# Optimal teacher strategies for automated curriculum learning

A THESIS PRESENTED
BY
WILLIAM L. TONG
TO THE
INSTITUTE FOR APPLIED COMPUTATIONAL SCIENCE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE
IN THE SUBJECT OF
DATA SCIENCE

HARVARD UNIVERSITY
CAMBRIDGE, MASSACHUSETTS
MAY 2023

Advisors: Gautam Reddy, Venkatesh N. Murthy

# Optimal teacher strategies for automated curriculum learning

## Abstract

Curriculum learning is a process whereby a difficult task is learned through a succession of gradually harder and more accurate approximations to the desired behavior. For example, when training a dog to track trails, the handler might start the dog on short, easy trails that align well with the animal's innate tracking capacity. The handler may then increase the tracking length, introduce obstacles, distractions, or obfuscate odor signals, incrementally rewarding the dog as it progresses. Before long, the dog is fully capable of tracking on command, and can navigate the difficult situations it will encounter during real-world tracking scenarios. Curriculum learning is also relevant to reinforcement learning, where difficult tasks offer reward signals that are too sparse for effective learning. By starting on easier tasks where rewards are more frequent, the agent more efficiently learns the overall behavior.

Despite its importance, little is understood about constructing optimal curricula. Typical approaches use ad-hoc methods or simple heuristics to design plausible learning schedules, but may ultimately be inefficient. In this thesis, we consider optimal curriculum design under a teacher-student framework, where a teacher agent proposes suitable tasks based on its student's transcript of successes and failures. We apply our framework to a simple sequence-learning task to derive approximately optimal teacher algorithms. We then validate these results on real-world, naturalistic tracking tasks. Broadly, this analysis highlights the basic elements that underpin effective curriculum design strategies.

To Mom and Dad.

And also MES,
for agreeing to another 5-6ish years.

# Acknowledgments

MANY, MANY THANKS TO Gautam Reddy for his guidance and mentorship throughout this project. Without Gautam, none of this project would have been remotely possible. I have learned much from him not only about olfaction, trail tracking, and curriculum learning, but also about the process of science and the responsibilities of theorists within a scientific community. His insights and advice will continue to shape my thinking far beyond the timeline of this project.

Huge thanks also to Venki Murthy, for introducing me to the wide world of trail tracking and supporting this project throughout the many metamorphoses it underwent *en route* to this document. Many thanks also to the rest of the thesis committee and its organizers at IACS — Lina Li, Weiwei Pan, Daniel Weinstock, and Pavlos Protopapas, for their feedback and support throughout this process. I am deeply grateful for the individuals in both the Murthy Lab and Pehlevan Group, as well as my colleagues at NTT Research, who have listened, critiqued, and ideated on portions of the project as it developed. In alphabetical order, they are Siddharth Jayakumar, Ugne Klibaite, Ziyin Liu, Cengiz Pehlevan, Shanshan Qin, Ekdeep Singh Lubana, Hidenori Tanaka, Ningjing Xia, and Jacob Zavatone-Veth.

Last but most certainly not least, many thanks to my family and friends who have supported this journey every step of the way. To my parents, who are a constant source of unconditional love and support. To my sister Anne, for the wisdom that comes only from an older sister. And my brother Alan, for being a mostly-excellent roommate. To Bhawesh, Tsybzhid, Annesya, Nam, and Selin, and for countless hours spent sampling fine Lehman cuisine, and tolerating my boardgames. To Albert and Shubham, for the good times, and the commisery. To all the good folks from Boardgames, D&D, Hiking, and Bill's Biceps, who made these two years spectacular. And of course, to MaryElena, for being there, and agreeing to another 6 years (tops, I promise) of shenanigans.

# Contents

# Listing of figures

*My colleagues and I knew that, in the eyes of the world,*
*we were crazy.*

BF Skinner

# 1 Introduction

It was 1943, Minneapolis. On the top floor of an inconspicuous flour mill, a scrawny scientist toiled away behind a large box containing a very confused pigeon. The hapless bird was strapped before a translucent screen, behind which was the image of a ship at sea. By tapping its beak to the left, the image shifts right. By tapping its beak above, the image shifts down. As time passes, the image seems to zoom in. If the hungry pigeon happens to "land" on the ship, it receives a reward in delicious hemp seeds. Ideally, through repetition, the pigeon becomes adept at landing on ships, at which point it can be strapped into a "Pelican" missile (Figure 1.1) and unleashed on the enemy. In the days before modern homing missiles and remote-operated drones, pigeon-guided firepower promised an efficient, cheap, scalable approach to precision bombing.

The scientist, one Burrhus F. Skinner, was convinced it would work. The key to success seemed to lie in the particular schedule of reinforcement applied to the pigeon. An untrained bird would fail to hit any targets. But by exposing the pigeon first to short, easy approaches, reinforcing successes immediately with a food reward, then gradually increasing the distance and difficulty of the task, Skinner found he could train a bird to pilot missiles with stunning consistency. In fact, the bird operated so successfully that upon seeing the results, an observer exclaimed, "this is better than radar!"[30] Using the reinforcement schedules, pigeons could be trained quickly using an automated rig, allowing the US to deploy guidance missiles that are cheap to produce, work on a wide array of targets, and are perfectly resistant to jamming.

The project was ultimately canned — not because the pigeons were ineffective, but because Skinner had trouble getting people to take him seriously. "The spectacle of a living pigeon carrying out its assignment," Skinner reflects, "no mattery how beautifully, simply reminded the committee of how utterly fantastic our proposal was. I will not say that the meeting was marked by unrestrained merriment, for the merriment was restrained. But it was there, and it was obvious that our case was lost." Nonetheless, Skinner's training techniques lived on, and continue to shape animal training practices to this day.

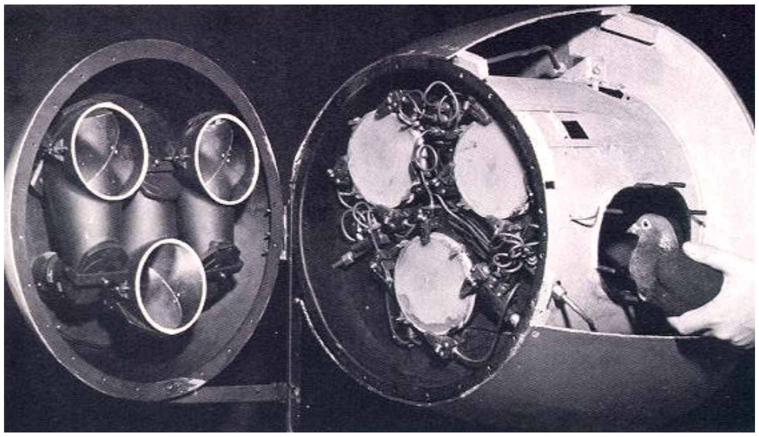

**Figure 1.1: "Project Pigeon" missile guidance system, called the Pelican.** Three pigeons are loaded into the cylindrical capsules in the nose of the missile. Placed in front of each cylinder is a transparent sheet sensitive to beak taps, allowing each pigeon to vote on a particular heading. The majority vote steers the missile, hopefully towards an unsuspecting vessel.

The key idea in this (and later pigeon projects) was the *curriculum*. The final task is too difficult for a pigeon to perform. If an experimenter were to wait for the pigeon to guide a missile perfectly before dispensing reward, they would be waiting all day, and the pigeon would never receive any reinforcement. Rather, by starting with an easier version of the task, and through gradual approximations attain the final version, the pigeon can be trained easily and simply on any variety of nontrivial behaviors. The same principle remains effective for training other animals, and are regularly employed today for training pets, working animals, animals for entertainment, and animals for science. Indeed, even humans are trained using curricula, progressing through grades of increasing difficulty until nontrivial literacy, reasoning, and social skills are acquired.

Beyond animal training, curricula are increasingly being used to train machine agents. In reinforcement learning (RL), agents routinely encounter difficult tasks with sparse rewards — precisely the setting for curricula. Curricula are also being used to train models for language processing, classification, and basic reasoning tasks, where the performance and sample efficiency of a model increases substantially when curricula are applied. The success of curricula for machine learning applications has spawned the nascent field of *curriculum learning*, the systematic study of curricula and curriculum design strategies for improving learning agents.

Yet despite its importance and rising popularity in ML, curriculum learning in implementation tends to be ad-hoc and heuristic. When Skinner trained his missile pigeons, he did not have a systematic theory of curriculum design in mind. Rather, he started the pigeon on an easy task, and as it got better, moved to harder tasks. Similarly, machine learning practitioners often approach curriculum design with a holistic mindset, applying a process that simply "feels right." Attempts to make the process more rigorous have led to the birth of a new field, *automated curriculum learning* (ACL), but existing strategies often rely on simplistic heuristics that may not generalize beyond simple situations. [15,18,20]

This thesis presents a novel approach to ACL that seeks to develop an optimality-based approach to curriculum design that generalizes to complex, real-world tasks. We develop a teacher-student framework for curriculum learning, and apply it to a simple sequence learning task. We then confirm that its principles generalize to a naturalistic tracking task. The optimal teachers developed under this framework can apply as naturally to a biological agent as an artificial one, and can be validated in a real-world animal training scenario. We therefore present this framework as a first step towards building a rigorous foundation for curriculum learning, improving the efficiency of both animal and machine training.

## 1.1 Structure of the thesis

- **Chapter 2** presents an overview of the basic concepts essential to curriculum learning, and introduces the teacher-student framework we use in the remainder of the thesis.
- **Chapter 3** introduces a simple sequence learning task that we use as a testbed for developing ideas about optimal teacher strategies
- **Chatper 4** tests the strategies developed in Chapter 3 on a naturalistic tracking task, to measure how well they may hold in a complex real-world scenario
- **Chapter 5** explores a generalizing of the teacher-student framework for tasks with continuous curricula
- **Chapter 6** concludes with a summary of the main results and proposes future directions.

*Teaching, it is often said, is an art, but we have increasing reason to hope that it may eventually become a science.*

BF Skinner

# 2 Background

As a formal topic of study, curriculum learning has its intellectual roots in behaviorism, the school of psychological thought that positions the origin of behavior in the positive and negative interactions between an agent (either human or animal) and its environment. The earliest notion of curriculum learning can be traced back to B.F. Skinner's "shaping" experiments, in which he cultivated complex, highly nontrivial behaviors in pigeons (among other animal subjects) by reinforcing gradual approximations to the desired outcome.[30,28,9] In subsequent years, curriculum learning has been adopted by the reinforcement learning community for training agents on difficult tasks with sparse rewards[25,26,5], and applied broadly in machine learning as a training technique[3,10]. Skinner's original behavioral shaping techniques remain foundational for animal training programs to this day.

In this chapter, we briefly review shaping both in animals and artificial agents, and motivate an automated approach to curriculum learning. We also present an overview of our teacher-student framework for curriculum learning.

## 2.1 Shaping behavior

Animal trainers routinely 'shape' an animal's behavior towards a specific sequence of actions.[28] For example, consider training a dog to perform a three-action sequence: circle $\rightarrow$ jump $\rightarrow$ sit, after which it receives reward. Prior to training, it is highly unlikely that the animal will perform these actions in sequence, even though it is innately capable of executing each action separately. One intuitive teaching strategy is to first reinforce the animal for circling. Once the animal circles reliably, the reinforcement for circling is stopped (or made variable) and is instead presented if circling is followed by a jump. This procedure is repeated until all three actions are performed in sequence. In some cases, the trainer may have to break down the task further or "lure" the animal to execute the right action.

For many tasks, such a shaping procedure is necessary to ensure that the animal makes progress, simply because the animal is unlikely to perform the right sequence through innate behavior and will thus never be reinforced for performing the task correctly. This reflects the basic notion that learning a specific behavioral trajectory through random, unconstrained exploration is impractical when the the dimensionality of behavior is large, irrespective of the learning rule. Shaping addresses this challenge by iteratively approximating longer portions of the trajectory, thereby reducing the search space at each step of the procedure. Partly inspired by shaping, several approaches (collectively termed as curriculum learning) have been successfully used to train artificial agents on complex sequential tasks.

A key element of effective shaping is to modulate *when* the animal receives reinforcement or punishment such that the animal learns the full task as quickly as possible. A common heuristic approach used during laboratory training is to hand-design a curriculum of sub-tasks leading up to the full task. Reward is delivered when the animal successfully completes the assigned sub-task in each trial. The curriculum progresses once the animal reaches a desired level of performance on the current sub-task. However, the design of effective shaping strategies and curricula remains an open problem. It is unclear whether existing heuristics are close to optimal even in simple scenarios and under what conditions these strategies will fail. Elucidating the principles behind effective shaping combined with the development of closed-loop training strategies has the potential to significantly reduce the time spent training laboratory animals (which often spans several weeks to months), while providing much-needed insight into the factors that underpin slow or fast learning.

## 2.2 Shaping in reinforcement learning

Just as training an animal to learn some nontrivial behavior — like bowling[29] — faces a sparsity issue, so too do many reinforcement learning tasks suffer from sparse rewards. The odds that a pigeon will, through innate behavior alone, knock a wooden bowling ball into the target pins is vanishingly small. If an experimenter were to wait for the pigeon to accomplish the whole task before dispensing reward, they would wait for a very long time indeed, and the pigeon would never learn. In precisely the same way, if an experimenter were attempting to train an RL agent to bowl, the agent is exceedingly unlikely to complete the task successfully through random chance. Hence, the agent will virtually never receive reinforcement, and never learn to bowl. The reward signal is simply too sparse to induce productive learning.

There are two common ways to address the sparsity issue: incentives and curricula.[5] Incentives decrease the sparsity of the reward signal by simply adding more rewards. Rather than reward the agent only if it successfully bowls down all 10 pins, perhaps the experimenter will dispense a smaller reward if the agent simply hits the ball. Because the chances of the agent hitting the ball through random exploration are far higher than the chances of bowling successfully, the agent receives more reward signal, learns to hit the ball, and will hopefully make the next step to hitting pins.

The primary weakness of engineering the reward signal directly is the propagation of unintended consequences. Because the agent is rewarded for hitting the ball, perhaps the agent now exploits the reward signal by tapping the ball constantly, accruing higher rewards while ignoring the pins altogether. Unintended consequences like these are prevalent across reinforcement learning. Techniques like shaping reward potentials or evolving reward signals ameliorate some of these effects, but they cannot be applied to every situation, and do not always prevent all unintended consequences.[19,1,32] In general, designing reward signals remains a very challenging task with often unpredictable outcomes.

The alternative is to design curricula. In much the same spirit as Skinner's shaping techniques, an experimenter can design a curriculum where the agent is first exposed to an easy approximation to the final task, then gradually increase the task difficulty through closer approximations. Such an approach avoids the need to engineer additions to the reward signal, and relies on the task difficulty itself to ensure rewards are dispensed with sufficient frequency. Curriculum learning has already garnered substantial interest in machine learning[3,7,6], and been applied with great success in reinforcement learning.

However, one clear issue about curriculum learning concerns the process of creating a curriculum in the first place. How should an experimenter choose the correct set of intermediate tasks, and order their presentation? Selecting a sub-optimal curriculum may result in prolonged training, or a failure to learn the task at all. Many experimenters will design curricula using ad-hoc heuristics, going by what "feels right," but the process of designing curricula can itself be optimized. By treating the curriculum design process as its own meta-learning problem, we can approach some notion of optimality in curriculum learning. This is the subject of the nascent field *automated curriculum learning*.

## 2.3 Automated curriculum learning

Automated curriculum learning (ACL) refers to a broad, loosely defined collection of algorithms and frameworks for training agents or other algorithms on progressively more difficulty tasks.[18,10] ACL is fundamentally a meta-learning procedure where the experimenter must optimize the outer, curriculum-building algorithm in conjunction with the inner target task. Approaches to ACL are diverse, but tend to rely on ad-hoc heuristics that may not always generalize beyond their target domain.[20] A primary goal of this work is to identify

ACL strategies from a first-principles basis, grounding their optimality in a study of the underlying task dynamics. To that end, we frame ACL as a teacher-student interaction.

One feature of many ACL frameworks is the need to access detailed state information about the student. In the context of reinforcement learning, common approaches might require direct information about the student's state, action space, or reward signals.[18,20] However, animal training setups rarely have access to this level of detail. Hence, one goal for our framework is to develop a "model-free" approach to ACL that requires minimal assumptions about the student. Ideally, the teacher has access only to the student's transcript of successes and failures, and *not* the student's internal state, policy, or even learning rule. We clarify the details of this approach below, and in Chapter 3.

### 2.3.1 A teacher-student framework

A natural way to think about curriculum learning is as an interaction between a teacher and a student. The teacher proposes tasks for a student to master, and by monitoring the progress of the student, adjusts the difficulty of the task. This approach is in some ways similar to the work of Matiisen et al.[15], which employs a similar teacher-student framework for curriculum learning. We later demonstrate in Section 3.6 that their heuristics tend to generalize poorly, and may not work as effectively as our own proposals on a wide array of tasks.

At the start of every interaction, the teacher receives as input a transcript $\mathbf{t}$ of the student's prior successes and failures. This can be as simple as a sequence of binary digits, where 0 corresponds to a failure and 1 corresponds to a success. The teacher then proposes a task, and the student attempts the task for $T$ fixed rounds, generating a new performance transcript.[1] This dialogic interaction proceeds until the student attains a satisfactory level of success on the final task. Algorithm 1 formalizes this process.

**Algorithm 1** Teacher-student learning framework

```
Require: task family $\mathcal{M}(\theta)$, parameterized by difficulty level $\theta \in 1, 2, \ldots, N$
  while Student is unsuccessful on $\mathcal{M}(N)$ do
      $\mathbf{t} \leftarrow$ Evaluate(Student)
      $\theta \leftarrow$ Teacher($\mathbf{t}$)
      Train Student on $\mathcal{M}(\theta)$ for $T$ rounds
  end while
```

Note, the task difficulty $\theta$ is assumed to be discrete, which we allow for simplicity. While true for many tasks, not all tasks can be naturally parameterized into discrete levels. We explore continuous difficulty parameterizations in Chapter 5.

Optimality under this framework is measured by the time it takes for the student to succeed at the final task $\mathcal{M}(N)$. The optimal teacher is one who takes the shortest time to train a student. The framework overall is very general, and applicable to a wide range of scenarios. The Student algorithm can be any learning agent, biological or artificial. Indeed, this framework permits a student that can be a laboratory mouse, a human learner, or even a Skinnerian pigeon. The teacher algorithms we explore can be applied equally well to a reinforcement learning algorithm as to a real, biological organism.

In the subsequent chapters, we explore the construction of optimal Teacher algorithms. Chapter 3 presents a simple sequence learning task that we use as a launchpad for thinking about optimality. Chapter 4 extends our results to a real-world tracking task. Chapter 5 explore continuous generalizations of our teacher algorithms. Additional implementation

[1] In our setting, the length of the transcript is $|\mathbf{t}| = T$. Note, there is no requirement that the teacher is stateless. Hence, the teacher algorithm can (and does) maintain a history of student performance across time.

details on all the teacher algorithms are available in Appendix B.

*"Catch your rabbit" is the first item in a well-known recipe for rabbit stew. Your first move, of course, is to choose an experimental subject. Any available animal — a cat, a dog, a pigeon, a mouse, a parrot, a chicken, a pig — will do. (Children or other members of your family may also be available, but it is suggested that you save them until you have had practice with less valuable material.)*

BF Skinner

# 3 Task: sequence learning

Many a task that lends itself to a curriculum can be thought of as a sequence of steps. To play a sonata, a pianist first learns the exposition, development, then recapitulation. To make nigiri, a sushi chef must first master the preparation of rice, fish, sauce, then plating. To track a long trail, a dog must first successfully track the intervening segments.

Tasks that decompose into discrete steps lend themselves naturally to a curriculum. A student begins with the first step, then through progressive mastery, advances to later steps until succeeding at the entire task. In this sense, curriculum learning can be interpreted as a sequence learning task. Given a task of steps 1 through $N$, a student must learn to perform the sequence 1 - 2 - 3 - . . . - $N$. In this chapter, we use a simple sequence learning task to study the fundamental learning dynamics that underpin curriculum learning, and propose optimal teacher strategies.

## 3.1 Task setup

The task consists of a series of discrete choices, inspired by the setup in[22]. At each timestep, the student must select the correct action to advance. Selecting the wrong action at any timestep terminates the episode without reward. If the student advances $N$ times in a row, where $N$ is the pre-determined length of the environment, the episode terminates with fixed reward $R$.

Selecting the correct action is determined by the student's Q-values, which are updated using a simple temporal-difference (TD) learning rule described below (Section 3.1.1). Though the model is constructed with binary actions (correction action / incorrect), it describes any situation where actions can be partitioned into correct and incorrect groups. The probability of selecting an incorrect action is tuned using an additional parameter $\varepsilon$, which we describe further below. In this way, the task models any setting where a student must execute a consecutive sequence of correct actions, capturing a large portion of tasks classically suitable for curriculum learning.

See Figure 3.1 for a graphical depiction of the task. An MDP that summarizes this task can be given as

- **State space**: integers $1, 2, \ldots N$
- **Action space**: move forward, halt
- **Transitions**: For states $s, s'$ and action $a$, we have the following transition probabilities:

$$\begin{aligned}
&\Pr(s' = i + 1 \,|\, s = i, a = \text{move forward}) = 1, \text{for } i < N \\
&\Pr(s' = \varnothing \,|\, s = N, a = \text{move forward}) = 1 \\
&\Pr(s' = \varnothing \,|\, s, a = \text{halt}) = 1, \text{for any } s
\end{aligned}$$

- **Reward**: the reward function is simply $R(s = N, a = \text{move forward}) = R$ and 0 otherwise. For all sequence learning tasks, $R$ is fixed at $R = 10$.[1]

This setup lends itself naturally to a curriculum, where larger values of $N$ correspond with more difficult tasks. The goal of the teacher is to propose a number of intermediate tasks indexed by their lengths $N_1, N_2, N_3, \ldots$ such that the student succeeds at the final task $N_{\text{final}}$.

Note, we assume here that the task has already been broken down into $N$ discrete steps. For a generalization to the continuous setting where $N$ can vary along the real numbers, see Chapter 5.

[1]For sufficiently large $R$, the specific choice of $R$ will not qualitatively change the learning dynamics of the student. The student will learn quicker, but the learning curves will have the same shape. If $R$ is chosen too small, the student experiences a bottleneck and its Q-values never saturate.

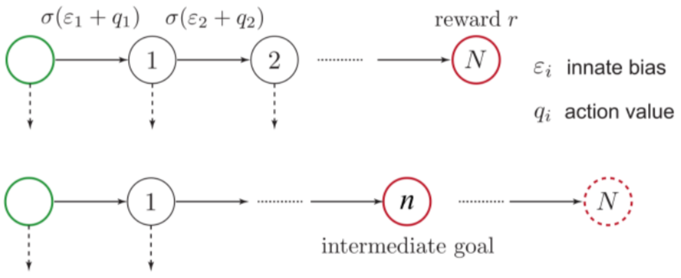
$\sigma(\varepsilon_1 + q_1)$
$\sigma(\varepsilon_2 + q_2)$
reward $r$
1
2
$N$
$\varepsilon_i$ innate bias
$q_i$ action value
1
$n$
$N$
intermediate goal

**Figure 3.1: The sequence learning task**. The task is broken into $N$ discrete steps, with some probability of advancing based on the student's Q-values and innate bias parameter $\varepsilon$. Reward is dispensed only if the student reaches the final state. If $N$ is very large, the probability of doing so becomes vanishingly small, necessitating a curriculum consisting of a sequence of intermediate tasks with length $n$. By scheduling progressively more difficult, longer tasks, the student will eventually learn the final task $N$. The optimal teacher is one which trains a student with the fewest number of intermediate tasks.

### 3.1.1 Model of the student

A student in the sequence learning setting is modeled as an Expected SARSA reinforcement learning agent.[32,33] For each state $i$, the student has two Q-values: $q_i^{\text{correct}}$ and $q_i^{\text{incorrect}}$. Because the incorrect action is never reinforced, we always have that $q_i^{\text{incorrect}} = 0$. For simplicity, we omit specifying $q_i^{\text{halt}}$ explicitly, and refer to $q_i^{\text{forward}}$ as simply $q_i$.

The student follows a softmax policy where the probability of moving forward from state $i$ is given by

$$\pi(\text{forward} \mid i) = \sigma(q_i) \tag{3.1}$$

where $\sigma$ is the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$.

In the real world, different students have different propensities for learning a task. One lab mouse may be easier to train than another. One dog may have a better sense of smell, and track a trail more efficiently than another. One student may learn quicker than another, and follow a correspondingly accelerated curriculum. Modeling a student's propensity for a task is therefore an essential consideration when designing curricula.

In the sequence learning setting, we incorporate this property by introducing a bias term $\varepsilon$. This term serves two related purposes:

1. **Model the student's innate propensity for success**: every student has a different set of innate talents and preferences. Modulating the $\varepsilon$ allows us to simulate students with abilities that are more or less aligned with the task.

2. **Model the student's extrinsic chance of success**: for more complex tasks, a student often has to choose one correct option among many incorrect choices. Modulating $\varepsilon$ controls the student's initial probability of success, capturing extrinsic factors that may aid or hinder the student's progress.

To model these effects, each Q-value decomposes as $q_i = q_i^r + \varepsilon_i$, where $\varepsilon_i$ remains fixed. The value of $\varepsilon$ can change for different steps $i$, but we typically allow $\varepsilon$ to be the same for all $i$ and simply have $q_i = q_i^r + \varepsilon$.[2] The value of $q_i^r$ is updated using the Expected SARSA update rule. Upon moving forward from state $i$, the value of $q_i^r$ becomes

$$q_i^r \leftarrow q_i^r + \alpha(r + \gamma \langle q_{i+1}^r \rangle_\pi - q_i^r) \tag{3.2}$$

Note, the expectation $\langle q_{i+1}^r \rangle_\pi$ is computed including bias $\varepsilon$, so

$$\langle q_{i+1}^r \rangle_\pi = \sigma(q_{i+1}^r + \varepsilon) q_{i+1}^r \tag{3.3}$$

[2]Practically, allowing $\varepsilon$ to vary from step to step has little impact on the qualitative behavior of each teacher algorithm.

To simplify the analysis, we assume the student has an infinite horizon, and allow the discount factor to be $\gamma = 1$.

## 3.2 Qualitative learning dynamics

With this task in hand, we can observe the qualitative dynamics of a student attempting to learn a sequence task, and get a feel for the requirements of an optimal teacher.

With $\gamma = 1$, the update rule in Equation 3.2 simplifies to

$$\Delta q_i^r = \begin{cases} \alpha(\langle q_{i+1}^r \rangle_\pi - q_i^r) & i < N \\ \alpha(R - q_N^r) & \text{i = N} \end{cases} \tag{3.4}$$

For $\alpha \ll 1$, we can approximate these dynamics with a system of differential equations

$$\begin{aligned} \dot{q}_N^r &= R - q_N^r \\ \dot{q}_{N-1}^r &= \langle q_N^r \rangle_\pi - q_{N-1}^r \\ \dot{q}_{N-2}^r &= \langle q_{N-1}^r \rangle_\pi - q_{N-2}^r \\ &\vdots \\ \dot{q}_1^r &= \langle q_2^r \rangle_\pi - q_1^r \end{aligned}$$

Suppose at $t = 0$, we initialize all Q-values as $q_N^r = q_{N-1}^r = \ldots = q_1^r = 0$. Then initially, $\dot{q}_N^r > 0$ and $q_N^r$ increases until it saturates at $R$. As $q_N$ increases, the value $\langle q_{N^r} \rangle_\pi$ also increases, which causes $q_{N-1}^r$ to increases. Eventually, $\langle q_N^r \rangle_\pi = q_N^r \sigma(q_N^r + \varepsilon) = R$ for $R \gg 1$, which causes $q_{N-1}^r$ to also saturate at $R$. And so it goes down the line, with subsequent values $q_i^r$ increasing towards $R$ until all Q-values are saturated. Reddy[22] describes this phenomenon as "reinforcement waves." Because the learning rule is local, reward updates propagate from the end of the task towards the beginning as a traveling wave, sequentially reinforcing each step along the sequence. Reinforcement waves can be witness in the trajectory plots below, e.g. Figure 3.3.

Once all Q-values are saturated and the task has been completely learned, what happens if we update the curriculum? In particular, what happens if we increase the difficulty of the task from $N$ to $N+1$? We update our learning dynamics slightly to accommodate the new step

$$\begin{aligned}
\dot{q}_{N+1}^r &= R - q_{N+1}^r \\
\dot{q}_N^r &= \langle q_{N+1}^r \rangle_\pi - q_N^r \\
&\vdots \\
\dot{q}_1^r &= \langle q_2^r \rangle_\pi - q_1^r
\end{aligned}$$

and initialize all Q-values at $t = 0$ as $Q_{N+1}^r = 0, Q_N^r = Q_{N-1}^r = \ldots Q_1^r = R$. Again, we have the effect where $\dot{q}_{N+1}^r > 0$, and $q_{N+1}^r$ increases gradually before saturating at $R$. However, because $\langle q_{N+1}^r \rangle_\pi$ is initially zero and $q_N^r = R$, we must have that $\dot{q}_N^r < 0$, so $q_N^r$ decreases initially. Once $q_N^r$ decreases, the value of $\langle q_N^r \rangle_\pi$ also decreases, so $q_{N-1}$ also decreases. And so the effect propagates back towards the start, decreasing each Q-value in turn. In this way, parallel to the reinforcement wave, there exists an extinction[3] wave that erases learning progress. Eventually, once $Q_{N+1}^r$ saturates towards $R$, a second reinforcement waves sweeps

[3] "Extinction" as in *extinguishing* a conditioned response, rather than an *extinction* of species.

through all Q-values, bringing them back to their original heights. Extinction waves can be witnessed in the plots below, e.g. again in Figure 3.3.

The goal of the teacher is to balance reinforcement with extinction. On the one hand, a teacher must select an easy enough task such that a reinforcement wave can propagate quickly through all Q-values. On the other, the teacher must propose more difficult tasks to facilitate efficient learning, yet not so difficult as to extinguish existing progress. As we will see, developing a teacher algorithm that achieves this balance is the heart of optimal curriculum learning.

## 3.3 First pass: an incremental teacher algorithm

As a baseline model, one intuitive strategy for building a curriculum is to assume an incremental approach: the teacher proposes tasks of incrementally increasing difficulty as the student masters each successive level. For the sequence learning task, if the goal is to learn a task of length $N$, the teacher would start with task of length 1. Once the student masters length 1, the teacher proposes length 2. Once the student masters length 2, the teacher proposes length 3. And so on until the student masters length $N$. See Algorithm 2 for a concrete description of this process.

At first glance, an incremental approach sounds very reasonable, and indeed, Incremental works well for students with $\varepsilon > 0$. However, for $e^{\varepsilon} \ll 1$, Incremental begins to fail (Figure 3.2). The reason is related to extinction waves. Upon proposing the next task, the student experiences a period of extinction before reinforcement catches up. For low values $\varepsilon$, the extinction wave becomes quite severe, and can erase much or even all of the student's progress up to that level (Figure 3.3). The problem is compounded by the fact that the rein-

**Algorithm 2** Incremental teacher for discrete curriculum

```
Require: target difficulty $N$, threshold success rate $\tau$
  $n \leftarrow 1$
  while $n \leq N$ do
    $s_n \leftarrow$ EVALUATE(Student)
    if $s_n > \tau$ then
      $n \leftarrow n + 1$
    else
      Train Student on task level $n$
    end if
  end while
```

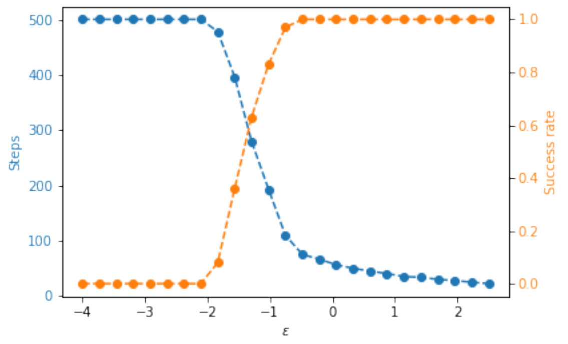

**Figure 3.2: Number of steps required to train a student on a task $N = 10$, across various levels of $\varepsilon$.** In blue is the number of steps it takes to train a successful student. In orange is the proportion of successful students the teacher was able to train. Incremental succeeds quite well for positive $\varepsilon$, but below $\varepsilon = -2$, Incremental fails to train the student at all, and the max allowed steps is reached.

forcement wave takes far longer to propagate for cases in which $e^{\varepsilon} \ll 1$ and the task level is high, further crippling the student's ability to recover after a large extinction wave. Hence, while Incremental performs well for high values of $\varepsilon$, it fails to propose effective curricula for students with low $\varepsilon$.

Ultimately, it is precisely those students with low $\varepsilon$ that benefit the most from a well-designed curriculum. Students with high $\varepsilon$ possess the ability to learn difficult versions of a task already, somewhat trivializing the need for curriculum learning altogether. Thus, we

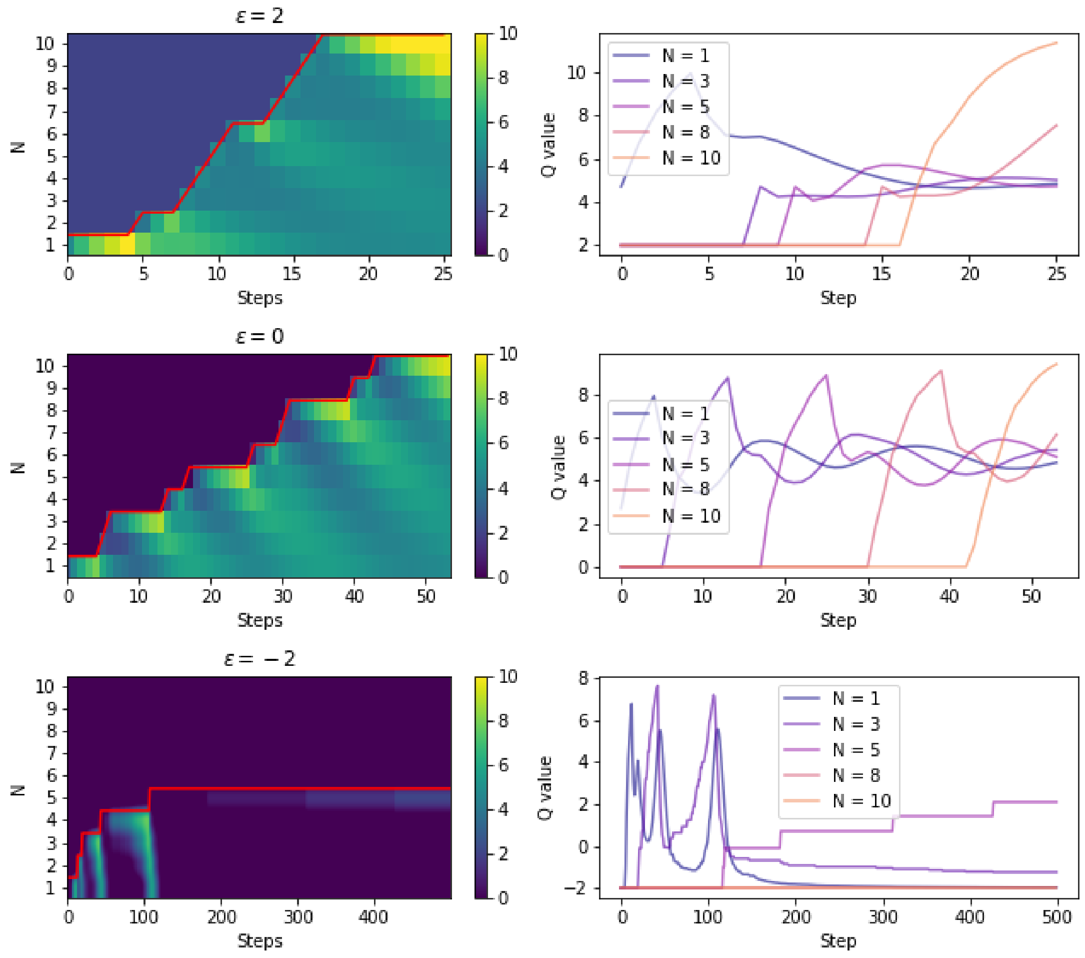

**Figure 3.3: Trajectory of the student across different levels of $\varepsilon$ under an Incremental teacher**. The student's trajectory is plotted in red, on the left. The Q-values ($q^r + \varepsilon$) are shaded in the background. On the right are trajectories of individual Q-values corresponding to different $N$, through time. As time moves from left to right, every time the curriculum increments, there is a brief dip in Q-values that propagates back towards the start. The dip is more pronounced for lower values of $\varepsilon$, to the point where the student's progress is wholly erased in the $\varepsilon = -2$ case.

focus particularly on students with $e^{\varepsilon} \ll 1$, for which Incremental is far from the perfect solution. We next explore an approximately optimal approach to proposing curricula that does succeed in training low-$\varepsilon$ students.

## 3.4 An approximately optimal teacher algorithm through POMCP

Just as the student is a reinforcement learning agent, the teacher can also be considered a reinforcement learning agent attempting to discover an optimal policy for a partially-observable Markov decision process (POMDP).[37] Informally, the teacher's observation space consists of the student performance history up to the current time. After observing the student perform for a time, the teacher must take an action: namely selecting the next level of task to present. The process is partially observable because the teacher lacks full knowledge of the student's state, and must infer the student's true level of performance from the trial history. Discovering an optimal policy for the resulting POMDP would shed insight on how an optimal curriculum should look for any particular student.

Formally, we define the following POMDP underlying the teacher's decision process:

- **State space**: states consist of a 3-tuple $(q^r, \varepsilon, \alpha)$ where $q^r$ represents all of the students learnable Q-values, $\varepsilon$ represents the student's innate ability bias, and $\alpha$ represents the student's learning rate. In effect, the state is the set of values that fully define the student's ability to perform the sequence learning task.

- **Action space**: integers $1, 2, \ldots N$, which correspond to the next task level that the student will see. In practice, to make solving the POMDP more tractable, we limit the action space to 3 values: 1) increment the task difficulty, 2) decrement the task difficulty, 3) keep the same task difficulty

- **Transitions**: transition probabilities from $(q^r, \varepsilon, \alpha)$ to $q^r_{\text{new}}, \varepsilon, \alpha)$ can be computed using the SARSA update rule in Equation 3.2

- **Reward function**: for a pre-determined success threshold $\tau$, if a student's rate of success at the final task exceeds $\tau$, the teacher receives a fixed reward $R$. Concretely, the student's rate of success at a sequence of length $k$ is

  $$\mu_k = \prod_{i=1}^{k} \sigma(q_i^r + \varepsilon) \tag{3.5}$$

  Let $s^* = (q^r_*, \varepsilon, \alpha)$ be a 3-tuple for which $\mu_N > \tau$. Then any transition that terminates in $s^*$ receives reward $R$, and the episode terminates. No other transition is rewarded, or terminates the episode (unless a pre-determined max number of iterations is exceeded).[4]

- **Observation space**: the teacher observes the true success rate of the student $\mu_k$ at a sequence task of the current length $k$. Hence the observation model can be given as

  $$O(o \,|\, s' = (q^r, \varepsilon, \alpha), a = k) = \Pr(\mu_k = o) \tag{3.6}$$

  Practically, to make solving the POMDP tractable, we sort the logit $\sigma^{-1}(\mu_k)$ into one of $\ell$ bins, and return the bin index as the observation. More bins provides greater resolution, but $\ell = 10$ seems to work well in practice.

[4]As before, we set $R = 10$. The magnitude of the reward has no practical impact on the qualitative behavior of the teacher algorithm, so long as it's positive.

We use the Partially Observable Monte-Carlo Planning (POMCP) algorithm[27] to approximately solve this POMDP. The resulting teacher proposes curricula that differ from Incremental in several striking ways (Figure 3.4). Most importantly, rather than monotonically increment the curriculum by steady intervals, the POMCP teacher backtracks to earlier levels in an oscillatory movement. By alternating between two or more levels, the POMCP teacher pre-empts an extinction wave before it propagates all the way back to the start, erasing the student's progress. In this way, the teacher encourages new reinforcement waves to form at higher levels while mitigating the impact of extinction, and successfully trains a student even for severely low $\varepsilon$.

One obstacle for fully-embracing the POMCP teacher is interpretability. POMCP is a tree-search algorithm that simulates an optimal policy through structured roll-outs, simulating the dynamics of a task far enough ahead to observe whether a particular action ends in reward. POMCP does not offer higher-level justifications for *why* a particular action is desirable, only that it seems to attract more reward on average. Hence, while we have a useful example of how an ideal teacher should operate, we are left without mechanistic principles for constructing such a teacher. We next turn our attention towards developing a teacher that produces the same high-level behavior of a POMCP teacher while also revealing mechanistic principles for structuring optimal curricula.

## 3.5 Adaptive Teacher

A key insight from the success of the POMCP teacher is that "backtracking" is essential to the success of a curriculum. Rather than increment in gradual, monotonic steps as the student learns, it is important to interleave easier levels periodically so as to counter extinction

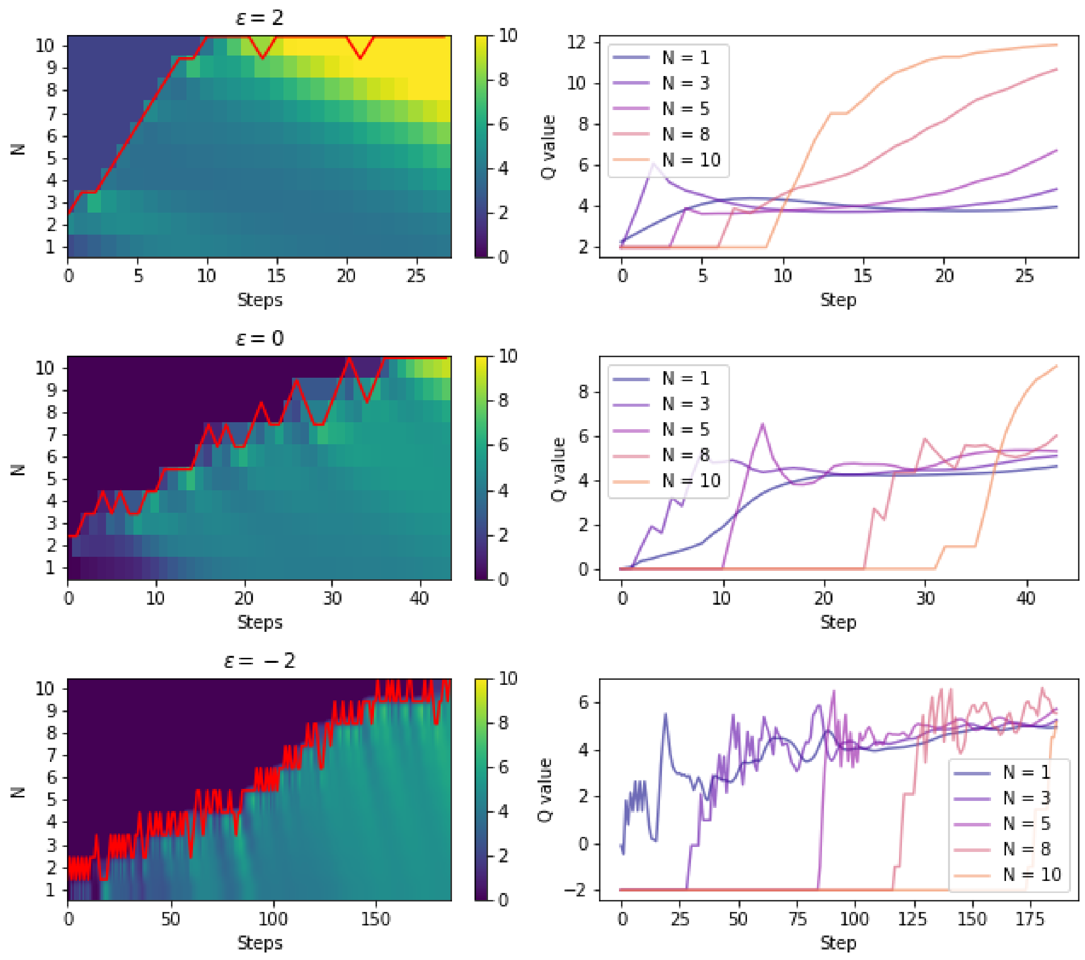

**Figure 3.4: Trajectory of the student across different levels of $\varepsilon$ under a POMCP teacher.** The student's trajectory is plotted in red, on the left. The Q-values ($q^r + \varepsilon$) are shaded in the background. On the right are trajectories of individual Q-values corresponding to different $N$, through time. The trajectory is noticeably oscillatory. Rather than increment in steady intervals, the curriculum oscillates regularly between two or more levels. In doing so, an extinction wave is pre-empted before propagating towards the start, and the student's progress is preserved. Compare with Figure 3.2.

effects. Whereas POMCP decides the the points at which to backtrack through a blackbox search procedure, we present an alternative algorithm that explicitly optimizes for points to backtrack: the Adaptive teacher.

The Adaptive teacher works by learning a decision tree that decides whether to increment or decrement the current task level. As input, the Adaptive teacher receives a history of success rates $s^{(1)}, s^{(2)}, \dots s^{(t)}$ estimated from the student at each step up to the current time $t$. A *first order* Adaptive teacher will additionally compute $\Delta s^{(1)}, \Delta s^{(2)}, \dots \Delta s^{(t)}$ where $\Delta s^{(i)} = s^{(i)} - s^{(i-1)}$ (and $s^{(j)} = 0$ for $j < 1$). When making a decision, the Adaptive teacher then compares the current success measures $(s^{(t)}, \Delta s^{(t)})$ against a decision tree whose leaf nodes correspond to one of three possible actions: 1) increment task level, 2) decrement task level, and 3) stay at current task level. See Figure 3.5 for a simple example of what this decision tree looks like.

The precise splits and leaves of the trees can be optimized using any number of popular search procedures[31,11,2,4]. The Adaptive teacher can be further customized with the addition of additional "features." In the first-order teacher, we include the first-order differences among estimated success rates as an additional feature for the decision tree. We could also implement a *second-order* teacher, that includes features $\Delta^2 s^{(i)} = \Delta s^{(i)} - \Delta s^{(i-1)}$. Or we could include arbitrary features $\Phi(s^{(i)}) = f(s^{(1)}, s^{(2)}, \dots s^{(i)})$ for some arbitrary function $f$. The possibilities are endless for tailoring Adaptive teachers of varying expressivity to the nuances of a particular task. For this simple sequence-learning task, the features $(s^{(t)}, \Delta s^{(t)})$ are perfectly adequate to produce a successful teacher. We optimize the decision tree using differential evolution[31].

See Figure 3.6 for plots of the student's trajectory under a curriculum provided by an

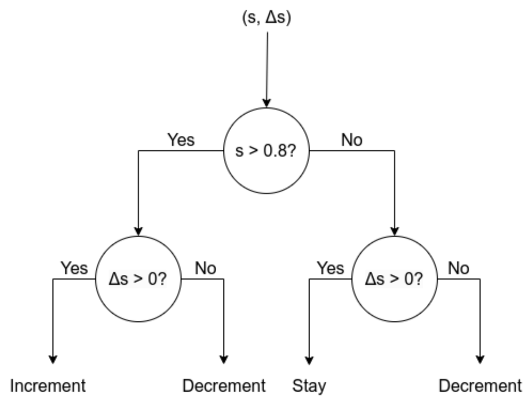

**Figure 3.5: Decision tree employed by Adaptive teacher for the sequence learning task.** At every iteration, the teacher receives as input an estimate of the student's current success $s$ as well as the difference between the current and last step $\Delta s$. The teacher then follows the decision branches to one of three actions: 1) increment the task difficulty, 2) decrement the task difficulty, 3) stay at the same difficulty.

Adaptive teacher. The trajectories are strikingly similar to those under the POMCP teacher, and mitigate exctinction waves through a similar oscillatory strategy.

Though both the POMCP and Adaptive teachers are developed through optimization, the Adaptive teacher is interpretable. Figure 3.5 illustrates the decision tree used by the Adaptive teacher on the sequence learning task. The teacher establishes a split on $s > 0.8$ that determines whether the student is performing well or poorly. If the student is performing well, improvement ($\Delta s > 0$) implies that the task should become harder. On the other hand, if the student is performing poorly ($s < 0.8$), improvement implies that the student should stay at the current level and get better. If the student is ever worsening ($\Delta s < 0$), the current task must be too difficult, and the teacher decrements the task to an easier level. These simple rules offer a *mechanistic* explanation for an optimal teacher's behavior, and reveal useful principles for scheduling curricula that may generalize to more complex settings (the topic of Chapter 4).

They also align well with our understanding of reinforcement and extinction waves. If

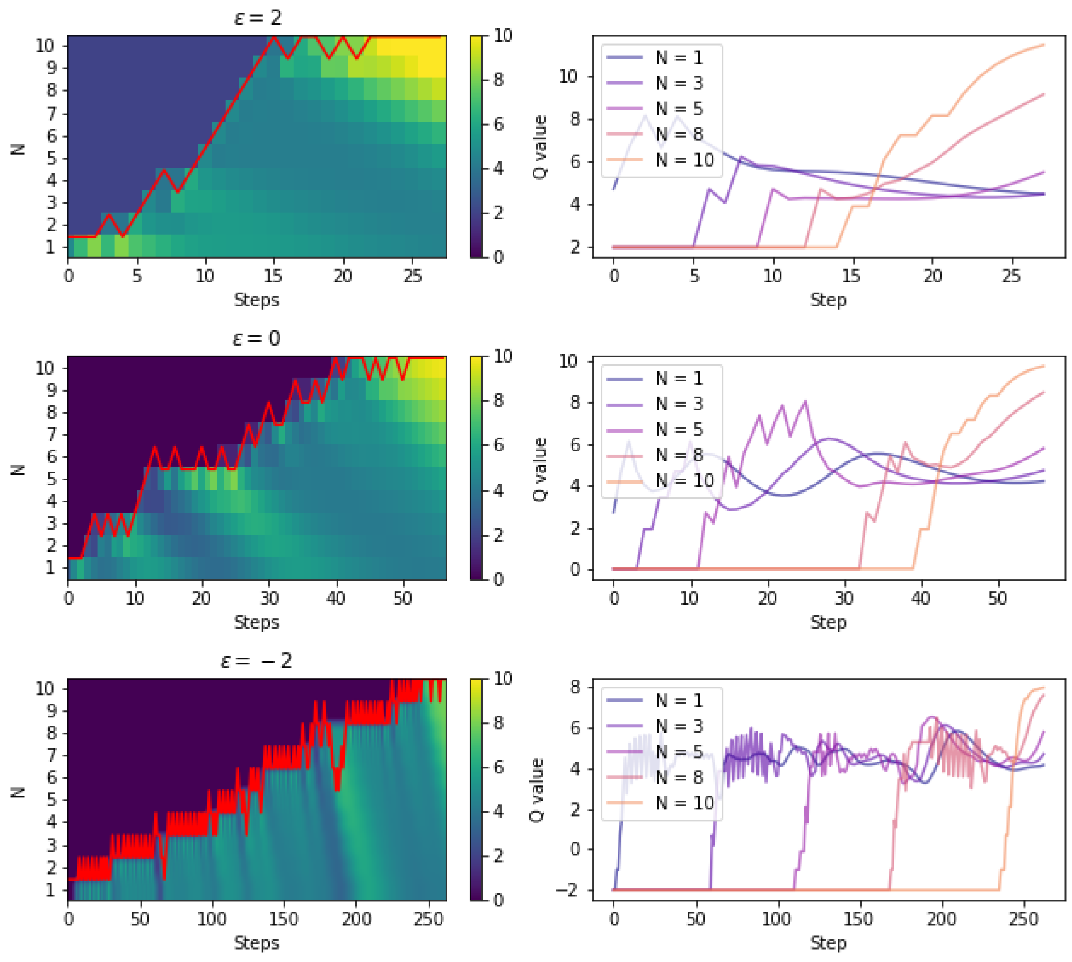

**Figure 3.6: Trajectory of the student across different levels of $\varepsilon$ under an Adaptive teacher.** The student's trajectory is plotted in red, on the left. The Q-values ($q^r + \varepsilon$) are shaded in the background. On the right are trajectories of individual Q-values corresponding to different $N$, through time. The trajectory is strikingly similar to that of the POMCP teacher, and oscillates frequently, preventing extinction waves from propagating back to the start. The resulting curriculum trains students even with very low $\varepsilon$.

a student is improving, a reinforcement wave is likely propagating, so the teacher should wait for the student to attain mastery (e.g. $s = 0.8$) before advancing to the next level. On the other hand, if the student is worsening, an extinction wave must be propagating, so the teacher should decrement levels to ameliorate the wave before it can erase all progress. Oscillatory dynamics arise naturally as a student attempts a new level. Upon first encountering a more difficult task, an extinction wave begins, causing the teacher to switch to an easier task. Now on the easier task, the extinction wave is squelched and the student improves, advancing it to the harder task. Oscillation continues until sufficient progress on the harder task obviates the genesis of further extinction waves.

In comparing the performance of Adaptive and POMCP teachers, it does not appear that Adaptive performs substantially worse (if at all). Hence, in choosing an interpretable framework over blackbox optimization, we lose little in the way of efficient teachers. Indeed, perhaps POMCP can be thought of as ultimately following a strategy not dissimilar to the decision process outlined in Figure 3.5. To conclude our discussion of teacher algorithms in this simplified sequence learning setting, we next compare performance across our teacher algorithms for a range of students and task lengths, and against teacher algorithms proposed in Matiisen et al.[15].

## 3.6 Comparison of teacher algorithms

Overall, we have identified a simple sequence learning task that captures many common features across tasks amenable to curriculum learning. Implementing an optimal curriculum is challenging in its need to balance reinforcement with waves of extinction. A naïve incremental strategy struggles to handle students with low $\varepsilon$ that generate large extinction waves.

Recasting the problem as a POMDP and optimizing directly reveals the need for oscillatory dynamics that prevent extinction waves from propagating to the start, erasing all progress. Optimizing an interpretable Adaptive teacher based on a decision tree reveals mechanistic principles that an optimal teacher applies for constructing efficient curricula. To observe how these teachers perform on a wide variety of settings, we test these teachers on many different task lengths across a wide variety of students in Figure 3.7.

As expected, the Incremental teacher tends to fail by $\varepsilon = -2$, but otherwise performs quite well. The POMCP and Adaptive teachers are comparable in performance, though Adaptive is occasionally outperformed by a slight margin compared to POMCP. All teachers perform relatively well for high $\varepsilon$. Low $\varepsilon$ and high task lengths tend to differentiate the more efficient teachers.

We also compare the performance of these teachers to alternatives proposed in Matiisen et al.[15]. To our knowledge, this work is the only other systematic study of teacher algorithms for curriculum design in a reinforcement learning setting. Their premise is based on a simple heuristic: a student should attempt the task level at which it makes the fastest progress (as measured by the slope of its learning curve). In particular, they explore four algorithms implementing this principle:

- **Online algorithm**: each task level $a$ is assigned a Q-value $Q_t(a)$. Upon performing a task, the $Q$ value is updated by an exponential moving average $Q_{t+1}(a) = \alpha r_t + (1 - \alpha)Q_t(a)$. The "reward" $r_t$ is calculated as the difference between the current and previous attempts of the task $r_t = x_t^{(a)} - x_{t'}^{(a)}$. The next task is then sampled using a softmax policy with inverse temperature parameter $\beta$:

$$\pi(a) = \frac{e^{\beta|Q_t(a)|}}{\sum_a e^{\beta|Q_t(a)|}}$$

In this way, the teacher prioritizes tasks for which the greatest progress seems to be made. Note: Q-values are computed using their absolute values in the softmax policy, which will also prioritize tasks that are worsening quickly.

- **Naive algorithm**: rather than use the simple difference to calculate $r_t = x_t^{(a)} - x_{t'}^{(a)}$, the Naive algorithm repeats each task $K$ times, and performs a linear regression on the resulting rewards. The slope of the scores is used as $r_t$ in the Q-value calculation.

- **Window algorithm**: the same strategy as the Naive algorithm is pursued, except rather than discard each run upon completion, all runs are kept up to a predetermined history length $K$. Regression is then performed including the timestamps of each reward.

- **Sampling algorithm**: the same strategy as the Online algorithm is pursued, except rather than maintain an exponential moving average $Q$, the last $K$ reward are kept. Upon selecting the next task, a reward $r_t$ is sampled from each task, and the task with the largest $|r_t|$ is scheduled next.

There is some justification for Matiisen's heuristic under our reinforcement / extinction wave framework. By selecting tasks that are improving quickly, the heuristic encourages the propagation of reinforcement waves. By selecting tasks that are also worsening quickly, the heuristic plausibly halts extinction waves. However, the heuristic is insensitive to order

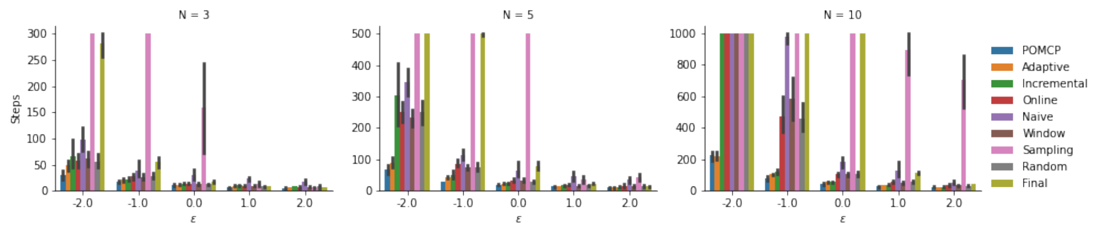

**Figure 3.7: Benchmarks comparing performance on the sequence learning task across teachers.** Teacher algorithms are named in the legend. "Random" refers to a random curriculum, where levels are select uniformly by chance. "Final" refers to the absence of any curriculum — the student is trained directly on the final, hardest task. Algorithms from Matiisen et al. [15] are routinely outperformed by other teachers, including "Final" in many cases, calling into question their true efficacy. Among our algorithms, POMCP illustrates the performance ceiling, but Adaptive is remarkably close. As expected, Incremental fails for low $\varepsilon$.

among tasks — all tasks are treated the same, regardless of difficulty or intrinsic sequence. The heuristic also fails to scale with curricula containing many tasks. In these cases, much time is wasted exploring the tasks sufficiently to develop a good estimate of $r_t$.

These weaknesses are evident in our benchmarks comparing our teacher algorithms against those of Matiisen et al. [15] (Figure 3.7). Algorithms from Matiisen et al. [15] tend to perform poorly, particularly for nontrivial cases with low $\varepsilon$ and high task length $N$. Indeed, in many cases, the Sampling algorithm is outperformed by using no curriculum at all. Simpler algorithms like Online and Window tend to perform somewhat better, though not quite as efficiently as Adaptive. Altogether, while Matiisen's heuristic sounds plausible, it does not exploit the order and structure of tasks, losing much time estimating performance on tasks that should already be considered too difficult.

All in all, our collection of teacher algorithms works well in this simple sequence learning setting, but it remains to be seen whether they generalize to more complex settings. In the next chapter, we turn our attention to two naturalistic tasks that measure how well some of the basic principles developed in this chapter might transfer to a real-world setting.

*Reality was no match for mathematics.*

BF Skinner

# 4 Task: naturalistic tracking

Trail tracking is a behavioral paradigm in which an animal executes a series of discrete odor measurements in pursuit of a target. Odors can be terrestrial or airborne, concentrated along a single thin trail or diffused along turbulent air currents. Examples include a dog tracking the scent of deer, a moth following pheromones to find a mate, or a mouse pursuing fragrant hints of last night's leftovers. This behavior is prevalent and essential across species, necessary for navigation, foraging, and mating[8,12,35]. Yet despite the importance and ubiquity of trail tracking, strategies supporting the behavior remain relatively poorly understood.

Trail tracking offers a novel experimental testbed for reinforcement learning algorithms. The task is nontrivial to perform, has important biological relevance, and lends itself naturally to curricula. By parameterizing the task difficulty based on distance between the agent's starting location and the destination, we can employ the same teacher strategies developed in the the sequence learning setting, and observe whether the same high level principles continue to hold for trail tracking.

In this chapter, we explore two specific variants of trail tracking: 1) ground-based terrestrial trails, which we refer to as simply "trail tracking" (Section 4.2), and 2) plume-based airborne trails, which we call "plume tracking" (Section 4.3). For both settings, we employ the same deep reinforcement learning framework to simulate naturalistic tracking scenarios, described next.

## 4.1 A deep RL framework for naturalistic tracking

To handle the full complexity of a naturalistic tracking scenario, we move beyond the simple sequence learning task constructed in Chapter 3 and pursue a deep reinforcement learning framework. Given the astonishing successes of various deep RL algorithms on video game tasks[16], we simulate trail tracking directly in pixel data, and allow the agent to learn both a sensory stack for interpreting the image inputs, as well as a navigational policy for finding the target.

### 4.1.1 Observation space

For both trail and plume tracking, the agent is centered on a flat, 2D surface without landmarks or obstructions. Odor is either distributed along a thin trail (for trail tracking) or diffused in simulated plume (for plume tracking). As the agent navigates this landscape, it encodes its position history together with the last few odor detections within each individual image observation.

The observation space itself consists of a 50 by 50 grid of pixels, with 3 color channels per pixel, producing a full RGB image and 7500 possible unique states. Although the agent's position is recorded continuously during each episode, when producing the pixel observations, it is discretized to the pixel grid. The three color channels of the image encode different information. The first channel (i.e. the "red" channel) encodes the agent's position history. The second channel (i.e. the "green" channel) encodes the agent's odor measurements. The last channel (i.e. the "blue" channel) is left unused.

At each timestep, the agent moves to a different location and performs an odor measurement. Position history is recorded as a thin continuous line in the image observation's red channel, linearly interpolating between discontinuous points. Odor history remains discrete, and is depicted as colored patches at each location where the agent performed an odor measurement. This difference reflects the fact that motion through a space is continuous, but odor measurements occur only when the agent "sniffs" the ground. Hence, odor measurements in history must be discrete. The strength of the odor is proportional to the pixel value in the green channel. The image overall remains centered on the agent, egocentric with respect to the agent's position and heading. As the agent moves in a particular direction, the whole "viewport" moves and rotates with the agent, leaving a track that spreads

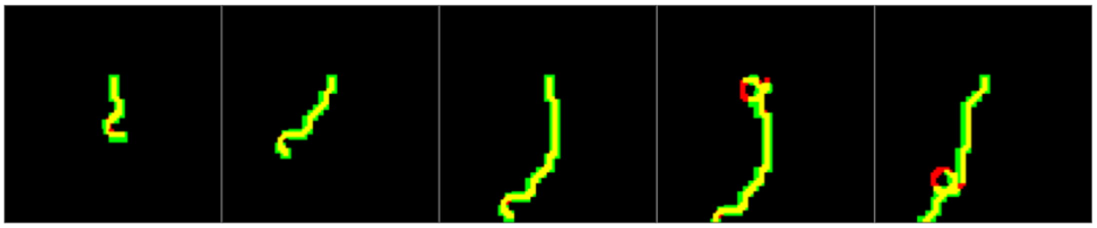

**Figure 4.1: Example observations sampled from various points along an agent's trajectory**. The agent is always fixed at the center of image, facing north. All other coordinates are translated along this egocentric reference frame. The agent's past positions are traced in red. Past odor samples are plotted in green.

out away from the center recording the agent's past. See Figure 4.1 for an example observation that the agent receives.

### 4.1.2 Action space

At each timestep, the agent can take one of three actions: 1) move left, 2) move right, 3) move forward. Choosing the move left (right) action will decrement (increment) the agent's heading by 45°. [1] After any heading updates, the agent's position is then incremented by three units along its new direction.

### 4.1.3 Reward

If the agent lands within 3 units of the target (either the end of a trail or source of a plume), the episode terminates with a large, fixed reward. Otherwise, if the agent fails to reach the target within a predetermined max number of iterations, the episode terminates without reward. No other actions are reinforced, nor are any negative rewards ever applied.

Note, the reward scheme is intentionally sparse. If an agent were to attempt a long trail

[1] Heading angles are calculated like a compass. North corresponds to 0°, with angles increasing in the clockwise direction.

without additional aid, it will fail to converge towards a successful strategy due to the lack of sufficient reinforcement signals. A traditional approach to addressing this sparsity is through reward shaping[19,13], a process whereby the designer supplies supplementary rewards that guides the agent towards successful behavior. However, reward shaping is difficult to implement in practice, may require significant assumptions about the student, and different shaping strategies may have unpredictable impacts on the agent's overall behavior.[32,36] Instead, we use curriculum learning to overcome the sparsity issue. By starting the agent on short, easy trails for which the reward scheme is not sparse, then gradually lengthening the distance to the target, the agent naturally learns a successful tracking strategy without the heavy-handed additions of reward shaping.

#### 4.1.4 Model

We use Proximal Policy Optimization (PPO)[24,21], a popular deep RL algorithm that achieves state of the art on a wide variety of discrete and continuous tasks. The agent uses a deep convolutional neural network to extract operable features from each image observation, followed by several fully-connected layers to infer state values and output actions. For specific implementation details and hyperparameter settings, please see Appendix B.

## 4.2 Trail tracking

Surface-bound odor trails are long, thin segments of concentrated odor with minimal diffusion through the air. To sense these trails, an animal must be relatively close, and may not always sustain contact. Further, these trails often have breaks: significant stretches during which odor is absent, and the animal must execute a search strategy to regain contact.

Terrestrial trail-tracking is therefore a highly nontrivial behavior, and presents the first challenge for our deep RL agent to learn.

To construct naturalistic trail geometries, we use the procedure described in Reddy et al.[23]. Trails are shaped through the following parameters:

- **Length**: the distance between the agent's starting position and the target
- **Width**: the scale at which odor diminishes along the axis perpendicular to the trail. With width parameter $\sigma$ and a distance $x$ from the trail, the concentration of odor $o$ is proportional to
$$o \propto \exp\left(-\frac{x^2}{\sigma}\right)$$
- **Heading**: the angle between the agent's initial heading, and the direction of the target.
- **Shape**: the shape of the trail is governed by two parameters: curvature and diffusion rate. Curvature governs how the high-level shape of the trail evolves over time. Diffusion rate governs how "kinky" the trail is on short intervals.
- **Breaks**: a trail can have one or more breaks of variable length, along which there is no odor

The agent perceives the exact magnitude of odor at its current location, which is represented in the magnitude of the green color channel in each image observation. This is in contrast to Reddy et al.[23], which uses a Poisson-based odor detection model. See Figure 4.3 for a plot of an example trail, including the trajectory of a successful agent.

For the purpose of building curricula, the difficulty of a trail is rated by a combination of its length, width, and breaks. The heading of the trail is allowed to vary across the entire compass rose, and the shape parameters are held constant across all episodes. Curricula are discretized by hand, with predetermined difficulty levels that specify particular parameter combinations. [2] Because the teacher algorithms developed in Chapter 3 all rely on discrete curricula, this extra manual intervention is necessary to apply these algorithms. In an ideal case, teacher algorithms should generalize to continuous curricula, which we explore further in Chapter 5. At the start of every episode, a new trail is sampled using the parameter combination specified by the curriculum, and the agent must essay an unseen trail from the beginning.

### 4.2.1 Results

We evaluate the Incremental and Adaptive teachers, compared to a baseline random curriculum, as well as no curriculum at all (the agent is trained directly on the final task). Both algorithms are applied as-is, with no changes made between the sequence and trail tracking settings. However, slight tweaks were made to their hyperparameters to accommodate the more challenging nature of trail tracking, which are documented in Appendix B.

See Figure 4.2 for a plot comparing the performance of each teacher. Adaptive performs the best, followed by Incremental, then Random. Their trajectories show that Adaptive oscillates like before, presumably reducing any extinction or forgetting effects associated with the transition to more difficult tasks. Incremental performs comparable, but experiences a somewhat greater degree of forgetting as we see in the longer time it spends at the highest

[2] The specific difficulty levels are listed in Appendix B.

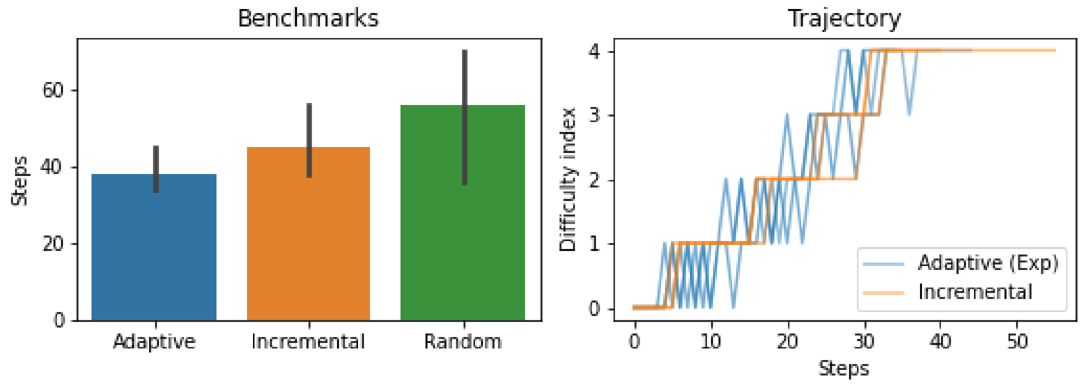

**Figure 4.2: Benchmarks across three teacher algorithms on the trail tracking task**. On the left are the number of steps required to train a successful student across the three algorithms. On the right are the trajectories of Adaptive and Incremental teachers (Random is random, so its trajectory is omitted). We see that Adaptive executes its characteristic oscillations, reducing forgetting effects and boosting its performance ahead of Incremental. A final-task-only teacher fails to learn on this setting, so is not plotted.

difficulty level. Random is surprisingly close to the other two teachers, but increasing the overall difficulty of the task will likely widen the differences between Random from the rest.

See Figure 4.3 for a plot of a trajectory made by a successful agent. Strikingly, though it was not the focus, the agent exhibits a surprisingly naturalistic trajectory. Like a real mouse or ant, the agent tracks the trail by casting back and forth along its heading, and exhibits a preference for the edge of the odor gradient rather than the center. The agent smoothly handles breaks and curves in the trail, and the length is no issue. Hence, despite receiving reward only at the destination, the agent nonetheless learns a set of rich navigation strategies for handling the complexities of trail tracking. Apart from efficiency metrics around any one particular teacher algorithm, curriculum learning remains a very robust strategy for teaching difficult tasks.

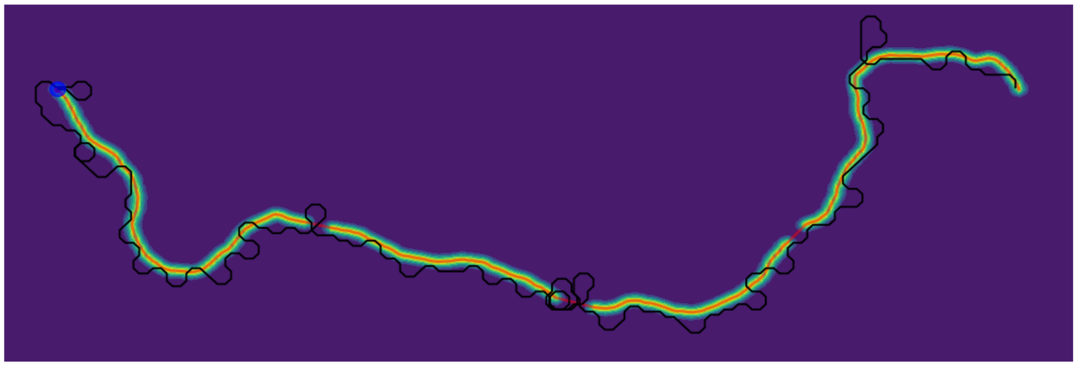

**Figure 4.3: Example of an agent following a surface-bound odor trail.** The agent's trajectory is outlined in black. The start point is highlighted with a blue dot. The red line plots the center of the trail. The color gradient corresponds to odor concentration. Note, the agent exhibits several behaviors commonly observed in animals. These including casting, in which the agent alternates back and forth along the trail, as well as edge-tracking, in which the agent displays a preference to follow the edge of the odor gradient rather than the center.

## 4.3 Plume tracking

Often times, odor diffuses through the air in turbulent plumes. For humans, this mode of olfaction is perhaps more commonly experienced than terrestrial odor trails: the smell of baked goods at the local pastry shop, fragrance from a spring garden, or the comforting scent of your home are all plume-based odors. Like terrestrial trails, odor plumes are essential for survival and reproduction across animal species. However, tracking the source of an odor plume presents its own unique difficulties. Plumes tend to be rarefied and clumpy, where air turbulence partitions regions of odor concentration into random, disconnected patches. An animal attempting to locate the source of a plume must infer its location based on sporadic contacts and indirect cues. Tracking odor plumes is by no means a trivial task, and presents another challenge for our deep RL agent.

To simulate naturalistic plumes, we use the model described in Vergassola et al.[34]. Odor plumes are shaped through the following parameters:

- **Wind speed**: a high wind speed produces long, elongated plumes. A low wind speed produces squat, round plumes. Zero wind produces a spherical plume.
- **Start rate**: the initial rate of detections at the agent's starting position. A low start rate implies that the agent will start further away from the source of the plume, and have a correspondingly harder task.
- **Particle properties**: additional parameters like diffusivity, lifetime, and emission rate are related to the properties of individual particles, and influence the overall shape of the plume.

In contrast to the terrestrial trails, where the agent perceives the exact magnitude of odor at its current location, the odor detection model in the plume setting is probabilistic, accounting for the random influence of turbulence. Odor detection is Poisson distributed, with rate given by the plume model. See Figure 4.5 for an example plume, including the trajectory of a successful agent.

For the purpose of building curricula, the difficulty of a plume is rated by its start rate, with lower start rates corresponding to more difficult trails. To ensure the agent is always downwind of the source, the agent's location is initialized within a fixed sector of a particular start rate. As in the terrestrial trail setting, because our teacher algorithms can only accommodate discrete curricula, we discretize the start rate into discrete levels for curriculum learning, with the specific levels given in Appendix B. A framework for handling continuous curricula is explored further in Chapter 5. The agent's starting *location* is re-sampled at the start of every new episode, though the starting *rate* remains the same for a particular task level. Particle properties and wind speed remain fixed for the entire duration of training.

### 4.3.1 Results

We evaluate the Incremental and Adaptive teachers from before, compared to a baseline random curriculum as well as no curriculum at all (in which the agent is trained directly on the final task). Both algorithms are applied as-is, with no changes between the sequence and plume tracking settings. However, slight tweaks were made to the hyperparameters to accommodate the more challenging nature of plume tracking, which are documented in Appendix B.

See Figure 4.4 for a plot comparing the performance of each strategy. Strikingly, Incremental seems to perform quite poorly in this setting, worse even than a random curriculum. Examining the trajectories gives a possible suggestion why. Though an Incremental teacher will often train a successful student quickly, in a nontrivial fraction of runs, the student experiences significant behavioral extinction, apparently erasing all progress. Secured by its backtracking oscillations, Adaptive continues to avoid major extinction events, and continues to train students efficiently.

See Figure 4.5 for a plot of a trajectory made by a successful agent. As in the terrestrial trail tracking case, an agent trained on a naturalistic plume task exhibits behaviors comparable to those observed in its animal counterparts. In particular, the agent exhibits "casting" and "zigzagging," counter-turning maneuvers in which the agent oscillates along the crosswind axis towards the odor source. Such behaviors are thought to help an animal regain contact with a rarefied odor[14], and we see it in abundance in the agent's trajectory.

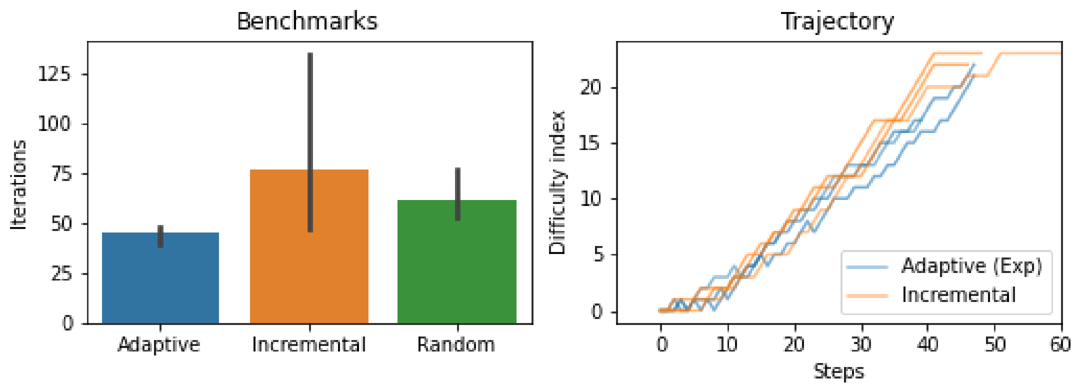

**Figure 4.4: Benchmarks on the plume tracking task.** Adaptive performs the best, followed by Random, then Incremental. Incremental's surprisingly poor performance seems to be in part due to its failure to prevent significant forgetting in the student. The trajectory on the right is truncated for easier viewing. However, several Incremental trajectories extend significantly to the right, (though the majority finish early), indicating that in a small but significant subset of runs, Incremental's student fails to prevent an extinction episode that erases significant progress. In contrast, Adaptive's oscillations continue to support efficient learning, putting it in the lead.

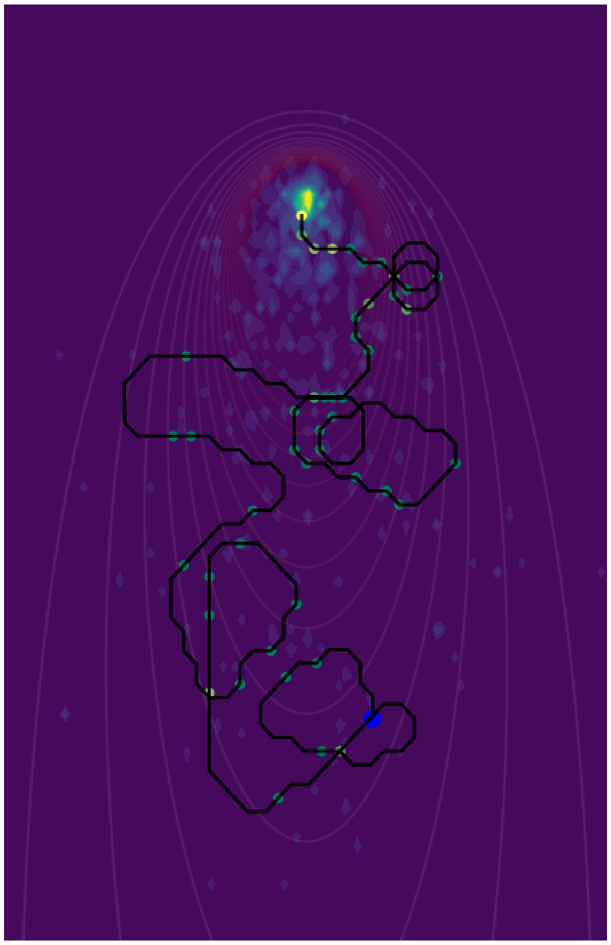

**Figure 4.5: Example of an agent following an airborne plume trail.** The agent's trajectory is outlined in black. The start point is highlighted with a blue dot. The red lines plot the odor detection rate contours. The color gradient represents a sample of odors at each point, and illustrates the sparsity of the plume. The colored dots along the agent's trajectory represent contacts at each point, where the brighter the dot the stronger the odor. As before, the agent exhibits behaviors commonly observed in animals, including "casting" and "zigzagging."

*The usual PhD thesis is a model of compulsive cautiousness, advancing only the most timid conclusions thoroughly hedged about with qualifications.*

BF Skinner

# 5 Continuous curricula

An important limitation of the discussion up to this point is that we assume curricula are always discrete. Whether we use an Incremental, POMCP, or Adaptive teacher, the task difficulty is discretized, allowing curricula to be built along fixed, distinct levels. In the case of the simple sequence-learning task developed in Chapter 3, this assumption was a non-issue as the task difficulty is naturally parameterized by the length of a particular sequence task. Indeed, many every-day examples may already be divided into a discrete difficulty levels: school subjects, performing arts, and certain sports like weightlifting all feature elements of discrete curricula.

However, as we witnessed in the naturalistic trail and plume tracking tasks of Chapter 4, there are also many compelling situations in which the underlying task difficulty is not discrete. The length of a trail or concentration of odor varies continuously. In these cases, we manually discretized task difficulty into prescribed difficulty levels, artificially defining ranges for length or odor concentration that map to a particular task level. However, defining these ranges by hand risks imposing substantial sub-optimalities on the curriculum. For example, if the jump from one level to the next is too large, the student will struggle to learn the next task regardless of the teacher algorithm. If the jump from one level to the next is too small, the student will learn easily, but take longer to pass through many levels where one might have been sufficient, wasting time and resources as the teacher evaluates the student more frequently than necessary. The "optimal" jump size must be one that balances ease-of-learning with efficiency.

Intuitively, the appropriate size for each jump is also related to the student's $\varepsilon$, the "innate bias" towards succeeding in a task. A large $\varepsilon$ implies that the student struggles with the task, and should observe smaller jumps. A small $\varepsilon$ implies the opposite, that the student is ready for larger jumps. On a continuous curriculum, the optimal teacher would therefore adjust its jump size to match the student. Hence, using a continuous curriculum introduces on additional optimization objective to the teacher algorithm: selecting the appropriate size of the increment from one task level to the next.

Introducing continuous curricula is a complex and nuanced topic. In this chapter, we introduce a set of preliminary ideas for addressing the issue, but defer a more complete study to future analysis.

## 5.1 Continuous sequence learning

It is not immediately obvious how to generalize the sequence learning task to a continuous curriculum. Here, we propose one approximate scheme that preserves many (but not all) of the intuitions from discrete sequence learning, though it is by no means the only possible scheme.

We begin with the following notion. In the discrete sequence setting, for an untrained student with fixed bias $\varepsilon$, the probability that the student advances from step $n$ to step $n+1$ is given by $\pi(n+1|n) = \sigma(\varepsilon)$. Suppose the student is allowed to move continuously. Then the probability that the student first moves to $n + \frac{1}{2}$, then from $n + \frac{1}{2}$ to $n+1$ must also be $\sigma(\varepsilon)$ overall. Hence, we must maintain that $\pi(n{+}1/2|n) \cdot \pi(n{+}1|n{+}1/2) = \sigma(\varepsilon)$. In other words, there exists a value $\varepsilon_{1/2}$ such that $\pi(n{+}1/2|n) \cdot \pi(n{+}1|n{+}1/2) = \sigma(\varepsilon_{1/2})\sigma(\varepsilon_{1/2}) = \sigma(\varepsilon)$. In this example, we can see quite clearly that

$$\varepsilon_{1/2} = \sigma^{-1}\left(\sqrt{\sigma(\varepsilon)}\right)$$

In the general case, if a student with *effective* bias $\varepsilon$ moves an interval $\Delta x \ll 1$, then we can identify a value $\varepsilon_{\Delta x}$ such that $\pi(n + \Delta x|n) = \sigma(\varepsilon_{\Delta x})$, where

$$\varepsilon_{\Delta x} = \sigma^{-1}\left(\sigma(\varepsilon)^{\frac{1}{\Delta x}}\right) \tag{5.1}$$

If we allow $\Delta x \to 0$, then we approach a continuous curriculum. In practice, for our simulations, we set $\Delta x$ to be a very small but nonzero quantity[1] to approximate a truly continuous setting.

In this way, we transfer much of the same intuition from the discrete sequence learning task. For an *effective* task length $N$ and bias $\varepsilon$, which matches $N$ and $\varepsilon$ from a discrete curriculum, the student operates on a "continuous" environment with $N_{\Delta x} = \frac{N}{\Delta x}$ and $\varepsilon_{\Delta x}$. Ideally, a student operating on $(N_{\Delta x}, \varepsilon_{\Delta x})$ should perform identically to a student operating on $(N, \varepsilon)$, as long as each curriculum increment is $1/\Delta x$. However, if we apply Equation 3.2 directly to update the student, we run into an important issue: only the previous Q-value is updated. For the continuous student, each Q-value represents an infinitesimal slice of ability. Applying the old rules as-is means we update only singular, infinitesimal slices at a time on the student. Intuitively, improving at a particular level should improve multiple Q-values simultaneously, and in a fashion that remains consistent with the discrete case. We model this effect by making a small change to the student's update rule.

### 5.1.1 N-step student

For a student on our continuous approximation of the simple sequence learning task, the number of Q-values grows as the unit of discretization $\Delta x$ decreases. Indeed, for a curriculum of effective length $N$, the number of Q-values becomes $N_{\Delta x} = \frac{N}{\Delta x}$. Without changes to the update rule described in Equation 3.2, the student is limited to updating one Q-value at a time. As $\Delta x \to 0$, the number of Q-values to-be-updated approaches infinity, requiring infinite time for any reinforcement wave to propagate to the start.

[1] $\Delta x = 0.01$ for all experiments that follow.

Intuitively, as a student develops one particular Q-value, neighboring Q-values should also be updated. After all, because $N \approx N - \Delta x$, if $q^r_N$ changes, $q^r_{N-\Delta x}$ should also change by approximately the same amount.[2] A natural way to incorporate this intuition is to change from a single-step update rule as described previously to $K$-step expected SARSA.[32] That is, rather than update each Q-value using its immediate next neighbor, The update rule now becomes

$$q^r_i \leftarrow q^r_i + \alpha \left( \sum_{j=1}^{K} \gamma^{j-1} r_j + \gamma^K \langle q_{i+K} - q^r_i \rangle \right)$$

where $r_j$ is the reward observed $j$ steps ahead of the current step $i$. Because reward is only dispensed at the very end of a successful run, we have that $r_j = 0$ for $j < N_{\Delta x}$. Using a discount $\gamma = 1$ as before, the update simplifies to

$$q^r_i \leftarrow q^r_i + \alpha(r_K + \langle q_{i+K} - q^r_i \rangle) \tag{5.2}$$

In effect, rather than updating from the immediate next-neighbor Q-value, the $K$-step student now updates using the Q-value $K$ steps ahead. Hence, upon encountering a reward, all previous $K$ Q-values are updated rather than just the immediately preceding one. On a later pass, as the student approaches this block of updated Q-values, the preceding block of $K$ values also receives updates as the intervals overlap. If we allow $K = 1/\Delta x$, the range of updated values correspond precisely with the updated values in the discrete sequence learning task.

[2]Note: progress may be symmetric such that $q^r_{N+\Delta x}$ also change by a similar amount. Such an approach implies some form of a smoothing strategy for the update rule. However, for simplicity and consistency with the discrete case, we assume asymmetric updates where only Q-values less than the current task level $N$ have an opportunity to be updated.

For sufficiently high $\varepsilon$, the continuous $K$-step student corresponds precisely in learning rate with the discrete student. However, one important difference is the propagation of mistakes. In the discrete case, if the student halts, the error is not backpropogated. Rather, the (unwritten) Q-value associated with choosing the halt action is "updated," and remains 0. However, in the $K$-step scenario, because all previous $K$ Q-values are updated, if the student halts, the last $K - 1$ Q-values are depressed slightly with a zero updated, initiating a secondary extinction wave distinct from the ones generated through curriculum changes. Hence, learning is slower and more difficult for continuous students, and the potential for extinction waves to erase all progress correspondingly higher. The situation is exacerbated by low $\varepsilon$, in which the probability of halting is higher.[3]

## 5.2 Teacher algorithms

We explore generalizations of both the Incremental and Adaptive teacher algorithms developed in Chapter 5.[4] Incremental generalizes directly to this setting. We allow the curriculum to increment in fixed intervals of $1/\Delta x$, monotonically increasing the task difficulty as the student progressively attains mastery. The procedure otherwise remains identical to the one described in Algorithm 2. Figure 5.1 shows trajectories for the Incremental teacher adapted for the continuous setting (compare to Figure 3.3).

[3] An easy way to fix this situation is to simply alter the learning rules such that halting actions do not propagate depressed Q-values. However, it remains unclear whether this effect is a feature or bug — perhaps it is more realistic for repeated failures to depress the performance of the student, rather than leave performance unimpacted. Perhaps the correct action is to modify the discrete student such that errors also propagate. Ultimately, we leave this distinction as an additional layer of complexity for the continuous student, and an additional obstacle an optimal teacher must overcome.

[4] Unfortunately, POMCP does not scale to massive size of the POMDP in the continuous case. We rely on the behavior of the Adaptive teacher to give a flavor for what optimality looks like in this setting. Note, none of the algorithms from Matiisen et al. [15] generalize at all to continuous curricula, so we do not investigate

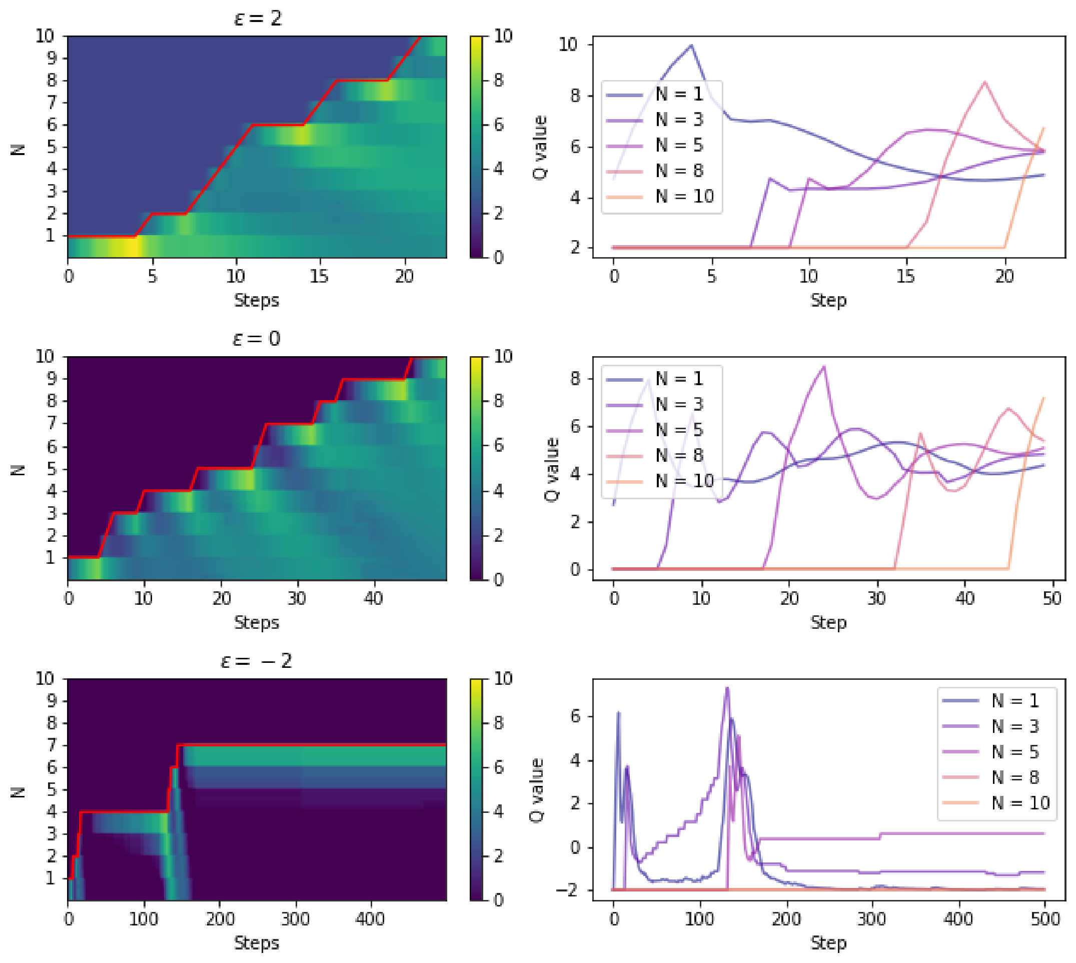

**Figure 5.1: Trajectory of a student through a continuous curriculum across different levels of $\varepsilon$ under an Incremental teacher**. The student's trajectory is plotted in red, on the left. The Q-values ($q^r + \varepsilon$) are shaded in the background. On the right are trajectories of individual Q-values corresponding to different $N$, through time. The plot is strikingly similar to the ones shown in Figure 3.3, right down to the "blockiness" of the Q-values. The $K$-step TD rule updates Q-values based on signals many steps ahead, creating groups of Q-values that update together in a similar way to the individual Q-values of the discrete setting. Again, we see that extinction waves tend to propagate freely under an Incremental teacher, wiping out progress and preventing low-$\varepsilon$ students from learning.

### 5.2.1 Continuous Adaptive teacher

The Adaptive teacher is similar in spirit to its discrete counterpart but extended to accomodate the particulars of a continuous curriculum. At the start of an interaction, the experimenter proposes an initial "rough guess" at an appropriate increment interval for the teacher to use.[5] As the Adaptive teacher progresses, it tweaks the size of the increment according the student's performance. Hence, rather than just three actions (increment, decrement, stay), we introduce a second set of three actions: increase the increment interval, decrease the increment interval, retain the same interval. Adjustments to the interval are made multiplicatively by a pre-determined percentile.

With this adjustment, at every iteration, the Adaptive teacher must now select from one of nine possible actions: an `increment`, `decrement`, `stay`, paired with a `grow interval`, `shrink interval`, and `keep interval`. To select the correct action, one can use the same optimization procedure described in Section 3.5 to learn the best actions for each situation. Figure 5.2 shows trajectories for the Adaptive teacher adapted for the continuous setting (compare to Figure 3.6).

### 5.2.2 Benchmarks

See Figure 5.3 for a comparison of Incremental and Adaptive teachers on the continuous sequence learning task. The magnitude of steps is close to that of the discrete case, pictured in Figure 3.7, though somewhat higher for larger $N$. Adaptive has a decisive edge over Incremental, particularly for high $N$ and low $\varepsilon$. Random and final-task-only curricula are not

them further in this setting.

[5] Such an increment can be considered a prior that the experimenter assumes about the difficulty of a task. For the continuous sequence task, we use an initial increment of $1/\Delta x$

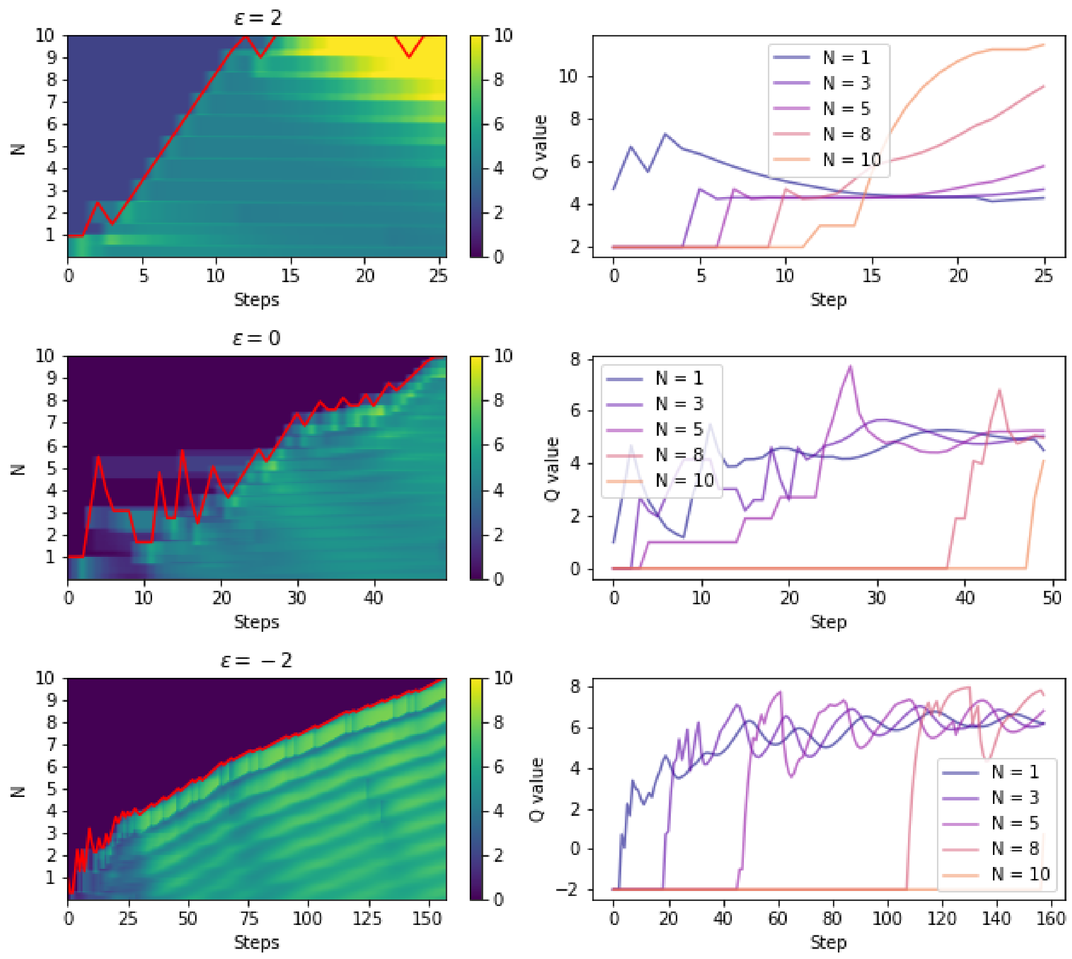

**Figure 5.2: Trajectory of a student through a continuous curriculum across different levels of $\varepsilon$ under an Adaptive teacher**. The student's trajectory is plotted in red, on the left. The Q-values ($q^r + \varepsilon$) are shaded in the background. On the right are trajectories of individual Q-values corresponding to different $N$, through time. The plot is similar to the ones shown in Figure 3.6. The Adaptive teacher is able to backtrack, as well as adjust its increment intervals, allowing it to continue training low-$\varepsilon$ students. The $K$-step learning rule tends to produce smooth ripples across the Q-values as Adaptive adjusts the task difficulty continually.

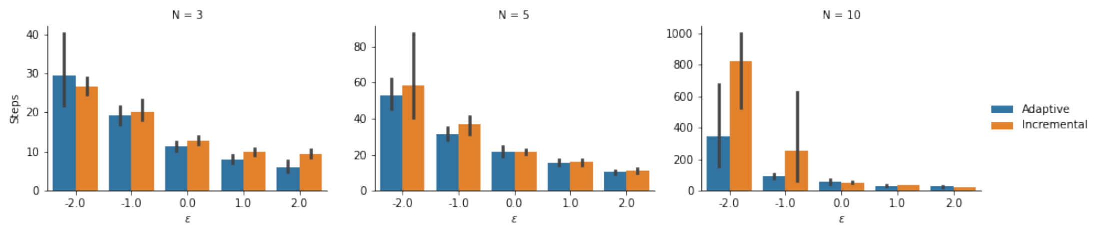

**Figure 5.3: Benchmarks on the continuous sequence-learning task.** Adaptive tends to outperform Incremental, particularly for high $N$ and low $\varepsilon$. Algorithms from Matiisen et al. [15] fail to learn in this setting, as do Random and final-task-only curricula.

plotted, as learning fails to occur even for the easiest tasks pictured.

Overall, intuitions from the discrete case translate naturally to the continuous case, though learning overall tends to be more difficult in the latter. We presented here only an initial exploration of one particular continuous generalization. Future work will need to examine more deeply nuances particular to the continuous case, what optimal teachers may look like in this setting, and validate these results on a naturalistic task like trail tracking.

*One virtue in crackpot ideas is that they breed rapidly and their progeny show extraordinary mutations. Everyone is talking about teaching machines nowadays, but Sidney Pressey can tell you what it was like to have a crackpot idea in that field 40 years ago.*

BF Skinner

# 6 Conclusion

From Skinner's missile guidance pigeons to today's latest-and-greatest RL agents, curriculum learning plays a foundational role in training agents for complex tasks. In this thesis, we have explored a teacher-student framework for developing optimal strategies for curriculum design.

We first studied the student's learning dynamics as it attempts new tasks. Dual waves of reinforcement and extinction modulate the student's performance, necessitating a careful strategy that cultivates reinforcement while preventing extinction from propagating. A naïve Incremental teacher fails to stop extinction waves, which tend to erase all progress particularly for students with low $\varepsilon$. The optimal teacher strategy, discovered through both a black-box optimization procedure (POMCP) as well as a tree-based Adaptive teacher, appears to rely on frequent oscillations between the current and previous task difficulty levels, which ameliorates extinction.

We then applied these teacher algorithms to a complex, naturalistic tracking environment. Trail tracking is a behavioral paradigm in which an animal follows a ground-based terrestrial trail or an airborne odor plume. The task is nontrivial, and highly difficult for an RL agent to learn from scratch, making it an ideal candidate for curriculum learning. We apply our teacher algorithms to these tracking settings, and find that they duplicate the same behaviors observed in the simple sequence learning task. The RL agents also appear to exhibit competing reinforcement and extinction effects, though the underlying mechanisms may be very different.

The previous work has relied on the assumption that the task can be naturally decomposed into discrete difficulty levels. For many real-world tasks (like trail tracking), the task difficulty is continuous. We therefore explore one particular continuous generalization of our teacher algorithms that relies on finite approximations to continuous intervals, coupled with a $K$-step TD learning rule. The resulting teacher algorithms reproduce many of the behaviors observed in the discrete setting, though the continuous setting itself is in many ways considerably harder.

## 6.1 Future directions

Looking ahead, there are several fruitful directions ripe for additional exploration.

**Continuous curricula.** While we have begun to explore a continuous generalization of the sequence learning task, there are many unanswered questions remaining in this setting. For instance, rather than using a $K$-step TD learning rule, perhaps some kernel-based smoothing method is more appropriate for propagating correlations between Q-values. Perhaps an alternative continuous generalization scheme is more appropriate or more natural than the finite approximation we essay. Vanilla POMCP will not scale to the continuous setting, but alternative search methods might. What would an optimal teacher look like in this setting, and does our Adaptive teacher approach it?

**Multifactor curricula.** The curricula we explore in this thesis have all involved only a single axis of difficulty. For many real-world task, there are multiple axes that must all be optimized simultaneously. For example, in addition to a trail's length, other factors that influence its difficulty include its curvature, smoothness, and the quantity and size of breaks. In designing a curriculum for this setting, we have simplified all these factors into a single difficulty scale, when ideally, the teacher should choose how to modulate the difficulty along each factor. One avenue for future work is to explore how to generalize our teacher algorithms for a setting where the curriculum varies across multiple independent (or correlated) factors.

**Validation in animal training.** Curriculum learning has just as much (or perhaps even more) relevance for animal training as it does for training RL agents. Concepts like task difficulty levels, innate bias $\varepsilon$, and behavioral extinction have natural analogs in biological agents. The teacher algorithms developed in this setting can be readily deployed on real animals, and their efficacy measured. Developing better teacher algorithms for animal training may result in significant savings in time and cost to produce well-trained subjects, and might even inform curriculum design for human students. Who knows, perhaps in some distant future where pigeons are once more the favored pilots of missile guidance systems, an Adaptive teacher can mean a decisive strategic advantage for more rapidly training these “unwitting heroes.”

# References

[1] Ackley, D. & Littman, M. (1991). Interactions between learning and evolution. *Artificial life II*, 10, 487–509.

[2] Barros, R. C., Basgalupp, M. P., De Carvalho, A. C., & Freitas, A. A. (2011). A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(3), 291–312.

[3] Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning* (pp. 41–48).

[4] Bergstra, J. & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2).

[5] Christian, B. (2020). *The Alignment Problem: Machine Learning and Human Values*. New York, NY: W. W. Norton I& Company.

[6] Cornacchia, E. & Mossel, E. (2023). A mathematical model for curriculum learning. *arXiv preprint arXiv:2301.13833*.

[7] Dekker, R. B., Otto, F., & Summerfield, C. (2022). Curriculum learning for human compositional generalization. *Proceedings of the National Academy of Sciences*, 119(41), e2205582119.

[8] Draft, R. W., McGill, M. R., Kapoor, V., & Murthy, V. N. (2018). Carpenter ants use diverse antennae sampling strategies to track odor trails. *Journal of Experimental Biology*, 221(22).

[9] Fester, C. B. & Skinner, B. F. (1957). *Schedules of Reinforcement*. New York, NY: Appleton-Century-Crofts.

[10] Graves, A., Bellemare, M. G., Menick, J., Munos, R., & Kavukcuoglu, K. (2017). Automated curriculum learning for neural networks. In *International Conference on Machine Learning* (pp. 1311–1320).: PMLR.

[11] Hansen, N. (2016). The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*.

[12] Hepper, P. G. & Wells, D. L. (2005). How many footsteps do dogs need to determine the direction of an odour trail? *Chemical Senses*, 30(4), 291–298.

[13] Hu, Y., Wang, W., Jia, H., Wang, Y., Chen, Y., Hao, J., Wu, F., & Fan, C. (2020). Learning to utilize shaping rewards: A new approach of reward shaping. *Advances in Neural Information Processing Systems*, 33, 15931–15941.

[14] Kennedy, J. (1983). Zigzagging and casting as a programmed response to wind-borne odour: a review. *Physiological Entomology*, 8(2), 109–120.

[15] Matiisen, T., Oliver, A., Cohen, T., & Schulman, J. (2019). Teacher-student curriculum learning. *IEEE transactions on neural networks and learning systems*, 31(9), 3732–3740.

[16] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[17] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540), 529–533.

[18] Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., & Stone, P. (2020). Curriculum learning for reinforcement learning domains: A framework and survey. *The Journal of Machine Learning Research*, 21(1), 7382–7431.

[19] Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99 (pp. 278–287).

[20] Portelas, R., Colas, C., Weng, L., Hofmann, K., & Oudeyer, P.-Y. (2020). Automatic curriculum learning for deep rl: A short survey. *arXiv preprint arXiv:2003.04664*.

[21] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268), 1–8.

[22] Reddy, G. (2022). A reinforcement-based mechanism for discontinuous learning. *Proceedings of the National Academy of Sciences*, 119(49), e2215352119.

[23] Reddy, G., Shraiman, B. I., & Vergassola, M. (2022). Sector search strategies for odor trail tracking. *Proceedings of the National Academy of Sciences*, 119(1), e2107431118.

[24] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[25] Selfridge, O. G., Sutton, R. S., & Barto, A. G. (1985). Training and tracking in robotics. In *IJCAI* (pp. 670–672).

[26] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587), 484–489.

[27] Silver, D. & Veness, J. (2010). Monte-carlo planning in large pomdps. *Advances in Neural Information Processing Systems*, 23.

[28] Skinner, B. F. (1951). How to teach animals. *Scientific American*, 185(6), 26–29.

[29] Skinner, B. F. (1958). Reinforcement today. *American Psychologist*, 13(3), 94.

[30] Skinner, B. F. (1960). Pigeons in a pelican. *American Psychologist*, 15(1), 28.

[31] Storn, R. & Price, K. (1997). Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4), 341.

[32] Sutton, R. S. & Barto, A. G. (2018). *Reinforcement Learning: an Introduction*. MIT press.

[33] Van Seijen, H., Van Hasselt, H., Whiteson, S., & Wiering, M. (2009). A theoretical and empirical analysis of expected sarsa. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning* (pp. 177–184).: IEEE.

[34] Vergassola, M., Villermaux, E., & Shraiman, B. I. (2007). 'infotaxis' as a strategy for searching without gradients. *Nature*, 445(7126), 406–409.

[35] Wallace, D. G., Gorny, B., & Whishaw, I. Q. (2002). Rats can track odors, other rats, and themselves: implications for the study of spatial behavior. *Behavioural brain research*, 131(1-2), 185–192.

[36] Wiewiora, E. (2003). Potential-based shaping and q-value initialization are equivalent. *Journal of Artificial Intelligence Research*, 19, 205–208.

[37] Åström, Karl Johan (1965). Optimal Control of Markov Processes with Incomplete State Information I. 10, 174–205.

# A

# Estimation methods

A central aspect of our curriculum learning theory is that the teacher does not have access to the student's internal parameters. Rather, the teacher must estimate some intrinsic of quality of the student through extrinsic observables alone. In the context of our particular curriculum learning algorithms, the critical internal (or hidden) parameter of the student we use is its true success rate $s_n$ given a task of difficulty $n$. For example, in the case of the

Incremental teacher, if $s_n$ exceeds some threshold $\tau$, the student advances. Otherwise, the student remains on the current task level. In the case of Adaptive, $s_n$ and $\Delta s_n$ are used as inputs to a decision tree that decides the student's next task.

In all these cases, the teacher as access only to the student's transcript of successes and failures: $\mathbf{h} = (h_1, h_2, \dots, h_t)$, where $h_i = 1$ if the student succeeded on round $i$, and 0 otherwise. Given this transcript, the teacher must construct an estimate $\hat{s}_n(\mathbf{h}) \approx s_n$. In the main text, we consider only a simple exponential moving average (EMA) approach to computing $\hat{s}_n$. In this appendix, we describe the EMA in further detail, and explore two alternative estimation procedures motivated by a Bayesian approach.

## A.1 Overview

Under the student model described in Section 3.1.1, the true success rate of the student on a task of length $n$ is given by

$$s_n = \prod_{i=1}^{n} \sigma(q_i) = \prod_{i=1}^{n} \sigma(q_i^r + \varepsilon) \tag{A.1}$$

However, because the underlying parameters $q_i^r$ and $\varepsilon$ are unknown to the teacher, the quantity $s_n$ must be estimated from the student's transcript of successes and failures. Below, we compare three estimation procedures for $s_n$:

1. **Exponential moving average**: we apply an EMA to the student's transcript to estimate $s_n$. This estimate functions as a baseline with which to compare our other two, Bayesian motivated approaches

2. **Beta posterior inference**: we assume the successes and failures in a student's tran-

script are approximately i.i.d from a Bernoulli distribution. Because the student's success rate evolves as it learns, $s_n$ is a nonstationary quantity, and so this assumption cannot hold over long periods of time. However, to make posterior inference tractable, we assume that negligible training occurs over short time periods, and that $s_n$ is locally stationary.

3. **Particle filtering**: we relax the local stationarity assumption from Method 2 and develop a particle filtering algorithm to estimate $s_n$. Particles are sampled from priors over $q_i^r$ and $\varepsilon$, then forward-simulated to produce posteriors over $s_n$.

In the follow sections, we develop each method in detail, and compare their ability to estimate $s_n$. To remove the confounding influence from complex teacher strategies, we use only the Incremental teacher as a framework for comparing these estimation procedures, and later compare Incremental (with sophisticated estimation techniques) to the gold-standard POMCP teacher.

## A.2 Exponential moving average

Recall that at time $t$, the teacher observes a transcript of the student's performance $\mathbf{h_t} = (h_t, h_{t_1}, \dots h_1)$, where $h_i = 1$ if the student succeeded on trial $i$, and 0 otherwise. The EMA algorithm constructs an estimate $\hat{s}_n$ through the following recursive update rule as new observations $h_t$ are made:

$$\hat{s}_n \leftarrow \begin{cases} 0 & t = 0 \\ \gamma h_t + (1-\gamma)\hat{s}_n & t > 0 \end{cases} \tag{A.2}$$

which corresponds to the unrolled equation

$$\hat{s}_n(\mathbf{h}_t) = \gamma h_t + \sum_{k=1}^{t-1} (1-\gamma)^k \gamma h_{t-1}$$

The parameter $\gamma \in [0,1]$ is chosen by the experimenter prior to computing the EMA, and controls the degree to which past observations are discounted.

The exponential moving average overall is a simple, non-parametric approach to estimating a nonstationary quantity. See Figure A.1 for a plot of EMA estimates compared to the true underlying success rate on a task with length $N = 10$, using the teacher model described above. We compare EMA across different discounts $\gamma$ and across students with different bias parameters $\varepsilon$. Lower values of $\gamma$ resulted in smoother estimates, though with a greater lag behind the true value. Higher values of $\gamma$ resulted in noisier estimates, though with less lag. For $\gamma > 0.5$, the estimates became meaningless as they tended to alternate between extremes. Overall, a discount of $\gamma = 0.7$ seems to be the most appropriate, and tracks the true success rate with reasonable consistency.

## A.3 Beta posterior inference

We next consider a simple Bayesian approach to this estimation problem. One challenge of estimating $s_n$ is that this quantity is nonstationary. As the student learns the task, $s_n$ changes over time. This is evident from Figure A.1 above, where the value of $s_n$ (plotted in red) changes considerably over the course of an episode as the student learns and experiences new challenges on a task.

However, to make the analysis simpler, we can make the (big) assumption that over

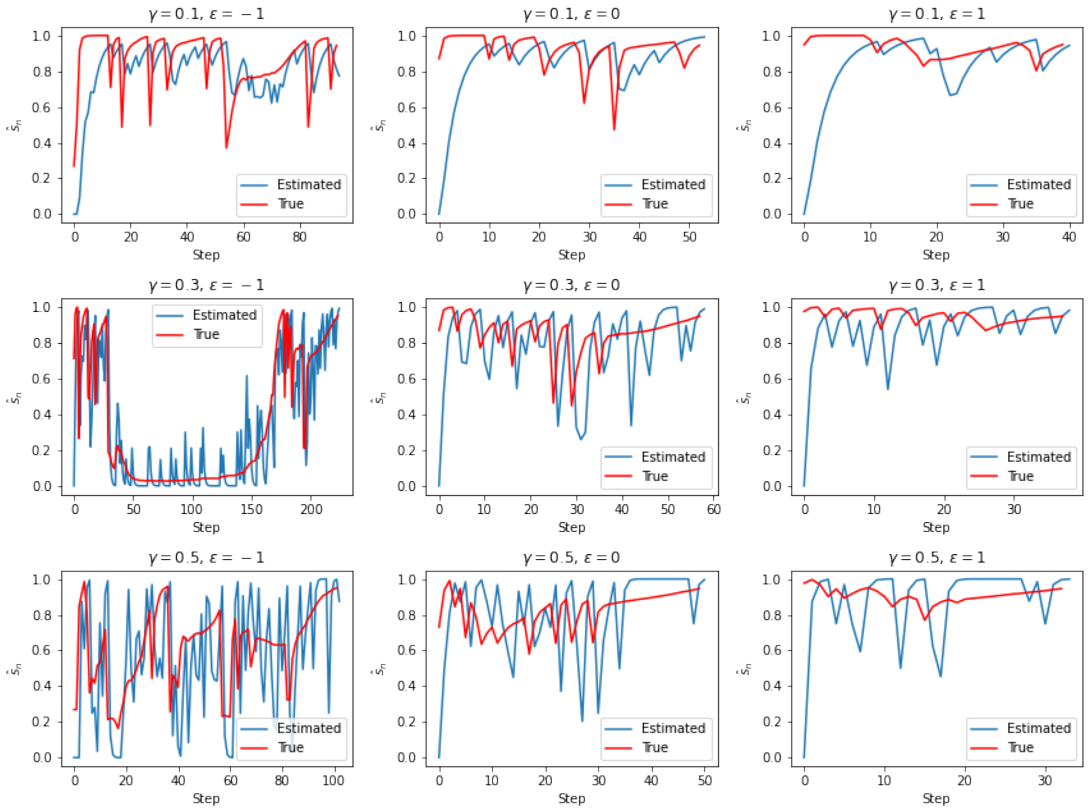

**Figure A.1: Exponential moving average estimate $\hat{s}_n$, across different discounts $\gamma$ and student bias $\varepsilon$.** Plotted in red are the student's true probability of success. In blue is the estimated success rate. The discount factor $\gamma$ varies across rows, and the student's innate bias parameter $\varepsilon$ varies across columns. We see that a discount of $\gamma = 0.3$ seems to work the best overall.

short intervals, $s_n$ is essentially stationary. In this case, suppose $s_n$ is approximately stationary over the time interval $[t - k, t]$, for some small integer $k$. Then for a transcript $\mathbf{h} = (h_1, h_2, \dots h_t)$, we have

$$h_t, h_{t-1}, \dots h_{t-k} \overset{\text{iid}}{\sim} \text{Bernoulli}(s_n)$$

Hence, if we impose the prior $s_n \sim \text{Beta}(\alpha, \beta)$, the posterior on $s_n$ becomes

$$s_n | \mathbf{h}_{(t-k):t} \sim \text{Beta}(\alpha + n_1(\mathbf{h}_{(t-k):t}), \beta + n_0(\mathbf{h}_{(t-k):t}))$$

where $n_1$ and $n_0$ count the number of ones and zeros respectively. This form suggests an estimator based on the posterior mean

$$\hat{s}_n = \mathbb{E}[s_n | \mathbf{h}_{(t-k):t}] = \frac{\alpha + n_1}{\alpha + \beta + k}$$

In this setting, we assume a uniform prior $s_n \sim \text{Beta}(1, 1)$. Selecting the value $k$ is somewhat trickier. We would ideally like to select a $k$ that is as small as possible, so as to ensure $s_n$ does not change too much over the interval $[t - k, t]$. At the same time, if $k$ is too small, our posterior has a higher variance, and the confidence in our estimate is correspondingly lower.

To address the latter issue, let us first consider the lowest value of $k$ we can tolerate. Suppose we use a threshold $\tau$ such that if $s_n > \tau$, then the student advances to the next level. Suppose we would like to be confident at the level $c$ such that $s_n$ exceeds $\tau$ before allowing the student to advance. Then we stipulate that $p(s_n | \mathbf{h}_{(t-k):t} > \tau) > c$. In the best case scenario, all observations in $\mathbf{h}$ are ones. The minimum $k$ we can tolerate is therefore the smallest $k$ such that $p(s_n|\mathbf{h}_{(t-k):t} > \tau) > c$ is true when $\mathbf{h} = 1, 1, \ldots 1$. To compute this $k$, we observe that

$$p(s_n|\mathbf{h}_{(t-k):t} < \tau) < 1 - c$$

and

$$p(s_n|\mathbf{h}_{(t-k):t} < \tau) = I_\tau(k+1, 1)$$

where $I_\tau(\alpha, \beta)$ is the incomplete Beta function with upper limit $\tau$. From here, we see that $I_\tau(k+1, 1) = \tau^{k+1} < 1 - c$. Solving for $k$ yields the final result

$$k > \frac{\log(1-c)}{\log \tau} - 1$$

so we can establish a lower bound on $k$ as $k_{\text{low}} = \lfloor \frac{\log(1-c)}{\log \tau} - 1 \rfloor$. In practice, we use a confidence $c = 0.5$, which corresponds to when the median of the posterior exceeds the threshold $\tau$, and yields reasonable results. Note, because $s_n$ changes for different difficulty levels $n$, $k$ should not be so large that it dips into episodes where $n$ was smaller. Hence, after the student advances, we must wait at least $k_{\text{low}}$ rounds before evaluating the student. For the EMA algorithm, this was a non-issue because EMA would adapt its estimate of the success rate to the current statistics. In contrast the Beta posterior procedure relies on an underlying stationary quantity, and so would be obviously invalid if it uses samples from $s_{n-1}$ in its estimation of $s_n$.

To determine what the upper bound on $k$ should be, it is unclear how to select a reasonable upper-bound analytically. However, because $s_n$ is presumably increasing over long timescales, the further back we look, the lower our estimate $\hat{s}_n$ will become. Hence, we establish a coarse upper bound $k_{\text{high}} = 3k_{\text{low}}$, and evaluate $p(s_n|\mathbf{h}_{(t-k):t} > \tau) > c$ for every

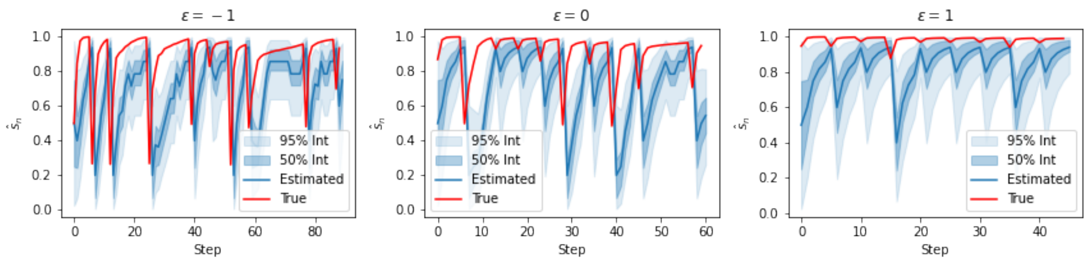

**Figure A.2: Beta posterior estimate $\hat{s}_n$, across different student bias parameters $\varepsilon$.** Plotted in red are the student's true probability of success. In blue is the estimated success rate, along with 50 and 95% intervals shaded in. Estimates are computed using $k_{\text{low}}$ unless the student advances, in which case the selected value of $k$ is used. The Beta estimate systematically underestimates the value of $s_n$.

$k$ in between $k_{\text{low}}$ and $k_{\text{high}}$. If at any point this expression evaluates to true, we advance the student to the next level. Because $\hat{s}_n$ decreases for larger values of $k$, there should ideally be a single $k$ between $k_{\text{low}}$ and $k_{\text{high}}$ that balances a large enough sample that produces a high-confidence estimate, and a small enough sample that does not yield an underestimate on $s_n$. As before, $k_{\text{high}}$ should not be so large as to contaminate the data with samples from $s_{n-1}$. The prefactor 3 was chosen as a hyperparameter setting that seems to work well in practice, though other factors may be used.

See Figure A.2 for a plot of the Beta posterior estimate, along with the true estimates of the student's performance. In general, the Beta posterior estimate tracks the trajectory of the student's true success rate, but underestimates it systematically. This may be in large part because our assumption of local stationarity on the scale of $k$ is invalid. Significant learning likely occurs even within just $k_{\text{low}}$ steps, motivating the need to develop an approach that does not rely on this tenuous assumption.

## A.4 Particle filtering

Our final estimation approach does not assume any stationarity in $s_n$; rather, it estimates $s_n$ by simulating the underlying learning dynamics using a particle filtering algorithm. Specifically, we estimate posteriors on the student's Q-values $q_i^r$ and innate bias parameter $\varepsilon$, then construct an estimate of $s_n$ based on Equation A.1.

The particle filtering algorithm proceeds as follows:

1. **Sample initial particles from the prior**. We apply a uniform prior on $\varepsilon$ over the interval $[-5, 5]$, which encompasses the range of reasonable values $\varepsilon$ can take. We apply a point-mass prior centered on 0 for all $q_i^r$, because each $q_i^r$ is initialized to 0 for every student. An initial set of particles $(\mathbf{q}_j^{r,(0)}, \varepsilon_j^{(0)})_{j=1}^M$ are sampled from these priors. In the simulations, we find that $M = 1000$ is sufficient.

2. **Simulate forward dynamics**. For each particle, we simulate $T$ trials of the student using the learning rules described in Equation 3.2, obtaining a set of updated parameters $(\mathbf{q}_j^{r,(1)}, \varepsilon_j^{(1)})$ along with a set of corresponding simulated transcripts $(\mathbf{h}_j^{(1)})$

3. **Filter particles consistent with observation.** After receiving an observed transcript $\mathbf{h}$ from the student, we keep all particles $j$ such that $\mathbf{h}_j^{(1)} = \mathbf{h}$. Note, contrary to previous descriptions where we assume $\mathbf{h}$ corresponds to the entire history of the student's performance, here we assume $\mathbf{h}$ corresponds only to the student's trials since the last interaction, and $|\mathbf{h}| = T$. If $T$ is small, checking direct equality works well.[1] However if $T$ is large, the probability that an observation will match a simu-

[1] In our case, $T = 3$, which is sufficiently small to check for equality directly.

lated transcript diminishes accordingly (even if the underlying parameters match), in which case we might compare a summary statistic on $\mathbf{h}$ like the mean.

4. **Resample remaining particles.** For our remaining particles, to remove any outliers, we perform a resampling weighted by each particle's likelihood. For a particle $(\mathbf{q}_j^{r,(1)}, \varepsilon_j^{(1)})$ and observation $\mathbf{h}$, if the student's learning rate $\alpha$ is sufficiently small, the likelihood of the particle can be given as

$$p(\mathbf{h}|\mathbf{q}_j^{r,(1)}, \varepsilon_j^{(1)}) = \hat{s}_n^{n_1(\mathbf{h})}(1 - \hat{s}_n)^{n_0(\mathbf{h})}$$

where $\hat{s}_n$ is the particle filtering estimate of the true success rate at level $n$, and follows from equation A.1

$$\hat{s}_n(\mathbf{q}, \varepsilon) = \prod_{i=1}^{n} \sigma(q_i^r + \varepsilon) \tag{A.3}$$

With these weights, all particles are resampled until we have a set of the original size $M$ particles.

5. **Particle reinvigoration.** Particularly for long runs, $\varepsilon$ will tend to drift over time as unexpected observations are encountered (the Q-value parameters tend to remain fairly close, as we will see below). To forestall any posterior collapse, it is essential to reinvigorate $\varepsilon$. We use a simple reinvigoration strategy where a small random jitter is applied to every $\varepsilon$ parameter. Specifically, we use a normally-distributed jitter centered at 0 with variance 0.25.

6. **Repeat.** Encountering one observation yields a set of particles $(\mathbf{q}_j^{r,(1)}, \varepsilon_j^{(1)})$, which represents samples from the posterior $\mathbf{q}^r, \varepsilon \mid \mathbf{h}^{(1)}$. Upon encountering a second ob-

servation, we repeat our calculations from step 2 using the current posterior sample to obtain a new set of particles $(\mathbf{q}_j^{r,(2)}, \varepsilon_j^{(2)})$, which represents samples from the posterior $\mathbf{q}^r, \varepsilon \mid \mathbf{h}^{(2)}, \mathbf{h}^{(1)}$. This process is repeated for all new observations as they arrive.

Using this particle filtering algorithm, we generate posteriors on the parameters $q_i^r$ and $\varepsilon$. At step $i$, the posterior samples $(\mathbf{q}_j^{r,(i)}, \varepsilon_j^{(i)})_{j=1}^M$ can be used to construct posterior samples on estimated success rate $\hat{s}_n$ using Equation A.3. From here, we can adopt the same approach as before, and if more than $c$ proportion of the posterior samples are greater than a predetermined threshold $\tau$, we advance the student to the next level. As for the Beta posterior inference approach, we use $c = 0.5$.

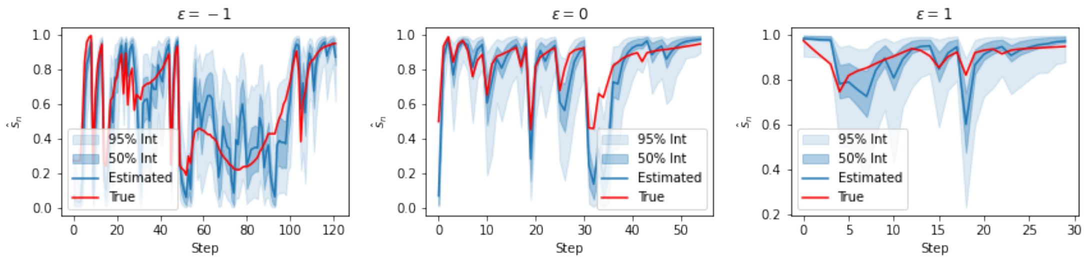

**Figure A.3: Particle filtering estimate $\hat{s}_n$, across different student bias parameters $\varepsilon$.** Plotted in red are the student's true probability of success. In blue is the estimated success rate, along with 50 and 95% intervals shaded in. The estimates are generally quite close to the true success rate, and certainly tend to capture it within the 50 percent interval.

See Figure A.3 for a plot of the estimated success rate during runs using the particle filtering algorithm. In general, the estimates fall quite close to the true success rate, outperforming both of the previous methods. Figure A.4 plots 95 percent intervals on the posterior samples of the underlying parameters, compared to the true values for each parameter. The particle filtering approach tends to capture the true values well, confirming that its inference is accurate.

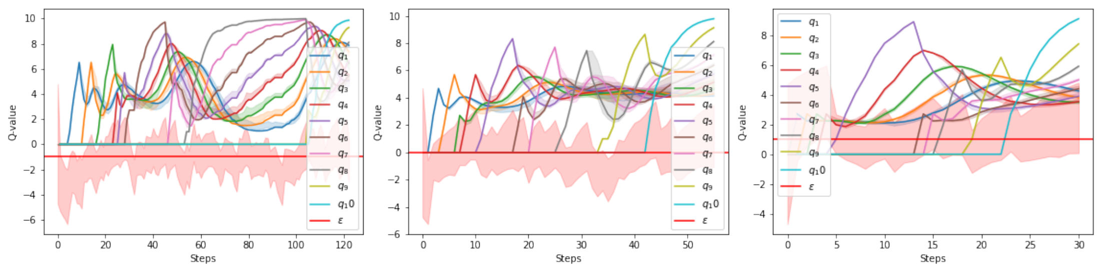

**Figure A.4: Particle filtering estimates of each $q_i^r$ and $\varepsilon$.** The solid line plotted in the color of each parameter is the true value of the parameter. The shaded regions represent the 95 percent interval of the posterior samples. In general, the intervals for the Q-values are extremely narrow around the true value. The intervals around $\varepsilon$ are wider, but tend to capture the true value nonetheless.

## A.5 Comparison of estimation procedures

Figure A.5 shows a comparison across all teacher algorithms, including the POMCP teacher. The POMCP teacher represents the approximately optimal teacher, and shows the lower bound that all other teachers approach. In general, our simple incremental teacher model approaches the optimal, though of course does not quite attain it, particularly for lower $\varepsilon$ and higher difficulty levels $N$. Somewhat surprisingly, it also appears that the accuracy of an estimation method does not matter critically, as all three estimation methods produce students with more-or-less the same efficiency, though the particle-filtering method retains a slight edge.

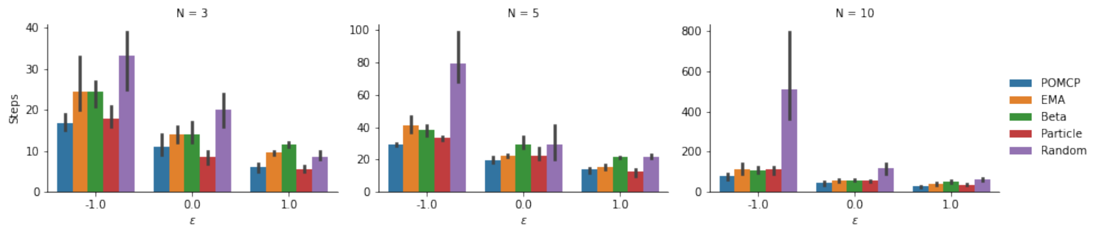

**Figure A.5: Comparison across POMCP and Incremental teachers using different estimation procedures.** The number of iterations it takes each teacher to train a student to full proficiency is compared across max task-lengths $N$ and innate bias parameters $\varepsilon$. The POMCP teacher represents approximately the best possible performance a teacher can achieve. Despite differences in their estimation accuracy, the CMA, Beta, and Particle teachers all perform comparably in terms of training efficiency, though Particle remains the best of this group. A random curriculum is also plotted as a comparison of the effect of an unstructured curriculum.

# B

# Implementation details

All code for this project is available on GitHub:

`https://github.com/wtong98/automated-curriculum-learning`

Below, we enumerate additional details regarding hyperparameter settings and implementation specifics about each experiment.

## B.1 Sequence learning

In the sequence learning setting, the student is an expected SARSA agent with the following parameters:

| Parameter | Value |
|---|---|
| Reward ($R$) | 10 |
| Learning rate ($\alpha$) | 0.1 |
| Discount ($\gamma$) | 1 |
| Episodes per interaction ($T$) | 3 |

The teacher uses an exponential moving average to measure the success rate of its student. (Alternative approaches are discussed in Appendix A.) The discount factor for EMA is $\gamma = 0.8$ for all teachers.

## B.2 Incremental teacher

The Incremental teacher has a single parameter: the success rate $\tau$ beyond which the student should advance to the next level. We use $\tau = 0.95$.

## B.3 POMCP teacher

The POMCP teacher uses the following parameter settings:

| Parameter | Value |
|---|---|
| Particles ($M$) | 5000 |
| Discount ($\gamma$) | 0.9 - 0.95 |
| MCTS stop threshold ($\varepsilon$) | 0.01 |
| MCTS explore factor ($e$) | 1 |
| Reinvigoration probability ($p$) | 1 |
| Reinvigoration scale ($\sigma$) | 0.5 |

Particle reinvigoration applies only to the estimation of the student's innate bias parameter $\varepsilon$. For a particle estimate $p_\varepsilon$, reinvigoration proceeds as $p_\varepsilon \leftarrow p_\varepsilon + \eta$, where $\eta \sim \mathcal{N}(0, \sigma)$. Estimation of the student's learned Q-values tend to be highly accurate, and hence do not require reinvigoration.

## B.4 Adaptive teacher

Figure 3.5 shows the decision tree used by the Adaptive teacher to output tasks for the student to learn. The decision tree learned by the Adaptive teacher on the continuous case is identical, except the split on $s$ occurs at 0.7. Further, if $s > 0.7$ and $\Delta s < 0$, the teacher decrements and shrinks the increment size to 60 percent of the current. If $s > 0.7$ and $\Delta s > 0$, the teacher increments and grows the increment size by 50 percent.

Optimizing the Adaptive teacher proceeds as a coordinated ascent. The procedure begins with an initial, reasonable set of actions selected by the experimenter. Differential evolution[31] is used to evolve the precise splits in the tree, followed by a comprehensive search through the entire space of possible actions. These two steps, evolution followed by action search, alternate until converging on a final tree.

## B.5 Matiisen teachers

We use a grid search to identify optimal hyperparameters for the teacher algorithms described in Matiisen et al.[15]. The final parameters we used are:

- **Online**: $\alpha = 0.05, \beta = 0.34$
- **Naive**: $\alpha = 0.16, \beta = 3.8$
- **Window**: $\alpha = 0.26, \beta = 4.83, k = 10$
- **Sampling**: $\alpha = 0.1, k = 3$

## B.6 Trail tracking

Trails are generated using generalized worm-like chain ensembles, using the procedure described in Reddy et al.[23]. The parameters used to sample each trail are

| Parameter | Value |
|---|---|
| Width | 5 |
| Diffusion rate | 0.02 |
| Curvature radius | 70 |
| Heading | $0$ - $2\pi$ |
| Break point | 0.5 - 0.6 |

Note, the `break point` parameter is expressed as a proportion of the trail's total length. The schedule used to produce Figure 4.2 varies based on the trail's length. The specific lengths used are:

$$\begin{pmatrix} 10 & 30 & 50 & 70 & 90 \end{pmatrix}$$

The agent is a PPO[24] deep reinforcement learning model with the following hyperparameters

| Parameter | Value |
|---|---|
| Steps per rollout | $1024 \times 8$ |
| Batch size | 256 |
| Epochs | 5 |
| Learning rate | 1e-4 |
| Feature model | Nature CNN[17] |
| Action model | MLP, 2 layers, 128 units per layer |
| Value model | MLP, 2 layers, 128 units per layer |
| Entropy coefficient | 0.1 |
| Discount | 0.98 |
| GAE weight | 0.9 |
| Gradient clip range | 0.2 |
| Max gradient norm | 1 |
| Value function clip range | 0.36 |

The agent receives as input a pixel observation of the world with a square view distance of up to 40 units in the horizontal or vertical directions, scaled up by a factor of 2 to produce the direct pixel observations. At each step, the agent advances three units in the forward direction, or 45 degrees to the left or right. The agent's heading rotates left/right by the same angle. The agent is allowed up to 200 steps before the episode terminates.

## B.7 Plume tracking

Plumes are generated using the plume model in Vergassola et al.[34]. The parameters used to generate each plume are

| Parameter | Value |
|---|---|
| Length scale | 20 |
| Diffusivity | 1 |
| Emission rate | 1 |
| Particle lifetime | 150 |
| Wind speed | 5 |
| Sensor size | 1 |

The schedule used to produce Figure 4.4 varies based on the starting odor detection rate. The specific rates used are computed as $1/r_k$, where $r_k = 0.5 + 0.1k$ and $k$ increases from 0 to 23, for a total of 24 difficulty levels.

As before, the agent is a PPO[24] deep reinforcement learning model, with the following hyperparameters

| Parameter | Value |
|---|---|
| Steps per rollout | $1024 \times 8$ |
| Batch size | 256 |
| Epochs | 5 |
| Learning rate | 1e-4 |
| Feature model | Nature CNN[17] |
| Action model | MLP, 2 layers, 128 units per layer |
| Value model | MLP, 2 layers, 128 units per layer |
| Entropy coefficient | 0.25 |
| Discount | 0.98 |
| GAE weight | 0.9 |
| Gradient clip range | 0.2 |
| Max gradient norm | 1 |
| Value function clip range | 0.36 |

The agent's mechanics are identical to those of the trail case. The max steps the agent can take in the environment scales as 3 times the starting distance to the odor source.