n+1$. The policy is realizable: for example, $w = [0, 0, 1]$ induces this policy. The feature difference vectors induced by $\pi^\dagger$ on $\mathcal{S}$ is given as $\Psi(D_{\mathcal{S}}) = \{[s \cdot \cos(\frac{2\pi a}{s}), s \cdot \sin(\frac{2\pi a}{s}), s] : s \in \mathcal{S}, a \neq n+1\}$. These difference vectors lie on elevated polygons as shown in Figure A.2(a). In particular, state s induces a s-gon of radius s centered at $(0, 0, s)$. Figure A.2(b) shows the top view of the extreme rays of the primal cone $\text{cone}(\Psi(D_{\mathcal{S}}))$. The extreme rays are shown as dots, and the states that cover each extreme ray are labeled.



(a) **(b)**

Figure A.3: Polygon Tower. a) TIE running time on the polygon tower with increasing n. b) The teaching dimension (optimal) vs. the demonstration set size found by TIE. They overlap. In fact, TIE finds the exact correct optimal teaching sets on the polygon tower.

Optimal Teaching The polygon tower problem has an interesting structure that allows us to characterize the minimum demonstration set.

Proposition A.10. *The optimal teaching set* T^* *of the polygon tower consists of all states in* S *that are not divisible by any other states in* S.

Proof. For any pair of states s, s' such that $s' > s$ and $s' \bmod s = 0$, then s' fully covers the induced difference vectors of the characteristic of s so teaching state s is not required if s' is taught. For example, state 6 in Figure A.2(b) covers all the extreme rays induced by states 2, and 3. Conversely, if a state s is not a factor of any other states in S then it must be taught because s induces the extreme ray $[s \cdot \cos(\frac{2\pi}{s}), s \cdot \sin(\frac{2\pi}{s}), s]$ that can only be covered by s. □

We run TIE with greedy set cover on a family of polygon towers with $n \in \{3, 4, 5, 6, 7, 8, 12, 16, 20, 24, 32, 44\}$. We verify TIE's solution to the ground truth minimum demonstration set established in Proposition A.10.

We observe that TIE always recovers the correct minimum demonstration set. This can be observed from the overlap curve of the optimal size of the teaching set (shown in orange) and the size of the teaching set found by TIE (shown in green) in Figure A.2(d).

We also observe that TIE runs quickly. We plot the running time of TIE over instance size n in a log-log plot in Figure A.3(c). For each n, we average the running time over 3 independent trial runs. The straight line of this log-log plot shows that our algorithm

indeed runs in polynomial time. The empirical estimate of the slope of the linear curve (after omitting the first three outlier points for small n) turns out to be 4.67, implying a running time of order $O(n^5)$ on this family of instances. Our algorithm has a worst-case running time of order $O((|\mathcal{S}||\mathcal{A}|)^3)$ and for $|\mathcal{S}| = |\mathcal{A}| = n$ as in this example, it is $O(n^6)$.

A.2.2 Pick the Right Diamond

The MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, P, \gamma, \mu)$ that describes the Block Programming problem is defined as follows :

1. A state $s \in \mathcal{S}$ is specified by an n size board where the cells are indexed $\{1, \cdots, n\}$. Each cell contains one of the four diamonds or is empty, leading to a total of $5^n - 1$ states of non-empty boards.

2. The action space is given as $\mathcal{A} = \{1, \cdots, n\}$ where each action a represents picking an object at location a and removing it from board.

3. The learner receives a reward of -1 for picking the rightmost diamond with the largest edge and -2 for all other actions on a non-empty board. Once the board is empty, it receives a reward of 0. The discount factor γ is 0.9.

4. The environment transitions deterministically to update the board if the agent picked the right object, i.e., the rightmost object with the largest edge otherwise, it remains the same. The initial state distribution μ is uniform on $\mathcal{S}$.

The optimal policy defined by the reward structure above is to pick the diamond in order of decreasing the number of edges. In the case of ties, the rightmost diamond should be picked.

A.2.3 Visual Block Programming in Maze with Repeat Loop

The MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, P, \gamma, \mu)$ that describes the Block Programming problem is defined as follows :

1. A state $s \in \mathcal{S}$ is specified by an $n \times n$ board with a turtle cell and a goal cell $\in [n^2] \times [n^2]$ and a turtle orientation $\in \{L, R, U, D\}$ denoting whether the turtle is facing left, right, up and down, refer to figure 1.6 for an example state. There is also a partial code of up to a constant size c, giving us a total of $4c(n^4 - n^2)$ states.

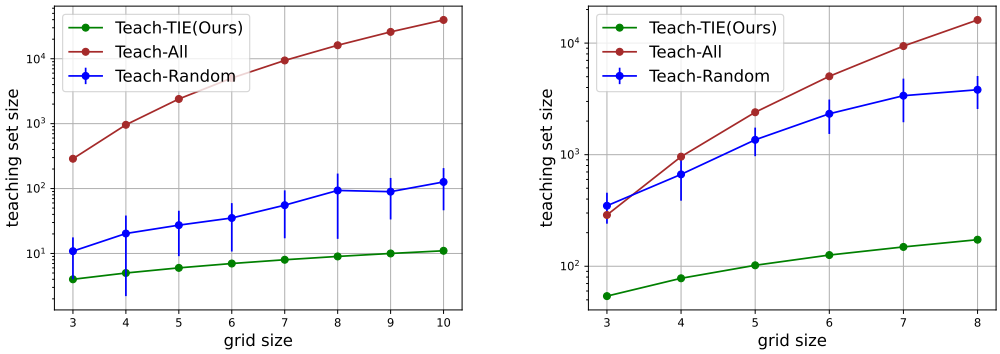


Figure A.4: Performance of TIE compared to other baselines on the visual programming task with local(on the left) and global features(on the right).

2. The abstract action space is given as $\mathcal{A} = \{TL,\ TR,\ MV\} \cup \{R_k\text{-}MV : k \in \{3, \cdots, n-1\}\}$ where *TL, TR, MV* represent simple code block that when executed allows the turtle to turn left, right and move forward actions respectively and R_k-MV represents a complex block of repeat loop that allows the turtle to move forward by k-step. In total, we have n actions where taking an action means adding the corresponding code block to the partial code in the state.

3. The learner receives a reward of -1 for using a simple code block action i.e. action $a \in \{TL,\ TR,\ MV\}$ and -2 for taking complex action R_k-*MV*. The disocunt factor γ is 0.9.

4. The environment transitions deterministically to update the orientation/position of the agent based on its chosen action. The initial state distribution μ is uniform on $\mathcal{S}$.

The goal of the teacher is to teach the optimal policy to write a succinct piece of code which, when executed, helps to lead the learner to the goal cell. The teacher has to do this by showing the smallest size of the (state, action) demonstration dataset.

A.3 Feature Representation for Visual Programming

A.3.1 Local Feature Representation

This feature representation effectively captures the spatial relationship between the turtle and the goal, as well as the impact of different actions.

A.3.1.1 State and Action description

- `board`: A 2D array representing the game board with cells indicating the agent's orientation (U for up, D for down, L for left, R for right).
- `agent_pos`: A tuple (x, y) represents the agent's current position on the board.
- `goal_pos`: A tuple (x, y) representing the goal's position on the board.
- `action`: A string representing the specific action taken by the agent (e.g., *'TL'*, *'TR'*, *'MV'*, *'R_k-MV'* etc.).

A.3.1.2 Feature Vector Construction

1. **Relative Quadrant of Goal**: Compute the relative quadrant of the goal from the agent's orientation.
2. **Forward Distance**: Compute the distance from the agent to the goal in the direction the agent is facing.

We refer interested readers to the supplementary material of the published paper for the code.

Appendix B

Optimal Teaching of RL Agents

B.1 The Computational Complexity of Finding METaL

In this section, we discuss another aspect of teaching, namely the computational complexity of finding the exact minimum expected teaching length of a particular teaching problem instance, i.e. $\mathtt{METaL}(M, L, Q_0, \pi^\dagger)$. Note this differs from TDim in that it is instance-specific.

For Level 1 and Level 2 teachers, the exact METaL can be found with polynomial-time algorithms Alg. 6 and Alg. 7. Now, we show that for the less powerful Level 3 teacher, finding METaL of a particular instance is NP-hard. In particular, it is as hard as the Asymmetric TSP problem.

Definition B.1. *An Asymmetric TSP problem [83], characterized by a directed graph* $G = (V, E)$ *and a starting vertex* $v \in V$*, is defined as finding the minimum length path that starts from* v *and visits all vertices* $v' \in V$ *at least once.*

Theorem B.2. *Finding the METaL of a Level 3 teaching problem instance is at least as hard as the Asymmetric Traveling Salesman Problem(ATSP), which is NP-hard; This also means that the best polynomial-time approximation algorithm can only achieve a constant-factor approximation.*

Proof. We show a polynomial-time reduction from the ATSP problem to a Level 3 METaL problem. Specifically, we show that for every ATSP problem instance $G = (V, E)$, there exists a Level 3 METaL problem instance $(M, L, Q_0, \pi^\dagger)$ such that the ATSP problem instance has a solution l if and only if the corresponding METaL instance has a solution l.

The reduction is as follows. Given an ATSP problem instance $\{\text{Graph } G = (V, E),\ \text{start vertex} = s_0\}$, we provide a construction to a level 3 METal problem instance $(M, L, Q_0, \pi^\dagger)$. We start by constructing the MDP first. The vertex set V forms the state space of the MDP. Each state s has exactly two actions $a^{(0)}$ and $a^{(1)}$. The support of the transition probability distributions $P(\cdot \mid s, a^{(0)})$ and $P(\cdot \mid s, a^{(1)})$ are the same: they are the outgoing edges of s in the graph G. The exact value of these probabilities and the reward function does not matter, since a level 3 teacher has the power to override them. The initial state distribution μ_0 is concentrated on s_0. We construct a Q_0 that favors action $a^{(0)}$ in each state, and the target policy $\pi^\dagger(s) = a^{(1)}$ for each state $s \in \S$. The horizon is $H = D^2$, where D is the diameter of the graph G. The learner is in $\mathcal{L}$.

Claim 1: If an ATSP problem instance $\{G = (V, E), s_0\}$ has a solution l, then the level 3 METaL problem instance $(M, L, Q_0, \pi^\dagger)$ has a solution l.

To verify Claim 1, note the teacher needs to make the learner visit every state exactly once to teach the target action $a^{(1)}$ in that state. This is because initially every state is untaught (by construction Q_0 prefers $a^{(0)}$). Further, each state s has exactly two actions and no matter which action the learner takes, the teacher can provide a suitable reward to push the target action $a^{(1)}$ to the top of Q-value ordering. If the ATSP problem has a solution $s_{i_0} = s_0 \to s_{i_1} \to \cdots s_{i_{l-1}}$, it is possible for the teacher to provide the state transitions $s_{i_0} = s_0 \to s_{i_1} \to \cdots s_{i_{l-1}}$ that visit all the states in the least number of time steps and thus teach the target policy optimally. This is because for every edge $s_i \to s_j$ in the graph, the transition $P(\cdot \mid s_i, a)$ supports s_j for both actions.

Claim 2: If the level 3 METaL problem instance $(M, L, Q_0, \pi^\dagger)$ has a solution l, then the ATSP problem instance $\{G = (V, E), s_0\}$ has a solution l.

We prove this by contradiction. Let say the METal problem instance $(M, L, Q_0, \pi^\dagger)$ has a solution l. Clearly, all states must have been visited in this optimal teaching length l at least once. So, the corresponding ATSP problem instance must have a solution $\leqslant l$. But if ATSP has a solution $m < l$, by Claim 1, the METaL problem instance will have a solution $m < l$, thus a contradiction. Hence, the ATSP problem has a solution l.

By establishing this reduction, we prove that the METaL problem for a level 3 teacher is at least as hard as the ATSP problem, which is itself NP-hard. □

B.2 Level 1: Algorithm and Proof

Algorithm 6: Optimal Level 1 Teaching Algorithm

def Teach($M, L, Q_0, \pi^\dagger$**):**

1: A state s needs to be taught if $Q_0(s, \pi^\dagger(s)) \leqslant \max_{a \neq \pi^\dagger(s)} Q_0(s, a)$. Terminate if the MDP has no state to be taught. Otherwise, arbitrarily order all MDP states that need to be taught as $s^{(0)}, s^{(1)}, \cdots, s^{(n)}$ where $0 \leqslant n \leqslant S-1$.
2: The teacher provides the state $s_0 \leftarrow s^{(0)}$.
3: **for** $t = 0, 1, \cdots, n$ **do**
4: The agent performs an action according to its current behavior policy $a_t \leftarrow \pi_t(s_t)$.
5: The teacher replaces the chosen action with target action $a_t \leftarrow \pi^\dagger(s_t)$.
6: The teacher provides the reward r_t, and next state s_{t+1}
7: where $s_{t+1} \leftarrow s^{(\min(t+1,n))}$
8: $r_t : Q_{t+1}(s_t, a_t) > \max_{a \neq a_t} Q_{t+1}(s_t, a)$.
9: The agent performs an update $Q_{t+1} \leftarrow f(Q_t, e_t)$ using experience $e_t = (s_t, a_t, r_t, s_{t+1})$

Proof of Theorem 3.6. For a level 1 teacher, the worst-case teaching problem instance is the one in which for all states $s \in \S$, the target action $\pi^\dagger(s)$ is not the top action in the $Q_0(s, \cdot)$. In that case, the teacher would need to make the learner visit each state s at least once so that the learner has a chance to learn $\pi^\dagger$ as s, i.e. to produce and maintain the eventual condition $Q_T(s, \pi^\dagger(s)) > \max_{a \neq \pi^\dagger(s)} Q_T(s, \cdot)$. Thus, $\mathtt{TDim} \geqslant S$. On the other hand, a level-1 teacher can teach a state in just a single visit to it by replacing the agent chosen action with the target action and rewarding it with a sufficiently high reward (step 8 in the algorithm). Further, at any time step, it can also make the agent transition to an untaught state to teach the target action in that state. Thus, for the worst teaching problem instance, the level-1 teacher can teach the target policy in S steps and hence $\mathtt{TDim} = S$. □

B.3 Level 2: Algorithm and Proof

Algorithm 7: Optimal Level 2 Teaching Algorithm

def Teach($M, L, Q_0, \pi^\dagger$**):**

1: A state s needs to be taught if $Q_0(s, \pi^\dagger(s)) \leqslant \max_{a \neq \pi^\dagger(s)} Q_0(s, a)$. Terminate if the MDP has no state to be taught. Otherwise, arbitrarily order all MDP states that need to be taught as $s^{(0)}, s^{(1)}, \cdots, s^{(n)}$ where $0 \leqslant n \leqslant S-1$.
2: $t \leftarrow 0, i \leftarrow 0$, the teacher provides initial state $s_0 \leftarrow s^{(0)}$
3: **while** $i \leqslant n$ **do**
4: The agent picks a randomized action $a_t \leftarrow \pi_t(s_t)$.
5: **if** $a_t = \pi^\dagger(s_t)$ **then**
6: $s_{t+1} \leftarrow s^{(\min(i+1,n))}$
7: $i \leftarrow i+1$ // move on to the next state
8: $r_t : Q_{t+1}(s_t, a_t) > \max_{a \neq a_t} Q_{t+1}(s_t, a)$ //promote action $\pi^\dagger(s_t)$ to top
9: **else**
10: **if** $\{a\text{: } Q_t(s_t, a) \geqslant Q_t(s_t, \pi^\dagger(s_t))\} = \{a_t, \pi^\dagger(s_t)\}$ **then**
11: $s_{t+1} \leftarrow s^{(\min(i+1,n))}$
12: $i \leftarrow i+1$ // move on to the next state
13: **else**
14: $s_{t+1} \leftarrow s^{(i)}$ // stay at this state
15: $r_t : Q_{t+1}(s_t, a_t) < \min_{a \neq a_t} Q_{t+1}(s_t, a)$ // demote action a_t to bottom
16: The agent performs an update $Q_{t+1} \leftarrow f(Q_t, e_t)$ with experience $e_t = (s_t, a_t, r_t, s_{t+1})$
17: $t \leftarrow t+1$

Remark: Line 10 checks whether a_t is the only no-worse action than $\pi^\dagger(s_t)$: if it is, its demotion also completes teaching at s_t.

Proof of Lemma 3.7. We focus on teaching the target action $\pi^\dagger(s)$ at a particular state s. In general let there be $n \in \{1, \ldots, A-1\}$ other actions better than $\pi^\dagger(s)$ in $Q(s, \cdot)$. For simplicity, we assume no action is tied with $\pi^\dagger(s)$, namely

$$Q(s, a_{i_1}) \geqslant \cdots \geqslant Q(s, a_{i_n}) > Q(s, a_{i_{n+1}} = \pi^\dagger(s)) > Q(s, a_{i_{n+2}}) \geqslant \cdots \geqslant Q(s, a_{i_A}). \quad \text{(B.1)}$$

Define the upper action set $U := \{a_{i_1} \cdots a_{i_n}\}$ and the lower action set $U := \{a_{i_{n+2}} \cdots a_{i_A}\}$. Define $T(n)$ to be the expected number of visits to s to teach the target action $\pi^\dagger(s)$ at state s, given that initially there are n other actions better than $\pi^\dagger(s)$. By "teach" we mean move the n actions from U to L. When the agent visits s it takes a randomized action according to $a_t \leftarrow \pi_t(s)$, which can be any of the A actions. We consider three cases:

Case 1: $a_t \in U$, which happens with probability $1-\varepsilon+(n-1)\frac{\varepsilon}{A-1}$. The teacher provides a reward to demote this action to the bottom of $Q(s,\cdot)$. Therefore, U has one less action after this one teaching step, and recursively needs $T(n-1)$ expected steps in the future.

Case 2: $a_t = \pi^\dagger(s)$, which happens with probability $\frac{\varepsilon}{A-1}$. The teacher provides a reward to promote a_t to the top of $Q(s,\cdot)$ and terminates after this one teaching step (equivalently, $T(0) = 0$).

Case 3: $a_t \in L$, which happens with probability $(A-n-1)\frac{\varepsilon}{A-1}$. The teacher can do nothing to promote the target action $\pi^\dagger(s)$ because a_t is already below $\pi^\dagger(s)$. Thus, the teacher provides a reward that keeps it that way. In the future, it still needs $T(n)$ steps.

Collecting the 3 cases together we obtain

$$T(n) = 1 + \left[\left(1-\varepsilon+(n-1)\frac{\varepsilon}{A-1}\right)T(n-1) + \frac{\varepsilon}{A-1}T(0) + (A-n-1)\frac{\varepsilon}{A-1}T(n)\right]. \tag{B.2}$$

Rearranging,

$$\left(1-\frac{A-n-1}{A-1}\varepsilon\right)T(n) = 1+\left(1-\frac{A-n}{A-1}\varepsilon\right)T(n-1). \tag{B.3}$$

This can be written as

$$\left(1-\frac{A-1-n}{A-1}\varepsilon\right)T(n) = 1+\left(1-\frac{A-1-(n-1)}{A-1}\varepsilon\right)T(n-1). \tag{B.4}$$

This allows us to introduce

$$B(n) := \left(1-\frac{A-1-n}{A-1}\varepsilon\right)T(n) \tag{B.5}$$

with the relation

$$B(n) = 1 + B(n-1). \tag{B.6}$$

Since $T(0) = 0$, $B(0) = 0$. Therefore, $B(n) = n$ and

$$T(n) = \frac{n}{1 - \frac{A-1-n}{A-1}\varepsilon}. \tag{B.7}$$

It is easy to show that the worst case is $n = A-1$, where $T(A-1) = A-1$ regardless of the value of ε. This happens when the target action is originally at the bottom of $Q(s,\cdot)$. □

Proof of Theorem 3.8. We construct a worst-case RL teaching problem instance. We design Q_0 so that for each state $s \in \S$ the target action $\pi^\dagger(s)$ is at the bottom of $Q_0(s,\cdot)$. By Lemma 3.7 the teacher needs to make the agent visits each state $A-1$ times in expectation. Thus a total $S(A-1)$ expected number of steps will be required to teach the target policy to the learner. □

B.4 Level 3 and 4: Algorithm and Proofs

Algorithm 8: The NavTeach Algorithm: Initialization

def Init(M):

$D \leftarrow \infty$. // select the initial state with the shortest tree
for s in $\{s \mid \mu_0(s) > 0\}$ **do**
  Construct a minimum depth directed tree $T(s)$ from s to all states in the underlying directed graph of M, via breadth first search from s. Denote its depth as $D(s)$.
  if $D(s) < D$ **then**
    depth $D \leftarrow D(s)$; root $s^S \leftarrow s$.
Let $s^1, \ldots, s^{S-1}, s^S$ correspond to a post-order depth-first traversal on the tree $T(s^S)$.

Proof of Tighter Lower and Upper bound for Level 3 Teacher. We hereby prove the claimed matching $\Theta\left((S-D)AH(1-\varepsilon)^{-D} + H\frac{1-\varepsilon}{\varepsilon}[(1-\varepsilon)^{-D} - 1]\right)$ lower and upper bounds for Level 3 Teacher. The key observation is that for an MDP with state space size S and diameter D, there must exist D states whose distance to the starting state is $0, 1, \ldots, D-1$,

Algorithm 9: The NavTeach Algorithm: Complete Algorithm

def NavTeach($M, L, Q_0, \pi^\dagger, \delta$**):**

$t \leftarrow 0, s_t \leftarrow s^S$, ask for randomized agent action $a_t \leftarrow \pi_t(s_t)$
for $i = 1, \ldots, S$ **do**
  // subtask i: teach target state s^i with the help of navigation path p^i
  Let $p^i \leftarrow [s_{i_0} = s^S, s_{i_1}, ..., s_{i_d} = s^i]$ be the ancestral path from root s^S to s^i in tree $T(s^S)$
  while $\arg\max_a Q_t(s^i, a) \neq \{\pi^\dagger(s^i)\}$ **do**
    if $s_t = s^i$ **then**
      // $s_t = s^i$: the current subtask, establish the target policy.
      Randomly pick $s_{t+1} \in \{s' : P(s' \mid s_t, a_t) > 0\}$.
      if $a_t = \pi^\dagger(s_t)$ **then**
        $r_t \leftarrow$ CarrotStick('promote', $a_t, s_t, s_{t+1}, Q_t, \delta$)
      else
        $r_t \leftarrow$ CarrotStick('demote', $a_t, s_t, s_{t+1}, Q_t, \delta$)
    else if $s_t \in p^i$ **then**
      // build navigation if MDP allows
      if $P(p^i.\text{next}(s_t) \mid s_t, a_t) > 0$ **then**
        $s_{t+1} \leftarrow p^i.\text{next}(s_t)$.
        $r_t \leftarrow$ CarrotStick('promote', $a_t, s_t, s_{t+1}, Q_t, \delta$).
      else
        Randomly pick $s_{t+1} \in \{s' : P(s' \mid s_t, a_t) > 0\}$.
        $r_t \leftarrow$ CarrotStick('demote', $a_t, s_t, s_{t+1}, Q_t, \delta$).
    else
      // s_t is off subtask i or an already taught state, maintain the $Q(s_t, a_t)$
      Randomly pick $s_{t+1} \in \{s' : P(s' \mid s_t, a_t) > 0\}$.
      $r_t \leftarrow$ CarrotStick('maintain', $a_t, s_t, s_{t+1}, Q_t, \delta$).
    Give experience $e_t \leftarrow (s_t, a_t, r_t, s_{t+1})$ to the agent.
    $t \leftarrow t + 1$
    if $t\%H = H - 1$ **then**
      $s_t \leftarrow s^S$. // episode reset
    Ask for randomized agent action $a_t \leftarrow \pi_t(s_t)$

Algorithm 10: The NavTeach Algorithm: Carrot Stick Procedure

def CarrotStick(g, a, s, s', Q, δ**):**

1: // make a unambiguously the worst action or the best action (with margin δ)or keep it as it is.
2: **if** $g =$ 'promote' **then**
3: Return $r \geqslant 0$ such that $Q_{t+1}(s,a) = \arg\max_{b \neq a} Q_{t+1}(s,b) + \delta$ after $Q_{t+1} = f(Q_t, (s,a,r,s'))$.
4: **else if** $g =$ 'demote' **then**
5: Return $r \leqslant 0$ such that $Q_{t+1}(s,a) = \arg\min_{b \neq a} Q_{t+1}(s,b) - \delta$ after $Q_{t+1} = f(Q_t, (s,a,r,s'))$.
6: **else**
7: Return r such that $Q_{t+1}(s,a) = Q_t(s,a)$ after $Q_{t+1} = f(Q_t, (s,a,r,s'))$.

respectively. As a result, the total time to travel to these states is at most

$$\sum_{d=0}^{D-1} H(\frac{1}{1-\varepsilon})^d = H\frac{1-\varepsilon}{\varepsilon}[(\frac{1}{1-\varepsilon})^D - 1] \tag{B.8}$$

In the lower bound proof of Theorem 3.9, we only count the number of teaching steps required to teach the tail states. Now, if we assume in addition that the neck states also need to be taught, and the target actions are similarly at the bottom of the $Q_0(s,a)$, then it requires precisely an additional $H\frac{1-\varepsilon}{\varepsilon}[(\frac{1}{1-\varepsilon})^D - 1]$ steps to teach, which in the end gives a total of

$$(S-D-1)(A-1)H(\frac{1}{1-\varepsilon})^D + H\frac{1-\varepsilon}{\varepsilon}[(\frac{1}{1-\varepsilon})^D - 1] \tag{B.9}$$

steps.

In the upper bound proof of Theorem 3.10 we upper bound the distance from s_0 to any state by D. However, based on the observation above, at most $S-D$ states can have distance D from s_0, and the rest D states must have distance $0, 1, ..., D-1$. This allows us to upperbound the total number of teaching steps by

$$(2S-1-2D)(A-1)H(\frac{1}{1-\varepsilon})^D + H\frac{1-\varepsilon}{\varepsilon}[(\frac{1}{1-\varepsilon})^D - 1] \tag{B.10}$$

These two bounds matches up to a constant of 2, and thus gives a matching $\Theta\left((S-D)AH(1-\varepsilon)^{-D} + H\frac{1-\varepsilon}{\varepsilon}[(1-\right.$

lower and upper bound. Setting $\varepsilon = 0$ (requires taking the limit of $\varepsilon \to 0$) induces Corollary 3.11. □

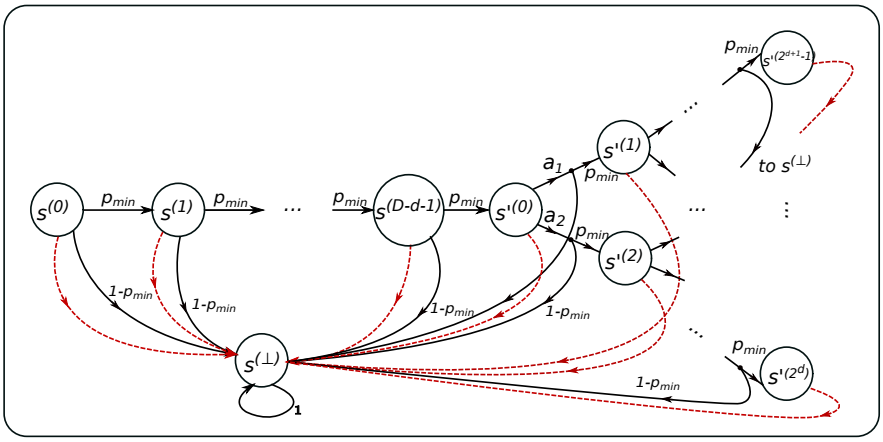


Figure B.1: The "peacock tree" MDP

Proof of Theorem 3.12. We construct a hard level 4 teaching problem instance, very similar to "peacock MDP" and call it "peacock tree MDP". We then show that this MDP admits the given lower bound. The "peacock tree MDP" has a linear chain of length $D-d-1$(the "neck") and a d depth binary tree(the "tail") attached to the end of the neck. For a given (S, D), we can always find d such that $2^d + (D-d+1) \leqslant S \leqslant 2^{d+1} + (D-d)$. Note that the depth of this MDP is D. To simplify the analysis of the proof, from now on, we will assume that the binary tree is complete and full, i.e., $S = 2^{d+1} + (D-d)$.

As in the case of "peacock MDP", every state has A actions. The action a_1 in the chain transits to the next state with probability p_{min} and to the absorbing state $s^{(\perp)}$ with probability $1-p_{min}$. The action a_1 in the non-leaf states of the binary tree transits to its top child with probability p_{min} and to $s^{(\perp)}$ with probability $1-p_{min}$, the action a_2 there transits to the bottom child with probability p_{min} and to $s^{(\perp)}$ with probability $1-p_{min}$. All other $A-1$ actions in the non-leaf states and the chain states lead to $s^{(\perp)}$ with probability 1. Further, all A actions in the leaf states lead to $s^{(\perp)}$ with probability 1. The target policy is to select a_1 at every state. We consider an initial Q_0 which favors the target policy at all non-leaf and chain states. For all the leaf states s, the target action a_1 is $\arg\min_a Q_0(s,a)$, namely at the bottom, and needs to be taught.

For the lower bound analysis, we consider teaching each leaf state when the traversal path to it is already optimal (Note that in reality, the path has to be taught for each leaf state, but that will eventually add to the lower bound, so we omit it for this analysis). For a leaf state s, there exists a path from the root to it. This requires the teacher to provide the correct transition to the next state along the path, and the learner to choose actions a_1 all along the chain and then a combination of a_1 and a_2 actions to reach that leaf s. Given that the traversal path to the leaf is already optimal, a successful episode consists of the learner choosing the greedy action at each step and the teacher transitioning the learner to the correct next state on the path to the leaf, which happens with a probability of $(p_{min}(1-\varepsilon))^D$. Thus, the expected number of episodes required to make the learner visit the leaf and teach it once there is $(\frac{1}{p_{min}(1-\varepsilon)})^D$. Note that in a successful episode, the learner takes D steps to reach the leaf and the rest of the steps in that episode is wasted, thus accounting for a total of H steps. Similarly, any failed episode wastes a total of H steps. Hence, the expected number of steps required to visit and teach a leaf state once is at least $H(\frac{1}{p_{min}(1-\varepsilon)})^D$. The teacher has to make the learner visit all 2^d leaf states $A-1$ times in expectation (since by our construction, the target action of each leaf is at the bottom of the Q-value ordering). Collectively, this would require at least $2^d(A-1)H(\frac{1}{(p_{min}(1-\varepsilon)})^D$ steps. We note that, $S = 2^{d+1} + (D-d) \leqslant 2^{d+1} + D = 2 \cdot 2^d + D \implies 2^d \geqslant \frac{1}{2}(S-D)$. Thus, the expected number of steps to teach the target policy is $\geqslant \frac{1}{2}(S-D)(A-1)H(\frac{1}{p_{min}(1-\varepsilon))^D}) \implies TDim \geqslant \Omega((S-D)AH(\frac{1}{p_{min}(1-\varepsilon))^D}))$. □

Proof of Theorem 3.13. The proof follows similarly to the upper bound proof for the teaching dimension of a level 3 teacher and uses the NavTeach algorithm algorithm 9. For a given MDP, the teacher first creates a breadth-first tree and then starts teaching the states in a post-order depth-first traversal. Note that the breadth-first tree is still constructed using the transition edges that are supported by the underlying MDP. A level 4 teacher, while transitioning out from a particular state, can only choose a desired transition-edge with a probability $\geqslant p_{min}$. Thus, the probability that the teacher can make the learner transit from one state to another using a greedy action chosen by the learner is at least $p_{min}(1-\varepsilon)$.

The teaching goal is broken into S subtasks, one for each state. The sub-task for a state further consists of teaching a navigation path to reach that state and then teaching the target action in that state. Because of the post-order depth-first teaching strategy, a large part of the navigation path is shared between two subtasks. Also, this strategy requires a

navigation action at each non-leaf state to be taught just once. We further note that in the depth-first teaching strategy, a navigation action from a parent state s_i to a child state s_j is taught only after a navigation path to the parent s_i is laid. Similarly, the target action at a state s_j is taught only after a navigation path to it is laid. Thus, the expected number of steps required to reach a state at depth i and teach once there is at most $(\frac{1}{p_{min}(1-\varepsilon)})^i$. For a simpler analysis, we assume that once the agent falls off the path leading to the target state, the remaining steps in that episode are wasted. Similarly, once an agent reaches a target state and is taught by the teacher, the remaining episode steps are wasted. Thus, the expected number of steps required to visit a state at depth i and teach the navigation action there is $(A-1)H(\frac{1}{p_{min}(1-\varepsilon)})^i \leqslant (A-1)H(\frac{1}{p_{min}(1-\varepsilon)})^D$. Noting the fact that there are at most $S-1$ non-leaf states and the teacher needs to teach the navigation action at each of them exactly once, the expected number of steps required to teach all the navigation actions is at most

$$(S-1)(A-1)H\Big(\frac{1}{p_{min}(1-\varepsilon)}\Big)^D. \tag{B.11}$$

Similarly, the expected number of steps required to visit a state at depth i and teach the target action there is $(A-1)H(\frac{1}{p_{min}(1-\varepsilon)})^i \leqslant (A-1)H(\frac{1}{p_{min}(1-\varepsilon)})^D$. Adding it up, the expected number of steps required to teach the target action at all states is at most

$$S(A-1)H\Big(\frac{1}{p_{min}(1-\varepsilon)}\Big)^D. \tag{B.12}$$

Combining B.11 and B.12, we conclude that the expected number of steps required to teach the target policy using algorithm 9 is at most

$$(2S-1)(A-1)H(\frac{1}{p_{min}(1-\varepsilon)})^D \implies TDim \leqslant O(SAH\Big(\frac{1}{p_{min}(1-\varepsilon)}\Big)^D). \tag{B.13}$$

□

Remark: A more careful analysis that leads to a tight lower and upper bound is also possible for the level 4 teacher, but the calculation and the eventual bound one gets become much more complicated, and thus we defer it to future works.

B.5 Generalization to SARSA

Algorithm 11: Machine Teaching Protocol for SARSA

Entities: MDP environment, learning agent with initial Q-table Q_0, teacher.

1: MDP draws $s_0 \sim \mu_0$ after each episode reset. But the teacher **may** override s_0.
2: **for** $t = 0, ..., H-1$ **do**
3: The agent picks an action $a_t = \pi_t(s_t)$ with its current behavior policy π_t. But the teacher **may** override a_t with a teacher-chosen action.
4: **if** $t = 0$ **then**
5: The agent updates $Q_{t+1} = Q_t$.
6: **else**
7: The agent updates $Q_{t+1} = f(Q_t, e_t)$ from experience $e_t = (s_{t-1}, a_{t-1}, r_{t-1}, s_t, a_t)$.
8: The MDP evolves from (s_t, a_t) to produce immediate reward r_t and the next state s_{t+1}. But the teacher **may** override r_t or move the system to a different state s_{t+1}.

SARSA is different from standard Q-learning in that its update is delayed by one step. In time step t, the agent is updating the (s_{t-1}, a_{t-1}) entry of the Q table, using experience $e_t = (s_{t-1}, a_{t-1}, r_{t-1}, s_t, a_t)$. This delayed update makes the student learn slowly. In particular, we show that it can take twice as many visits to a state to enforce the target action compared to Q-learning.

Lemma B.3. *For a Level 2 Teacher, any SARSA learner, and an MDP family $\mathcal{M}$ with action space size A, it takes at most $2A-2$ visits in expectation to a state s to teach the desired action $\pi^\dagger(s)$ on s.*

Proof Sketch: The key in proving Lemma B.3 is to see that if the agent visits the same state two times in a row, then the lesson provided by the teacher during the first visit has not been absorbed by the learner, and as a result, during the second visit, the learner will still prefer the same (undesirable) action. This, in the worst case ($\varepsilon = 0$), will be a completely wasted time step, which implies that the total number of visits required will double compared to Q-learning, giving us $2A-2$. □

The wasted time step in Lemma B.3 will only occur when the agent visits one state twice in a roll. This can be avoided in Level 1 and 2 teachers as long as $S \geqslant 2$. Therefore, the teaching dimension for level 1 and 2 teachers will only increase by 1 due to the delayed update of the learner. For Level 3 and Level 4 teachers, the new Lemma B.3 only results in

at most two times increase in the teaching dimension, which does not change the order of our results. Therefore, Level 3 and Level 4 results still hold for SARSA agents.

Appendix C

Nurture-then-Nature Teaching

C.1 Proofs of Theorems and Lemmas

C.1.1 Finite Binary Hypothesis Class

Budgeted Maximum Coverage Problem: Given a finite universe of items U and a finite collection of subsets of the universe $V = \{V_x \subseteq U : x \in \mathcal{X}\}$, where $\mathcal{X}$ is a finite set, the goal is to find a subcollection of V of size upto B, that covers maximum number of elements of U. This problem is known to be NP-hard [43]. However, a greedy algorithm that greedily chooses a subset to reduce U maximally is approximately optimal, and achieves an approximation ratio of $1 - \frac{1}{e}$. This leads to the following guarantee on the optimal reduction of the version space size.

Theorem C.1 (Theorem 2 of main text)**.** *There exists an algorithm that reduces the version space size of a finite hypothesis class up to an approximation ratio of* $1 - \frac{1}{e}$.

Proof. We note that each demonstration $(x, h^*(x))$ eliminates a subset of hypothesis, $V_x = \{h \in \mathcal{H} : h(x) \neq h^*(s)\}$ from $\mathcal{H}$. Maximally reducing the size of the version space requires eliminating as many hypotheses from $\mathcal{H} \backslash \{h^*\}_{P_{\mathcal{X}}}$ as possible under budget B. This is nothing but budgeted maximum coverage problem with $U = \mathcal{H} \backslash \{h^*\}_{P_{\mathcal{X}}}$, $\{V_x : x \in \mathcal{X}\}$ as defined above and the result follows from [43]. □

C.1.2 Axis-aligned rectangles on $\mathbb{Z}^2-$grid

Definition C.2 (Extending h to h')**.** *A rectangle h' is said to extend h if $\{x : h(x) = 1\} \subsetneq \{x : h'(x) = 1\}$. An extension can occur along one or more of the four sides[1] of the rectangle — namely, top, bottom, left, or right.*

Definition C.3 (Fixing sides and degrees of freedom)**.** *Given a version space $\mathcal{V}$ that contains h^*, we define the **degrees of freedom** of $\mathcal{V}$ w.r.t. h^* as the number of sides along which h^* can be **extended** to another rectangle h', such that $h' \in \mathcal{V}$. If no such extension is possible along a particular side, we say that that side is **fixed** in $\mathcal{V}$.*

Remark C.4 (Reducing degrees of freedom)**.** *The original hypothesis class has four degrees of freedom corresponding to the four sides along which h^* can be independently extended while still remaining within the version space $\mathcal{V}$. When $k \in \{1,2,3,4\}$ sides of h^* are **fixed** in $\mathcal{V}$, the degrees of freedom of the version space reduce by k.*

Lemma C.5. *Let $\mathcal{H}$ be the class of axis-aligned rectangles on $\mathbb{Z}^2$-grid. For any rectangle, $h \in \mathcal{H}$, fixing one (two) of its sides requires two (three) labelled examples.*

Proof. (Fixing one side). Without loss of generality, consider ***fixing*** $y^*_{\min}$, corresponding to the bottom side of the target rectangle h^*. This can be done using exactly two labelled examples: $\{((x, y^*_{\min}), +), ((x, y^*_{\min} - 1), -)\}$ where $x_{\min} \leqslant x \leqslant x_{\max}$. The '$+$' and '$-$' examples force every consistent hypothesis h to satisfy $y_{\min} \leqslant y^*_{\min}$ and $y_{\min} > y^*_{\min} - 1$, respectively, thereby enforcing $y_{\min} = y^*_{\min}$. Thus, no ***extensions*** are possible along the bottom side, thereby ***fixing*** this side. With only one labelled example there is always an ***extension*** h of h^* possible along the bottom side such that h is consistent with the labelled example — enlarge downward (shrink upward) given a single positive (negative) example. Hence, two examples are necessary.

(Fixing two sides). Naively, by the reasoning above, we can use four examples to ***fix*** two sides. But we can do better by using just three examples: labeling a corner point of the rectangle as '$+$' and two adjacent points just outside the rectangle as '$-$'. For e.g., if the corner is $(x_{\max}, y_{\min})$, then the following set suffices as a teaching set: $\{((x_{\max}, y_{\min}), +), ((x_{\max} +$

[1] By a side of a rectangle, we mean one of the 4-tuple values that defines any rectangle $h = \{x_{\min}, x_{\max}, y_{\min}, y_{\max}\} \in \mathcal{H}$. For e.g., $x_{\min}$ refers to the bottom side of the rectangle $h \in \mathcal{H}$. We follow this convention in the subsection C.1.2 for readability of the proofs.

$1, y_{\min}), -), ((x_{\max}, y_{\min} - 1), -)\}$. The necessity of three labelled examples is apparent, given the need for two labelled examples to ***fix*** a single side (as seen above). □

Theorem C.6 (Theorem 3 of main text)**.** *The VC dimension of axis-aligned rectangles in* $\mathbb{Z}^2$ *can be optimally reduced as shown below:*

Budget B	1	2	3	4	5	$\geqslant 6$
min VC	4	3	2	2	1	0

Minimum VC achievable by B-budgeted teaching on axis-aligned rectangle class in $\mathbb{Z}^2$.

Proof. We will proceed by starting with the case of $B = 2$ and ending with the case $B = 5$ in that increasing order. The case $B \geqslant 6$ follows from classical Teaching Dimension [30] as $TD = 6$.

Case $B = 1$: We cannot ***fix*** any side of the target h^* with $B = 1$ (Lemma C.5) and, hence, VC-dimension remains 4.

Case $B = 2$: We can ***fix*** exactly one of the sides of the target h^* with two examples as per Lemma C.5. This means the reduced version space $\mathcal{H}'$ has 3 degrees of freedom (Remark C.4) and $VC(\mathcal{H}') = 3$: Consider the rightmost side (i.e. $x_{\max}$) is ***fixed*** and take four points in general position. If one point lies within the convex hull of the other three, fixing $x_{\max}$ prevents labeling the outer three points as '+' and the interior point as '−'. Otherwise, if no point is inside the convex hull of the remaining three points, label the two points farthest apart (along the axis of the ***fixed*** side) as '+' and the remaining two as '−'. In both cases, at least one labeling is impossible, implying $VC(\mathcal{H}') < 4$. However, the set $\{(x, y), (x+1, y+1), (x+1, y-1)\}$, where $x < x_{\max}$, can be shattered.

Case $B \in \{3, 4\}$: ***Fix*** two opposite sides (e.g., $x_{\min}$ and $x_{\max}$) via four examples (Lemma C.5). Thus, the reduced version space $\mathcal{H}'$ has 2 degrees of freedom (Remark C.4) and $VC(\mathcal{H}') = 2$: Any three collinear points lying between these sides cannot all be labelled arbitrarily (one of them becomes the 'middle' point). Alternatively, for a triplet in general position, if one of the points falls outside these sides, we cannot flip its label without contradiction. Hence, no triple is shattered, but pairs are.

Alternatively, by Lemma C.5, use three labelled examples to ***fix*** two sides that meet at a corner of the target rectangle (e.g., $(x_{\max}, y_{\min})$). Again, we have reducedthe version space $\mathcal{H}'$ with 2 degrees of freedom (Remark C.4). This reduction ensures no set of three points

can be shattered: if they are collinear, the 'middle' point cannot be labelled differently from the other two; if they are in general position, at least one labelling is impossible (e.g. two points that are closest to one of the sides to be ***fixed*** are labelled as '$+$' and the remaining point as '$-$'). However, two points remain shatterable (for instance, by choosing a suitable (x, y) for the top-right corner). Hence, $VC(\mathcal{H}') = 2$.

Case $B = 5$: With three sides ***fixed*** by, for instance, ***fixing*** a corner and one of the sides corresponding to the opposite corner (Lemma C.5) using three and two labelled examples, respectively, the version space reduces to $\mathcal{H}'$ with 1 degree of freedom (Remark C.4) and $VC(\mathcal{H}') = 1$: No two-point set can be shattered as one labeling always becomes impossible depending on which three of the 4-tuple values have been taught. However, we can construct a single point set that can be labeled in any way.

Case $B = 6$: Since $TD(h^*; \mathcal{H}) = 6$, we can simply use a teaching set of size six so that the version space is reduced from $\mathcal{H}$ to $\mathcal{H}' = \{h^*\}$. Thus, $VC(\mathcal{H}') = 0$. □

C.1.3 Homogenous Linear Classifiers

Let D be a dataset of size m with p negative labels generated by a target hypothesis $w^* \in \mathcal{H}_{\text{linear}}$, given as follows,

$$D := D^- \cup D^+ := \{(x_i, -1) : i \in [p]\} \cup \{(x_i, +1) : i \in \{p+1, \cdots, m\}\}. \tag{C.1}$$

Note that D induces a polyhedral cone as a version space,

$$\mathcal{V}(D) = \{w : w^\top x_i < 0, \forall x_i \in D^-, w^t x_i \geqslant 0, \forall x_i \in D^+\}.$$

Since a cone lies in the subspace spanned by its vectors, we have that $VC(\mathcal{V}(D)) \leqslant d(\mathcal{V}(D))$, where d denotes dimensionality. The next lemma shows that $VC(\mathcal{V}(D))$ is also lower bounded by $d(\mathcal{V}(D)) - 1$.

Lemma C.7. *For a dataset* D *containing all positive labels, i.e.,* $D = D^+$*, we have that* $VC(\mathcal{V}(D)) \geqslant d(\mathcal{V}(D))$*. Otherwise,* $VC(\mathcal{V}(D)) \geqslant d(\mathcal{V}(D)) - 1$.

Proof. Let $\mathcal{V}(S) = \{w \in \mathbb{R}^D : w^\top x_i \geqslant 0, \forall i \in [n]\}$ be a closed polyhedral cone formed by an all-positive dataset S and let l be its dimensionality. We show that $VC(S) = l$.

We construct a set V consisting of l points in $\mathbb{R}^d$ and show that it can be shattered by S. For each labeling $s \in \{0,1\}^l$, we construct a labeling vector $w_s \in S$ that achieves label s on V. Consider a set of l orthogonal vectors in S and arrange them as columns of matrix $A = [v_1, v_2, \cdots, v_l] \in \mathbb{R}^{d \times l}$. Since S is l-dimensional, we can always find such a set of vectors.

Now, we show that the set of l points, $V = \{-A^{-\top}e_1, -A^{-\top}e_2, \cdots, -A^{-\top}e_l\}$ can be shattered by S. We use pthe seudo-inverse if $l < d$.

Let $s \in \{0,1\}^l$ be a labeling vector. We will show that $w_s = \left(\sum_{i:s_i=0} Ae_i\right)$ achieves the labeling s on V. First, note that since S is a convex cone, $w_s = \sum_{i:s_i=0} v_i \in S$.

We have that, $(-A^{-\top}e_j)^\top w_s = \sum_{i:s_i=0} -e_j^\top A^{-1}Ae_i = \sum_{i:s_i=0} -e_j^\top e_i = -\mathbb{1}[s_j = 0]$. Thus, $h_{w_s}(-A^{-\top}e_j) = \mathbb{1}[(A^{-\top}e_j)^\top w_s \geqslant 0] = \mathbb{1}[s_j = 1]$ and so w_s realizes the labeling s. Since, $dim(S) = l, S \subseteq \mathbb{R}^l$, and we have that $VC(S) \leqslant VC(\mathbb{R}^l) = l$. Thus, $VC(S) = l$.

Now, if S has an open halfspace, i.e., the corresponding dataset contains a negative labeled point, then all the labeling except all positive ones can be realized, i.e., $w_{\not\Vdash} = 0$ does not lie in S, while rest of all w_s still lie in S and the above proof proceeds. Thus, $l - 1 \leqslant VC(S) \leqslant l$. □

Next, we characterize the dimensionality of the version space and assert that maximal dimensionality reduction can be achieved by B positively labeled demonstrations as stated in point 3 below.

Lemma C.8. *The following statement hold true for a consistent dataset* D *generated by* $w^* \in \mathbb{R}^D$ *:*

1. *Negative points do not help in reducing dimensionality, i.e.,* $d(\mathcal{V}(D)) = d(\mathcal{V}(D^+))$.
2. $\forall D : |D| \leqslant B$*, we have that* $d(\mathcal{V}(D)) \geqslant d - |D| + 1$.
3. $\forall B \leqslant d$*, the dataset* D_T^B *achieves optimal reduction in VC by* $B-1$*, thus,* $d(\mathcal{V}(D_T^B)) = d - B + 1$.

$$D_T^B = \{(v_1, +1), \ldots, (v_{B-1}, +1), (-\sum_{i \in [B-1]} v_i, +1)\}$$

where $\{v_1, \ldots, v_{B-1}\}$ *is a* B*-basis of* $w^{*\perp}$ *subspace.*

Proof. To prove 1, we start with a basis set of $\mathcal{V}(D^+)$ and a feasible $x_0 \in \mathcal{V}(D)$. Translating the basis set by x_0 yields a basis set for $\mathcal{V}(D)$. To prove 2, we can construct a dataset with $|D|$ points that kills $|D| - 1$ vectors in the orthogonal subspace of w^*. For prove 3, it is easy

to see that a classifier w is consistent on D_T^B if and only if $w \notin \text{span}(v_1, \cdots, v_{B-1})$. Thus, the dimensionality of the version space $\mathcal{V}(D_T^B)$ reduces from $d(\mathbb{R}^d) = d$ to $d(\mathcal{V}(D_T^B)) = d - B + 1$. □

Theorem C.9 (Theorem 4 of main text)**.** *There exists an algorithm that* $\forall B \leqslant d+1$*, optimally reduces the VC of the linear class to* $d - B + 1$ *and the optimal teaching set is given as* $D_T^B = \{(v_1, +1), \ldots, (v_{B-1}, +1), (-\sum_{i \in [B-1]} v_i, +1)\}$*, where,* $\{v_1, \ldots, v_{B-1}\}$ *is a* B*-basis of* $w^{*\perp}$ *subspace.*

Proof. We make the following observations:

1. For a dataset with all positive demonstrations, we have that $VC(\mathcal{V}(D)) \geqslant d(\mathcal{V}(D)) \geqslant d - B + 1$ and the lower bound is achieved by D_T^B in Lemma C.8.

2. For a dataset with at least one negative demonstration, $VC(\mathcal{V}(D)) \geqslant d(\mathcal{V}(D)) - 1 = d(\mathcal{V}(D^+)) - 1 \geqslant d - |D^+| \geqslant d - B + 1$. The first inequality follows from Lemma C.7 while others follow from Lemma C.8.

Thus, for $B \leqslant d$, the optimal strategy is to teach dataset D_T^B to kill off $B - 1$ dimensional subspace. We refer to Figure [4.2] for an illustrative teaching example in $w^* \in \mathbb{R}^3$ and $B = 2, 3$. For $B \geqslant d + 1$, our optimal teaching dataset matches with the unconstrained teaching dataset proposed by [49]. □

C.1.4 Polynomial Hypothesis Class

Theorem C.10 (Theorem 5 of main text)**.** *For any target polynomial* $h^* \in \mathcal{H}$*, the optimal teaching set that reduces the VC dimension of the polynomial version space by* $B - 1$ *is given as* $D_T^B = \{(x_i, +1) : \phi(x_i) \perp w^*, i \in [B-1], \forall i \neq j, \phi(x_i) \perp \phi(x_j)\} \cup \{(\phi(x_B) = -\sum_{i=1}^{B-1} \phi(x_i), +1)\}$

Proof. The teacher can computes $B - 1$ orthonormal bases functions to θ^* and their negative summand represented by vectors $v_1, \cdots, v_{B-1}, -\sum_{i \in [B-1]} v_i$ in the standard bases. However, to be a valid teaching set, these vectors must be induced by the feature function ϕ on a certain input set (preimages under ϕ) $\{x_i \in \mathbb{R}^d : i \leqslant [B]\}$. Assuming that there exists a set of such inputs, the teacher can construct the optimal teaching set given by the dataset $D_B^T = \{(x_i, 1) : i \in [B-1]\} \cup \{(-\sum_{i \in [B-1]} x_i, 1)\}$. □

Computing the preimages for teaching vectors: We propose an iterative algorithm that can compute the orthogonal preimages efficiently, assuming they exist. At the start of iteration $k \in [B]$, say we have already computed the $k-1$ orthogonal bases vectors in $w^{*\perp}$, the optimization problem to find the next orthobasis vector v_k is as follows:

$$\begin{aligned} x_k, v_k \leftarrow \min_{x,v} \quad & \|v - \phi(x)\|^2 \\ \text{s.t. } & \alpha^\top v = 0, \forall \alpha \in \{v_i : i \leqslant k-1\} \cup \{w^*\} \end{aligned}$$

Once all $\{v_i\}|_{i \leqslant B-1}$ vectors have been computed, we compute the preimage of their negative summand by solving

$$x_B, v_B \leftarrow \min_{x,v} \|v - (-\sum_{j \leqslant B-1} \phi(x_j))\|^2 + \lambda \|v - \phi(x)\|^2.$$

Remark C.11. *This method extends to any finite-dimensional kernel. However, not all kernel mappings may admit a pre-image set, thereby limiting this approach.*

C.2 Experiments for Instance-Aware Teaching using Datamodels

Overview: We first train a datamodel using meta dataset $D_M := (D_i, R(\mathcal{A}(D_i)))$ computed by training perceptron algorithms on various data subset D_i sampled from $P_D \in \Delta(2^{\mathcal{X}})$ and tested on a held out test set to get the meta label $R(\mathcal{A}(D_i))$. We use a uniform distribution P_D over all subsets of size $\leqslant \alpha \cdot |\mathcal{X}|$ where $\mathcal{X}$ is a finite set of points in $\mathbb{R}^2$.

Once we have the meta dataset, we train the datamodel parameter $\hat{w}_P$ using sparse linear regression on D_M. We then compute the optimal teaching set as B points in $\mathcal{X}$ with minimum weights $\hat{w}_{P,x}(1-P_x)^n$.

We then evaluate the NtN performance on this dataset for various values of n_{iid}. The teaching code for VC reduction teaching for linear and axis-aligned rectangles can also be found in the supplementary materials (data_models.ipynb).

Now we describe our setup and pseudo-code along with hyperparameter configurations for estimating linear datamodels, computing the teaching sets, and evaluating NtN performance.

C.2.1 Setup, Data Generation and Evaluation

We consider teaching a homogeneous linear classifier in $\mathbb{R}^2$. The learner is trained via an ERM procedure using a *perceptron loss* (as a computationally convenient surrogate to 0–1 loss) on a finite universe $\mathcal{X} \subset \mathbb{R}^2$.

We let $\mathcal{X}$ be a set of 16 uniformly-spaced points on the unit circle in $\mathbb{R}^2$. Each point $x \in \mathcal{X}$ is labeled via a target linear separator w_{true}[2]. In the *nature phase*, the learner receives n i.i.d. draws from the uniform distribution $P_{\mathcal{X}}$. Since we have access to all of $\mathcal{X}$ and we know that $P_{\mathcal{X}}$ is uniform, we evaluate the test performance of a linear classifier as the average 0-1 loss on the entire feature space $\mathcal{X}$.

C.2.2 Teaching through Linear Datamodels (OPT-DM)

As discussed in Section 4.5, we use linear datamodels to approximate the learner's risk as a linear function of the dataset. We then use these estimated data models' weights to find the budgeted teaching set, which is to be used for teaching the learner under the prescribed teaching budget.

C.2.2.1 Estimating Linear Datamodels

- **Meta-Dataset Construction:** We sample N_{subsets} subsets $S_i \subset \mathcal{X}$ of size $\alpha \cdot |\mathcal{X}|$ via a distribution P_{subsets}. For each subset S_i, we train a *perceptron* on S_i using the perceptron loss and measure its 0–1 test loss y_i on the entire space $\mathcal{X}$. We use $\alpha = 0.25$ and P_{subsets} to be a uniform distribution in our experiments.

- **Sparse Linear Fit (ℓ_1-regularization):** We collect pairs $(\mathbf{1}_{S_i}, y_i)$ and solve

$$\theta = \arg\min_{\theta'} \frac{1}{N_{\text{subsets}}} \sum_{i=1}^{N_{\text{subsets}}} \left(\theta'^{\top} \mathbf{1}_{S_i} - y_i\right)^2 + \lambda \|\theta'\|_1.$$

  We use `scikit-learn`'s `LassoLarsCV` solver with 4-fold cross-validation to automatically select λ and perform the ℓ_1-regression. Here, θ_x measures how strongly point $x \in \mathcal{X}$ influences the overall risk. The pseudo-code for estimating datamodels is outlined in Algorithm 12.

[2]For simplicity, we have chosen w_{true} to be one of these 16 points in our simulations.

Algorithm 12: EstimateDataModel

Require: Universe $\mathcal{X} \subseteq \mathbb{R}^3$, subsampling fraction $\alpha \in (0,1)$, test-set size m, number of subsets $N_{subsets}$, distribution over subsets $P_{subsets}$

$T \leftarrow []$ ▷ Initialize datamodel training set
$S = \{(x,y) \mid x \in \mathcal{X}, y = 2 \cdot \text{sign}(w_{true} \cdot x) - 1\}$
for $i = 1$ to $N_{subsets}$ **do**
  Sample subset $S_i \subset S$ as per $P_{subsets}$ with $|S_i| = \alpha \cdot d$
  Train $\mathcal{A}$ on S_i
  Sample $D_{test} \sim P_{\mathcal{X}}^m$
  $y_i \leftarrow \frac{1}{m} \sum_{(x,y) \in D_{test}} \ell_{0-1}(\mathcal{A}(x; S_i), y)$
  Define $\mathbf{1}_{S_i} \in \{0,1\}^d$ where $(\mathbf{1}_{S_i})_j = 1$ if $x_j \in S_i$ else 0
  $T \leftarrow T \cup \{(\mathbf{1}_{S_i}, y_i)\}$
$\theta \leftarrow$ RunRegression(T)
return θ

C.2.2.2 Computing the teaching set

Having estimated the linear datamodel using Algorithm 12 above, we use its weights θ along with the given teaching budget B, a nature budget n, and underlying data distribution $P_{\mathcal{X}}$, we can compute the limited-budget teaching set by finding B points that minimize the following

$$\sum_{x \in D} \theta_x \left(1 - P_{\mathcal{X},x}\right)^n.$$

We thus pick the B smallest values of $\theta_x(1 - P_{\mathcal{X},x})^n$ as proved in Theorem 4.26. This is outlined below as Algorithm 13.

Algorithm 13: ComputeTeachingSet

Require: Teaching budget B, weight vector θ, distribution $P_{\mathcal{X}}$, nature budget n

$D_T \leftarrow \arg\min_B \{\theta_x \cdot (1 - (P_{\mathcal{X},x})^n)\}$
return D_T

C.2.2.3 NtN Evaluation

Given that we have estimated the datamodel and computed the limited-budget teaching set D_T, we now measure performance $(R_{NtN}(D_T, n_{iid}))$ of the learnt classifier as a function of

the nature budget n_{iid}. In particular, by sampling $K = 50$ distinct i.i.d. subsets of size n_{iid}, training a perceptron on $D_T \cup D_k^n$ for each $k \in [K]$, and computing average test-set error $R_{NtN}(D_T, n_{iid})$ as averaging their 0–1 loss on the entire space $\mathcal{X}$ (Equation (C.2) below). We repeat this for nature budgets $n_{iid} = 1, \ldots, N$ for $N = 16$. Note that $K = 50$ subsets are sampled for each n_{iid} so that we can compute the 95% confidence intervals (shaded regions) as seen in Figure 4.8.

$$R_{NtN}(D_T, n_{iid}) = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}} \ell_{0-1}(\mathcal{A}_k^{n_{iid}}(x), y_x) \tag{C.2}$$

where $\{\mathcal{A}_k^n\}$ denotes trained models using the k^{th} subset of i.i.d. training set with n_{iid} nature budget and y_x denotes the true label of any point $x \in \mathcal{X}$ as per w_{true}.

C.2.2.4 Combining everything: Training a linear classifier through NtN

The `OPT-DM` procedure as outlined above in B.2.1 – B.2.3 can be collectively expressed as Algorithm 14 below.

Algorithm 14: TrackNtNPerformance

Require: $\mathcal{X} \subset \mathbb{R}^2$, $|\mathcal{X}| = d$, uniform $P_{\mathcal{X}}$, teaching budget B, max nature budget N, α, $N_{subsets}$, $P_{subsets}$, test-set size m, number of models K

1: $\theta \leftarrow$ EstimateDataModel($\mathcal{X}$, α, m, $N_{subsets}$, $P_{subsets}$)
2: **for** $n_{iid} = 1$ to N **do**
3: $\quad D_T \leftarrow$ ComputeTeachingSet(B, θ, $P_{\mathcal{X}}$, n)
4: $\quad$ **for** $k = 1$ to K **do**
5: $\quad\quad$ Sample $D_k^{n_{iid}} \sim P_{\mathcal{X}}^{n_{iid}}$
6: $\quad\quad$ Train $\mathcal{A}_k^{n_{iid}} = \mathcal{A}(D_T \cup D_k^{n_{iid}})$
7: $\quad$ Evaluate $R_{NtN}(D_T, n_{iid})$ as per Equation (C.2)

C.2.3 Extending to Neural Datamodel in Instance-Aware Setting

Our Instance-Aware method based on the datamodel is very generic and in fact it can be extended to any datamodel that can be optimized over input space and that includes deep neural networks as well.

We briefly outline the procedure for using a neural datamodel for NtN teaching. For teaching purposes, we assume that we already have access to a neural datamodel (one can

easily train one similar to how we trained a linear datamodel using the same underlying meta-dataset, just by substituting the linear function class with a neural network function class).

$$\theta^* \leftarrow \arg\min_{\theta} \frac{1}{m} \sum_{i=1}^{m} \ell_2(f_\theta(\mathbb{1}_{D_i}), R(\mathcal{A}(D_i))) + \lambda \|\theta\|_2^2.$$

Once we have a trained neural datamodels parameterized by θ^*, we use the following projected gradient descent algorithm to find the best NtN teaching set under budget constraints.

Algorithm 15: ProGrad-NtN : Projected Gradient for Nurture-then-Nature

Require: Model function $f_\theta : x \to \mathbb{R}$, Target $y^* = 0$, Budget B, Learning rate η
Ensure: Optimized dataset $D^* \in \{0,1\}^d$ with $|D^*|_0 \leqslant B$.

```
Initialize D ∈ {0,1}^d randomly.
while not converged do
    Compute gradient: ∇ ← ∇_D ℓ(f_θ(D), y*)
    Perform gradient step: D ← D − η∇
    Project D onto ℓ_0 ⩽ B ball: Keep the top-B entries of D and set others to 0.
return D
```

We tested this method on a simple threshold classification problem with input space $\mathcal{X} = \{-4, -3, -2, -1, 0, 1, 2, 3\}$. The ground truth classifier is $h(x) = \mathbb{1}[x \geqslant 0]$.

We apply our algorithm ProGrad-Ntn using Adam optimizer with a regularization coefficient of 0.1 and learning rate of 0.01 for 10K iterations or until iterates converge to a local minima. The resulting x^* so obtained is $x^* = [-0., 0.01, 0.37, 0.23, 0., 0.01, 0.]$. Projecting on $l_0(x^*) = 2$, yields the dataset $x = [-0.33, 0]$ as a teaching set, which indeed is an optimal teaching set for the problem.

We would like to emphasize that the aim of this and the main experiments in the chapter is to serve as an empirical proof of concept for the usefulness of the datamodel. A complete treatment of these methods on complex problems is beyond the scope of this paper, and we hope that future works could build on our work to solve more real-world NtN problems using our algorithm.