Learning From Almost No Data

by

Ilia Sucholutsky

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Statistics

Waterloo, Ontario, Canada, 2021

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Pascal Tyrrell Professor, Medical Imaging, University of Toronto
Supervisor:	Matthias Schonlau Professor, Statistics and Actuarial Science, University of Waterloo
Internal Member:	Mu Zhu Professor, Statistics and Actuarial Science, University of Waterloo
Internal Member:	Samuel Wong Professor, Statistics and Actuarial Science, University of Waterloo
Internal-External Member:	Jimmy Lin Professor, Computer Science, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The chapters of this thesis describe and contain Ilia Sucholutsky's work, under the supervision of Dr. Matthias Schonlau, that has been published or pre-printed in the following venues. Exceptions to sole authorship of material are noted.

- Chapter 3:
  - Ilia Sucholutsky, Apurva Narayan, Matthias Schonlau, and Sebastian Fischmeister. Deep learning for system trace restoration. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2019a. doi: 10.1109/IJCNN.2019.8852116. Pre-print at arXiv:1904.05411
  - Ilia Sucholutsky, Apurva Narayan, Matthias Schonlau, and Sebastian Fischmeister. Pay attention and you won't lose it: a deep learning approach to sequence imputation. *PeerJ Computer Science*, 5:e210, August 2019b
  - **Note**: Section 3.5 was co-authored by Ilia Sucholutsky and Dr. Apurva Narayan. Dr. Apurva Narayan conducted the experiments involving TREM algorithm.
- Chapter 4:
  - Ilia Sucholutsky and Matthias Schonlau. Soft-label dataset distillation and text dataset distillation. *2021 International Joint Conference on Neural Networks (IJCNN)*, 2019. Forthcoming. Pre-print available at arXiv:1910.02551
- Chapter 5:
  - Ilia Sucholutsky and Matthias Schonlau. Optimal 1-NN prototypes for pathological geometries. *PeerJ Computer Science*, 7, 2021b. doi: 10.7717/peerj-cs.464
  - Ilia Sucholutsky and Matthias Schonlau. 'Less than one'-shot learning: Learning N classes from M< N samples. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021a. Forthcoming. Pre-print available at arXiv:2009.08449
  - Ilia Sucholutsky, Nam-Hwui Kim, Ryan P Browne, and Matthias Schonlau. One line to rule them all: Generating LO-shot soft-label prototypes. *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021. Forthcoming. Pre-print available at arXiv:2102.07834
  - **Note**: 'Component 1: Finding Lines' in Section 5.4.2 was jointly authored by Ilia Sucholutsky and Nam-Hwui Kim. Nam-Hwui Kim developed the Recursive Regression and Distance-Based Attraction algorithms described in this section.

- Appendix C:
  - Ilia Sucholutsky and Matthias Schonlau. SecDD: Efficient and secure method for remotely training neural networks (Student Abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021c. Forthcoming. Pre-print available at arXiv:2009.09155

Abstract

The tremendous recent growth in the fields of artificial intelligence and machine learning has largely been tied to the availability of big data and massive amounts of compute. The increasingly popular approach of training large neural networks on large datasets has provided great returns, but it leaves behind the multitude of researchers, companies, and practitioners who do not have access to sufficient funding, compute power, or volume of data. This thesis aims to rectify this growing imbalance by probing the limits of what machine learning and deep learning methods can achieve with small data.

What knowledge does a dataset contain? At the highest level, a dataset is just a collection of samples: images, text, etc. Yet somehow, when we train models on these datasets, they are able to find patterns, make inferences, detect similarities, and otherwise generalize to samples that they have previously never seen. This suggests that datasets may contain some kind of intrinsic knowledge about the systems or distributions from which they are sampled. Moreover, it appears that this knowledge is somehow distributed and duplicated across the samples; we intuitively expect that removing an image from a large training set will have virtually no impact on the final model performance.

We develop a framework to explain efficient generalization around three principles: information sharing, information repackaging, and information injection. We use this framework to propose 'less than one'-shot learning, an extreme form of few-shot learning where a learner must recognize N classes from $M < N$ training examples. To achieve this extreme level of efficiency, we develop new framework-consistent methods and theory for lost data restoration, for dataset size reduction, and for few-shot learning with deep neural networks and other popular machine learning models.

Acknowledgements

There are so many people I need to give thanks to for helping me get here today.

To Dr. Matthias Schonlau for his tireless supervision, extensive consultation, and meticulous feedback. Thank you for always giving me your time and attention even when I would show up unannounced at your office. Your carefully thought-out advice helped me focus my constant excitement for new ideas into actual progress and contributions. Most importantly, thank you for always giving me the freedom to explore the ideas that interested me and being incredibly supportive even when my interests were far away from yours. I could not possibly have asked for a better supervisor or a better PhD experience.

To the members of my committee - Dr. Pascal Tyrrell, Dr. Jimmy Lin, Dr. Samuel Wong, Dr. Mu Zhu - for all the time and valuable feedback that they gave me. Thank you for helping me make this thesis into something I can truly be proud of.

To Dr. Sebastian Fischmeister and Dr. Apurva Narayan for making my first year of grad school high-paced, productive, and action-packed. You taught me that theory is very different from practice when it comes to actually making machine learning useful and that it is just as important to explore engineering considerations as it is to derive equations. Thank you for making my first foray into the research world so unforgettable.

To all the other teachers, tutors, lecturers, instructors, professors, and mentors in my life who went above and beyond. Patrick who was the first teacher to treat me like an adult and encourage me to have strong opinions of my own. Evans who suggested I might one day get a PhD in math long before I had any idea what career I would want to pursue (he just thought it would be at Georgia Tech rather than at Waterloo). Chris who changed my life by telling me to learn how to code when I told him I was bored in his class. David who not only made me into a mathematician with the breadth and depth of the material he covered, but also gave life advice that I still remember today. Ilya who guided my research directions over the years beginning with his simple statement that 'everything is irrelevant other than deep learning'. Alex who taught me that the secret to productivity is collaboration. Thank you to these educators and all the others who helped me learn and grow.

To my family for a lifetime of unwavering love, support, and heated dinner-time debates. My grandparents who filled my childhood with love and warmth and were my first teachers in every subject from language to math. My parents who uprooted their entire lives twice just to make sure my sister and I could grow up with every possible opportunity to be safe, successful, and happy. My sister who was always there to guide me and lend a helping, and very artistic, hand whenever I needed it, including designing the beloved rhinocorn

that graces the cover and inner pages of this thesis (see Note below). Thank you all for making me who I am today.

To Ellie for being my better half all of these years. You love and support me in more ways than I could possibly list here, but most importantly, you make me strive to be better every single day. Thank you for making all of our adventures together so incredible. I could not imagine undertaking this or any other adventure without you.

To Gennady for not only providing countless ideas but also thoroughly proofreading this thesis and in so doing becoming the first person to read it from beginning to end.

To Jon for being the best roommate, chef, and friend throughout all our years in Waterloo. To Nam for the countless hours we spent discussing math, scheming in our offices, and walking to get tea.

To all my other wonderful friends and colleagues, who I miss already, for making my time in university so much richer, more fun, and more enjoyable.

Since I spend so much of my productive time thinking about small data, let me distill these acknowledgements into one final sentence.

Thank you to everyone who helped put me on my current trajectory.

Note: The front cover, the back cover, the black and white image preceding Chapter 1, and the image in Figure 2.1 were created by Asya Sucholutsky (http://www.AsyaSucholutsky.com/) for use in this thesis and are reproduced here with the creator's permission. The background used in the design of the black and white image preceding Chapter 1 and in the design of the full-color image Figure 2.1 was created by iStock.com/Matt_Gibson and was used with permission and appropriate licensing.

Table of Contents

List of Figures

List of Tables

'It has been said that everything everywhere affects everything else.
This may be true.
Or perhaps the world is just full of patterns.'
Terry Pratchett, *Wings*

Learning From Almost No Data

ILIA SUCHOLUTSKY

Chapter 1

Introduction

Conceptually, a dataset is just a collection of samples that can be images, text, time series, or a multitude of other formats. When we train models on these datasets, we see that these models are often able to find patterns, make inferences, detect similarities, and otherwise generalize to samples that they have previously never seen. This suggests that datasets may contain some kind of intrinsic knowledge about the systems or distributions from which they are sampled. For many machine learning tasks, it is often the objective to approximate this underlying true distribution. Deep learning models are increasingly approaching and exceeding human performance on a number of such tasks [LeCun et al., 2015]; however, in order to achieve human-level or better performance on these tasks, models generally need to be trained on datasets containing a very large number of examples. If these datasets contain noise or loss, then model performance will likely be impacted and an even larger training set may be needed. These restrictions add a lot of overhead when using deep learning to automate or solve various tasks as a large, clean dataset must first be collected and annotated, and then the model must be trained on this entire dataset. The computational overhead is even higher if the model architecture or hyper-parameters need to be optimized, as many architecture search or hyper-parameter search algorithms require that the candidate models be retrained each time [Zoph and Le, 2016, Liu et al., 2018, Pham et al., 2018].

The old school of thought in the fields of statistical and machine learning was that the number of trainable parameters in a model could not exceed the number of training examples without causing overfitting. Combined with the success of applying deep learning to big data, this has led to the recent trend of training increasingly large models on increasingly large datasets. Unfortunately, the computational costs of this approach have led to a de-democratization of AI [Ahmed and Wahed, 2020]: only a shrinking number

of stakeholders (researchers, companies, etc.) are able to reap the benefits, or participate in the development, of new deep learning methods. However, these same developments in deep learning have also shown that neural networks with millions and even billions of parameters can be successfully trained without overfitting. Recent studies have even found that, in certain settings, more data can hurt model performance while larger model sizes improve it; a phenomenon called 'deep double descent' [Nakkiran et al., 2019]. These recent results along with the human ability to solve many learning tasks given only a small number of examples [Lake et al., 2015] suggest that there is only a relatively small amount of task-specific knowledge or information needed to find good solutions to a task. It also suggests that knowledge is duplicated across the observations in a dataset. If knowledge is duplicated across a dataset and the actual amount of knowledge contained within a dataset is relatively small, then it is important to determine whether there is a more efficient way to train our models than by simply showing them these redundant examples.

To this end, we think of a dataset as a set of information (about an underlying distribution) that has been packaged in a particular way. The features, labels, and every observation are all just different aspects of that packaging. Much of the work in machine learning can then be described as finding 'better' ways to re-package that information, where the definition of 'better' depends on the type of task being solved. Feature selection and engineering, dataset reduction and augmentation, and representation learning are all methods for re-packaging information. In certain cases, we can improve the re-packaging process by also injecting some external information using methods like transfer learning and pre-training. This also includes any method that requires input from an 'expert' (human or AI) with prior knowledge, such as data labeling and annotation, manual feature selection and engineering, knowledge distillation, and many more.

Throughout this thesis we approach the problem of learning from almost no data from this perspective of re-packaging information. In the next chapter, we formalize our approach to efficient learning by developing a framework that ties generalization to different forms of information sharing and redundancy within data. In the remaining chapters, we use this framework to propose and study novel methods of re-packaging information and their implications for generalizing from small data. This includes restoring lost data by adapting sequence-modeling deep neural networks, reducing training dataset sizes by multiple orders of magnitude while maintaining the same model performance, establishing a connection between data-efficiency in deep neural networks and classical models like k-Nearest Neighbor classifiers, and providing both empirical and theoretical evidence that a model can learn to separate N classes from $M < N$ training examples.

We use the rest of this chapter to provide some definitions and terminology, as well as to give a more detailed outline of the remainder of this thesis.

1.1 Definitions and Terminology

- We interchangeably use the terms (data-)point, sample, example, and observation to refer to individual records (e.g. an image, a text sequence, etc.) within a dataset.
- We use 'prototype' to refer to a representative example which may either be synthetic or come from the true data distribution.
- When discussing statistical sampling, we provide context around the word 'sample' to avoid ambiguity (e.g. '...randomly sample from the distribution...').
- We generally use the term 'model' to refer to whichever machine learning model we are discussing (e.g. neural network, kNN classifier, etc.) when there is no ambiguity.
- We interchangeably use the terms 'sample efficiency' and 'data efficiency' to refer to the amount of training data required for a model to properly generalize.
- We use 'information sharing' as a broad non-technical term for when a data object (e.g. an observation, a class of observations, a sequence, etc.) reveals information about another data object. The closest technical term would be 'mutual information', which quantifies how much information we can receive about one random variable through observations of another [Kullback, 1997, Cover, 1999, Zeng, 2015], which in turn is based on Claude Shannon's foundational concepts of rate of transmission and joint entropy [Shannon, 1948]. We intentionally avoid framing information sharing in terms of random variables, and instead suggest that it can cover both stochastic and deterministic settings. As a result, we consider concepts/measures like mutual information, directed information, joint entropy, partial redundancies, correlation, causation, duplication, symmetry, set membership, and structural similarity to be special cases of information sharing.
- We use 'label' to refer to any desired (or 'target') output, that a machine learning model must learn to produce given the associated input. This generalized definition allows us to discuss all supervised learning settings (e.g. classification, regression, autoencoding, self-supervised learning, etc.) without having to re-define terminology for each one.
- In the context of classification, a **hard label** is a vector of length N representing a point's membership to exactly one out of N classes.

$$y^{hard} = e_i = \begin{bmatrix} 0 & ... & 0 & 1 & 0 & ... & 0 \end{bmatrix}^T$$

Hard labels can only be used when each point belongs to exactly one class. If there are n classes, and some point x belongs to class i, then the hard label for this point is the i^{th} unit vector from the set of standard basis vectors.

- In the context of classification, a **soft label** is the vector-representation of a point's simultaneous membership to several classes. Soft labels can be used when each point is associated with a distribution of classes. We denote soft labels by y^{soft}.

$$y^{soft} = \begin{bmatrix} 0.2 & ... & 0.1 & 0.6 & 0 & ... & 0.1 \end{bmatrix}^T$$

1.2 Outline

In Chapter 2, we develop a framework for understanding and achieving sample-efficient generalization in machine learning in terms of three principles: information sharing, information repackaging, and information injection. We argue that various forms of information sharing, or redundancy, in datasets are what enable generalization and that the best way to leverage this during model training is through rich labels that properly describe the shared information.

We use this framework to propose 'less than one'-shot (LO-shot) learning, a new machine learning task where a model must learn to recognize N classes from $M < N$ training examples. We also discuss different ways of using this framework to improve sample efficiency, and even achieve LO-shot learning, in practice. Specifically, we suggest methods consistent with our framework that can improve sample efficiency in various machine learning settings involving small data or requiring efficient generalization.

The consequent chapters of this thesis flow out of this framework as visualized in Figure 1.1 and can be summarized as follows:

- In Chapter 3, we focus on a typical applied machine learning workflow, which usually involves analyzing a custom real-world dataset and training various neural network architectures to perform some specific task using this data. Specifically, we show that deep learning architectures that are typically used for language modelling can be used for imputation of missing observations in any kind of sequence data, even when a large fraction of the sequence is missing. We provide a practical demonstration of this with an application to improving anomaly detection in safety-critical systems (like cars) that rely on clean data. More generally, the study in this chapter demonstrates that generalization in practical machine learning settings is accurately described by

our proposed framework: information sharing between different observations within a dataset enables imputation of missing observations from the remaining ones.

- In Chapter 4, we turn our focus to studying how our framework can be used to enable models to generalize from small data. As a first step in this direction, we explore whether large information-rich training datasets can be reduced, or repackaged, into small information-rich training datasets that models can still effectively generalize from. Dataset distillation is a method for reducing training dataset sizes for neural networks by learning a small number of synthetic observations that contain all the 'useful' information of a large dataset [Wang et al., 2018]. We develop a soft-label version of dataset distillation and show it can leverage the shared information, or features, between different classes in order to reduce a training dataset to less than one example per class. This provides the first empirical evidence that LO-shot learning is possible. We also extend dataset distillation to work with sequential data like text.

- In Chapter 5, we aim to provide theoretical validation for LO-shot learning to confirm that models can indeed generalize from small datasets. We consider a seemingly simple classification task that turns out to be particularly difficult for kNN classifiers, requiring a large amount of training data to properly fit the classifiers. We demonstrate a connection between sample efficiency in neural networks and in kNN classifiers and prove that similarly to the dataset distillation case with neural networks, soft labels can also greatly reduce the amount of data required to fit a kNN. Unexpectedly, we prove that a classifier can learn to separate any finite number of classes after fitting on as few as two soft-label prototypes. We use this result to develop the first algorithm for generating LO-shot prototypes for kNN classifiers.

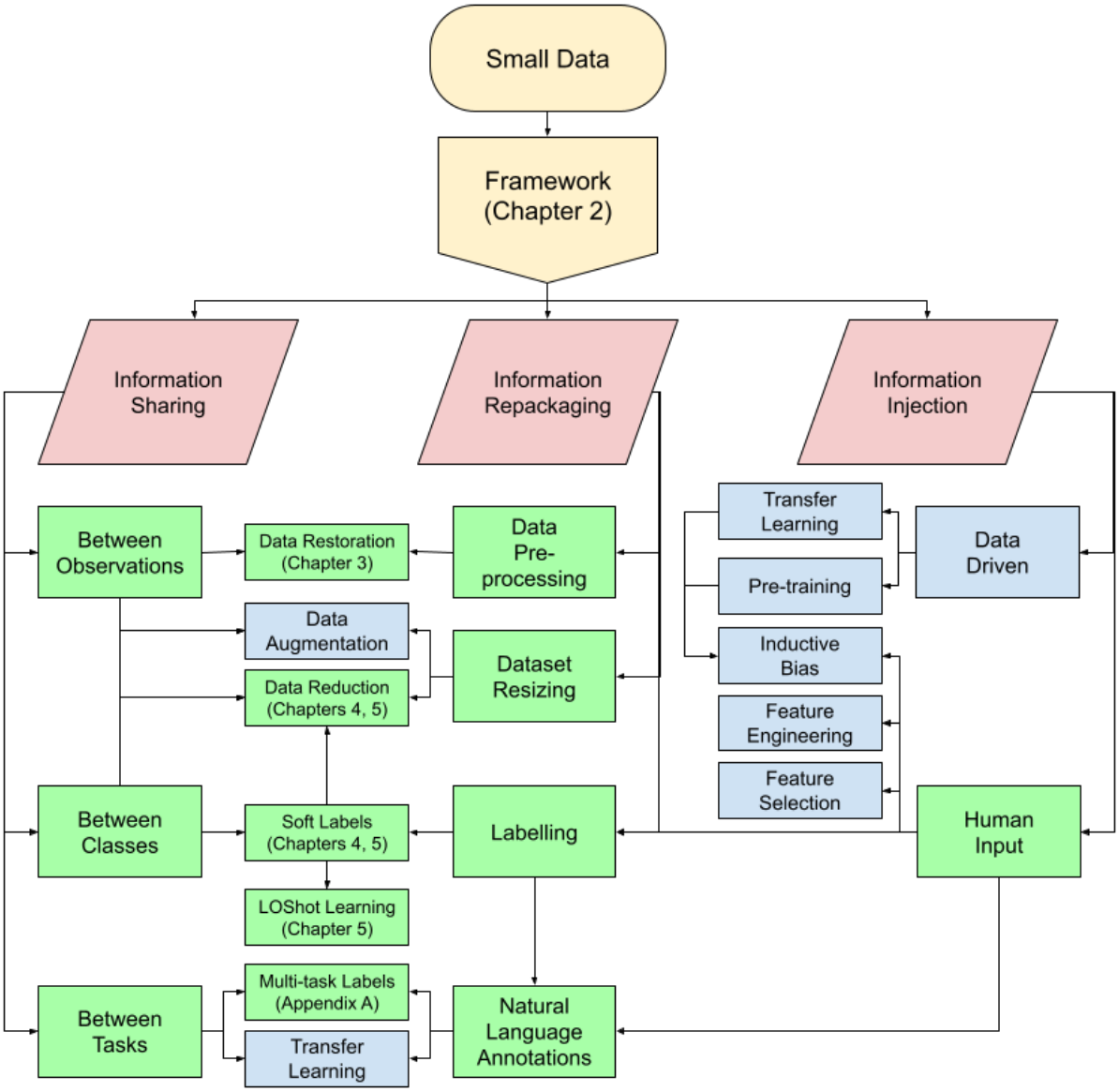


Figure 1.1: The consequent chapters of this thesis flow out of the framework defined in Chapter 2. Red cells denote the three core premises of the framework. Green cells denote topics to which this thesis makes novel contributions.

Chapter 2

Extreme Data Efficiency and 'Less Than One'-Shot Learning

2.1 Introduction

As mentioned above, deep supervised learning models are extremely data-hungry, generally requiring a very large number of samples to train on. Meanwhile, it appears that humans can quickly generalize from a tiny number of examples [Lake et al., 2015]. Getting machines to learn from 'small' data is an important aspect of trying to bridge this gap in abilities.

In Section 2.2, we propose a new machine learning setting we call 'Less Than One'-Shot Learning (abbreviated as 'LO-shot learning' or 'LOSL'), where a model must learn to recognize N classes from M $<$ N training examples. We also compare and contrast this proposed setting to existing, related settings like few-shot and one-shot learning. Few-shot learning settings like LOSL require a paradigm shift away from the current trend of big data machine learning.

Since many current deep learning methods were developed for use with big data, it can be difficult to achieve the extreme level of sample-efficiency with them that is required for the LO-shot learning setting. In addition, the methodology suggested for increasing sample efficiency in a particular setting often depends on the dataset and task being considered. In Section 2.3, we develop a general, unified framework for understanding and achieving sample-efficient generalization.

Finally, in Section 2.4, we discuss a number of practical settings as motivating examples of how our framework can be applied in practice to solve problems related to data-efficiency.

2.2 'Less Than One'-Shot Learning

Few-shot learning (FSL) is one approach for making models more sample efficient. In this setting, models must learn to discern new classes given only a few examples per class [Lake et al., 2015, Snell et al., 2017, Wang et al., 2020]. Further progress in this area has enabled a more extreme form of FSL called one-shot learning (OSL); a difficult setting where models must learn to discern a new class given only a single example of it [Fei-Fei et al., 2006, Vinyals et al., 2016]. We now propose 'less than one'-shot learning ('LO-shot learning' or 'LOSL'), a setting where a model must learn N new classes given only $M < N$ examples, less than one example per class. At first glance, this appears to be an impossible task, but throughout this thesis we both theoretically and empirically demonstrate feasibility.

Some readers may notice a similarity of this formulation to zero-shot learning (ZSL). ZSL is a few-shot learning setting where models must learn to recognize previously unseen classes during inference time, typically using meta-information or latent features [Xian et al., 2018]. One key difference between ZSL and LOSL is that ZSL only restricts the number of training examples available for the unseen classes (i.e. some classes have zero associated examples but the remaining classes can have any number of training examples), but LOSL restricts the total number of training examples (i.e. N classes but fewer than N training examples). In addition, LOSL assumes that the user has knowledge of all classes at training time. While some of those classes may not be covered by training examples (similarly to ZSL), the *a priori* knowledge of their existence can be used during training to improve the model's ability to detect them at inference time. In particular, throughout this thesis and especially in Chapter 5, we show that soft labels can be used to form a distribution over all known classes (even if training examples are unavailable for some of those classes) resulting in coverage of N classes by a set of $M < N$ training examples. The assumption that a user has prior knowledge of all classes is typically not used in ZSL frameworks though there are some that also leverage it, for example, to develop attribute-based profiles (or 'signatures') of unseen classes that can be used to detect them at inference time [Romera-Paredes and Torr, 2015].

As an analogy, consider an alien zoologist who arrived on Earth and is being tasked with photographing a unicorn. The alien has no familiarity with local fauna and there are, of course, no photos of unicorns. As a result, humans show the alien a photo of a horse and a photo of a rhinoceros, and say that a unicorn is something in between. With just two examples, the alien has now learned to recognize three different animals. This is the essence of LO-shot learning, generalizing to more (known) classes than the number of available training examples.

However, with the limited information that was shared, the alien's mental represen-

Figure 2.1: An artist's rendition of what a rhinocorn might look like in it's natural habitat. Describing a unicorn as being somewhat similar to both horses and rhinoceroses may be helpful for identification when no photos of unicorns are available, but it may also be insufficient for differentiating it from other hybrids such as this one. *(Created by Asya Sucholutsky)*

tation, or conception, of a unicorn may be different than what the humans had in mind. One such possible conception, a 'rhinocorn', is pictured on the front cover of this thesis as well as in Figure 2.1. In order to avoid such misunderstandings, approaches to extreme few-shot learning must be more careful and complex than the simplistic one described in our analogy. Throughout this thesis we aim to develop such approaches and analyze their capabilities and limitations.

2.3 A Framework for (Efficient) Generalization

In in this section, we aim to create a unified and general framework that explains sample-efficient generalization in terms of three core principles: information sharing, information repackaging, and information injection. This framework will prove to be helpful for explaining few-shot learning and beyond that, LO-shot learning.

Learners aim to generalize from training data by forming 'beliefs' (e.g. patterns, priors, etc.) that continue to hold true when applied to previously unseen test data. Our framework postulates that, regardless of whether these beliefs are learned exclusively from training data or come as priors or inductive bases from external sources, the underlying mechanism that enables them is the same: shared information, or redundancy, within the data (see Section 1.1 for a full definition of information sharing). In particular, we find that the following three major types of information sharing enable many modern machine learning settings: sharing between observations, sharing between classes, and sharing between tasks. We explain each type of information sharing in the next section.

Without information sharing between current and future observations, generalization would not be possible. Therefore, we consider information sharing to be the root of generalization. However, we still need a mechanism for surfacing insights and prior knowledge about shared information to our models. We believe that creating information-rich labels (see Section 1.1 for a generalized definition of labels) is an information repackaging and injection technique that can quantify, or provide prior knowledge about, each such type of information sharing. We discuss information repackaging in Section 2.3.2, and information injection in Section 2.3.3. Our results throughout this thesis suggest that using rich labels to simultaneously leverage multiple types of information sharing can greatly improve sample-efficiency and even enable LO-shot learning.

2.3.1 Types of Information Sharing

We generally divide information sharing into three broad categories: between observations, between classes, and between tasks. We summarize what we mean by each one and how they connect to the rest of this thesis.

Information Sharing Between Observations

Correlation, knowledge duplication, and/or partial redundancy between different observations are required for pattern-matching, inter/extrapolation, and any kind of generalization from training to inference-time. Broadly speaking, these forms of information sharing between observations is what enables learning from data in the first place. Data restoration (e.g. imputation of missing values in a sequence) and data reduction are two clear examples of machine learning problems that are only solvable due to information sharing between the observations.

Data Restoration If knowledge is highly distributed, or mostly redundant, across a dataset, then we would expect that losing some fraction of the examples would not result in a large amount of knowledge being lost. In other words, models can use the remaining samples to restore lost samples because of information sharing between observations. One such way to restore lost data is to train a generative model on the available samples and use it to generate samples from missing portions of the distribution, a technique that is also often used for data augmentation [Antoniou et al., 2017]. In Chapter 3, we propose another way to investigate the idea that knowledge is duplicated within datasets by studying the ability of state-of-the-art (SOTA) deep learning models to impute lost or missing sequential data.

Data Reduction If knowledge is indeed duplicated, then we can investigate how much knowledge is actually contained within a dataset. If there is actually only a small amount of knowledge, or relevant information, contained within a dataset, then it should be possible to design a small number of synthetic samples that contain all the information and can thus train a model to close to its original accuracy. Dataset distillation is a method for reducing dataset sizes: the goal is to learn a small number of synthetic samples (or prototypes) containing all the information of a large dataset [Wang et al., 2018]. One way to think about these synthetic samples is as highly-efficient training examples that are optimized for training a particular model. As a result, we can use them to study the limits of few-shot

learning with different models. In addition, these highly-efficient representations can not only be used to better understand a dataset and the knowledge it contains, but also to greatly speed up a large range of machine learning tasks. The benefits include speeding up deep learning model training, reducing energy consumption, and reducing required storage space.

While this may seem unintuitive at first, these two machine learning problems (restoration and reduction) are very similar both to each other as well as to related problems like dataset augmentation; they are all forms of information repackaging that aim to change the number of samples in a dataset without changing the amount of information contained therein. We discuss these and other forms of information repackaging in more detail in Section 2.3.2.

Information Sharing Between Classes

In our alien zoologist analogy, information sharing between classes is what allows us to describe a unicorn by its common features with horses and rhinoceroses. Horses and unicorns represent distinct classes, but they share information such as common visual features; the same is true for the unicorn and rhinoceros classes.

One way to quantify shared information between different classes is by using *soft labels* (see definitions in Section 1.1) to associate each observation simultaneously with multiple classes. As such, soft labels allow us to encode and decode more information from each example than is otherwise possible. By exploiting the shared information between different classes, and using soft labels to quantify this relationship, we can more efficiently repackage the information necessary for solving classification tasks.

For example, soft labels can be used to improve dataset reduction algorithms that are intended for use with datasets that have implicit (latent) or explicit (user-defined) classes, by increasing the representational power of each synthetic observation. We use this idea in Chapter 4 to propose the Soft-Label Dataset Distillation algorithm which extends dataset distillation to simultaneously optimize the synthetic samples as well as the labels assigned to them. In our experiments with this algorithm, we present the first empirical evidence that LO-shot learning is feasible. Specifically, we show that it is possible to design a set of five soft-labelled synthetic images, that train neural networks to over 90% accuracy on the ten-class MNIST task (less than one training image per class).

While dataset distillation is intended for use with neural networks, information sharing between classes can also be leveraged by other machine learning algorithms. In Chapter 5, we study the increase in sample efficiency enabled by switching from hard labels to soft

labels. In order to study this problem analytically, we focus on a simpler machine learning model, the k-nearest neighbors classifier, and unexpectedly show that soft labels enable kNN classifiers to separate any finite number of classes given just two carefully-designed training prototypes.

Information Sharing Between Tasks

While both of the previous information sharing settings focused on single-task datasets, it is much more common that real-world datasets are used for a multitude of tasks. In real-world settings, models trained to perform different downstream tasks often rely on detecting and learning many of the same patterns. Information sharing between tasks allows us to find these task-independent patterns in datasets which enables major learning settings like transfer learning, pre-training/fine-tuning, and other forms of multi-task learning. For example, the same image dataset could be used for learning different tasks like semantic segmentation, object counting, and classification, but performance on each of these tasks is likely to be improved by the ability to detect edges or other such inherent, structural patterns in the dataset. Multi-task learning systems often aim to learn these patterns independently of the downstream task, so that they can then be re-used for each task without the need to re-learn them from scratch [Zhang and Yang, 2021].

In Appendix A, we discuss how this type of information sharing can be leveraged using rich natural language annotations and the practical implications of this finding. We do not focus on information sharing between tasks in the remaining chapters of this thesis.

2.3.2 Information Repackaging

We consider a dataset to be a set of information that has been packaged in a particular way. Various aspects of that packaging include the observations, features, and labels chosen for inclusion in the dataset. Information repackaging refers to any transformation that modifies these and any other aspects of a dataset while preserving the information contained therein. From this perspective, data pre-processing methods can be seen as a way to repackage the information contained within the dataset. The packaging of a dataset should not have an effect on the performance of a 'perfect' learner. If a dataset contains all of the knowledge or information required to solve a particular task to some desired standard, then a 'perfect' learner should be able to solve the task given any repackaging of the dataset. However, developing such 'perfect' learners that can extract all task-relevant information while discarding any spurious patterns, features, or noise may be equivalent to achieving artificial general intelligence (AGI).

Until such models are created that can always extract all relevant information from a dataset, much of the research and development work in machine learning will continue to revolve around finding 'better' ways to re-package information, where the definition of 'better' depends on the type of task being solved. For example, when training a model to discern dogs and wolves from photographs of them, we may wish to remove backgrounds from the images to prevent models from using presence of snow as an explanatory variable, in order to improve accuracy when testing on previously unseen images (e.g. a wolf indoors, a husky in the snow, etc.). Obvious examples of this include any kind of pre-processing or preparation of the data such as feature selection/engineering and dataset reduction/augmentation. However, even research that focuses on model development is often about finding ways to improve information re-packaging, for example by getting deep neural networks to learn better representations of the data in their intermediate layers.

Resizing Datasets

Since information repackaging transforms a dataset without changing what information is available within it (unlike information injection which we discuss in the next section), this suggests the unintuitive conclusion that more data may not always be better for machine learning [Nakkiran et al., 2019]. For example, one way to augment a dataset is to train a generative model on that dataset and then use it to generate additional samples. This process introduces no new information (aside from inductive biases on the generative model) but simply duplicates and recombines the existing information already contained within the original dataset. Our framework predicts that sufficiently flexible/general models (i.e. models that approach the performance of a 'perfect' learner on the specific dataset being considered) should not perform better when trained on this augmented dataset than when trained on the original dataset.

A recent study provided empirical evidence for this prediction by finding that the performance of large deep learning models in the 'real' world (where data are limited and models see the same training examples multiple times) is similar to their performance in the 'ideal' world (where there is unlimited data and each observation is viewed only once) [Nakkiran et al., 2020]. The 'ideal' world was simulated by training a generative model on CIFAR-10 [Krizhevsky et al., 2009] and using that model to augment the dataset by generating several million additional samples called 'CIFAR-5m'. When testing several different models with different configurations (e.g. regular training, pre-training, data augmentation, etc.) on CIFAR-5m, their results suggest that test error is the same in both the ideal and real worlds, which is consistent with our framework's expectations.

It is also possible to repackage information in such a way that the resulting dataset is

smaller (i.e. has fewer samples) than the original dataset, but still contains all the same information. Conceptually, this is easier to grasp when thinking of a dataset as just one possible, potentially inefficient, packaging of a set of information that may contain various redundancies and noise as byproducts of the sampling and dataset creation methodology that were employed. Throughout this thesis, we extensively explore this idea of reducing the size of datasets as we believe that studying the reduced datasets may provide insights into how sample-efficient generalization can be achieved. In particular, in Chapter 4 we study how large datasets can be reduced into small datasets while maintaining model performance, and then in Chapter 5 we use our findings to find the theoretical lower limit on the number of training examples required for a model to learn N classes.

2.3.3 Information Injection

Additional information can be injected into a dataset in various ways: input from an 'expert' (human or AI) with prior knowledge, such as data labeling and annotation, manual feature selection and engineering, knowledge distillation, pre-training, transfer learning, inductive biases, and many more. If the information contained within a dataset is insufficient for even a 'perfect' learner to solve a task up to some standard, or there are too few samples to distinguish signal from noise, the only available remedy may be to supplement the training process by injecting some external information.

In settings where a large amount of related data (containing relevant external information) is available, data-dependent information injection methods, like pre-training or transfer learning, can be used with 'imperfect' learners to achieve large performance improvements in few-shot learning. In other words, we can bypass the need for 'perfect' learners by throwing large amounts of related data at 'imperfect' learners until they manage to extract all the relevant external information. For example, when working with language models, we often first (pre-)train the model on a large text corpus to teach it how to efficiently represent the language, and then use it to classify domain-specific text into domain-specific classes [Radford et al., 2018]. Encoding the domain-specific text into those efficient representations is a form of information repackaging, but the pre-training injected information into this process that dictates what these representations should be.

Meanwhile, in settings where related data are limited or are highly redundant (i.e. they contain only a small amount of additional external information), information injection methods that use expert input to increase the amount of information contained within existing observations, like labelling and annotation, are far more effective than data-dependent information injection methods. For any type of supervised learning, including

classification, regression, and self-supervised learning, labels (see definition in Section 1.1) are a way to communicate, or encode, the intent of the task. Labels guide the training of the model by specifying what outputs it must learn to produce for given inputs. Labels also allow us to inject the annotator's prior knowledge about the dataset (e.g. knowledge about shared information between observations and shared information between classes) into the training process.

Throughout this thesis, we argue that information-rich labels enable more sample-efficient generalization. In Chapters 4 and 5 we show that in the classification setting, transitioning from hard labels to soft labels already provides a sufficiently large increase in sample-efficiency to enable LO-shot learning. In general, we believe that information repackaging/injection techniques related to labels (e.g. automatic re-labelling, expert annotations, etc.) are the most effective methods for improving few-shot learning performance by simultaneously leveraging the different types of information sharing that occur in datasets. As a result, we advocate for the development of novel methods for collecting increasingly information-rich labels. For example, in Appendix A, we propose a possible method for packing more information into labels by using natural language annotations instead of numerical labels. We believe that this is an important direction for future research as it would not only be a more natural way for annotators to share their insights but would also provide even more information about each observation than soft labels.

2.4 Motivating Examples: Efficient Generalization in Practice

We consider some practical settings where sample-efficient generalization is useful. For each setting, we discuss the underlying assumptions, discuss a few examples of problems that are often encountered within the setting, and suggest solutions to these problems that are consistent with our framework.

2.4.1 Practical Setting 1 - Neural Architecture Search

Neural Architecture Search (NAS) refers to a family of algorithms used for optimizing neural network architectures [Elsken et al., 2019]. These algorithms are typically distinguished by three components: their search space, their search strategy, and their performance evaluation strategy. As deep learning began seeing broad adoption into industry, AutoML and

NAS became increasingly popular methods for designing neural networks for various applications with much less manual effort and domain knowledge than previously required. Unfortunately, NAS tends to be especially computationally expensive as it requires frequent re-training of candidate architectures on the training dataset in order to evaluate their quality.

A popular approach for mitigating the prohibitive computational costs of NAS is to use small proxy datasets when evaluating candidate architectures. As mentioned above, dataset distillation is a powerful information repackaging technique that takes advantage of information sharing between observations in order to reduce the size of training datasets while preserving the information contained within [Wang et al., 2018].

The original dataset distillation algorithm assigns each synthetic sample a single hard label, which limits the accuracies models trained on distilled datasets can achieve. In Chapter 4, we propose to simultaneously distill both images and their labels, and thus to assign each synthetic sample a soft label (a distribution of labels) rather than a single hard label. Our improved algorithm increases accuracy by 2-4% over the original dataset distillation algorithm for several image classification tasks. We show that a variety of popular datasets can be distilled down to very small synthetic datasets and still train deep learning models to close to their original accuracies. For example, training a LeNet model with just 10 distilled images (one per class) results in over 96% accuracy on the MNIST data. Using soft labels also enables distilled datasets to consist of fewer samples than there are classes as each sample can encode information for more than one class. For example, we show that LeNet achieves almost 92% accuracy on MNIST after being trained on just 5 distilled images. As far as we can tell, this is the first empirical evidence that LO-shot deep learning is possible.

We also propose an extension of the dataset distillation algorithm that allows it to distill sequential datasets including texts. We demonstrate that text distillation outperforms other methods across multiple datasets. For example, we are able to train models to almost their original accuracy on the IMDB sentiment analysis task using just 20 distilled sentences.

2.4.2 Practical Setting 2 - Federated Learning

Federated learning systems typically consist of two components: a central node with high compute power where any intensive processing takes place, and multiple edge nodes with limited compute power where neural networks are deployed. When any new data are collected, it is aggregated at the central node, and then distributed to the edge nodes

where it is used to update each of the deployed networks. However, frequent transmissions of large amounts of data may be both expensive and slow. An additional concern in this setting is that the channels used for transmissions may be insecure.

As a motivation example, consider a medical diagnosis smartphone app that runs on users' phones. This app could require transmitting private information, be limited to the compute power available on a smartphone, and have high cost/slow rate for large transmissions. In such a case, it is important to simultaneously reduce the size of the transmission, preserve any private information, and reduce the compute power required for training.

Dataset Distillation and Soft-Label Dataset Distillation have already been shown to be good solutions for reducing transmission sizes of training data for federated learning [Goetz and Tewari, 2020, Zhou et al., 2020]. In Appendix C we use other recent extensions of Dataset Distillation to propose Secure Dataset Distillation (SecDD). To summarize briefly, SecDD intentionally overfits the soft-label distillation process to a particular neural network with known weights. As a result, not only is the resulting synthetic training dataset small, but it is also privacy-preserving as it does not contain recognizable features and can only be used for training by the specific combination of neural network and weights to which it was overfitted.

2.4.3 Practical Setting 3 - Expensive Inference

Instance-based learning algorithms like k-nearest neighbors are incredibly popular due to their simplicity and interpretability. However, these algorithms tend to be very computationally expensive at inference time as they usually involve comparing new observations against a large number of previously seen observations.

As a motivating example, consider a movie recommendation engine with millions of existing users and a large selection of movies. When a new user joins the service and enters some information about themselves (e.g. a set of movies or genres that they like), the engine wants to find the movies they are most likely to be interested in. One straightforward way to do this would be to compare the new user's profile against every existing user in the system and make a recommendation based on the preferences of the most-similar existing users. While straightforward and likely effective in terms of predictive power, this kind of kNN-type approach involves performing millions of comparisons and an expensive sorting of the resulting similarity scores every time that a new user joins the system.

As mentioned above, using prototype methods to reduce the size of training datasets can drastically reduce the computational cost of classification with instance-based learning

algorithms like the k-Nearest Neighbour classifier [Bezdek and Kuncheva, 2001, Triguero et al., 2011a, Garcia et al., 2012, Kusner et al., 2014]. The number and distribution of prototypes required for the classifier to match its original performance is intimately related to the geometry of the training data. As a result, it is often difficult to find the optimal prototypes for a given dataset, and heuristic algorithms are used instead.

In Chapter 5.2, we consider a particularly challenging setting where commonly used heuristic algorithms fail to find suitable prototypes and show that the optimal number of prototypes can instead be found analytically. We also propose an algorithm for finding nearly-optimal prototypes in this setting, and use it to empirically validate the theoretical results. Finally, we show that a parametric prototype generation method that normally cannot solve this pathological setting can actually find optimal prototypes when combined with the results of our theoretical analysis.

In Chapter 5.3, we use a soft-label generalization of the k-Nearest Neighbors classifier to explore the intricate decision landscapes that can be created in the 'less than one'-shot learning setting. We analyze these decision landscapes to derive theoretical lower bounds for separating N classes using $M < N$ soft-label samples and investigate the robustness of the resulting systems. We find that assigning soft labels to prototypes can allow increasingly small sets of prototypes to accurately represent the original training dataset.

In Chapter 5.4, we use this to propose a novel, modular method for generating soft-label prototypical lines that still maintain representational accuracy even when there are fewer prototypes than the number of classes in the data. In addition, we propose the Hierarchical Soft-Label Prototype k-Nearest Neighbor classification algorithm based on these prototypical lines. We show that our method maintains high classification accuracy while greatly reducing the number of prototypes required to represent a dataset, even when working with severely imbalanced and difficult data.

2.4.4 Practical Setting 4 - Sequence Imputation

In most areas of machine learning, it is assumed that data are fairly consistent between training and inference. Unfortunately, in real systems, production/inference data are plagued by noise, loss, and various other quality reducing factors while training datasets are typically well-curated and clean. While a number of deep learning algorithms solve end-stage problems of prediction and classification, fewer aim to solve the intermediate problems of data pre-processing, cleaning, and restoration at inference time. These inter-

mediate problems are clear examples of the 'information re-packaging' concept we previously discussed.

Self-driving cars are the latest example of automotive systems becoming increasingly complex and algorithm-dependent. Cars and other automotive systems typically contain dozens of small computers that must communicate between each other to keep the vehicle functioning. The event data exchanged by these computers, called system traces, can be monitored to perform predictive analytics tasks like anomaly detection. As mentioned above, real systems like cars are plagued with data problems like data loss, but downstream algorithms like anomaly detection often require clean data to function properly. Thus, a solution must accurately restore lost events given only the remaining events in a sequence as input.

In Chapter 3, we develop a method for accurately reconstructing discrete temporal or sequential system traces affected by data loss, using Long Short-Term Memory Networks (LSTMs). The model works by learning to predict the next event in a sequence of events, and uses its own output as an input to continue predicting future events. As a result, this method can be used for data restoration even with streamed data. Such a method can reconstruct even long sequence of missing events, and can also help validate and improve data quality for noisy data. The output of the model will be a close reconstruction of the true data, and can be fed to algorithms that rely on clean data. We demonstrate our method by reconstructing automotive data (CAN traces) consisting of long sequences of discrete events. We show that given even small parts of a CAN trace, our LSTM model can predict future events with an accuracy of almost 90%, and can successfully reconstruct large portions of the original trace, greatly outperforming a Markov Model benchmark. However, Long Short-Term Memory (LSTM) networks suffer from a major bottleneck: a large number of sequential operations. To remedy this, we use attention mechanisms from the recently proposed Transformer models [Vaswani et al., 2017] to entirely replace the recurrent components of these data-restoration networks. We demonstrate that such an approach leads to reduced model sizes by as many as 2 orders of magnitude, a 2-fold to 4-fold reduction in training times, and 95% accuracy for automotive data restoration. We also show in a case study that this approach improves the performance of downstream algorithms reliant on clean data.

The ability to restore lost data, even when faced with high levels of loss, suggests that information is indeed shared across observations. This duplication may be useful when we are dealing with high levels of noise or loss as it allows our models to still extract the true signal or pattern from the dataset. However, this duplication also leads to a large number of samples being required to store a relatively small amount of knowledge.

Chapter 3

Data Restoration

Introduction

The growth of the Internet of Things has led to a sharp increase in the number of real-world datasets being collected, processed, and analyzed in a decentralized way. Unfortunately such data are rarely as clean as in lab conditions and are often plagued by noise, loss, and various other quality reducing factors. However, in many areas of machine learning, algorithms are still designed with the assumption that data at time of inference is of the same quality as training data. As a result, when algorithms are trained on clean, pre-processed samples, they may fail to function as desired when performing inference on raw data. In particular, data loss is a serious issue for many algorithms. This is especially important for safety-critical systems like cars, where a failure can have very serious consequences for users. Anomaly detection, and other end-stage algorithms, need to have very high performance in such systems, but lost data may be a large factor in preventing this. As a result, for systems that analyze streams of data, a method is needed to quickly impute the missing values as closely as possible to what the true values would have been.

Deep learning techniques have been proven to be effective at learning temporal structure of data [LeCun et al., 2015]. However, while a number of deep learning algorithms have been successfully shown to solve numerous end-stage problems like prediction and classification [Glorot et al., 2011, LeCun et al., 2015, Abadi et al., 2016], very few attempts have been made to use them for solving the intermediate problems of data pre-processing [Kotsiantis et al., 2006, García et al., 2015], cleaning [Kotsiantis et al., 2006, García et al., 2015],

and restoration [Efron, 1994, Lakshminarayan et al., 1996], even though from a machine learning perspective these end-stage and intermediate problems can be very similar.

Many real-world systems produce discrete datasets such as text or event logs during their operation. Automotive Controller Area Network (CAN) traces, for example, consist of a long sequence of discrete events. A CAN bus is a communication system that allows various types of devices like microcontrollers to communicate with each other in real-time without needing a host. It is a message-based protocol, designed originally for multiplex electrical wiring within automobiles to save on the cost of copper wires, but is also used in many other applications. The messages in a CAN trace can be considered as 'words' produced by the CAN bus. Therefore, various supervised learning techniques from natural language processing (NLP) are likely to be effective in analyzing the patterns in sequences - or "sentences"- of these events.

Recurrent Neural Networks (RNNs), are neural networks that have an internal memory. Long Short-Term Networks (LSTMs) are a type of RNN that have a superior ability to learn long-term dependencies in data. LSTMs have been used extensively for work with Natural Language Processing (NLP) and event based data [Sutskever et al., 2014], they have been used for data compression [Filippova et al., 2015], and they have recently even been used - with varying degrees of success - directly for anomaly detection [Malhotra et al., 2015]. The use of LSTMs for restoration of lossy system traces has not been explored. We propose a method for using LSTMs to accurately reconstruct discrete temporal traces affected by data loss. The reconstructed traces can then be used by algorithms that rely on 'perfect' data. As a case study, we tested the performance of a specification mining framework that uses timed regular expressions and deterministic finite state automata for extracting system behavior.

The model works by learning to predict the next event in a sequence of events from a large database of CAN traces, and uses its own output to continue prediction for multiple points ahead in time, allowing for even large chunks of lost data to potentially be restored. We separately feed the original, lossy, and reconstructed traces into a Timed-Regular Expression Mining (TREM) framework [Cutulenco et al., 2016, Narayan et al., 2018, Schmidt et al., 2017] to gauge the effectiveness of our LSTM-based reconstruction approach. We show that given even small parts of a CAN trace, the LSTM model can accurately reconstruct large portions of the original trace thereby permitting the use of algorithms like TREM, that are reliant on 'perfect' data, with 'imperfect' data that would otherwise cause them to perform poorly.

Unfortunately, LSTMs suffer from major bottlenecks like requiring large numbers of sequential operations that cannot be parallelized. Recently, Transformer [Vaswani et al.,

2017], a novel encoder-decoder model that heavily uses attention mechanisms [Luong et al., 2015], was proposed as a replacement for encoder-decoder models that use LSTM or convolutional layers, and was shown to achieve state-of-the-art translation results with orders of magnitude fewer parameters than existing models. Inspired by this impressive result, we additionally propose using a version of Transformer with modified hyper-parameters, that we will refer to in this chapter as "Restorer". Restorer solves the intermediate problem of restoring missing elements in sequences of discrete data while entirely replacing the recurrent components of existing solutions. We demonstrate that such an approach leads to reduced model sizes, faster training times, and higher-quality reconstruction when compared to the LSTM model.

Our contributions are as follows:

- We implement and empirically validate LSTMs as a deep learning approach to restoring lost sequential data.
- We implement and empirically validate the neighbourhood-restriction on Transformer that was theoretically proposed by Vaswani et al. [2017].
- We propose and create Restorer, a Transformer-based model tailored for imputing sequential data.
- We match the LSTM accuracy in restoring lost automotive CAN data with a Restorer model that has just 0.6% of the number of parameters.
- We use Restorer to improve the accuracy in restoring lost automotive CAN data by up to 7% while requiring 2-4 times less training time compared to the LSTM
- We demonstrate Restorer's positive impact on downstream algorithm performance in a case study where it outperforms both the benchmark and LSTM models.

3.1 Background

3.1.1 Sequence Modelling with Deep Learning

The most popular deep learning architecture for sequence modelling is Recurrent Neural Networks (RNNs), a type of neural network with an internal feedback mechanism that can be used as a form of memory [Williams and Zipser, 1989]. Probably the most successful extension of RNNs are LSTMs, which increase the flexibility of the internal feedback

mechanism [Gers and Schmidhuber, 2001]. More recently, LSTM-based encoder-decoder models like Seq2Seq were shown to improve the state-of-the-art in sequence modelling and the addition of attention mechanisms was shown to further increase their performance [Sutskever et al., 2014]. In such models, the encoder performs an embedding of the entire input sequence before the decoder begins to use this embedding as input when generating an output sequence.

There has recently been much discussion about whether recurrent models provide any intrinsic advantage over feed-forward models [Miller and Hardt, 2018]. For example, Vaswani et al. [2017] recently proposed that an encoder-decoder model they call Transformer, can be built that entirely eschews recurrent components. Transformer makes use of several attention mechanisms to form an architecture that significantly outperforms the state-of-the-art on machine translation tasks without resorting to recurrent layers [Vaswani et al., 2017]. While Transformer was demonstrated to work specifically with the task of machine translation, its impressive performance suggests that it is not unreasonable to expect it can be adapted to work with other sequence modelling tasks.

3.1.2 Data Restoration with Deep Learning

Classical techniques for imputing lost values typically revolve around using a local or global mean wherever a missing value occurs [Donders et al., 2006, Schmitt et al., 2015]. When working with discrete data, instead of means, class values that occur with high frequency locally or globally are often used in techniques like hot-deck imputation [Andridge and Little, 2010, Myers, 2011, Aljuaid and Sasi, 2016], k-nearest neighbours [Cover and Hart, 1967], decision trees [Safavian and Landgrebe, 1991], etc. Other, more complex, techniques work only with multivariate, and primarily continuous, data [Raghunathan et al., 2001, Buuren and Groothuis-Oudshoorn, 2010]. However, all of these techniques fail to properly address the problem of imputing missing discrete data in sequences [Schafer and Graham, 2002].

Interestingly, while deep learning has been repeatedly shown to be effective at sequential modelling, there has been little work in applying it to data restoration. Of the methods that do address data restoration, many are designed specifically for the restoration of continuous data [Duan et al., 2014, Zhou and Huang, 2017, Niklaus et al., 2017]. Some methods are intended for use only with image data through techniques like in-painting [Xie et al., 2012, Lehtinen et al., 2018, Altinel et al., 2018]. More recently, Generative Adversarial Networks (GANs) have been used for super-resolution and denoising of images, although their primary use case is typically image generation [Goodfellow et al., 2014, Ledig et al.,

2017]. There are also a number of data restoration methods that assume that all the data is available at once [Blend and Marwala, 2008, Leke et al., 2015, Gondara and Wang, 2017, Beaulieu-Jones and Moore, 2017].

3.2 Setup

3.2.1 Data

A simple dataset was needed that was guaranteed to have temporal patterns without being trivial. Many messages in automotive CAN data occur in a periodic way so temporal patterns are present; however, other messages in CAN data are event-triggered and thus predicting upcoming messages is a non-trivial problem. A CAN trace is a long sequence of timestamped messages that consist of an ID and a payload of several bytes of data. We strip away the payload and timestamps and keep only the message IDs in order to obtain a dataset consisting of discrete, sequential data.

Our CAN data comes from a Lexus RX450h hybrid SUV. The data are split into a number of maneuvers that were repeated multiple times. To reduce some of the variance and reduce training time while testing our approaches, we used traces from a single maneuver, the vehicle driving at 20 km/h and decelerating down to 0 km/h. 20 clean traces of this maneuver were available. Of the 20 traces, 15 were used for training and validation, while 5 were held out for testing.

The traces used for training were examined to compose a dictionary of possible message IDs. A total of 43 different message IDs were found in the training set. We added a 44th element to the dictionary to designate any "other" message IDs that may not have been present in the training set. We then one-hot encoded all traces using this dictionary, replacing the one-dimensional message ID with a 44-dimensional vector of indicator variables where 43 elements have the value 0 and 1 element the value 1.

3.2.2 Benchmark

In order to understand what the results mean, it is important to have a benchmark. We create a benchmark using a fast Markov Model that could be described as a history search, or as a conditional probability method. It has been shown that in situations where data or computational power are limited, Hidden Markov Models can match the performance

of LSTMs [Panzner and Cimiano, 2016]. As a result, the Markov Model will act as a benchmark that our models can be compared against.

The basic premise of this method is that the next message is highly correlated with the preceding messages in a sequence. However, this correlation is likely to degrade as we look further into the past. As a result, we can say that the system is a Markov process of order n, where the previous n messages form a state that influences the next one. Sequences of n consecutive messages are often called "n-grams", and their analysis is common in sequence modelling domains like Natural Language Processing (NLP) [Schonlau et al., 2017, Lin et al., 2012, Lesher et al., 1999]. The most straightforward method of using this property is to perform a history search where every time we want to make a prediction, we look at the previous n messages, and then search our entire training dataset to find the most commonly occurring message after this n-gram. This of course, can involve performing a large number of repeated searches since we do not store the results between searches.

Instead we can build a model of the system by selecting an n, and creating a separate state for each possible combination of the d unique messages. Of course, initializing all possible states at once means that our model will contain n^d states even if some of these are never seen in the data, which may quickly become too memory-intensive for large n or d. To mitigate this issue, we iteratively fill a dictionary D by traversing the training data using the following algorithm:

During inference, each time we would like to impute a value, we take the n preceding values to create a state and look it up in the dictionary to find the highest probability transition. If there are multiple missing values within n steps of each other, then we impute them chronologically and use our previously imputed values when imputing the consecutive ones. One issue that arises for larger n and d or smaller training datasets, is that there may be states in the testing data that are not found in the training data. To address this, we updated D to include sub-sequences of length less than n as states; D now contains k-grams where $k \leq n$. When we want to impute a missing value, we find the maximal length state in D that matches the preceding values.

After experimenting with different values for the hyper-parameter n, we found that accuracy reached its peak and stayed constant above $n \geq 30$; for consistency with the LSTM experiments we used $n = 40$. The maximal next-message prediction accuracy achieved with this Markov model was 76.55%, but when predicting multiple messages consecutively, the accuracy rapidly dropped off, falling to just 37.01% when predicting 20 events into the future. Tables 3.8 and 3.11 offer some more benchmark results and some comparisons in accuracy between Restorer, LSTM, and this benchmark model.

Algorithm 1: Iteratively learn transition frequencies

Result: Dictionary D containing all states and transition frequencies found in the training data

```
i = 0;
while i + n < length(train_data) do
    K = train_data[i : i + n];
    if K ∉ D.keys() then
        Let S_K be a new dictionary;
        for each m_i in the set of the d unique messages do
            S_K[m_i] := 0;
        end
        D[K] := S_K;
    end
    m* = train_data[i + n];
    S_K[m*] = S_K[m*] + 1;
    i = i + 1;
end
```

3.3 LSTM

3.3.1 RNNs and LSTMs

One of the major limitations of non-recurrent neural networks is that they operate on fixed-size vectors, performing a limited number of transformations to derive another fixed-size vector. This limits the effectiveness of non-recurrent networks in identifying features dispersed over sequences or over time. Recurrent neural networks (RNNs) are a type of neural network that use a feedback mechanism to allow the network to operate on a sequence of inputs as well as outputs by selectively keeping information about previous states. In this manner, the i^{th} output vector y_i of an RNN layer is a function of the i^{th} input x_i to the layer as well as the previous output y_{i-1} of the layer. As with regular layers, the function is generally a non-linear transformation over a weighted sum of the terms involved.

$$y_i = f(y_i, x_i, y_{i-1}) = f(V * y_{i-1} + W * x_i + c)$$

Of course in such a recursive function, every previous output and input must be con-

sidered when adjusting weights during training, as during backpropagation they would all have an impact on the error gradient. These incredibly long chains of derivatives pose two problems: computational intensity and vanishing or exploding gradients.

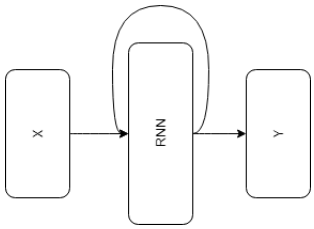


Figure 3.1: **Single RNN Unit:** A recurrent neural network uses a feedback mechanism to access information about previous states.

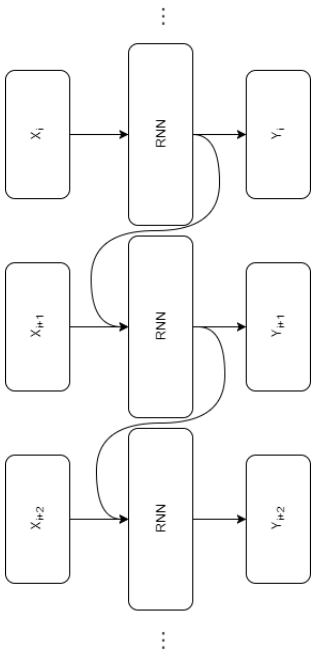


Figure 3.2: **Unrolled RNN Unit:** The feedback loop in a recurrent neural network can be unfolded for an alternative, sequential representation of the repeated transformations it performs.

One solution to these problems is known as **unrolling** the RNN. This requires consid-

ering that if an RNN is essentially a loop with signal flowing through, it can also be equally represented by 'unfolding' the loop as a long series of the same transformations applied to the signal. The structures in Figure 3.1 and Figure 3.2 are thus equivalent ways of representing an RNN. Unrolling is the process of using the second, 'unfolded' representation of an RNN and having the option to consider only a fixed number of previous states when predicting the current output. If we use the analogy of short-term memory to describe the feedback mechanism of an RNN, then unrolling would be displaying this memory as a sequence of events and having the option to cut off the sequence at some event, ignoring any events that came before it.

A second solution to the problem of vanishing or exploding gradients in recurrent networks was the creation of two gates: input and forget. The input gate controls which information from the current state's input will be used, while the forget gate controls which information will be used from the recurrent outputs of previous states. In such a way, the input and forget gates allow the network to **selectively** remember or forget information about previous and current states. Hochreiter and Schmidhuber (1997) proposed a new type of RNN that makes use of these two gates in their recurrent units to solve the vanishing gradient problem and gave it the name Long Short-Term Memory network (LSTM) [Hochreiter and Schmidhuber, 1997].

3.3.2 Architecture

We built our LSTM model in Tensorflow. After experimenting with a few different architectures, we found that a smaller model with five layers in the hidden portion as seen in Figure 3.3 was sufficient. Additional hidden layers did not improve performance.

The input layer has one node for each entry in the trace dictionary, and one additional "other" node to account for rare events that were not seen in the training data that was used to compile the dictionary. Input dropout was set to 0.2.

It is followed by two densely-connected hidden layers with double the number of nodes as in the input layer. Each of these hidden layers uses a hyperbolic tangent activation function, and has a dropout of 0.4.

The hidden layers are followed by 2 LSTM layers, using Tensorflow's LayerNormBasicLSTMCell. These cells perform layer normalization based on Lei Ba et al. [2016]. Each one is unrolled for 40 steps; increasing unrolling further did not seem to increase performance on this dataset, but may be useful for others such as those with higher dimensionality. Both LSTM layers also use the hyperbolic tan activation function, and recurrent dropout

of 0.4 based on Semeniuta et al. [2016]. Each has four times as many nodes as each of the hidden layers, or eight times as many as the input layer.

Finally, the last hidden layer is another densely connected layer with a sigmoid activation function and our final configuration has the same number of nodes as the input layer. However, in an earlier configuration explained below, this layer had n times the number of nodes as the input layer, where n is the number of future events the user wishes to simultaneously predict. The output of this layer is the output of the model.

To increase training speed, LayerNormBasicLSTMCell layers could be replaced with either the peephole [Sak et al., 2014] or the non-peephole [Hochreiter and Schmidhuber, 1997] implementation of CoupledInputForgetGateLSTM layers that couple the input and forget gate as described by Greff et al. [2015] resulting in less computational operations but higher variance in performance. For our CAN trace experiment, the vocabulary size was 43, our input layer had 44 nodes, our hidden layers had 88 nodes each, our LSTM layers had 352 nodes each, and the last hidden layer had 44 nodes whose output was considered the output of the model.

3.3.3 Training

The LSTM was trained using the following procedure:

1. Select 2 random traces from set (1 for training, 1 for validation)
2. Train the model on these 2 traces for 10 epochs with a learning rate of 0.2 and a logloss loss function
3. Train the model on these 2 traces for an additional 20 epochs with learning rate decay of 1/1.1 and a logloss loss function
4. Reset learning rate, preserve weights, repeat from step 1 selecting 2 new random traces

Dropout for regularization and randomized input order prevent overfitting, so this procedure should be repeated until accuracy reaches a plateau since early termination due to detection of overfitting is unlikely to occur. Alternatively, the procedure can be repeated a fixed number of times if so desired.

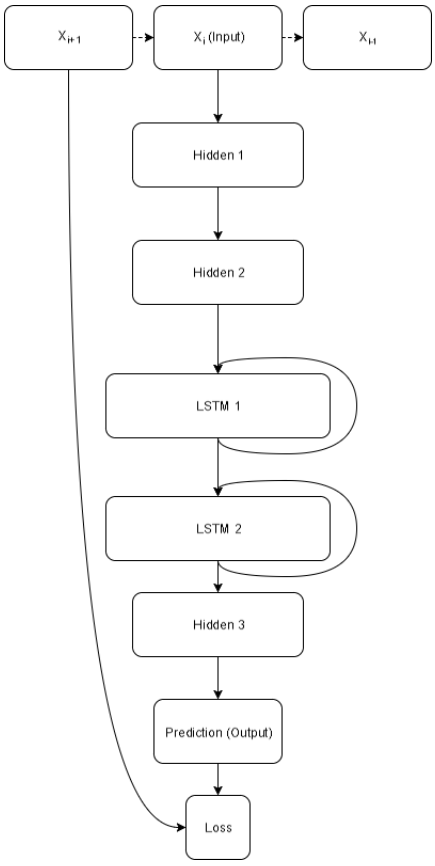


Figure 3.3: **LSTM Model Architecture:** The LSTM model consists of two hidden layers followed by two recurrent LSTM layers and one additional hidden layer. Input is a sequence of events, output is a prediction of next event in the sequence. Loss is calculated as a logloss function comparing the true next event to the predicted one.

3.3.4 Results

Accurately predicting multiple steps forward is a challenging but important problem in lossy data restoration, as multiple consecutive events may have been lost. Two ways of predicting multiple events forward were considered.

The first method was to increase the size of the output layer by a factor of n to directly predict n events forward. In other words, the output layer was modified to have n*44 nodes where n was number of events to simultaneously predict. The output at each step was reshaped into n vectors of 44 nodes each, and each vector predicted 1 future event. Within each vector, the element with the highest value was considered as the prediction. The true positive rate was used as a measure of accuracy. A set of predictions at step i was considered to be a true positive if all n of n predictions made at that step were made correctly. If c_i denotes the number of steps in epoch i where all n predictions were correct, and w_i denotes the number of steps in epoch i where at least 1 of the n predictions was incorrect, then the accuracy for that epoch is denoted by:

$$acc_i = \frac{c_i}{c_i + w_i}$$

One of the issues with this setup is that each time n is changed, a new network needs to be initialized and trained. As such, we retrained the model several times with increasingly large values of n. Table 3.1 documents the average performance of the model for different values of n as well as the range 95% of the measured accuracy levels were within. The average accuracy with this method decayed rapidly as n was increased, while the ranges increased quickly. This suggests that the random initialization of the network plays an increasingly large role in the quality of predictions as n increases.

n	Avg. accuracy	95% range	Benchmark
1	0.895	0.88-0.91	0.766
10	0.58	0.52-0.65	0.454
20	0.48	0.4-0.54	0.370

Table 3.1: n-forward prediction accuracy using the direct method

A second method was developed to address the shortfalls of the first method. The second method predicts only one output at a time; however, the code was altered to allow the model to use its predictions as inputs to itself. In such a way, the model uses its own outputs, one at a time, to make long sequences of predictions. Using the numbers

from Table 3.1, the expected accuracy for this method when predicting n steps forward would be the accuracy for 1 step prediction, to the power of n, under the assumption that the model's predictions derail after it makes even a single mistake. Table 3.2 details the expected accuracy levels for this second method based on this calculation.

n	Expected accuracy	Benchmark
1	0.895	0.766
10	0.330	0.454
20	0.109	0.370

Table 3.2: Expected n-forward prediction accuracy using the step-by-step method assuming model cannot recover after a mistake

n	Accuracy	Benchmark
1	0.895	0.766
10	0.892	0.454
20	0.889	0.370

Table 3.3: True n-forward prediction accuracy using the step-by-step method

While the expected accuracy values for this method would be very low, the convergence in logloss in the model suggested that this accuracy estimate may be underestimating model performance. In fact, when the model was run on the test traces using this step-by-step method, the true results were drastically different as shown in Table 3.3. This suggests that even if the model makes a mistake in its predictions, it is robust enough to continue predicting correctly contrary to the assumption above. However, it is clear that the assumption does hold for the benchmark model and as a result it is not nearly as robust to its own mistakes as the LSTM is.

As mentioned above, the LSTM uses 40 steps of unrolling for the recurrent portion meaning the model requires 40 inputs to fill its internal feedback sequence. Impressively, when given just the first 40 events of the shortest CAN trace held out for testing, the model was able to step-by-step predict the remaining 3500 events of this trace with only several omissions of rarely occurring events as shown in Table 3.4 and some localized mistakes in the order of predicted events as shown in Table 3.5.

Figures 3.4 & 3.5 are visualizations of a sequence of true one-hot embedded events and a sequence of predicted one-hot embedded events, respectively, pulled from the same portion

	Predicted Event	True Event
1	B4	B4
2	25	25
3	22	22
4	23	23
5	B0	**340**
6	320	B0
7	B2	320
8	2D0	B2
9	2C4	2D0
10		2C4

Table 3.4: Example of omitted rare event; rarely occurring event '340' was incorrectly omitted by the model, causing all predicted events beginning from the fourth one to be shifted one up from their true counterparts.

	Predicted Event	True Event
1	25	25
2	22	22
3	23	23
4	2C6	**2C4**
5	B0	2C6
6	320	B0
7	B2	320
8	**2C4**	B2
9	20	20
10	223	223

Table 3.5: Example of local ordering mistake; '2c4' was incorrectly predicted as the 8th event instead of 4th, causing all of the other events from 4th to 8th to also appear misclassified. In reality, true events 5-8 were shifted up by one and predicted as events 4-7.

of a test trace. It is noticeable that mismatches between the two Figures get increasingly worse as the index increases. This is due to the omission problem: each time an event is omitted in the predictions, the entire sequence of predicted events is shifted to the left, causing increasingly large mismatches. When gauging model performance, each omission was recorded, and then the predicted sequence re-aligned at that point in order to once again match the true sequence. Similarly, each ordering mistake was recorded, and the one point detected to be in the wrong position moved to its correct location, to identify whether other mistakes were made in the same area.

On average, the model omitted 5.058% of points and had one local ordering mistake every 11.1 events. Curiously, both these mismatches and ordering mistakes occur at only a slowly increasing rate throughout all of the 3500 predictions, suggesting once again that our model's flexibility makes it at least partially resistant to mistakes in the input, as prediction quality did not decrease even if some local mistakes in output prediction were made and then fed in as input. The LSTM-based approach is thus sufficiently robust to work with and restore not only lossy, but also noisy, data.

Layer Type	**Complexity per Layer**	**Sequential Operations**	**Maximum Path Length**
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Table 3.6: Comparison of different layers where n is sequence length, d is dimension of representation, k is kernel size, and r is neighbourhood size [Vaswani et al., 2017].

3.4 Restorer

3.4.1 Attention

In RNNs, sequences are represented within their hidden states so that the output of each hidden state needs to be computed before the model can access the next step in the sequence. This results in a large bottleneck since a number of sequential operations have to be performed that cannot be parallelized. In addition, this results in long path lengths that elements of the sequence have to traverse, which has been shown to be undesirable

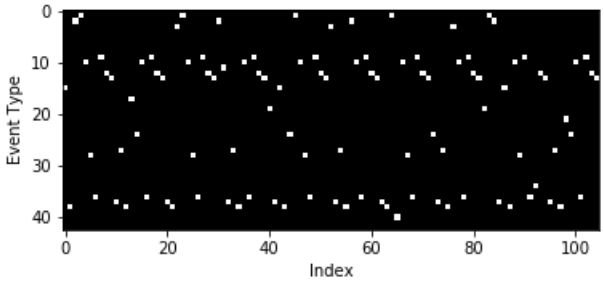


Figure 3.4: **One-hot Encoded True Events:** Visualization of a sequence of just over 100 true events pulled from a testing trace. White pixels correspond to the one active element in that column.

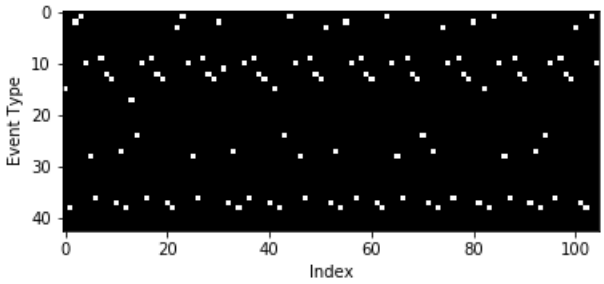


Figure 3.5: **One-hot Encoded Predicted Events:** Visualization of a sequence of just over 100 predicted events pulled from the predictions on a testing trace. White pixels correspond to the one active element in that column.

behaviour when trying to model long-term dependencies [Hochreiter et al., 2001]. Meanwhile, not only can convolutional layers be computationally expensive [Chollet, 2017], but they also need to be stacked in order to connect all inputs to all outputs [Kalchbrenner et al., 2016], which in turn increases path length.

Both recurrent and convolutional layers can instead be replaced with self-attention layers as described in the Transformer model in Vaswani et al. [2017]. This removes the bottleneck of sequential operations and reduces path length as dot-product attention provides access to the entire history at once. As a result, training time should be decreased, while the ability to learn long-term dependencies increased. We confirm that this result holds in practice by demonstrating it empirically. This improvement does come at a cost: an increase in complexity relative to sequence length since the entire sequence is operated on at once and every element attends to every other input. To avoid this new bottleneck, a version of Transformer can be used where input is restricted to neighbourhoods of size r instead of operating on the entire sequence at once [Vaswani et al., 2017]. Table 3.6 summarizes the theoretical differences between these layer types across three key metrics.

3.4.2 Architecture

Restorer is not an RNN or LSTM; it instead follows the Transformer architecture illustrated in Figure 3.6 but with modified hyper-parameters as seen in Table 3.7. For the various Restorer model versions, N_{blocks} is the number of blocks in the encoder and decoder, d_{model} is the dimension of the embedding and consequently is used as the scaling factor in our scaled dot product attention (instead of d_k as described in Vaswani et al. [2017]), d_k is the dimension that keys in attention are projected to, d_v is the dimension that values in attention are projected to, d_h is the size of the hidden layers, and n_{head} is the number of parallel heads in each multi-head attention layer. For the LSTM model, n_{hid} is the number of non-LSTM hidden layers, n_{lstm} is the number of LSTM layers, d_h is the size of the non-LSTM hidden layers, d_{lstm} is the size of LSTM layers, and n_{steps} is the number of steps for which the LSTM is unrolled.

As mentioned above, one of the theoretical advantages of the proposed approach, is a large reduction in the number of trainable parameters. In fact, we show empirically that our Restorer architectures have as many as two orders of magnitude fewer parameters than the LSTM model as seen in Table 3.7.

We implement our Restorer model in Keras based on the implementation of Transformer by Lsdefine [2018].

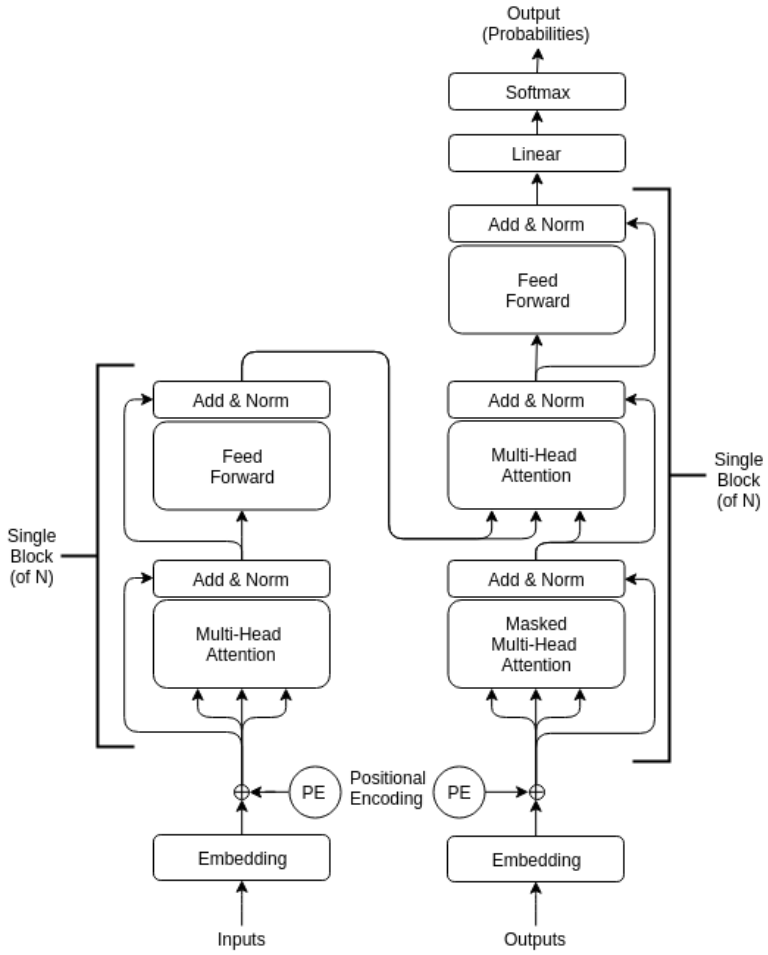


Figure 3.6: **Transformer Model Architecture** [Vaswani et al., 2017]**:** The Transformer consists of an encoder and decoder each made up of N blocks. Input is a sequence of events, output is a sequence of predicted events.

Model							**params**
LSTM	$\mathbf{n_{hid}}$	$\mathbf{n_{lstm}}$	$\mathbf{d_h}$	$\mathbf{d_{lstm}}$	$\mathbf{n_{steps}}$		
	3	2	88	352	40		1,640,892
Restorer	$\mathbf{N_{blocks}}$	$\mathbf{d_{model}}$	$\mathbf{d_k}$	$\mathbf{d_v}$	$\mathbf{d_h}$	$\mathbf{n_{head}}$	
(main)	1	15	64	64	512	4	80,824
	2	15	64	64	512	4	158,873
	3	15	64	64	512	4	236,922
	5	15	64	64	512	4	393,020
	10	15	64	64	512	4	783,265
	15	15	64	64	512	4	1,173,510
	20	15	64	64	512	4	1,563,755
(1-head)	1	15	64	64	512	1	46,264
	2	15	64	64	512	1	89,753
	3	15	64	64	512	1	133,242
(mini)	1	15	32	32	88	1	14,216
	2	15	32	32	88	1	25,657
	3	15	32	32	88	1	37,098
(mini2)	1	32	16	16	32	1	16,704
	2	32	16	16	32	1	27,488
	3	32	16	16	32	1	38,272
(micro)	1	16	8	8	16	1	5,792
	2	16	8	8	16	1	8,624
	3	16	8	8	16	1	11,456

Table 3.7: Size comparison of different model versions in terms of number of parameters

Model							**30-ep. acc**	**300-ep. acc**	**3000-ep. acc**
Benchmark							30.07	30.07	30.07
LSTM	$\mathbf{n_{hid}}$	$\mathbf{n_{lstm}}$	$\mathbf{d_h}$	$\mathbf{d_{lstm}}$	$\mathbf{n_{steps}}$				
	3	2	88	352	40		67.32	86.72	88.59
Restorer	$\mathbf{N_{blocks}}$	$\mathbf{d_{model}}$	$\mathbf{d_k}$	$\mathbf{d_v}$	$\mathbf{d_h}$	$\mathbf{n_{head}}$			
(main)	1	15	64	64	512	4	62.88	90.01	92.56
	2	15	64	64	512	4	74.76	88.12	93.41
	3	15	64	64	512	4	74.19	89.56	**94.00**
	5	15	64	64	512	4	74.17	83.01	84.44
	10	15	64	64	512	4	9.46	9.47	-
(1-head)	1	15	64	64	512	1	69.40	88.22	90.04
	2	15	64	64	512	1	**74.79**	89.87	92.71
	3	15	64	64	512	1	72.01	90.00	92.83
(mini)	1	15	32	32	88	1	68.91	87.01	88.85
	2	15	32	32	88	1	71.67	85.83	91.49
	3	15	32	32	88	1	71.66	89.85	93.15
(mini2)	1	32	16	16	32	1	71.66	91.02	93.28
	2	32	16	16	32	1	73.74	90.89	94.01
	3	32	16	16	32	1	74.60	**91.27**	93.69
(micro)	1	16	8	8	16	1	59.09	84.80	86.45
	2	16	8	8	16	1	64.87	85.35	90.33
	3	16	8	8	16	1	64.29	88.96	92.22

Table 3.8: Maximum percent accuracy after n epochs when trained with input and output lengths of 40

3.4.3 Results

The models are trained using Adam optimizer [Kingma and Ba, 2014] with a fixed learning rate for 30 epochs on a randomly selected trace, before moving on to another randomly selected trace and repeating the procedure. Restorer is the restricted neighbourhood version of Transformer that was briefly mentioned as a future direction in Vaswani et al. [2017], so input sequence length can be limited to a neighbourhood of size 40 to be consistent with the LSTM model. As a result, each model received 40 consecutive messages as input and was asked to predict the next 40 messages as output. One important change when using the restricted version of Transformer is that the model no longer needs to learn to use an end of sequence token as length of the output is predetermined.

Model	**30-ep. time**	**300-ep. time**
LSTM	77.64	807.25
Restorer		
(main, $N_{blocks} = 3$)	36.22	231.95

Table 3.9: Training time in seconds on single V100 GPU

Table 3.8 illustrates that the majority of the different Restorer combinations outperform the LSTM model by several percent, and the benchmark by much more. Interestingly, the best performance seems to be achieved by smaller architectures with either 2 or 3 identical blocks, although the smallest architecture, micro, has a noticeable drop in accuracy. The larger architectures seem to primarily suffer from overfitting, as training accuracy quickly begins to exceed validation accuracy during their training. Models with 10 or more blocks did not converge at all past 10% test accuracy, likely due to the fixed learning rate being used. Figure 3.7 shows the full testing accuracy for the top configuration of each Restorer model version. While the increase in testing accuracy certainly slows down by 3000 epochs, we found that even after 6000 epochs small increases were still being made.

On an Amazon Web Services (AWS) p3.2xlarge machine with a single V100 GPU and a batch size of 512, we found that the Restorer main model with 3 blocks was on average 2-4 times as fast as the LSTM model, as shown in Table 3.9.

While this method of training by selecting one random trace at a time does achieve high accuracy and lead to faster training times, we found that by concatenating all of the training data and training Restorer on it all at once, higher accuracy can be achieved.

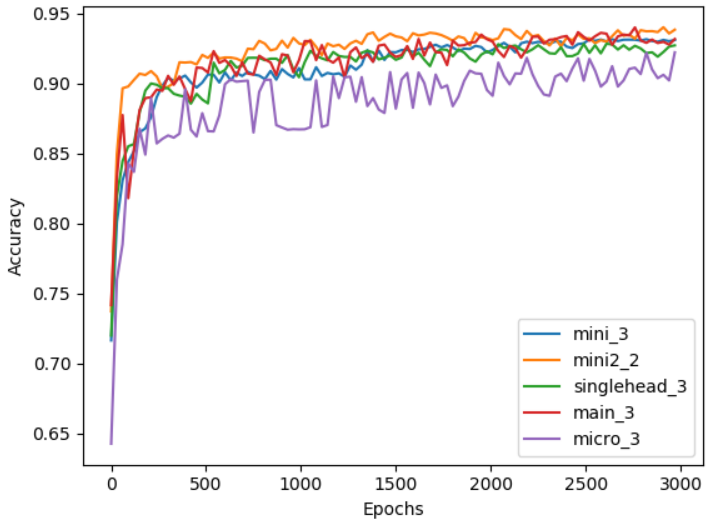


Figure 3.7: Testing accuracy of best Restorer configurations measured every 30 epochs of training. Model titles in the legend follow the format [model name]_[# of blocks].

Table 3.10 shows that the Restorer achieves one to two percent higher accuracy when trained in this way instead of the trace-by-trace method.

Model							**30-ep. acc**	**300-ep. acc**	**3000-ep. acc**
Restorer	$\mathbf{N_{blocks}}$	$\mathbf{d_{model}}$	$\mathbf{d_k}$	$\mathbf{d_v}$	$\mathbf{d_h}$	$\mathbf{n_{head}}$			
(main)	1	15	64	64	512	4	93.72	94.69	94.79
	2	15	64	64	512	4	**94.16**	**95.06**	**95.10**
	3	15	64	64	512	4	91.48	94.72	94.82
	5	15	64	64	512	4	84.36	84.43	93.11

Table 3.10: Maximum percent accuracy after n epochs when trained with input and output lengths of 40 and using all training data at once

We also found that when Restorer was trained exclusively with output lengths of 40, it generalized poorly to test cases with longer or shorter output sequences. In order to train models that would generalize to other output lengths, we found that changing the output length every 30 epochs to a random integer between 1 and 100 was an effective method.

Table 3.11 summarizes the average accuracy achieved by Restorer models when tested with different output lengths. In this scenario, models with more blocks generalized better than those with fewer blocks. It appears that when the length of the output sequence is fixed and known in advance, smaller models should be used, but when this length is either unknown or variable, it is better to use larger models and train in this stochastic way.

Model		**1 out acc**	**10 out acc**	**20 out acc**	**40 out acc**	**60 out acc**	**80 out acc**	**100 out acc**
Benchmark		76.55	45.41	37.01	30.07	25.70	22.99	21.10
Restorer	$\mathbf{N_{blocks}}$							
(main)	1	91.22	92.09	92.23	92.16	75.21	56.41	45.13
	2	**91.98**	**92.49**	**92.54**	92.44	78.65	58.98	47.18
	3	91.89	92.37	92.44	**92.58**	**92.56**	**92.56**	**88.41**
	5	91.66	91.99	91.94	92.08	92.21	92.31	86.96

Table 3.11: Accuracy (as a percentage) when using different output lengths. Training was performed using all data at once for 3000 epochs with input length of 40 and a new target output length randomly selected every 30 epochs

3.5 Case Study: Timed Regular Expression Mining on CAN Traces

This section was jointly written with Dr. Apurva Narayan. The TREM algorithm was previously proposed by him and his co-authors. He ran this algorithm on the 3 trace types I provided him with in order to get the results documented in this section.

It is important to verify that the quality of data restoration is not just high in terms of accuracy, but also in terms of the effect on end-stage algorithms that would make use of the restored data.

Time of occurrence of an event plays a key role in the domain of real-time systems [Lamport, 1978, Dwyer et al., 1999]. Recently, there has been tremendous activity in reverse engineering of complex real-time systems by mining temporal properties, typically by making use of template-based mining frameworks, that reflect the common behavior of these systems [Lemieux et al., 2015]. Since most programs in industry and elsewhere lack formal specifications, mined specifications play a key role in the life of software as they can be

used for tasks like testing [Dallmeier et al., 2010] or verification [Kincaid and Podelski, 2015].

State-machine based approaches have become quite popular in the domain of both active and passive learning of system specifications. The behavior of a system is learned in the form of a state machine that can be employed for numerous tasks such as run-time monitoring, run-time verification, debugging etc. In the context of real-time systems, the focus of these temporal specifications is inclined towards a quantitative (actual duration of time between events) notion of time rather than qualitative (ordering of events) notion of time. For example, the response of an interrupt handler should be bound within its predefined time constraints. These time constraints are critical to the safe operation of real-time systems since a delayed response can lead to a faulty system operation.

System traces used for mining temporal specifications for a system are quite often lossy due to various software and hardware issues such as a buffer overflow or a memory leak. Therefore, recovery of data will play a very important role in not only improving understanding of system behavior but also performing better fault diagnosis. Mining of Timed Regular Expressions (TREs) [Eugene Asarin, 2002] was further extended by Cutulenco et al. [2016], Narayan et al. [2018].The method proposed by Eugene Asarin [2002] to synthesize a timed automaton for a given TRE is the basis of the proposed approach. The timed automaton is then used as a checker to verify whether traces satisfy the corresponding TRE.

The mining algorithm is executed on three types of traces: a) Normal traces, b) Lossy traces, and c) Restored traces. The Normal traces in this case correspond to the original traces obtained from a real vehicle as described above. We do not have a large corpus of lossy CAN traces so instead lossy traces were generated from Normal traces by introducing random loss of events. We introduced loss of events at five levels: 5%, 10%, 15%, 20%, and 25%. Higher levels of loss are extremely unlikely to occur in a real vehicle aside from total failure of multiple components and as such are not included; nonetheless, exploratory experiments suggested that final results for higher levels of simulated loss were consistent with the pattern seen among these 5 selected levels. All losses were random to ensure that we are able to approximate a real-world scenario, as bugs, sensor defects, buffer overflows, cyberattacks, disconnects, and other similar events can all cause loss at any point within a trace. Lastly, the Restored traces were obtained from the Restorer main model with $N_{blocks} = 3$ that was trained on the Normal traces. We ensure that timestamps remain in order with the Normal traces since both time and order of occurrence of events is of key importance for specification in the form of TREs. Two TRE templates from Narayan et al. [2018] are used with a time interval parameter of 0 to 1,000.

T-1(response): $(\curlywedge(P)^*.(\langle P.\curlywedge(S)^*.S\rangle[0,1000]).\curlywedge(P)^*)+$

T-2(alternating): $(\curlywedge(P|S)^*.(\langle P.\curlywedge(P|S)^*.S.\hat{}(P|S)^*\rangle[0,1000]))+$

The mining framework extracts all rules from the system traces that take the form of the above two templates. A ranking module reduces the mined set of rules to a set of most commonly occurring TRE-instances in the system trace. We compared the presence and absence of the mined TRE instances from normal, lossy, and restored traces for evaluation of our restoring algorithms.

Trace Type	**5% loss**	**10% loss**	**15% loss**	**20% loss**	**25% loss**
Lossy Traces	6.2%	17.8%	21.6%	33.37%	53.99%
LSTM-Restored Traces	5.9%	8.2%	9.8%	12.6%	10.8%
Restorer-Restored Traces	1.16%	4.6%	6.55%	7.96%	9.53%

Table 3.12: Percentage deviation in total number of mined TRE instances in restored traces and lossy traces at each level of loss when compared to the number mined in normal traces

In Table 3.12, we present the percent of instances that were not found in the lossy and restored traces when compared with normal traces. Low deviation values are desirable as deviation is indicative of information loss. For this case study, we use the Restorer 'main' model with $N_{blocks} = 3$.

Clearly, performing restoration improves the mining performance greatly over just using lossy traces. When the traces are restored using either of our models, the number of TRE instances found is still reduced but much less so than when mining lossy traces, particularly when the message loss is high. For example, when 25% of the messages are lost, performing the LSTM restoration leads to a loss of only 10.8% of the TRE-instances as compared to 53.99% of the instances without restoration. Meanwhile, Restorer is even better at reducing the deviation in number of mined instances: from 6.2% to 1.16% in case of low levels of loss, and a remarkable reduction of almost 45% in the deviation of number of instances mined in cases with high levels of loss.

3.6 Conclusion

Restoring lost data is an important intermediate problem in machine learning as many algorithms rely on clean data for input. We have developed an LSTM-based approach for restoring lost or noisy data in discrete settings such as CAN traces. This approach can be

used to restore or predict arbitrarily large sequences of missing data, with output feedback used as input to predict further into the future. A major advantage of this approach is that few assumptions need to be made about the structure of the data, and so it can be applied in any setting where there is discrete data with some form of temporal or sequential dependence.

We have also demonstrated that models making use of attention mechanisms based on the recently proposed Transformer model are faster, smaller, and achieve better accuracy than the LSTM restoration model. We have shown that the exact architecture can be tailored to the task at hand based on the associated constraints. For example, for compute-constrained or time-constrained tasks, a smaller architecture can be chosen to minimize training or inference times and number of parameters. Meanwhile, for tasks where the length of the output sequence is variable or unknown beforehand, a larger architecture with more blocks achieves better accuracy.

There are several limitations of this study that merit discussion. First off, we avoided using bidirectional methods because they would have a large impact on the latency of algorithms running downstream from the LSTM model or Restorer when working with streams of data. However, in order to improve accuracy, it may be useful to develop a mechanism for using newly processed steps of the input sequence to retroactively correct predictions made by the model. Second, while we have tested our models on real data, it will be important to conduct additional studies with other datasets to further establish them as an effective solution for data restoration. Finally, we have shown that Transformer-based models can be used for even more sequence modelling tasks than the already wide-range described in Vaswani et al. [2017].

In general, we have shown that Restorer can achieve accuracies of up to 95% when restoring long sequences of missing CAN data, beating out the benchmark method by a wide margin. When comparing Restorer to LSTM-based methods for restoring CAN data, we demonstrated that using Restorer leads to a reduction in model sizes by up to 2 orders of magnitude, an up to 4-fold reduction in training times, and an increase of up to 7% in accuracy of data restoration. We additionally demonstrated in a case study that our method successfully improves the performance of complex downstream algorithms.

Chapter 4

Dataset Distillation

4.1 Introduction

Deep learning is computationally intensive and this poses several issues: high energy consumption [Strubell et al., 2019], high financial cost, and long training times. This is particularly problematic for settings like federated learning where edge devices are computationally constrained and may not be able to work with large datasets [Li et al., 2020b]. One path for mitigating these issues is to reduce network sizes. Hinton et al. [2015] proposed knowledge distillation as a method for imbuing smaller, more efficient networks with all the knowledge of their larger counterparts. Instead of decreasing network size, a second path to efficiency may be to decrease dataset size. Dataset distillation (DD) has recently been proposed as an alternative formulation of knowledge distillation to do exactly that [Wang et al., 2018].

Dataset distillation is the process of creating a small number of synthetic samples that can quickly train a network to the same accuracy it would achieve if trained on the original dataset. It may seem counter-intuitive that training a model on a small number of synthetic images coming from a different distribution than the training data can result in comparable accuracy, but Wang et al. [2018] have shown that for models with known initializations this is indeed feasible; they achieve 94% accuracy on MNIST, a hand-written digit recognition task [LeCun et al., 1998], after training LeNet on just 10 synthetic images.

We propose to improve their already impressive results by learning 'soft' labels as a part of the distillation process. The original DD algorithm uses fixed, or 'hard', labels for the synthetic samples (e.g. the ten synthetic MNIST images each have a label corresponding

$$\begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix} \xleftarrow[\text{and rest to 0}]{\text{Set largest to 1}} \begin{bmatrix}0.01\\0.69\\0.02\\0.02\\0.03\\0.05\\0.03\\0.1\\0.01\\0.04\end{bmatrix} \xleftarrow[\text{softmax}]{\text{Apply}} \begin{bmatrix}0.8\\5.1\\1.5\\1.5\\2\\2.5\\2\\3.2\\0.8\\2\end{bmatrix}$$

Figure 4.1: **Left**: An example of a 'hard' label where the second class is selected. **Center**: An example of a 'soft' label restricted to being a valid probability distribution. The second class has the highest probability. **Right**: An example of an unrestricted 'soft' label. The second class has the highest weight. 'Hard' labels can be derived from unrestricted 'soft' labels by applying the softmax function and then setting the highest probability element to 1, and the rest to 0.

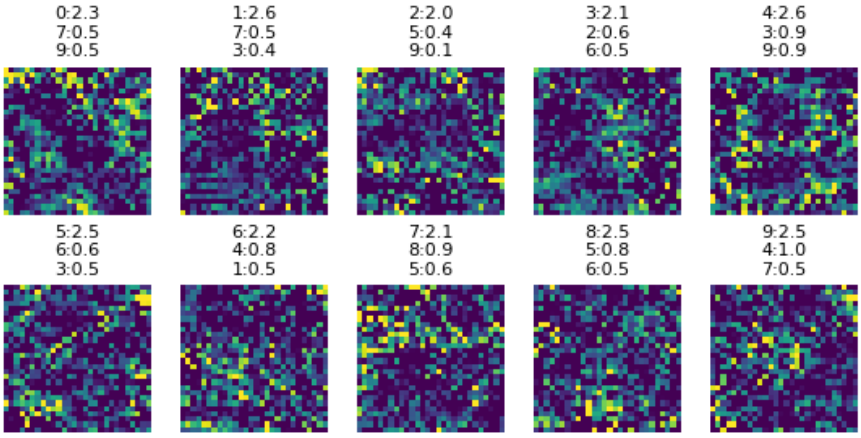


Figure 4.2: 10 MNIST images learned by SLDD can train networks with fixed initializations from 11.13% distillation accuracy to 96.13% ($r_{10} = 97.1$). Each image is labeled with its top 3 classes and their associated logits. The full labels for these 10 images are in Table 4.1.

Table 4.1: Learned distilled labels for the 10 distilled MNIST images in Figure 4.2. Distilled labels are allowed to take on any real value. If a probability distribution is needed, a softmax function can be applied to each row.

	Digit									
Distilled Label	0	1	2	3	4	5	6	7	8	9
1	2.34	-0.33	0.23	0.04	-0.03	-0.23	-0.32	0.54	-0.39	0.49
2	-0.17	2.58	0.32	0.37	-0.68	-0.19	-0.75	0.53	0.27	-0.89
3	-0.26	-0.35	2.00	0.07	0.08	0.42	0.02	-0.08	-1.09	0.10
4	-0.28	0.04	0.59	2.08	-0.61	-1.11	0.52	0.19	-0.20	0.32
5	-0.11	-0.52	-0.08	0.90	2.63	-0.44	-0.72	-0.39	-0.29	0.87
6	0.25	-0.20	-0.19	0.51	-0.02	2.47	0.62	-0.42	-0.52	-0.63
7	0.42	0.55	-0.09	-1.07	0.83	-0.19	2.16	-0.30	0.26	-0.91
8	0.18	-0.33	-0.25	0.06	-0.91	0.55	-1.17	2.11	0.94	0.47
9	0.46	-0.48	0.24	0.09	-0.78	0.75	0.47	-0.40	2.45	-0.71
10	-0.53	0.52	-0.74	-1.32	1.03	0.23	0.05	0.55	0.31	2.45

to a different digit). In other words, each label is a one-hot vector: a vector where all entries are set to zero aside from a single entry, corresponding to the correct class, which is set to one. We relax this one-hot restriction and make the distribution learnable for these synthetic labels. The resulting distilled labels are thus similar to those used for knowledge distillation as a single image can now correspond to multiple classes. An example comparing a 'hard' label to a 'soft' label is shown in Figure 4.1. A 'hard' label can be derived from a 'soft' label by applying the softmax function and setting the element with the highest probability to one, while the remaining elements are set to zero. Our soft-label dataset distillation (SLDD) not only achieves over 96% accuracy on MNIST when using ten distilled images (as seen in Figure 4.2), a 2% increase over the state-of-the-art (SOTA), but also achieves almost 92% accuracy with just five distilled images, which is less than one image per class. In addition to soft labels, we also extend dataset distillation to the natural language/sequence modeling domain and enable it to be used with several additional network architectures. For example, we show that Text Dataset Distillation (TDD) can train a custom convolutional neural network (CNN) [LeCun et al., 1999] with known initialization up to 90% of its original accuracy on the IMDB sentiment classification task [Maas et al., 2011] using just two synthetic sentences.

The remainder of this chapter is divided into four sections. In Section 4.2, we discuss related work in the fields of knowledge distillation, dataset reduction, and example generation. In Section 4.3, we propose improvements and extensions to dataset distillation and

associated theory. In Section 4.4, we empirically validate SLDD and TDD in a wide range of experiments. Finally, in Section 4.5, we discuss the significance of SLDD and TDD, and our outlook for the future.

4.2 Related Work

4.2.1 Knowledge Distillation

Dataset distillation was originally inspired by network distillation [Hinton et al., 2015] which is a form of knowledge distillation or model compression [Buciluǎ et al., 2006] that has been studied in various contexts including when working with sequential data [Kim and Rush, 2016]. Network distillation aims to distill the knowledge of large, or even multiple, networks into a smaller network. Similarly, dataset distillation aims to distill the knowledge of large, or even multiple, datasets into a small number of synthetic samples. 'Soft' labels were recently proposed as an effective way of distilling networks by feeding the output probabilities of a larger network directly to a smaller network [Hinton et al., 2015], and have previously been studied in the context of different machine learning algorithms [El Gayar et al., 2006]. Our soft-label dataset distillation (SLDD) algorithm also uses 'soft' labels but these are persistent and learned over the training phase of a network (rather than being produced during the inference phase as in the case of network distillation).

4.2.2 Sample Efficiency in Deep Learning

Deep learning models are notoriously data-hungry. In their landmark paper introducing GPT-3, Brown et al. [2020] write that a 'limitation broadly shared by language models is poor sample efficiency during pre-training'. Sample efficiency is also a widely discussed problem in reinforcement learning [Munos et al., 2016, Nachum et al., 2018, Buckman et al., 2018]. Deep supervised learning also generally requires a very large number of examples to train on. For example, MNIST and CIFAR10 both contain thousands of training images per class.

Meanwhile, it appears that humans can quickly generalize from a tiny number of examples [Lake et al., 2015]. Getting machines to learn from 'small' data is an important aspect of trying to bridge this gap in abilities. Few-shot learning aims to improve deep learning sample efficiency to the point where models can learn a new class given only a few examples [Lake et al., 2015, Ravi and Larochelle, 2017, Snell et al., 2017].

Dataset distillation allows us to probe the limits of a given model's sample efficiency by creating synthetic samples that are tailored specifically for efficiently training this exact model. Studying the distilled images produced by dataset distillation may enable us to identify what allows neural networks to generalize so quickly from so few of them. In some sense, dataset distillation can be thought of as an algorithm for creating dataset summaries that machines can learn from.

4.2.3 Dataset Reduction and Prototype Methods

Generally, when discussing dataset reduction, we broadly refer to any information re-packaging technique that aims to reduce the space taken up by a dataset. If we think of a dataset as a matrix where the width is the number of features and the length is the number of observations, then the space taken up by the dataset can be reduced along either of these axes. Throughout this thesis we focus specifically on reducing the length of datasets.

There are many methods that aim to reduce the size of a dataset with varying objectives. Active learning aims to reduce the required size of the labeled portion of a dataset by only labeling examples that are determined to be the most important [Cohn et al., 1996, Tong and Koller, 2001]. Several methods aim to 'prune' a dataset, or create a 'core-set', by leaving in only examples that are determined to be useful [Angelova et al., 2005, Tsang et al., 2005, Bachem et al., 2017, Sener and Savarese, 2017]. In the context of nearest-neighbor classification, prototype selection (PS) and prototype generation (PG) are studied extensively as methods of reducing storage requirements and improving the efficiency of nearest-neighbor classification [Triguero et al., 2011a, Garcia et al., 2012]. In general, all of these methods aside from PG, use samples from the true distribution, typically subsets of the original training set. By lifting this restriction and, instead, learning synthetic samples, DD requires far fewer samples to distill the same amount of knowledge. PG methods typically create samples that are not found in the training data; however, these methods are designed specifically for use with nearest-neighbor classification algorithms. In addition, most of the methods listed above use fixed labels. SLDD removes this restriction and allows the label distribution to be optimized simultaneously with the samples (or prototypes) themselves.

4.2.4 Federated Learning and Privacy Preservation

DD-like algorithms are naturally well-suited for federated learning: they provide the ability to transmit small, synthetic, privacy-preserving training datasets while still maintaining

high performance of the networks trained on them. Increased initial overhead is a small price to pay for smaller transmission sizes, faster training times, and privacy preservation. We initially released the code for SLDD and TDD in 2019. Since then, there has been significant work building on our research that demonstrates the applications of SLDD to federated learning [Goetz and Tewari, 2020, Zhou et al., 2020, Sucholutsky and Schonlau, 2021c] as well as to the transmission of private data [Sucholutsky and Schonlau, 2021c, Li et al., 2020a]. There has also been an increasing amount of literature discussing using the original DD algorithm in the context of federated learning [Kairouz et al., 2019, Song et al., 2020, Shen et al., 2020, Amiri et al., 2020]. We discuss this further in Appendix C.

4.2.5 Generative Adversarial Networks

Generative Adversarial Networks (GANs) have recently become a very widely used method for image generation and are primarily used to produce images that closely mimic those coming from the true distribution [Ledig et al., 2017, Goodfellow et al., 2014, Choi et al., 2018, Radford et al., 2015]. With dataset distillation we instead set knowledge distillation as the objective but do not attempt to produce samples from the true distribution. Using the generator from a trained GAN may be a much faster way of producing images than the gradient-based method employed by dataset distillation. However, since the number of distilled images we aim to produce is very small, solving the objective directly through gradient-based optimization is sufficiently fast, while also more straightforward. Additionally, while some GANs can work with text [Reed et al., 2016, Yu et al., 2017], they are primarily intended for image generation.

4.2.6 Measuring Problem Dimensionality

We may intuitively believe that one deep learning task is more difficult than another. For example, when comparing the digit recognition task MNIST, to the image classification task CIFAR10 [Krizhevsky et al., 2009], it seems that CIFAR is the tougher problem, but it is hard to determine to what extent it is tougher. It is possible to try to quantify what exactly it means for one problem to be more difficult than other. One approach is to compare state-of-the-art (SOTA) results on datasets. For example, the near-SOTA 'dropconnect' model on MNIST achieves a 0.21% error rate, while on CIFAR10 it achieves an error rate of 9.32% [Wan et al., 2013]. However, this approach reveals increasingly little as deeper networks approach perfect accuracy on multiple tasks. Li et al. [2018] instead derive a more model-independent metric for comparing the dimensionality of various problems

based on the minimum number of learnable parameters needed to achieve a good local optimum. Similarly, dataset distillation aims to find the minimum number of synthetic samples needed to achieve a good local optimum. The difference is that Li et al. [2018] constrain the number of searchable dimensions within the network weight space, while dataset distillation constrains them within the data space.

4.3 Extending Dataset Distillation

4.3.1 Motivation

As mentioned above, nearest-neighbors classification often involves accuracy-preserving data reduction techniques known as prototype selection (PS) and prototype generation (PG). We can use these concepts, along with the k-Nearest Neighbors (kNN) classification algorithm, to visualize the difference between classical dataset reduction methods, DD, and SLDD. When we fit a kNN model, we essentially divide the entire space into classes based on the location of points in the training set. However, the cost of fitting a kNN model increases with the number of training points.

PS methods use subsets of the training set to construct a reduced training set. In the first column of Figure 4.3, we visualize reduction of the training set for a three-class problem, the Iris flower dataset [Fisher, 1936], by selecting one point from each class and then fitting the kNN on these three selected points. Only being able to use a subset of the original points limits the ability to finely tune the shape of the resulting partitions. PG methods create synthetic points whose placement can be optimized to improve kNN performance. DD for neural networks works analogously. In the second column of Figure 4.3, we generate one point for each class and optimize the location of one of them. Using synthetic points allows for better control of the resulting landscape.

In the third column of Figure 4.3, we propose that the points have optimizable distributions of labels. In order to visualize the effect of changing a point's label distribution, we create one point per class, but then change the label distribution of one of these points, increasingly making it a mixture of the other two classes. In the final column of Figure 4.3, we combine the PG method with our soft-label modification, to visualize the effect of simultaneously changing a point's location and label distribution. This last case is the kNN counterpart to our proposed SLDD algorithm. The visualization illustrates that this method provides the most control over the resulting landscape. In fact, Figure 4.4 shows that we can even separate three classes using just two points when using soft labels with the kNN classifier.

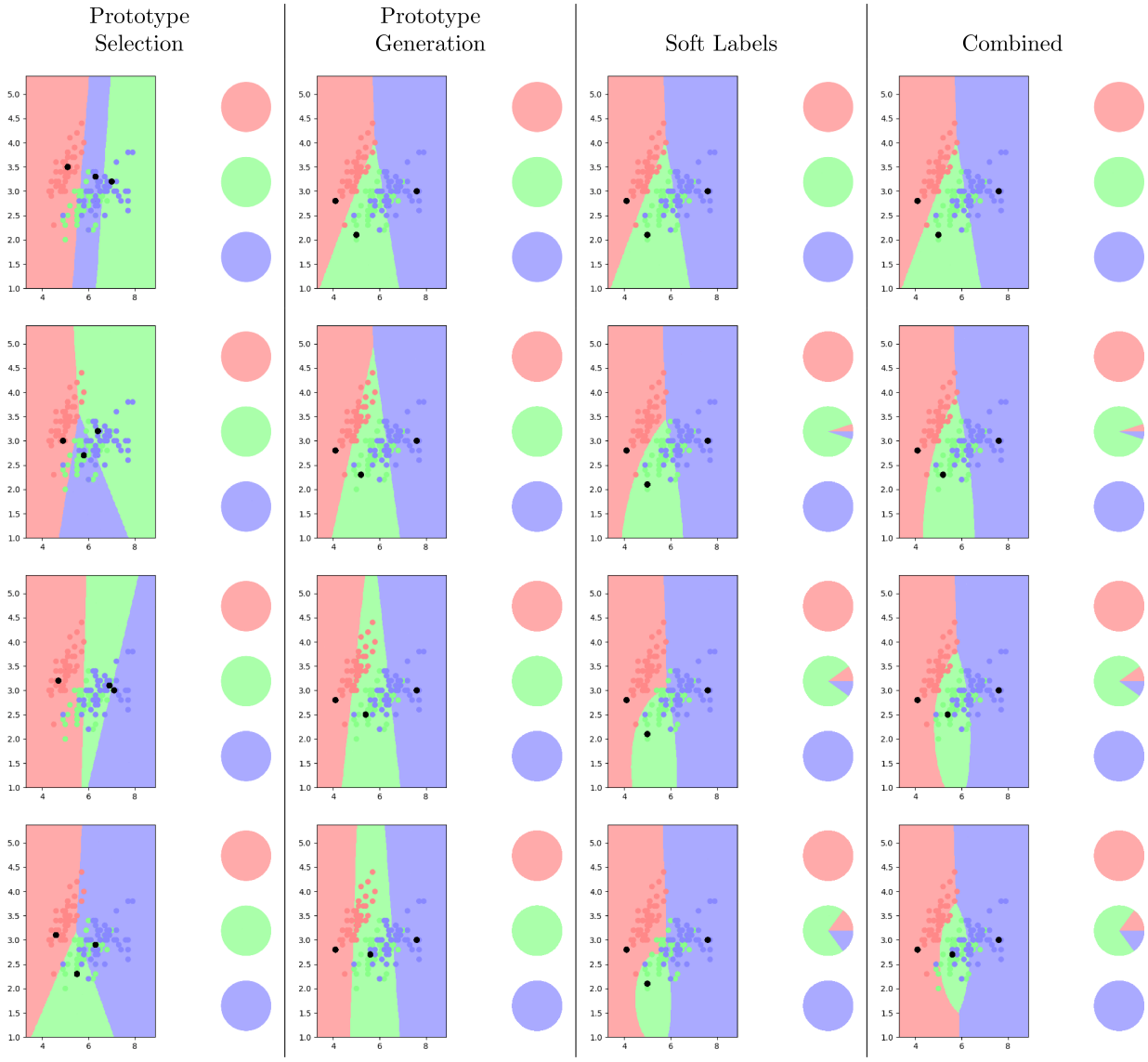


Figure 4.3: kNN models are fitted on 3 points obtained using four methods: PS, PG, soft labels, and PG combined with soft labels. Each column contains 4 steps of the associated method used to update the 3 points. The pie charts represent the label distributions assigned to each of the 3 points. **PS:** A different random point from each class is chosen to represent its class in each of the steps. **PG:** The model can select and adjust synthetic points to represent each class. In this case, the middle point associated with the 'green' label is moved diagonally in each step. **Soft Labels:** The label distribution of the middle point is changed each step to contain a larger proportion of both other classes. **Combined:** The middle point is simultaneously moved and has its label distribution updated in each step.

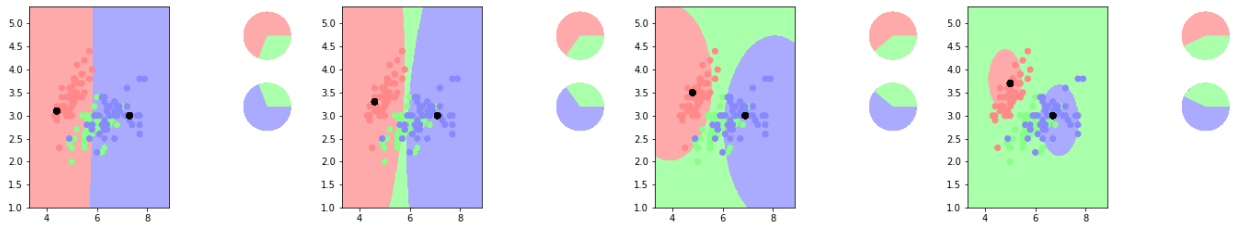


Figure 4.4: A kNN model fitted on 2 points obtained using the 'Combined' method. The pie charts represent the label distributions assigned to each of the 2 points. From the left plot to the right plot, the locations of the 2 points slightly shift and the 'green' portions of their label distributions are increased. By modifying the soft labels of the 2 points, the space can still be separated into 3 classes.

4.3.2 Basic Approach

In this section, we summarize the approach by Wang et al. [2018] in a slightly modified way to explicitly show the labels of the distilled dataset. In the next section, we expand this approach to work with soft labels.

Given a training dataset $\mathbf{d} = \{x_i, y_i\}_{i=1}^{N}$, a neural network with parameters θ, and a twice-differentiable loss function $\ell\left(x_i, y_i, \theta\right)$, our objective is to find

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \ell\left(x_i, y_i, \theta\right) \triangleq \arg\min_{\theta} \ell(\mathbf{x}, \mathbf{y}, \theta) \ . \tag{4.1}$$

In general, training with involves repeatedly sampling mini-batches of training data and updating network parameters by their error gradient scaled by learning rate η.

$$\theta_{t+1} = \theta_t - \eta \nabla_{\theta_t} \ell\left(\mathbf{x}_t, \mathbf{y}_t, \theta_t\right) \tag{4.2}$$

With DD, the goal is to perform just one such step while still achieving the same accuracy. We do this by learning a small number of synthetic samples $\tilde{\mathbf{x}}$ that minimize $\mathcal{L}$, a one-step loss objective, for $\theta_1 = \theta_0 - \tilde{\eta} \nabla_{\theta_0} \ell\left(\tilde{\mathbf{x}}, \theta_0\right)$.

$$\mathcal{L}\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\eta}; \theta_0\right) := \ell\left(\mathbf{x}, \mathbf{y}, \theta_0 - \tilde{\eta} \nabla_{\theta_0} \ell\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \theta_0\right)\right) \tag{4.3}$$

$$\tilde{\mathbf{x}}^*, \tilde{\eta}^* = \arg\min_{\tilde{\mathbf{x}}, \tilde{\eta}} \mathcal{L}\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\eta}; \theta_0\right) \tag{4.4}$$

We minimize this objective, or 'learn the distilled samples', by using SGD. Here we are optimizing over $\tilde{\mathbf{x}}$ and $\tilde{\eta}$, but not $\tilde{\mathbf{y}}$, as the labels are fixed for the original DD algorithm.

4.3.3 Learnable Labels

Soft labels exploit the fact that each training image contains information about more than one class (e.g. the digit '3' looks a lot like a '3' but also like an '8'). Using soft labels allows us to convey more information about each image.

The original dataset distillation algorithm was restricted to 'hard' labels for the distilled data; each distilled image has to be associated with just a single class. We relax this restriction and allow distilled labels to take on any real value. Since the distilled labels are now continuous variables, we can modify the distillation algorithm in order to make the distilled labels learnable using the same method as for the distilled images: a combination of backpropagation and gradient descent. With our modified notation, we simply need to change Equation 4.4 to also minimize over $\tilde{\mathbf{y}}$.

$$\begin{aligned}\tilde{\mathbf{x}}^*, \tilde{\mathbf{y}}^*, \tilde{\eta}^* &= \arg\min_{\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\eta}} \mathcal{L}\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\eta}; \theta_0\right) \\ &= \arg\min_{\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\eta}} \ell\left(\mathbf{x}, \mathbf{y}, \theta_0 - \tilde{\eta} \nabla_{\theta_0} \ell\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \theta_0\right)\right)\end{aligned}$$

In our experiments, we generally initialize $\tilde{y}$ with the one-hot values that 'hard' labels would have. We found that this tends to increase accuracy when compared to random initialization, perhaps because it encourages more differentiation between classes early on in the distillation process.

Algorithm 2 details this soft-label dataset distillation (SLDD) algorithm.

4.3.4 Text and Other Sequences

It is difficult to use gradient methods directly on text data as they are discrete. In order to use SLDD with text data, we first embed the text data into a continuous space using pre-trained GloVe embeddings [Pennington et al., 2014]. This is a common practice when working with many modern natural language processing models, though the embedding method itself can vary greatly [Ma and Hovy, 2016, Devlin et al., 2019, Peters et al., 2018]. Distilling embedded text is analogous to image distillation. If all sentences are padded/truncated to some pre-determined length, then each sentence can be viewed as a one-channel image of size [length]∗[embedding dimension], though this is not required when working with RNNs. Only sentences coming from the true dataset need to be embedded; the distilled samples are learned directly as embedded representations. As a result, finding the nearest sentences that correspond to the distilled embeddings may help

Algorithm 2: Soft-Label Dataset Distillation (SLDD)

Input: $p(\theta_0)$: distribution of initial weights; M: the number of distilled data; α: step size; n: batch size; T: number of optimization iterations; $\tilde{y}_0$: initial value for $\tilde{y}$; $\tilde{\eta}_0$: initial value for $\tilde{\eta}$

Initialize distilled data
$\tilde{\mathbf{x}} = \{\tilde{x}_i\}_{i=1}^{M}$ randomly,
$\tilde{\mathbf{y}} = \{\tilde{y}_i\}_{i=1}^{M} \leftarrow \tilde{y}_0$,
$\tilde{\eta} \leftarrow \tilde{\eta}_0$
for each training step t = 1 to T **do**
  Get a minibatch of real training data
  $(\mathbf{x}_t, \mathbf{y}_t) = \{x_{t,j}, y_{t,j}\}_{j=1}^{n}$
  One-hot encode the labels
  $(\mathbf{x}_t, \mathbf{y^*}_t) = \{x_{t,j}, \text{Encode}(y_{t,j})\}_{j=1}^{n}$
  Sample a batch of initial weights
  $\theta_0^{(t)} \sim p(\theta_0)$
  for each sampled $\theta_{0,i}^{(t)}$ **do**
    Compute updated model parameter with GD
    $\theta_{1,i}^{(t)} = \theta_{0,i}^{(t)} - \tilde{\eta} \nabla_{\theta_{0,i}^{(t)}} \ell\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \theta_{0,i}^{(t)}\right)$
    Evaluate the objective function on real training data: $\mathcal{L}_i^{(t)} = \ell\left(\mathbf{x}_t, \mathbf{y^*}_t, \theta_{1,i}^{(t)}\right)$
  end for
  Update distilled data
  $\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} - \alpha \nabla_{\tilde{\mathbf{x}}} \sum_j \mathcal{L}^{(j)}$,
  $\tilde{\mathbf{y}} \leftarrow \tilde{\mathbf{y}} - \alpha \nabla_{\tilde{\mathbf{y}}} \sum_j \mathcal{L}^{(j)}$, and
  $\tilde{\eta} \leftarrow \tilde{\eta} - \alpha \nabla_{\tilde{\eta}} \sum_j \mathcal{L}^{(j)}$
end for

Output: distilled data $\tilde{\mathbf{x}}$; distilled labels $\tilde{\mathbf{y}}$; optimized learning rate $\tilde{\eta}$

with interpretability. To compute the nearest sentence to a distilled embedding matrix, for every column vector in the matrix, the nearest embedding vector from the original dictionary must be found. These embedding vectors must then be converted back into their corresponding words, and those words joined into a sentence. To differentiate this modified procedure from SLDD, we refer to it as Text Dataset Distillation (TDD). The resulting algorithm for text dataset distillation (TDD) is detailed in Algorithm 3, which is a modification of the SLDD Algorithm 2.

4.3.5 Random initializations and multiple steps

The procedures we described above make one important assumption: network initialization θ_0 is fixed. The samples created this way do not lead to high accuracy when the network is re-trained on them with a different initialization as they contain information not only about the dataset but also about θ_0. In Figures 4.2 and 4.5, this can be seen as what looks like a lot of random noise. Wang et al. [2018] propose a generalization that works with network initializations randomly sampled from some restricted distribution.

$$\tilde{\mathbf{x}}^*, \tilde{\mathbf{y}}^*, \tilde{\eta}^* = \arg\min_{\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\eta}} \mathbb{E}_{\theta_0 \sim p(\theta_0)} \mathcal{L}\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \tilde{\eta}; \theta_0\right) \tag{4.5}$$

The resulting images, especially for MNIST, appear to have clearer patterns and less random noise. Our experimental results suggest that this method generalizes fairly well to other randomly sampled initializations from the same distribution.

Additionally, Wang et al. [2018] suggest that the above methods can work with multiple gradient descent (GD) steps and with multiple epochs over the distilled data. The experimental results suggest that multiple steps and multiple epochs improve distillation performance for both image and text data, particularly when using random network initializations.

4.4 Experiments

4.4.1 Metrics

The simplest metric for gauging distillation performance is to train a model on distilled samples and then test it on real samples. We refer to the accuracy achieved on real samples as the 'distillation accuracy'. However, several of the models we use in our experiments

Algorithm 3: Text Dataset Distillation (TDD)

Input: $p(\theta_0)$: distribution of initial weights; M: the number of distilled data; α: step size; n: batch size; T: number of optimization iterations; $\tilde{y}_0$: initial value for $\tilde{y}$; $\tilde{\eta}_0$: initial value for $\tilde{\eta}$; s: sentence length; d: embedding size

Initialize distilled data
$\tilde{\mathbf{x}} = \{\tilde{x}_i\}_{i=1}^{M}$ randomly of size $s \times d$,
$\tilde{\mathbf{y}} = \{\tilde{y}_i\}_{i=1}^{M} \leftarrow \tilde{y}_0$,
$\tilde{\eta} \leftarrow \tilde{\eta}_0$
for each training step t = 1 to T **do**
  Get a minibatch of real training data
  $(\mathbf{x}_t, \mathbf{y}_t) = \{x_{t,j}, y_{t,j}\}_{j=1}^{n}$
  Pad (or truncate) each sentence in the minibatch
  $(\mathbf{x^p}_t, \mathbf{y}_t) = \{\text{Pad}(x_{t,j}, \text{len} = s), y_{t,j}\}_{j=1}^{n}$
  Embed each sentence in the minibatch
  $(\mathbf{x}^*{}_t, \mathbf{y}_t) = \left\{\text{Embed}(x^p_{t,j}, \dim = d), y_{t,j}\right\}_{j=1}^{n}$
  One-hot encode the labels
  $(\mathbf{x}^*{}_t, \mathbf{y}^*{}_t) = \left\{x^*_{t,j}, \text{Encode}(y_{t,j})\right\}_{j=1}^{n}$
  Sample a batch of initial weights
  $\theta_0^{(t)} \sim p\left(\theta_0\right)$
  for each sampled $\theta_{0,i}^{(t)}$ **do**
    Compute updated model parameter with GD
    $\theta_{1,i}^{(t)} = \theta_{0,i}^{(t)} - \tilde{\eta} \nabla_{\theta_{0,i}^{(t)}} \ell\left(\tilde{\mathbf{x}}, \tilde{\mathbf{y}}, \theta_{0,i}^{(t)}\right)$
    Evaluate the objective function on real training data: $\mathcal{L}_i^{(t)} = \ell\left(\mathbf{x}_t, \mathbf{y}^*{}_t, \theta_{1,i}^{(t)}\right)$
  end for
  Update distilled data
  $\tilde{\mathbf{x}} \leftarrow \tilde{\mathbf{x}} - \alpha \nabla_{\tilde{\mathbf{x}}} \sum_j \mathcal{L}^{(j)}$,
  $\tilde{\mathbf{y}} \leftarrow \tilde{\mathbf{y}} - \alpha \nabla_{\tilde{\mathbf{y}}} \sum_j \mathcal{L}^{(j)}$, and
  $\tilde{\eta} \leftarrow \tilde{\eta} - \alpha \nabla_{\tilde{\eta}} \sum_j \mathcal{L}^{(j)}$
end for

Output: distilled data $\tilde{\mathbf{x}}$; distilled labels $\tilde{\mathbf{y}}$; optimized learning rate $\tilde{\eta}$

do not achieve SOTA accuracy on the datasets they are paired with, so it is useful to construct a relative metric that compares distillation accuracy to original accuracy. We define 'distillation ratio' as the ratio of distillation accuracy to original accuracy. The distillation ratio is heavily dependent on the number of distilled samples so the notation we use is $r_M = 100\% * \frac{\text{[distillation accuracy]}}{\text{[original accuracy]}}, M = \text{[number of distilled samples]}$. We may refer to this as the 'M-sample distillation ratio' when clarification is needed. It may also be of interest to find the minimum number of distilled images required to achieve a certain distillation ratio. We call this the '$A\%$ distillation size', and we write $d_A = M$ where M is the minimum number of distilled samples required to achieve a distillation ratio of $A\%$.

4.4.2 Image Data

While LeNet achieves 99% accuracy on MNIST, AlexCifarNet only achieves 80% on CIFAR10 so it is helpful to use the relative metrics when describing this set of results.
Baselines. We use the same baselines as Wang et al. [2018].
Random real images: We randomly sample the same number of real images per class from the training data. These images are used for two baselines: training neural networks and training K-Nearest Neighbors classifiers.
Optimized real images: We sample several sets of random real images as above, but now we choose the 20% of these sets that have the best performance on training data. These images are used for one baseline: training neural networks.
k-means: We use k-means to find centroid images for each class. These images are used for two baselines: training neural networks and training K-Nearest Neighbors classifiers.
Average real images: We compute the average image for each class and use it for training. These images are used for one baseline: training neural networks.

Each of these baseline methods produces a small set of images that can be used to train models. All four baseline methods are used to train and test LeNet and AlexCifarNet on their respective datasets. Two of the baseline methods are used to also train K-Nearest Neighbor classifiers to compare performance against neural networks. The results for these six baselines, as determined by Wang et al. [2018], are shown in Table 4.2.
Fixed initialization. When the network initialization is fixed between the distillation and training phases, synthetic images produced by dataset distillation result in high distillation accuracies. The SLDD algorithm produces images that result in equal or higher accuracies when compared to the original DD algorithm. For example, DD can produce 10 distilled images that train a LeNet model up to 93.76% accuracy on MNIST [Wang et al., 2018]. Meanwhile, SLDD can produce 10 distilled images that train the same model up to 96.13% accuracy (Figure 4.2). SLDD can even produce a tiny set of just 5 distilled images that

train LeNet to 91.56% accuracy. As can be seen in Figure 4.6, the 90% distillation size (i.e. the minimum number of images needed to achieve 90% of the original accuracy) of MNIST with fixed initializations is $d_A = 5$, and while adding more distilled images typically increases distillation accuracy, this begins to plateau after five images. Similarly, SLDD provides a 7.5% increase in 100-sample distillation ratio (6% increase in distillation accuracy) on CIFAR10 over DD. Based on these results detailed in Table 4.2, it appears that SLDD is more effective than DD at distilling image data into a small number of samples.

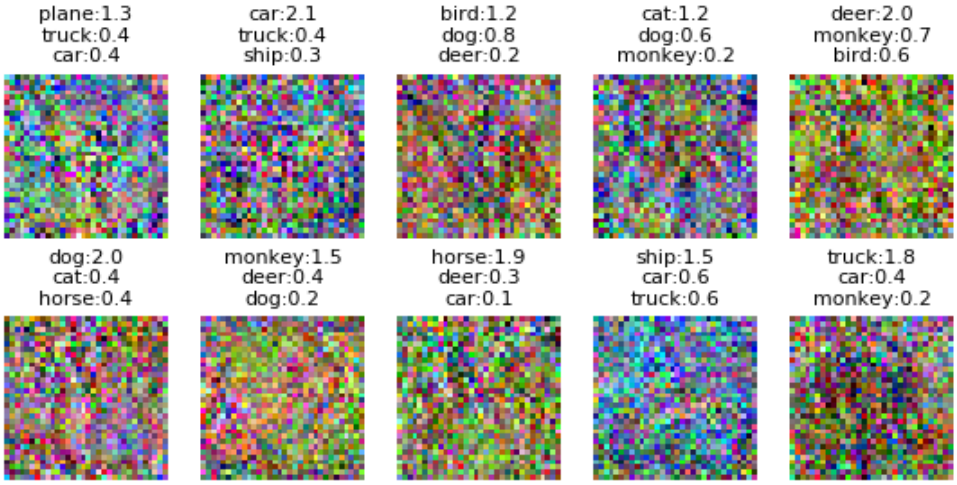


Figure 4.5: SLDD can learn 100 distilled CIFAR10 images (10 steps with 10 images each) that train networks with fixed initializations from 12.9% distillation accuracy to 60.0% ($r_{100} = 75.0$). Each image is labeled with its top 3 classes and their logits. Only the last step is shown.

Random initialization. It is also of interest to know whether distilled data are robust to network initialization. Specifically, we aim to identify if distilled samples store information only about the network initializations, or whether they can store information contained within the training data. To this end, we perform experiments by sampling random network initializations generated using the Xavier Initialization [Glorot and Bengio, 2010]. The distilled images produced in this way are more representative of the training data but generally result in lower accuracies when models are trained on them. On both datasets, distilled images produced by SLDD lead to higher distillation accuracies than those from

DD when the number of distilled images is held constant. These results are detailed in Table 4.2. It is also interesting to note that the actual distilled images, as seen in Figures 4.7 and 4.8, appear to have clearer patterns emerging than in the fixed initialization case. These results suggest that DD, and even more so SLDD, can be generalized to work with random initializations and distill knowledge about the dataset itself when they are trained this way.

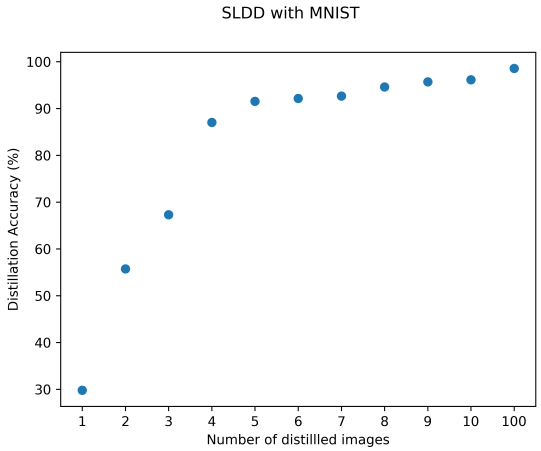


Figure 4.6: Distilled dataset size and MNIST accuracy

4.4.3 Text Data

We use the IMDB sentiment dataset, the Stanford Sentiment Treebank 5-class task (SST5) [Socher et al., 2013], and the Text Retrieval Conference question classification tasks with 6 (TREC6) and 50 (TREC50) classes [Voorhees et al., 1999]. Text experiments are performed with three architectures: a shallow but wide CNN (TextConvNet), a bi-directional RNN (Bi-RNN) [Schuster and Paliwal, 1997], and a bi-directional Long Short-Term Memory network (Bi-LSTM) [Hochreiter and Schmidhuber, 1997]. The accuracies on full datasets are detailed in Table 4.4 and distillation ratios in this section are calculated based on them.

Baselines. We consider the same six baselines as for image data but modify them slightly so that they work with text data (e.g. padding/truncating the sampled sentences). Each of the baseline methods produces a small set of sentences, or sentence embeddings, that can be used to train models. All four of the baseline methods are used to train and test our TextConvNet on each of the text datasets. Additionally, two of the baseline methods are

	SLDD accuracy		DD accuracy		Used as training data in same # of GD steps				Used in K-NN	
	Fix	Rand	Fix	Rand	Rand real	Optim real	k-means	Avg real	Rand real	k-means
MNIST	**98.6**	82.7 ± 2.8	96.6	79.5 ± 8.1	68.6 ± 9.8	73.0 ± 7.6	76.4 ± 9.5	77.1 ± 2.7	71.5 ± 2.1	**92.2 ± 0.1**
CIFAR	**60.0**	**39.8 ± 0.8**	54.0	36.8 ± 1.2	21.3 ± 1.5	23.4 ± 1.3	22.5 ± 3.1	22.3 ± 0.3	18.8 ± 1.3	29.4 ± 0.3

Table 4.2: Means and standard deviations of SLDD, DD, and baseline accuracies (as detailed by Wang et al. [2018]) on MNIST and CIFAR10 datasets. The first four baselines produce reduced datasets that are used to train the same neural network as in the distillation experiments. The last two baselines produce reduced datasets that are used to train a K-NN classifier. Experiments with random initializations have their results listed in the form [mean ± standard deviation] and are based on the resulting performance of 200 randomly initialized networks.

	TDD accuracy		Used as training data in 10 GD steps				Used in K-NN	
	Fixed	Random	Rand. real	Optim. real	k-means	Avg. real	Rand. real	k-means
IMDB	**75.0**	**73.4 ± 3.3**	49.7 ± 0.9	49.9 ± 0.8	49.9 ± 0.6	50.0 ± 0.1	50.0 ± 0.1	50.0 ± 0.0
SST5	**37.5**	**36.3 ± 1.5**	21.2 ± 4.9	24.6 ± 2.6	19.6 ± 4.5	21.3 ± 4.1	23.1 ± 0.0	20.9 ± 2.1
TREC6	**79.2**	**77.3 ± 2.9**	37.5 ± 10.1	44.6 ± 7.5	34.4 ± 13.0	28.0 ± 9.5	31.5 ± 9.9	50.5 ± 6.8
TREC50	67.4	42.1 ± 2.1	8.2 ± 6.0	9.9 ± 6.6	14.7 ± 5.5	12.5 ± 6.4	15.4 ± 5.1	**45.1 ± 6.6**

Table 4.3: Means and standard deviations of TDD and baseline accuracies on text data using TextConvNet. The first four baselines are used to train the same neural network as in the distillation experiments. The last two baselines are used to train a K-NN classifier. Each result uses 10 GD steps aside from IMDB with k-means (2 GD steps) and TREC50 (5 GD steps, 4 images per class) which had to be done with fewer steps due to GPU memory constraints and also insufficient training samples for some classes in TREC50. Experiments with random initializations have their results listed in the form [mean ± standard deviation] and are based on the resulting performance of 200 randomly initialized networks.

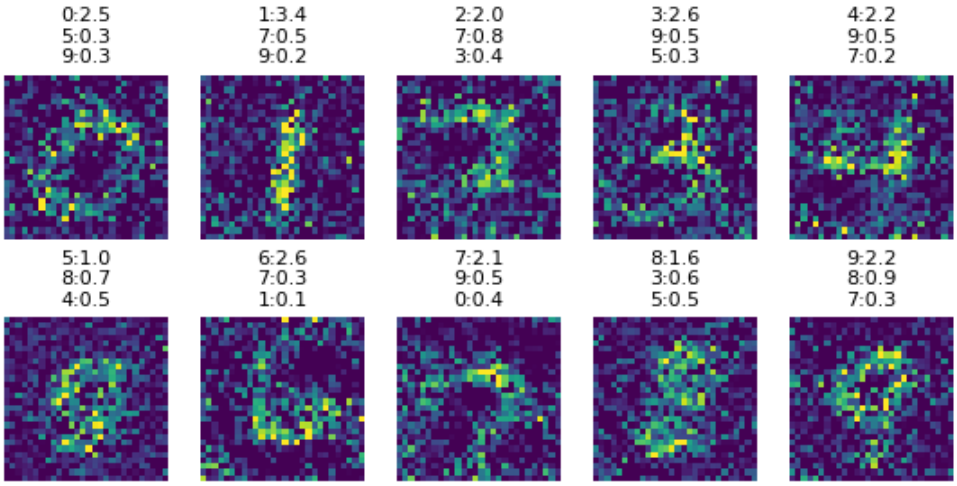


Figure 4.7: SLDD can learn 100 distilled MNIST images (10 steps with 10 images each) that train networks with random initializations from $10.09\% \pm 2.54\%$ distillation accuracy to $82.75\% \pm 2.75\%$ ($r_{100} = 83.6$). Each image is labeled with its top 3 classes and their logits. Only the last step is shown.

Model	Dataset	# of Classes	Accuracy
TextConvNet	IMDB	2	87.1%
Bi-RNN	SST5	5	41.0%
Bi-LSTM	TREC6	6	89.4%
TextConvNet	TREC50	50	84.4%

Table 4.4: Model accuracies when trained on full text datasets.

used to also train K-Nearest Neighbor classifiers to compare performance against neural networks. The baseline results are shown in Table 4.3.

- **Random real sentences:** We randomly sample the same number of real sentences per class, pad/truncate them, and look up their embeddings. These sentences are used for two baselines: training neural networks and training K-Nearest Neighbors classifiers.
- **Optimized real sentences:** We sample and pre-process different sets of random

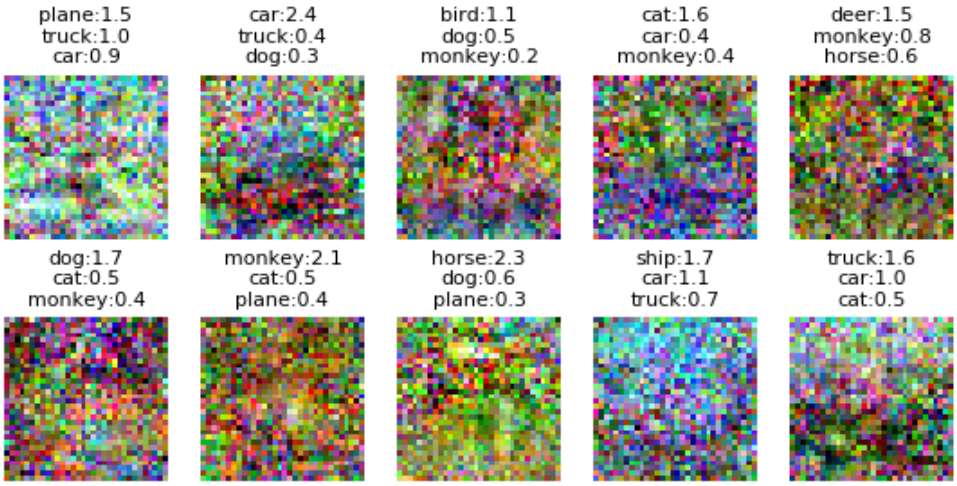


Figure 4.8: SLDD can learn 100 distilled CIFAR10 images (10 steps with 10 images each) that train networks with random initializations from $10.17\% \pm 1.23\%$ distillation accuracy to $39.82\% \pm 0.83\%$ ($r_{100} = 49.8$). Each image is labeled with its top 3 classes and their logits. Only the last step is shown.

Model	Dataset	M	Distillation Ratio (r_M)	
			Fixed	Random (Mean ± SD)
TextConvNet	IMDB	2	89.9	80.0 ± 6.3
TextConvNet	IMDB	20	91.5	85.2 ± 3.2
Bi-RNN	SST5	5	87.7	57.0 ± 5.7
Bi-RNN	SST5	100	89.8	66.8 ± 5.4
Bi-LSTM	TREC6	6	97.8	69.3 ± 9.8
Bi-LSTM	TREC6	120	98.2	78.9 ±6.3
TextConvNet	TREC50	500	57.6	11.0 ± 0.0
TextConvNet	TREC50	1000	67.4	42.1 ± 2.1

Table 4.5: Distillation ratios for text datasets and their associated neural networks. The number of distilled sentences, M, is specified ahead of time. Experiments with random initializations have their results listed in the form [mean ± standard deviation] and are based on the resulting performance of 200 randomly initialized networks.

real sentences as above, but now we choose the 20% of the sets that have the best performance. These sentences are used for one baseline: training neural networks.

- k**-means:** First, we pre-process the sentences. Then, we use k-means to learn clusters for each class, and use the resulting centroids to train. These sentences are used for two baselines: training neural networks and training K-Nearest Neighbors classifiers.

- **Average real sentences:** First, we pre-process the sentences. Then, we compute the average embedded matrix for each class and use it for training. These sentences are used for one baseline: training neural networks.

Distilled Sentence	**Label**
editor panoramic brewing swat regency medley arts fleet attained hilary novelist rugged medal who abbot has sweden new ensemble member understands alba archaeologist operatic intercontinental martian marshal paste smooth titular english-language adaptation songwriter historian enlightenment royal gaelic ceo author macarthur skipper honored excellence distance most endings collaborations his mythological polanski mayer choreographer grisham eminent brooke sympathies modelling vitality dictionary dedication farewell enjoys energetic jordan equality lectures sophia elijah maureen novelist	2.53
dump misled speculate bait substandard uncovered lump corpses 911 punched whopping discarded ref dollar were dough sided brink unconscious tomatoes locking trash burying punched diversion grenade overboard cashier wards agreeing brent prematurely knife randomly stupid buster wipe virus pap waste inflated loan patient peg eliminates nudity worms ?? rotten shoddy strangled substandard boil whopping tunnels steep unjust dummy satisfactory mistakenly adopt tightened bloated hacked misled dump 3-4 untrue contaminated bureaucracy waste	-2.21

Table 4.6: TDD can learn 2 distilled sentences of length 400 that train networks with random initializations from 50.0% to 69.6% $\pm$ 5.5% ($r_2 = 79.96$). Each sentence is accompanied by its associated soft label logit. Only a segment of 70 (out of 400) words is shown for each sentence. The first sentence corresponds to positive sentiment, and the second to negative.

Fixed initialization. TDD achieves impressive performance in fixed initialization experiments that are detailed in Tables 4.3 and 4.5. For example, TDD can produce 2 distilled sentences that train the TextConvNet up to a distillation ratio of almost 90% on

the IMDB dataset. However, TDD does require a larger number of distilled samples M for more complicated datasets like TREC50. In Table 4.7, we list four of the decoded sentences corresponding to the distilled embeddings from the six-sentence TREC6 experiments along with their respective label distributions. These sentences can contain any tokens found in the TREC6 dataset, including punctuation, numbers, abbreviations, etc. The sentences do not have much overlap which is consistent with the distilled labels suggesting that each sentence corresponds strongly to a different class. It appears that TDD encourages class separation; however, some mixing still occurs and some words do not seem to match the associated class with the highest logit. This may be due to the TDD algorithm overfitting to the noise of the model initialization, and is consistent with the fixed initialization results for image data.

Random initialization. TDD experiments with random initialization are detailed in Tables 4.3 and 4.5. On IMDB, TDD with random initialization has only a slight performance drop compared to fixed initialization. However, there is a larger drop in performance when going from fixed to random initializations in experiments involving the recurrent networks and the more difficult TREC50 task. We list the decoded distilled sentences corresponding to the distilled text embeddings from the six-sentence TREC6 experiments with random network initializations in Table 4.8, along with their respective label distributions. Once again, the decoded sentences do not have much overlap, though the last two both seem to focus on locations. This is consistent with the distilled labels which suggest that each sentence corresponds strongly to a different class, except for the last sentence which is also associated primarily with the location class. We believe that the model has mostly ignored the abbreviation class due to severe class imbalance; abbreviation is the smallest class comprising only 86 of the 5500 training examples. As such, a greater performance increase can be achieved by assigning the last sentence to a combination of the other classes. Compared to the fixed initialization case, the words in each sentence now seem to be more related to the associated class with the highest logit. This is consistent with the random initialization results for image data as the TDD algorithm is no longer overfitting to the noise of the model initialization. We show a sample of the two decoded sentences from the IMDB experiment with $M = 2$ in Table 4.6. As this is a binary classification task, each label is a scalar. TDD produced one sentence with a positive sentiment and one with a negative sentiment. The model appears to have overcome the challenge of having to describe long sentences with a single scalar by using duplication. For example in the negative sentence, words like 'contaminated' and 'misled' are repeated. In such a way, the algorithm may be assigning lower sentiment scores to individual words.

	Logits of Label Class					
	0	1	2	3	4	5
Distilled Sentence	ENT	HUM	DES	NUM	LOC	ABB
allan milk banned yellow planted successfully introduced bombay 1936 grass mines iron delhi 1942 male heir throne oath clouds 7th occur millennium smoking flows truth powder judiciary pact slim profit	**2.72**	-0.48	-0.07	-0.62	-0.53	-0.27
whom engineer grandfather joan officer entered victoria 1940s taxi romania motorcycle italian businessman photographer powerful driving u brilliant affect princess 1940s enemies conflicts southwestern retired cola appearances super dow consumption	-0.05	**3.21**	-1.15	-0.79	-0.71	-0.64
necessarily factors pronounced pronounced define bow destroying belonged balls 1923 storms buildings 1925 victorian sank dragged reputation sailed nn occurs darkness blockade residence traveled banner chef ruth rick lion psychology	-0.67	-0.66	**3.28**	-0.27	-0.77	0.47
accommodate accommodate peak 2.5 adults thin teenagers hike aged nurse policeman admit aged median philippines define baghdad libya ambassador admit baseman burma inning bills trillion donor fined visited stationed clean	-0.98	-0.14	-1.12	**5.57**	-0.85	-1.85
suburb ports adjacent mountains nearest compare hilton volcano igor nebraska correspondent 1926 suburb sailed hampshire hampshire gathering lesson proposition metric copy carroll sacred moral lottery whatever fix o completed ultimate	-0.29	-0.22	-0.36	-1.01	**3.86**	-0.44
advertising racism excuse d nancy solved continuing congo diameter oxygen accommodate provider commercials spread pregnancy mideast ghana attraction volleyball zones kills partner serves serves congressman advisory displays ranges profit evil	0.94	0.53	-0.85	0.08	-0.83	**1.99**

Table 4.7: TDD can learn 6 distilled sentences of length 30 that train networks with fixed initializations from 12.6% to 87.4% ($r_6 = 97.8$). The first column contains the nearest decoding for each distilled sentence. Each sentence is accompanied by the logits associated with each value of its distilled label. Classes are denoted by their standard abbreviations.

	Logits of Label Class					
	0	1	2	3	4	5
Distilled Sentence	ENT	HUM	DES	NUM	LOC	ABB
taste displays animated variety comic detroit adventures comic nickname silk golden circulation release photograph allegedly scandal allegedly lynn economist jeremy craig hockey chess pregnant ads abraham w chess stakes cliff	**4.11**	-2.01	0.42	-2.90	-3.31	0.40
who composer nephew conductor founders producer ally lady 1922 industry electrical company commander colonel hood addresses serial handle issues associates determine cathedral colleges domestic points la 1922 swing warren kenya	-2.36	**6.01**	-3.11	-3.03	-3.18	-2.13
necessarily happen happen proceed solved boom anymore mess ash facts disc fever mess illustrated suite ugly plane yankees ticket puerto dog senator stern floor boycott le lists igor articles streak	0.35	-3.38	**4.16**	-2.36	-2.94	0.34
size boom enrolled many seized span occurred floors detailed visitors m americas temperatures tropical certified tropical crew no. academy sponsor merged partner attraction ally floors justin herbert ivan victory chuck	-0.95	-2.49	-1.73	**5.26**	-2.32	-1.15
northeast region mountainous located cemetery airport beaches sand mountainous streets trinidad southern congo border seas britain beverly walks correspondent cooking pit premiered reviews airports audience moo continental musician founders actress	-1.66	-2.38	-1.74	-2.11	**4.54**	0.13
adjacent mozambique austria romania gaza highways erected tackle romania mile romania china hiv richest romania romania dream magazine burial detective cliff rhode anna toys beverly hearts square businessman photographer ad	1.19	0.55	0.20	1.14	**2.72**	1.07

Table 4.8: TDD can learn 6 distilled sentences of length 30 that train networks with random initializations from 16.60% ± 8.33% to 61.99% ± 8.79% ($r_6 = 69.33$). The first column contains the nearest decoding for each distilled sentence. Each sentence is accompanied by the logits associated with each value of its distilled label. Classes are denoted by their standard abbreviations.

4.5 Conclusion

By introducing learnable distilled labels, we have increased distillation accuracy across multiple datasets by up to 6%. By enabling text distillation, we have also greatly increased the types of datasets and architectures with which distillation can be used. In fact, as long as a network has a twice-differentiable loss function and the gradient can be back-propagated to the inputs, then that network is compatible with dataset distillation. However, there are still some limitations to dataset distillation. The network initializations come from the same distribution, and no testing has yet been done on whether a single distilled dataset can be used to train networks with different architectures. Further investigations are needed to determine more precisely how well dataset distillation can be generalized to work with more variation in initializations, and even across networks with different architectures.

As mentioned above, the initialization of distilled labels appears to affect performance. It is not clear whether it is better to separate similar classes (e.g. '3' and '8' in MNIST) thereby encouraging the network to discern between them, or to instead keep those classes together thus allowing soft-label information to be shared between them. It may be interesting to explore the dynamics of distillation when using a variety of label initialization methods. The pre-specified number of distilled samples also appears to have a large effect. Generally, we found that adding more distilled samples results in higher performance, but we quickly ran into issues such as overfitting and decreasing returns as we experimented with larger numbers of distilled samples.

Working with a very small number of distilled samples also produced several interesting results. For example, with the TREC6 experiment, it appears that the bottleneck created by restricting the number of distilled sentences leads to smaller classes being ignored. Meanwhile, reducing the number of distilled images for MNIST revealed that soft labels allow us to represent the dataset using less than one sample per class.

More broadly, distilled datasets enable faster training which is particularly useful for federated learning, as well as for other computationally intensive meta-algorithms like neural architecture search. When distilled datasets are a good proxy for performance evaluation, they may reduce training times by multiple orders of magnitude.

Chapter 5

Soft-Label Prototypes and k-Nearest Neighbors

5.1 Motivation and Related Work

As mentioned in the previous chapter, Wang et al. [2018] showed that Dataset Distillation (DD) can use backpropagation to create small synthetic datasets that train neural networks to nearly the same accuracy as when training on the original datasets. Networks can reach over 90% accuracy on MNIST after training on just one such distilled image per class (ten in total), an impressive example of one-shot learning. However, we showed that dataset sizes could be reduced even further by enhancing DD with soft, learnable labels. Soft-Label Dataset Distillation (SLDD) can create a dataset of just five distilled images (less than one per class) that trains neural networks to over 90% accuracy on MNIST. In other words, SLDD can create five samples that allow a neural network to separate ten classes. To the best of our knowledge, this is the first example of LO-shot learning and it motivates us to further explore this direction.

In this chapter, we aim to investigate the theoretical foundations of LO-shot learning so we choose to work with a simpler model that is easier to analyze than deep neural networks.

5.1.1 Prototype Methods for kNN

The k-Nearest Neighbour (kNN) classifier is a simple but powerful classification algorithm. There are numerous variants and extensions of kNN [Dudani, 1976, Yigit, 2015, Sun et al.,

2016, Kanjanatarakul et al., 2018, Gweon et al., 2019], but the simplest version is the 1NN classifier which assigns a target point to a class based only on the class of its nearest labeled neighbor. Unfortunately, the family of kNN classifiers can be computationally expensive when working with large datasets, as the nearest neighbors must be located for every point that needs to be classified. This has led to the development of prototype selection methods and generation methods which aim to produce a small set of prototypes that represent the training data [Bezdek and Kuncheva, 2001, Triguero et al., 2011a, Bien and Tibshirani, 2011, Garcia et al., 2012, Kusner et al., 2014]. Reducing the number of prototypes required to represent a training dataset is especially valuable for instance-based algorithms like kNN as the computational complexity at inference time depends on the number of training examples the classifier was fitted on. In addition to dataset distillation and prototype generation, there are numerous other methods making use of various tricks and heuristics to achieve impressive results in reducing the size of training datasets. These include active learning [Cohn et al., 1996, Tong and Koller, 2001], core-set selection [Tsang et al., 2005, Bachem et al., 2017, Sener and Savarese, 2017], and dataset pruning [Angelova et al., 2005]. There are even methods that perform soft-label dataset condensation [Ruta, 2006].

Using prototype methods speeds up the kNN classification step considerably as new points can be classified by finding their nearest neighbors among the small number of prototypes. Prototype selection methods select a subset of real points to use as prototypes while prototype generation methods are not similarly restricted and instead create synthetic points (that are not necessarily found in the original data) to act as prototypes. Generating synthetic prototypes allows for more efficient representations so our analysis focuses specifically on the generation of optimal or near-optimal prototypes. The number of prototypes required to represent the training data can be several orders of magnitude smaller than the number of samples in the original training data. However, in Section 5.2, we discuss particularly pathological geometric configurations where heuristic methods fail and analytical methods must be used instead.

5.1.2 Achieving LO-Shot Learning with kNN

In Section 5.3, we build on our previous success with soft labels and focus on theoretically establishing the link between soft-label prototypes and 'less than one'-shot learning. Specifically, we propose a generalization of kNN that can be fitted on soft-label points. Our analysis is centered around a distance-weighted kNN variant that can make use of soft-label prototypes. Distance-weighted kNN rules have been studied extensively [Dudani, 1976, Macleod et al., 1987, Gou et al., 2012, Yigit, 2015] since inverse distance weighting

was first proposed by Shepard [1968]. Much effort has also gone into providing finer-grained class probabilities by creating soft, fuzzy, and conditional variants of the kNN classifier [Mitchell and Schaefer, 2001, El Gayar et al., 2006, Thiel, 2008, El-Zahhar and El-Gayar, 2010, Kanjanatarakul et al., 2018, Wang and Zhang, 2019, Gweon et al., 2019]. We chose our kNN variant because it works well with soft-label prototypes but remains simple and easy to implement. We use it to explore the intricate decision landscapes that can still be created even with an extremely limited number of training samples. We first analyze these decision landscapes in a data-agnostic way to derive theoretical lower bounds for separating N classes using $M < N$ soft-label samples. Unexpectedly, we find that our model fitted on just two prototypes with carefully designed soft labels can be used to divide the decision space into any finite number of classes. Additionally, we provide a method for analyzing the stability and robustness of the created decision landscapes. We find that by carefully tuning the hyper-parameters we can elicit certain desirable properties. We also perform a case study to confirm that soft labels can be used to represent training sets using fewer prototypes than there are classes, achieving large increases in sample-efficiency over regular (hard-label) prototypes. In extreme cases, like the concentric circle dataset, using soft labels can even reduce the minimal number of prototypes required to perfectly separate N classes from $\mathcal{O}(N^2)$ down to $\mathcal{O}(1)$. In Section 5.4, we aim to extend the theory from Section 5.3 into algorithms that can be used for performing LO-shot prototype generation and learning in practice.

5.2 Optimal 1NN Prototypes for Pathological Geometries

5.2.1 Background

We consider a synthetic dataset consisting of N concentric circles where the points on each circle belong to a different class. We wish to select or generate the minimal number of prototypes such that the associated kNN classification rule will separate each circle as a different class. We find that commonly used prototype generation methods failed to find prototypes that would adequately represent this dataset, suggesting that the dataset exhibits pathological geometries. After applying analytical methods to this dataset, we find that $\mathcal{O}(N^2)$ hard-label prototypes are required for 1NN to separate the N classes.

Theorem 1 *Suppose we have N concentric circles with radius $r_t = t * c$ for the t^{th} circle. An upper bound for the minimum number of hard-label prototypes required for 1NN to*

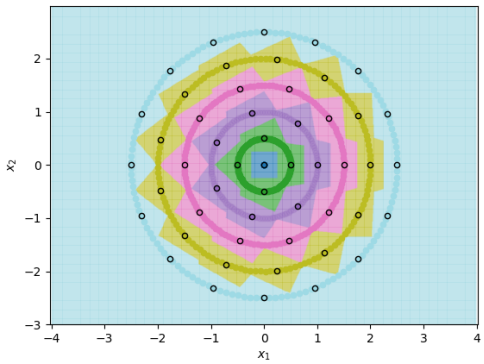


Figure 5.1: Decision boundaries of a vanilla 1NN classifier fitted on the minimum number of prototypes required to perfectly separate circle classes. From inner to outer, the circles have 1, 4, 7, 10, 13, and 16 prototypes.

produce decision boundaries that perfectly separate all circles, is $\sum_{t=1}^{N} \frac{\pi}{\cos^{-1}\left(1-\frac{1}{2t^2}\right)}$*; with the* t^{th} *circle requiring* $\frac{\pi}{\cos^{-1}\left(1-\frac{1}{2t^2}\right)}$ *prototypes.*

The proof can be found in the Appendix. We can use the approximation $cos^{-1}(1-y) \approx \sqrt{2y}$ to get that $\frac{\pi}{cos^{-1}(1-\frac{1}{2t^2})} \approx \frac{\pi}{(\frac{1}{t})} = t\pi$. Thus the upper bound for the minimum number of prototypes required is approximately $\sum_{t=1}^{N} t*\pi = \frac{N(N+1)\pi}{2}$. Figure 5.1 visualizes the decision boundaries of a regular kNN that perfectly separates six concentric circles given $\lceil t\pi \rceil$ prototypes for the t^{th} circle. It is possible that the number of prototypes may be *slightly* reduced by carefully adjusting the prototype locations on adjacent circles to maximize the minimal distance between the prototype midpoints of one circle and the prototypes of neighboring circles. However, this upper bound does not account for the possibility of rotating prototypes on adjacent circles as a method of reducing the number of required prototypes. We explore this direction to analytically find tighter bounds and an approximate solution for the minimal number of prototypes required for a 1-Nearest Neighbor classifier to perfectly separate each class after being fitted on the prototypes. In particular, we show that this problem actually consists of two sub-problems, or cases, only one of which is closely approximated by the previously proposed upper bound. We also propose an algorithm for finding nearly-optimal prototypes and use it to empirically confirm our theoretical results.

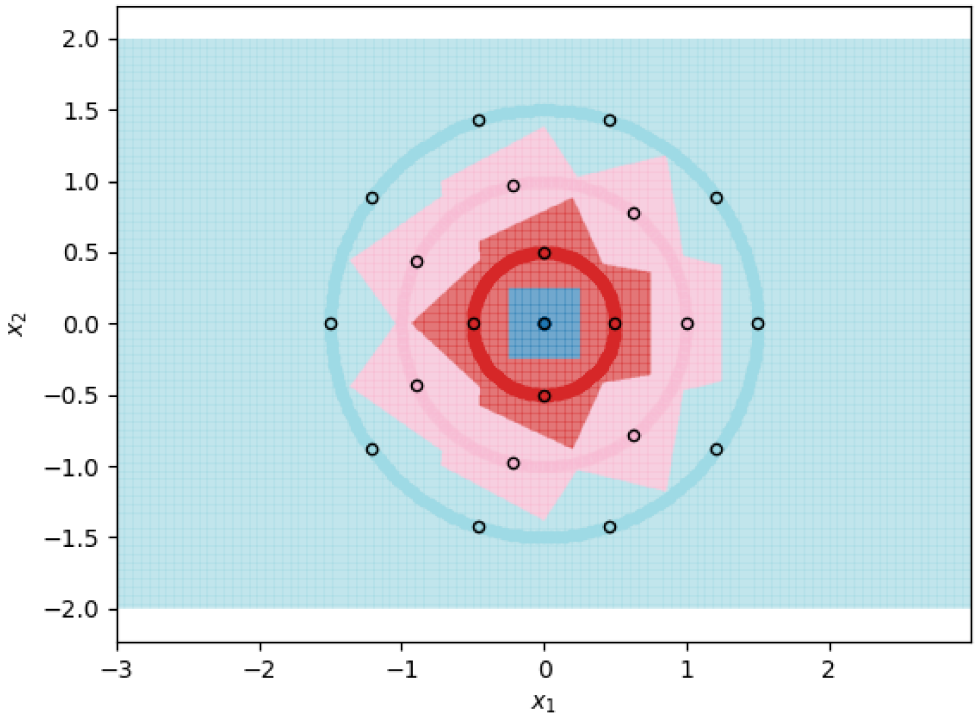


Figure 5.2: 1NN decision boundaries when fitted on $\lceil t\pi \rceil$ prototypes per class. Each shaded circle represents a different class and the outlined points represent the assigned prototypes. The colored regions correspond to the decision boundaries created by the 1NN classifier. The axes form a Cartesian plane whose origin coincides with the smallest class. Different rotations of prototype placements on adjacent circles can lead to changes in the decision boundaries.

5.2.2 Theory

Preliminaries

We first proceed to formalize the problem of having a 1-NN classifier separate the classes after being fitted on a minimal number of prototypes. Consistent with Sucholutsky and Schonlau [2021a], we define the t^{th} circle as having radius tc for $t = 0, 1, \ldots$. Because each class is fully separated from non-adjacent classes by its adjacent classes, it is sufficient to consider arbitrary pairs of adjacent classes when trying to find the optimal prototypes. For the rest of this section, we consider arbitrarily selected circles t and $t+1$ with the following radii.

$$r_1 = tc,\ r_2 = (t+1)c,\ t \in \mathbb{N}_0,\ c \in \mathbb{R}_{>0},$$

Because of the symmetry of each circle, we require that the prototypes assigned to each circle be spaced evenly around it. We assume that circle t and $t+1$ are assigned m and n prototypes respectively. We define θ^* as the angle by which the prototypes on circle $t+1$ are shifted relative to the prototypes on circle t. We record the locations of these prototypes in Cartesian coordinates.

$$a_i = (r_1 \cos(\frac{2\pi i}{m}), r_1 \sin(\frac{2\pi i}{m})),\ i = 1, ..., m$$
$$b_j = (r_2 \cos(\frac{2\pi j}{n} + \theta^*), r_2 \sin(\frac{2\pi i}{n} + \theta^*)),\ i = 1, ..., n$$

We can then find the arc-midpoints of these prototypes as follows.

$$a_i^* = (r_1 \cos(\frac{2\pi i + \pi}{m}), r_1 \sin(\frac{2\pi i + \pi}{m})),\ i = 1, ..., m$$
$$b_j^* = (r_2 \cos(\frac{2\pi j + \pi}{n} + \theta^*),\ r_2 \sin(\frac{2\pi i + \pi}{n} + \theta^*)),\ i = 1, ..., n$$

Letting $d(x, y)$ be the Euclidean distance between points x and y, we find the distances between prototypes on the same circle.

$$d_a(m) = d(a_i, a_i^*) = \sqrt{2t^2c^2 - 2t^2c^2\cos(\frac{\pi}{m})}$$
$$d_b(n) = d(b_i, b_i^*) = \sqrt{2(t+1)^2c^2 - 2(t+1)^2c^2\cos(\frac{\pi}{n})}$$

We also find the shortest distance between prototypes of circle t and arc-midpoints of circle $t+1$ and vice-versa.

$$\begin{aligned} d_1^*(m,n,\theta^*) &= \min_{i,j}\{d(a_i, b_j^*) \| \ {}^{i=1,\ldots,m}_{j=1,\ldots,n}\} \\ &= \min_{i,j}\{\sqrt{t^2c^2 + (t+1)^2c^2 - 2t(t+1)c^2\cos(\frac{2\pi i}{m} - \frac{2\pi j + \pi}{n} - \theta^*)} \| \ {}^{i=1,\ldots,m}_{j=1,\ldots,n}\} \\ d_2^*(m,n,\theta^*) &= \min_{i,j}\{d(a_i^*, b_j) \| \ {}^{i=1,\ldots,m}_{j=1,\ldots,n}\} \\ &= \min_{i,j}\{\sqrt{t^2c^2 + (t+1)^2c^2 - 2t(t+1)c^2\cos(\frac{2\pi i + \pi}{m} - \frac{2\pi j}{n} - \theta^*)} \| \ {}^{i=1,\ldots,m}_{j=1,\ldots,n}\} \end{aligned}$$

The necessary and sufficient condition for the 1-NN classifier to achieve perfect separation is that the distance between prototypes and arc-midpoints assigned to the same circle, be less than the minimal distance between any arc-midpoint of that circle and any prototype of an adjacent circle. This must hold for every circle. Given these conditions and some fixed number of prototypes assigned to the t^{th} circle, we wish to minimize n by optimizing over θ^*.

$$\begin{aligned} &\text{Given } m, t \ \min_{\theta^*} n \\ &\quad \text{s.t. } d_1^*(m,n,\theta^*) > d_b(n) \\ &\qquad d_2^*(m,n,\theta^*) > d_a(m) \end{aligned}$$

Inspecting the inequalities, we see that they can be reduced to the following system which we note is now independent of the constant c .

$$-\frac{2t+1}{2(t+1)} > t\cos(\frac{2\pi i}{m} - \frac{2\pi j + \pi}{n} - \theta^*) - (t+1)\cos(\frac{\pi}{n}) \tag{5.1}$$
$$\frac{2t+1}{2t} > (t+1)\cos(\frac{2\pi i + \pi}{m} - \frac{2\pi j}{n} - \theta^*) - t\cos(\frac{\pi}{m}) \tag{5.2}$$

It is clear that $n \geq m$, but we separate this system into two cases, $n = m$ and $n > m$, as the resulting sub-problems will have very different assumptions and solutions. The simpler case is where every circle is assigned the same number of prototypes; however, the total number of circles must be finite and known in advance. In the second case where larger circles are assigned more prototypes, we assume that the number of circles is countable but not known in advance. We also note that for $t = 0$, a circle with radius 0, exactly one prototype is required. Given this starting point, it can be trivially shown that for $t = 1$, a minimum of four prototypes are required to satisfy the conditions above (three if the strict inequalities are relaxed to allow equality). However for larger values of t, careful analysis is required to determine the minimal number of required prototypes.

Upper bounds

We first show how our setup can be used to derive the upper bound that was found by Sucholutsky and Schonlau [2021a].

Theorem 2 (Previous Upper Bound) *The minimum number of prototypes required to perfectly separate N concentric circles is bounded from above by approximately $\sum_{t=1}^{N} t\pi$, if the number of circles is not known in advance (each circle must have a different number of assigned prototypes).*

Proof. Given the setup above, we first consider the worst case scenario where a θ^* is selected such that $\cos(\frac{2\pi i}{m} - \frac{2\pi j+\pi}{n} - \theta^*) = \cos(\frac{2\pi i+\pi}{m} - \frac{2\pi j}{n} - \theta^*) = cos(0) = 1$. We can then solve Inequality 5.1 for n and Inequality 5.2 for m.

$$
\begin{aligned}
-\frac{2t+1}{2(t+1)} &> t\cos(0) - (t+1)\cos(\frac{\pi}{n}) \\
\cos(\frac{\pi}{n}) &> \frac{2(t+1)^2 - 1}{2(t+1)^2} \\
n &> \frac{\pi}{\arccos(\frac{2(t+1)^2-1}{2(t+1)^2})} \approx (t+1)\pi \\
\frac{2t+1}{2t} &> (t+1)\cos(0) - t\cos(\frac{\pi}{m}) \\
\cos(\frac{\pi}{m}) &> \frac{2t^2 - 1}{2t^2} \\
m &> \frac{\pi}{\arccos(\frac{2t^2-1}{2t^2})} \approx t\pi
\end{aligned}
$$

This is exactly the previously discovered upper bound. □

However, note that we assumed that there exists such a θ^*, but this may not always be the case for $n > m$. If we instead use the same number of prototypes for each circle (i.e. $m = n$), then we can always set $\theta^* = \frac{\pi}{n}$. This results in a configuration where every circle is assigned $n = \lceil \frac{\pi}{\arccos(\frac{2(t+1)^2-1}{2(t+1)^2})} \rceil \approx \lceil (t+1)\pi \rceil$ prototypes. While the minimum number of prototypes required on the t^{th} circle remains the same, the *total* minimum number of prototypes required to separate N circles is higher as each smaller circle is assigned the same number of prototypes as the largest one.

Corollary 3 (Upper Bound - Same Number of Prototypes on Each Circle) *The minimum number of prototypes required to perfectly separate N concentric circles is bounded from above by approximately $N^2\pi$, if the number of circles is finite and known in advance (each circle can have the same number of assigned prototypes).*

Lower bounds

An advantage of our formulation of the problem is that it also enables us to search for lower bounds by modifying the θ^* parameter. We can investigate the scenario where a θ^* is selected that simultaneously maximizes $d_1^*(m, n, \theta^*)$ and $d_1^*(m, n, \theta^*)$.

Theorem 4 (Lower Bound) *The minimum number of prototypes required to perfectly separate N concentric circles is bounded from below by approximately $\sum_{t=1}^{N} t^{\frac{1}{2}}\pi$, if the number of circles is not known in advance (each circle must have a different number of assigned prototypes).*

Proof. If $m \neq n$, the best case would be a θ^* such that $\cos(\frac{2\pi i}{m} - \frac{2\pi j+\pi}{n} - \theta^*) = \cos(\frac{2\pi i+\pi}{m} - \frac{2\pi j}{n} - \theta^*) = cos(\frac{\pi}{n})$. Solving the inequalities leads to the following values for m and n.

$$n > \frac{\pi}{\arccos(\frac{2t+1}{2(t+1)})} \approx (t+1)^{\frac{1}{2}}\pi$$

$$m > \frac{\pi}{\arccos(\frac{2t^2-t-1}{2t^2})} \approx \frac{t}{(t+1)^{\frac{1}{2}}}\pi$$

We note again that such a θ^* may not always exist. □

Exact and approximate solutions

In the case where $m = n$, we can always choose a θ^* such that $\cos(\frac{2\pi i}{m} - \frac{2\pi j+\pi}{n} - \theta^*) = cos(\frac{\pi}{n})$. Solving the inequalities, we get that $n > \frac{\pi}{\arccos(\frac{2t+1}{2(t+1)})} \approx (t+1)^{\frac{1}{2}}\pi$. Thus we have a tight bound for this case.

Corollary 5 (Exact Solution - Same Number of Prototypes on Each Circle) *The minimum number of prototypes required to perfectly separate N concentric circles is approximately $N^{\frac{3}{2}}\pi$, if the number of circles is finite and known in advance (each circle can have the same number of assigned prototypes).*

When $m > n$, we have that $\cos(\frac{2\pi i}{m} - \frac{2\pi j+\pi}{n} - \theta^*) > cos(\frac{\pi}{n})$ as $\frac{2\pi i}{m} - \frac{2\pi j}{n} = \frac{2\pi c_1 \gcd(m,n)}{mn}$, $c_1 \in \mathbb{N}_0$. Let $q := \frac{2\pi \gcd(m,n)}{mn}$, then $|\frac{2\pi i}{m} - \frac{2\pi j+\pi}{n} - \theta^*| \leq \frac{q}{2}$ and $|\frac{2\pi i+\pi}{m} - \frac{2\pi j}{n} - \theta^*| \leq \frac{q}{2}$. Thus $\cos(\frac{2\pi i}{m} - \frac{2\pi j+\pi}{n} - \theta^*) \geq cos(\frac{q}{2})$, and $\cos(\frac{2\pi i+\pi}{m} - \frac{2\pi j}{n} - \theta^*) \geq cos(\frac{q}{2})$.
Using the series expansion at $q = 0$ we can find that $\cos(\frac{q}{2}) = 1 - \frac{q^2}{8} + \frac{q^4}{384} - \frac{q^6}{46080} + O(q^8)$.

Theorem 6 (First Order Approximation - Different Number of Prototypes on Each Circle) *The minimum number of prototypes required to perfectly separate N concentric circles is approximately $1 + \sum_{t=1}^{N} t\pi$, if the number of circles is not known in advance (each circle must have a different number of assigned prototypes).*

Proof. For a first order approximation, we consider $\cos(\frac{q}{2}) = 1 - \frac{q^2}{8} + O(q^4)$ and $\cos(\frac{\pi}{n}) = 1 - \frac{\pi^2}{2n^2} + O(\frac{1}{n^4})$. Inequality 5.1 then becomes the following.

$$\begin{aligned} -\frac{2t+1}{2(t+1)} &> t(1 - \frac{q^2}{8} + O(q^4)) - (t+1)(1 - \frac{\pi^2}{2n^2} + O(\frac{1}{n^4})) \\ &= -1 - \frac{\pi^2}{2n^2}(t\frac{\gcd(m,n)^2}{m^2} - t - 1) + O(\frac{1}{n^4}) \\ n^2 &> -\pi^2(t+1)(t\frac{\gcd(m,n)^2}{m^2} - t - 1) + O(\frac{1}{n^2}) \end{aligned}$$

However, we know from our previous upper bound that $m + 1 \leq n \leq m + 4$.
Thus $\frac{4}{(n-4)^2} > \frac{\gcd(m,n)^2}{m^2} > \frac{1}{(n-1)^2}$ which means that $\frac{\gcd(m,n)^2}{m^2} = O(\frac{1}{n^2})$.

$$\begin{aligned} n^2 &> -\pi^2(t+1)(t\frac{\gcd(m,n)^2}{m^2} - t - 1) + O(\frac{1}{n^2}) \\ &= \pi^2(t+1)^2 + O(\frac{1}{n^2}) \end{aligned}$$

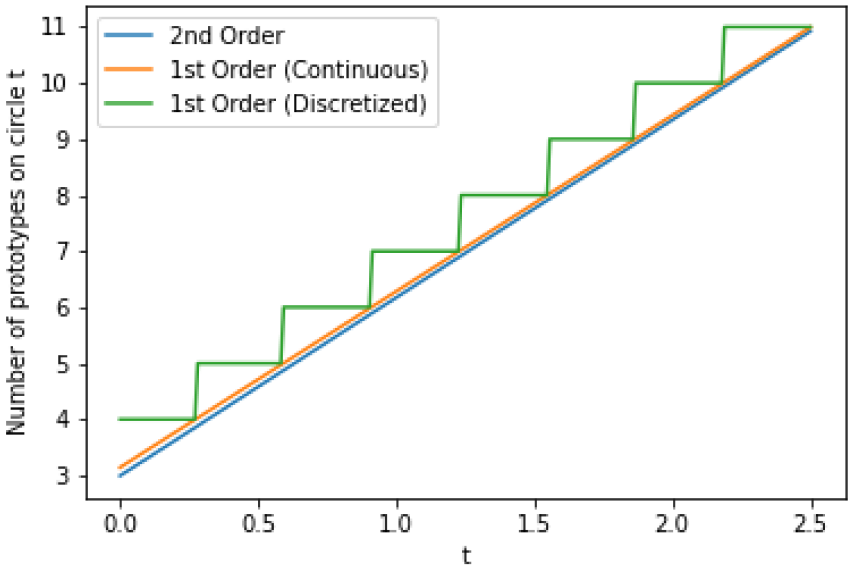
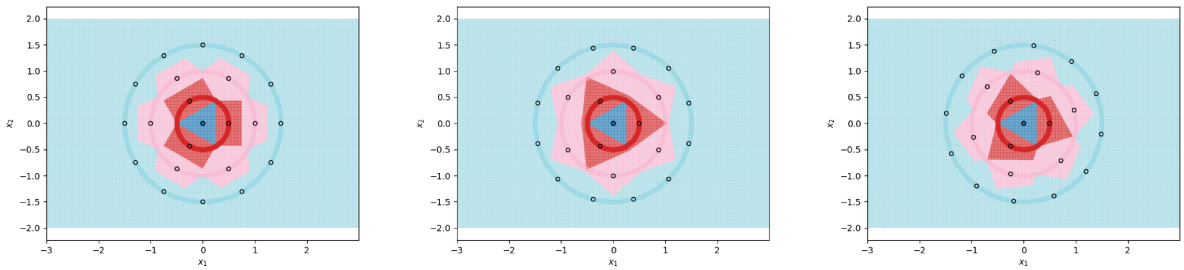


Figure 5.3: First order (before and after discretizing by rounding to nearest integer) and second order approximations for the minimal number of prototypes that must be assigned to circle t. The approximations are applied to continuous values of t to show the convergence behavior.

Therefore we have that $n + O(\frac{1}{n}) > (t+1)\pi$ as desired. □

We plot the second order approximation alongside the first order approximation from Theorem 6 in Figure 5.3, without rounding to show that the two quickly converge even at small values of t. Thus we can be confident that approximately $t\pi$ prototypes are required for the t^{th} circle since this approximation quickly approaches the true minimal number of required prototypes as t increases. Since we can only assign a positive integer number of prototypes to each circle, we assign $\lceil t\pi \rceil$ prototypes to the t^{th} circle; this is also shown in Figure 5.3. Applying this to the initial condition that the 0^{th} circle is assigned exactly one prototype results in the following sequence of the minimal number of prototypes that must be assigned to each circle. We note that the sequence generated by the second order approximation would be almost identical, but with a 3 replacing the 4.

$$1, 4, 7, 10, 13, 16, 19, 22, 26, 29, 32, 35, 38, 41 \ldots$$



(a) (b) (c)

Figure 5.4: 1NN decision boundaries when fitted on two sub-optimal prototype arrangements as well as near-optimal prototypes found using the FindPUGS algorithm. Each shaded circle represents a different class and the outlined points represent the assigned prototypes. The colored regions correspond to the decision boundaries created by the 1NN classifier. The axes form a Cartesian plane whose origin coincides with the smallest class. **Left and Center**: Prototypes on adjacent circles are not optimally rotated resulting in imperfect class separation in certain regions. **Right**: Prototypes are optimally rotated to maximize distances between the prototypes and prototype arc-midpoints of adjacent circles resulting in perfect class separation.

Corollary 7 (Approximate Solution - Different Number of Prototypes on Each Circle) *The minimum number of prototypes required to perfectly separate N concentric circles is approximately $\sum_{t=1}^{N} \lceil t\pi \rceil \approx \frac{N+N(N+1)\pi}{2}$, if the number of circles is not known in advance (each circle must have a different number of assigned prototypes).*

5.2.3 Computational Results

Algorithm

While Theorem 6 gives us the number of prototypes required for each circle, it does not give us the exact locations of these prototypes. Finding the locations would require us to know θ, the optimal rotation of circle $n+1$ relative to circle n. Unfortunately, the equations involving θ depend on greatest common denominator terms. Since this makes it difficult to find explicit analytical solutions, we instead turn to computational methods to find near-optimal prototypes. The theoretical results above enable us to develop computational methods to empirically find the minimum number of required prototypes. Based on the equations derived in the previous section, we propose an iterative, non-parametric al-

gorithm, Algorithm 4, that proceeds from the innermost circle to the outermost one finding a near-optimal number of required prototypes, and their positions, in a greedy manner.

The core of the algorithm consists of three loops: outer, middle, inner. The outer loop iterates over each circle, from smallest to largest. For each circle, m is set to be the minimum number of prototypes found during the loop for the previous circle. The middle loop then iterates over candidate values of n, starting from $m + 1$ and increasing by one each time, until it reaches the first value of n for which Equations 5.1 and 5.2 can be simultaneously solved given the current value of m. Whether the equations can be solved is determined in the inner loop which plugs different values of the rotation angle θ into the system. Since the distance equations are periodic over values of θ, and the length of the period depends on the greatest common divisor of m and n, we can speed the search up by only considering an interval of the length of the maximum period and iterating θ by an angle inversely proportional to the product of m and n. To avoid any potential floating-point precision errors, we use a wider than necessary interval and smaller than necessary update size for θ. At the end of each iteration of the outer loop, the n and θ that were found are recorded. We note that the rotation angle θ is relative to the rotation of the previous circle. In other words, the absolute rotation for a given circle can be found by adding its relative rotation to the relative rotations of all the preceding circles.

Our code for this algorithm can be found at the publicly available GitHub repositories associated with this thesis. As shown above, the choice of $c > 0$, the constant length by which the radius of each consecutive circle increases, does not affect the number of required prototypes. Nonetheless, we still include c as a parameter in our algorithm to verify correctness. Running the algorithm for some large T, with any choice of c, results in the following sequence.

$$1, 3, 6, 12, 13, 16, 19, 22, 26, 29, 32, 35, 38, 41 \ldots$$

This sequence appears to converge very quickly to the one predicted by our theorem. Curiously, the small differences between the first few steps of the two sequences cancel out and the cumulative number of required prototypes is identical when there are four or more circles. While requiring the algorithm to find numerical solutions to these equations is perhaps not computationally efficient, it does guarantee near-optimal performance, with the only sub-optimal portion occurring at the start of the sequence where the algorithm outputs $1, 3, 6, 12, 13$ rather than the optimal $1, 3, 7, 10, 13$ due to its greedy nature.

We visualize two sub-optimal prototype arrangements, and the near-optimal arrangement found by our algorithm, in Figure 5.4. The patterns seen in these visualizations

are largely dependent on the greatest common divisors of the number of prototypes on adjacent circles, as well as the relative rotations of the prototypes on adjacent circles. The particularly symmetrical patterns in Figure 5.4 are a result of the outer three circles having 3, 6, and 12 prototypes respectively, doubling each time. We show another example of the decision boundaries exhibited by 1NN when fitted on near-optimal prototypes in Figure 5.5.

Heuristic Prototype Methods

We compare the performance of our proposed algorithm against a variety of existing prototype selection and generation methods. Specifically, we compare against every undersampling method implemented by Lemaître et al. [2017] in the 'imbalanced-learn' Python package. We describe the prototype methods below and summarize their key properties in Table 5.1.

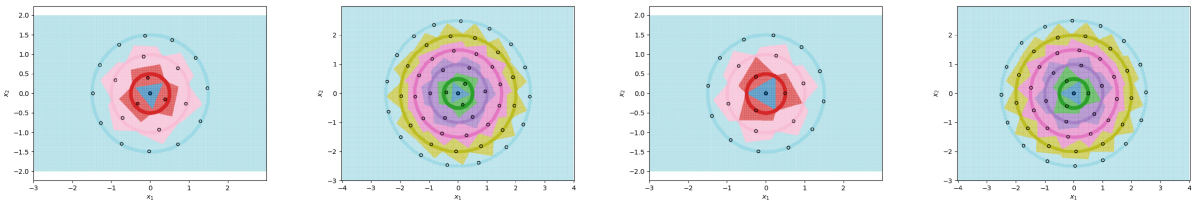
Name	Type	Choosing Number of Prototypes
TomekLinks	Selection	Automatic
RandomUndersampler	Selection	Automatic or Manual
OneSidedSelection	Selection	Automatic
NeighbourhoodCleaningRule	Selection	Automatic
NearMissV1-3	Selection	Automatic or Manual
InstanceHardnessThreshold	Selection	Automatic or Semi-Automatic
AllKNN	Selection	Automatic
EditedNearestNeighbours	Selection	Automatic
RepeatedEditedNearestNeighbours	Selection	Automatic
CondensedNearestNeighbours	Selection	Automatic
ClusterCentroids	Generation	Automatic or Manual

Table 5.1: A list of prototype selection and generation methods. The last column describes how the number of prototypes is chosen for each class.

- TomekLinks: Rebalances classes by removing any Tomek links [Tomek, 1976b].
- RandomUndersampler: Rebalances classes by randomly selecting prototypes from each class.
- OneSidedSelection: Rebalances classes by isolating each class and resampling the negative examples (composed of the remaining classes) [Kubat, 1997].

- NeighbourhoodCleaningRule: Improves on OneSidedSelection in settings where particularly small classes are present. As a result, it focuses more on improving data quality than reducing the size of the dataset [Laurikkala, 2001].
- NearMiss: All three versions rebalance classes by resampling the negative examples for a particular class. V1 selects the points from other classes which have the shortest distance to the nearest three points from the target class. V2 selects the points from other classes which have the shortest distance to the furthest three points from the target class. For every point in the target class, V3 selects a fixed number of the nearest points from other classes [Mani and Zhang, 2003].
- InstanceHardnessThreshold: Rebalances classes by fitting a classifier to the data and removing points to which the classifier assigns lower probabilities [Smith et al., 2014].
- EditedNearestNeighbours:Resamples classes by removing points found near class boundaries defined by a fitted classifier [Wilson, 1972].
- RepeatedEditedNearestNeighbours: Resamples classes by repeatedly applying EditedNearestNeighbours and refitting the classifier. [Tomek, 1976a].
- AllKNN: Resamples classes similarly to RepeatedEditedNearestNeighbours but increases the parameter k of the classifier each time [Tomek, 1976a].
- CondensedNearestNeighbours: Rebalances classes by repeatedly fitting a 1NN on the set of candidate prototypes and then adding all misclassified points to that set [Hart, 1968].
- ClusterCentroids: Rebalances classes by using kMeans to replace clusters with their centroids.

For each experiment, the dataset consists of 800 points divided as evenly as possible between the circles. We note that most methods are not able to reduce the number of prototypes much below the number of training points. This is in part due to the automatic class re-balancing that some of these methods attempt to do. Since all classes already have roughly the same number of points, and since none are misclassified when all 800 training points are used, several of the methods determine that little-to-no re-sampling is necessary. As a result, these methods provide at most a small reduction in the number of prototypes. We visualize some of the methods performing automatic undersampling in Figure 5.6, where two common failure modes can be seen: the methods either fail to reduce the number of prototypes but achieve good separation of classes, or reduce the number of prototypes but



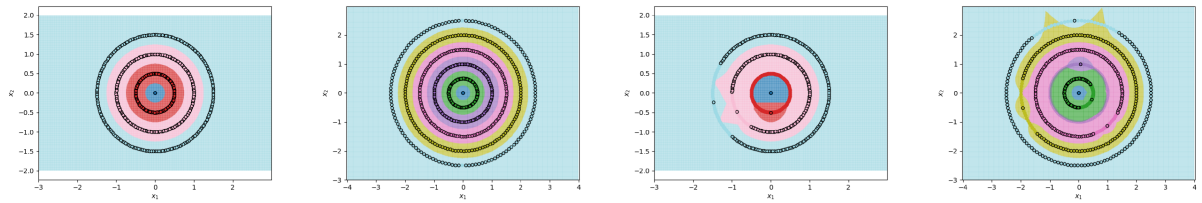
(a) ClusterCentroids **(b)** ClusterCentroids **(c)** Our Algorithm **(d)** Our Algorithm

Figure 5.5: The ClusterCentroids prototype generation method finds similar prototypes to our proposed algorithm when parametrized with the near-optimal number of prototypes per class.

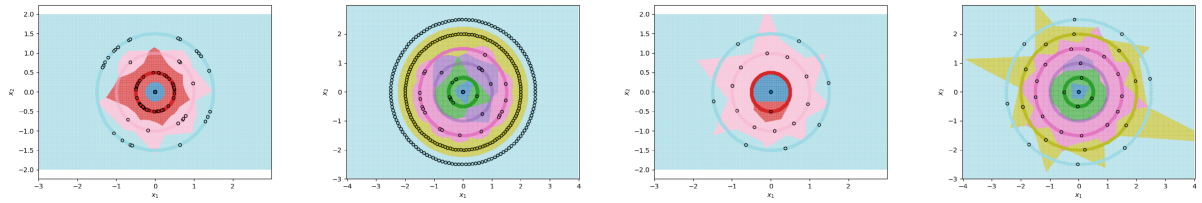
fail to separate the classes. However, the user can also override the automatic re-balancing for a few of the methods, those which include the 'Manual' option in Table 5.1, by passing the number of desired prototypes per class as a hyperparameter. We pass the optimal number of prototypes per class suggested by our earlier theoretical analysis, and the near-optimal number suggested by our algorithm, to these methods and document the results in Figure 5.7. Curiously, none of the prototype selection methods achieve perfect separation when restricted to this nearly optimal number of prototypes, even though the nearly-optimal prototypes found by our algorithm have extremely close-by neighbors among the training points. In other words, it is not theoretically impossible for the prototype selection methods to select prototypes close to where the optimal prototypes would be, and yet they do not. Meanwhile, the ClusterCentroids prototype generation method finds similar prototypes to the ones proposed by our algorithm as seen in Figure 5.5.

Additional Experiments

By using the results of our theoretical analysis to parametrize ClusterCentroids, we enable it to find efficient sets of high-quality prototypes. We combine our proposed algorithm with ClusterCentroids to produce a method that combines the benefits of both: a non-parametric algorithm that finds near-optimal prototypes in our pathological case but is robust to noise. We conduct additional experiments to show that this resulting method is indeed robust to noise. Each experiment still uses a dataset of 800 points that are spread over N concentric, circular classes with radius growth parameter $c = 0.5$; however, we now introduce Gaussian noise to each class. The level of noise is controlled by parameter σ, the standard deviation of the Gaussian distribution underlying the positioning of points within a class. The ratio of σ to c dictates how much overlap occurs between classes. For

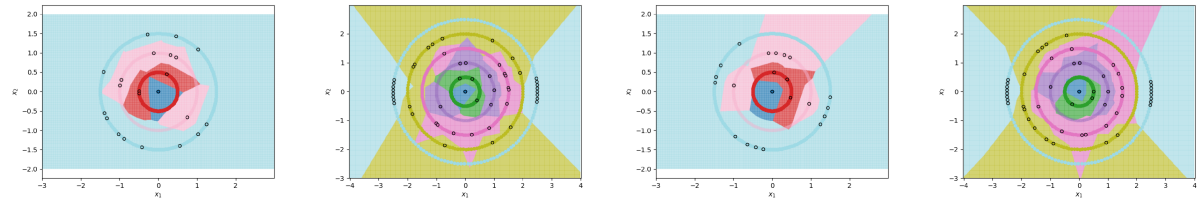


(a) NearMissV3 (4 classes) **(b)** NearMissV3 (6 classes) **(c)** OneSidedSelection **(d)** OneSidedSelection

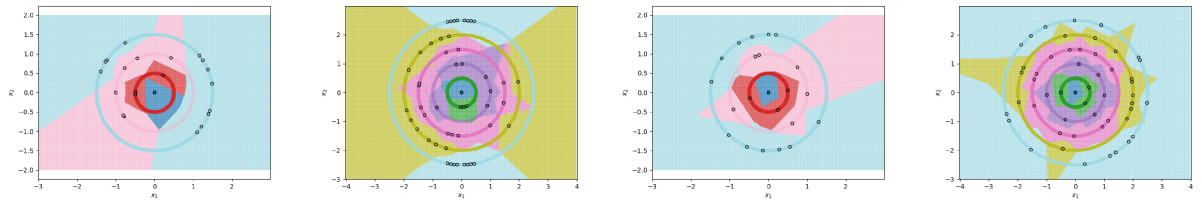


(e) InstanceHardness **(f)** InstanceHardness **(g)** CondensedNN **(h)** CondensedNN

Figure 5.6: Examples of failure modes on four and six-class concentric circles data using prototype methods where number of prototypes per class was found automatically (semi-automatically for the InstanceHardnessThreshold method).



(a) NearMissV1 (4 classes) **(b)** NearMissV1 (6 classes) **(c)** NearMissV2 **(d)** NearMissV2



(e) NearMissV3 **(f)** NearMissV3 **(g)** RandomUnderSampler **(h)** RandomUnderSampler

Figure 5.7: Examples of failure modes on four and six-class concentric circles data using prototype methods for which the number of prototypes per class was set manually.

example, when $\sigma = 0.25 = \frac{c}{2}$, only around 68% of points belonging to a particular class will be contained within the band of thickness $c = 0.5$ associated with that class. We use four levels of noise ($\sigma = 0.05$, 0.1, 0.2, 0.4) and five different numbers of classes (4, 6, 8, 10, 12), for a total of 20 generated datasets to which we apply the near-optimally parametrized ClusterCentroids algorithm and measure classification accuracy.

The results are detailed in Table 5.2 and visualized in Figure 5.8. As expected, increasing noise causes a decrease in classification accuracy. However, the decrease in accuracy is roughly equal to the percentage of points found outside of their class's band as they are indistinguishable from the points of the class whose band they are in. This suggests that the 1NN classifier fitted on prototypes designed by the near-optimally parametrized ClusterCentroids algorithm, approaches the Bayes error rate. We also note that as the number of classes increases, and hence the number of points per class decreases, the accuracy of the classifier stays stable or even increases at high levels of noise. The near-optimally parametrized ClusterCentroids algorithm is clearly robust to increases in the number of classes. It is also partially robust to noise, even though noise violates the underlying assumptions on which the nearly-optimal parametrization is based.

5.2.4 Conclusion

The kNN classifier is a powerful classification algorithm, but can be computationally expensive. While numerous prototype methods have been proposed to alleviate this problem, their performance is often strongly determined by the underlying geometry of the data. Certain pathological geometries can result in especially poor performance of these heuristic algorithms. We analyzed one such extreme setting and demonstrated that analytical methods can be used to find the minimal number of optimal prototypes required for fitting a 1NN classifier. We also found that in such pathological cases, theoretical analysis may not be able to provide the exact locations of the prototypes, but it can be used to derive systems of equations that when solved with numerical methods, produce optimal or near-optimal prototypes.

To demonstrate this approach, we proposed an algorithm for finding nearly-optimal prototypes in the particular pathological setting of concentric circular classes, and used it to validate our theoretical results. The algorithm outperformed all prototype selection methods that it was tested against. A prototype generation method was able to find the optimal prototypes, but only when parametrized using either the theoretical results or the outputs of our proposed algorithm. We further showed that this combination of our proposed algorithm with an existing prototype generation method exhibited the desirable

features of both: not only is it non-parametric, but it is also guaranteed to find near-optimal prototypes even in the examined pathological case. It is also general enough that it is robust to violations of the underlying assumptions of our theoretical analysis, such as the addition of Gaussian noise to the data.

We believe that identifying and studying further pathological geometries in kNN and other machine learning models is an important direction for understanding their failure modes and jointly improving training algorithms and prototype methods.

Noise (σ)	Out-of-band points	Accuracy (4 classes)	Accuracy (6 classes)	Accuracy (8 classes)	Accuracy (10 classes)	Accuracy (12 classes)
0.05	0%	0.975	0.98625	0.9775	0.9775	0.975
0.1	1%	0.94375	0.94	0.94	0.9425	0.93125
0.2	21%	0.77375	0.77625	0.76375	0.79375	0.81375
0.4	53%	0.53	0.4925	0.5325	0.58375	0.60875

Table 5.2: Accuracy of ClusterCentroids parametrized with near-optimal number of prototypes.

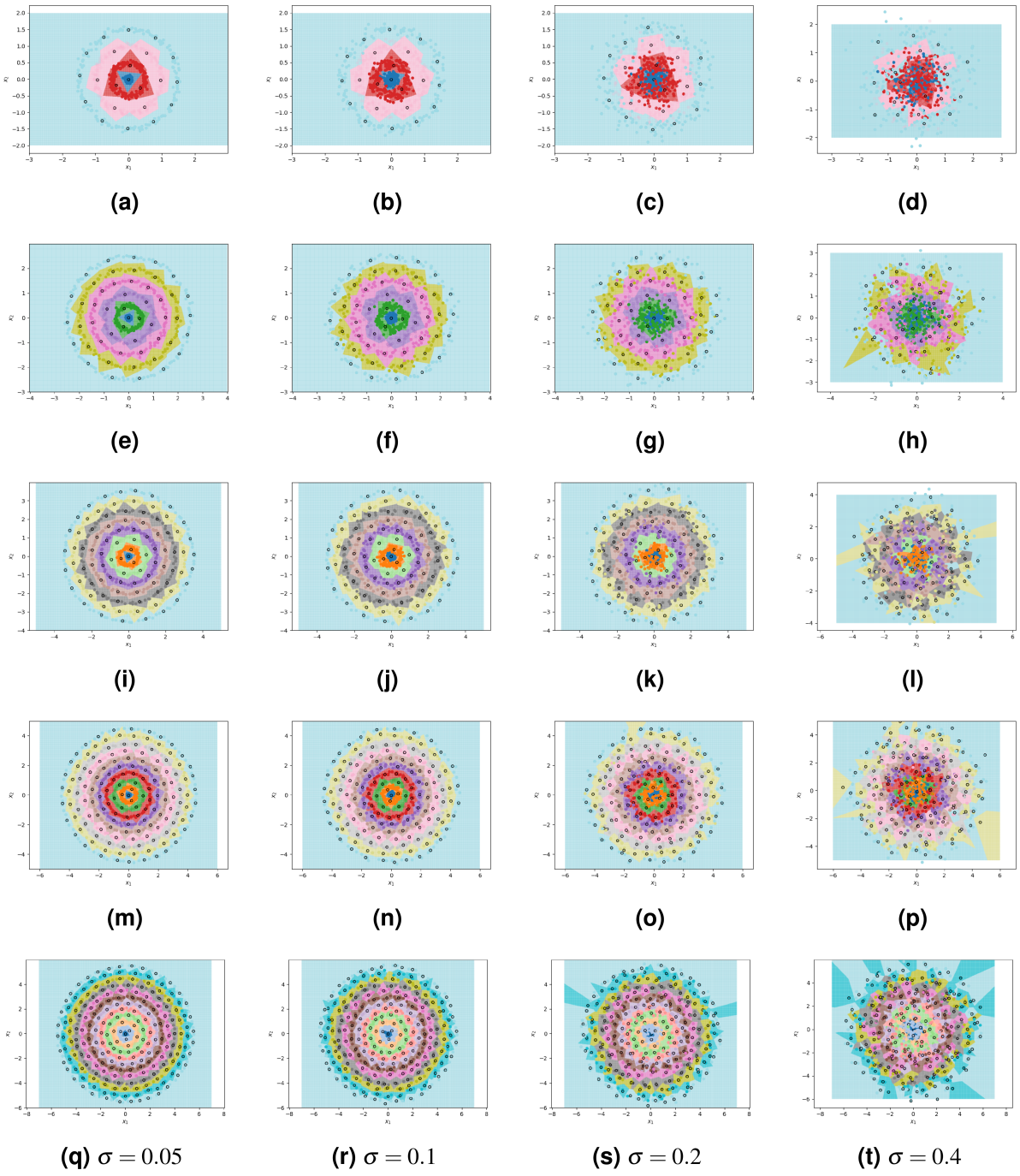


Figure 5.8: ClusterCentroids parametrized with near-optimal number of prototypes applied to various levels of noise. From top to bottom, the rows correspond to 4, 6, 8, 10, and 12 classes. From left to right, columns correspond to $\sigma = 0.05, 0.1, 0.2, 0.4$.

Algorithm 4: FindPUGS Algorithm: Finding (nearly-optimal) Prototypes Using Greedy Search

Result: Two ordered lists, **N** and **R**, of the minimum number of prototypes required for each circle and their rotations relative to the previous circle.

$T \leftarrow$ the number of circles;
$c \leftarrow$ the length by which radii should grow;
Algorithm `FindPUGS(`T, c`)`
    $\mathbf{N} \leftarrow [1]$;
    $\mathbf{R} \leftarrow [0]$;
    for $t = 1, 2, \ldots, T-1$ **do**
        $m \leftarrow \mathbf{N}[-1]$;
        $n \leftarrow m + 1$;
        $p \leftarrow 0$;
        while *True* **do**
            $d_a \leftarrow \sqrt{2t^2c^2 - 2t^2c^2\cos(\frac{\pi}{m})}$;
            $d_b \leftarrow \sqrt{2(t+1)^2c^2 - 2(t+1)^2c^2\cos(\frac{\pi}{n})}$;
            for $i = 0, 1, \ldots, 4mn$ **do**
                $\theta \leftarrow \frac{i\pi}{m*n*16}$;
                if *d1*$(t, c, m, n, \theta) > d_b$ ***and*** *d2*$(t, c, m, n, \theta) > d_a$ **then**
                    $p \leftarrow n$;
                    break;
                end
            end
            if $p > 0$ **then**
                N.append(p);
                R.append(θ);
                break;
            end
            $n \leftarrow n + 1$;
        end
    end
    return N, R;
Procedure `d1(`t, c, m, n, θ`)`
    dists $\leftarrow$ [] ;
    for $i = 0, \ldots, m-1$ **do**
        for $j = 0, \ldots, n-1$ **do**
            dist $\leftarrow \sqrt{t^2c^2 + (t+1)^2c^2 - 2t(t+1)c^2\cos(\frac{2i\pi}{m} - \frac{2j\pi}{n} - \frac{\pi}{n} - \theta)}$;
            dists.append(dist);
        end
    end
    return min(dists);
Procedure `d2(`t, c, m, n, θ`)`
    dists $\leftarrow$ [] ;
    for $i = 0, \ldots, m-1$ **do**
        for $j = 0, \ldots, n-1$ **do**
            dist $\leftarrow \sqrt{t^2c^2 + (t+1)^2c^2 - 2t(t+1)c^2\cos(\frac{2i\pi}{m} - \frac{2j\pi}{n} + \frac{\pi}{m} - \theta)}$;
            dists.append(dist);
        end
    end
    return min(dists);

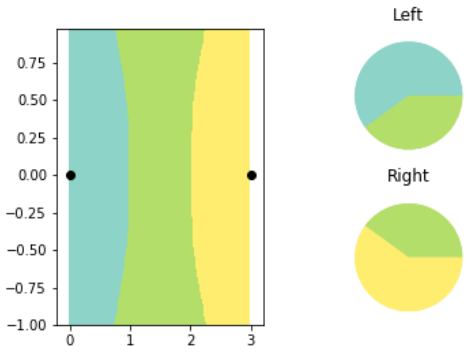


Figure 5.9: A SLaPkNN classifier is fitted on 2 soft-label prototypes and partitions the space into 3 classes. The soft label distribution of each prototype is illustrated by the pie charts.

5.3 Theoretical Foundations of 'Less Than One'-Shot Learning with kNN

In this section, we derive and analyze several methods of configuring M soft-label prototypes to divide a space into N classes using the distance-weighted soft-label prototype k-Nearest Neighbors classifier. An example can be seen in Figure 5.9 where two samples with soft labels are used to separate a space into three classes. All proofs for results in this section can be found in Appendix B.

5.3.1 Definitions

Definition 1 *A **hard label** is a vector of length N representing a point's membership to exactly one out of N classes.*

$$y^{hard} = e_i = \begin{bmatrix} 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{bmatrix}^T$$

Hard labels can only be used when each point belongs to exactly one class. If there are n classes, and some point x belongs to class i, then the hard label for this point is the i^{th} unit vector from the standard basis.

Definition 2 *A **soft label** is the vector-representation of a point's simultaneous membership to several classes. Soft labels can be used when each point is associated with a distribution of classes. We denote soft labels by* y^{soft}.

Definition 2.1 *A **probabilistic (soft) label** is a soft label whose elements form a valid probability distribution.*

$$\forall i \in \{1, ..., n\}\ y_i^{soft} \geq 0$$
$$\sum_{i=1}^{n} y_i^{soft} = 1$$

Definition 2.2 *An **unrestricted (soft) label** is a soft label whose elements can take on any real value, including negative values.*

Definition 3 *A **soft-label prototype (SLaP)** is a pair of vectors* (X, Y) *where* X *is a feature (or location) vector, and* Y *is the associated soft label.*

A probabilistic label can be derived from an unrestricted label by applying the softmax function. A hard label can be derived from a probabilistic label by applying the argmax function (setting the element with the highest associated probability to 1 and the remaining elements to 0). We illustrate this in Figure 4.1.

When using the classical k-Nearest Neighbors (kNN) classifier rule to partition a space, it is clear that at least n points are required to separate n classes (i.e. create n partitions). kNN uses only hard labels, so each of these points, or **prototypes**, contains information only about the single class to which it is assigned. We investigate whether a soft-label prototype generalization of kNN can more efficiently separate n classes by using only $m < n$ soft-label prototypes.

Definition 4 *The **distance-weighted soft-label prototype k-Nearest Neighbors (SLaPkNN)** classification rule takes the sum of the label vectors of the* k*-nearest prototypes to target point* x*, with each prototype weighted inversely proportional to its distance from* x*.* x *is then assigned to the class corresponding to the largest value of the resulting vector.*
More formally, assume we are given a set of M *soft-label prototypes representing a dataset with* N *classes. Let* $S = (X_1, Y_1), ..., (X_M, Y_M)$ *be the prototypes available for training where* X_i *is the position of the* i^{th} *prototype and* Y_i*, a vector of length* N*, is its soft label. Let* x *be*

the position of the target point that we wish to classify. Compute $D = \{d(X_i, x)\}_{i=1,...,M}$*, the set of distances between each prototype and the target. Let* $S' = (X_{(1)}, Y_{(1)}), ..., (X_{(M)}, Y_{(M)})$ *be a reordering of the prototypes such that* $d(X_{(1)}, x) \leq ... \leq d(X_{(M)}, x)$*. Then the distance-weighted sum of the k-nearest prototype labels is* $Y^* = \sum_{i=1}^{k} \frac{Y_{(i)}}{d(X_{(i)}, x)}$ *and* x *is assigned to class* $C^{SLaPkNN}(x) = \arg\max_j Y_j^*$ *where* Y_j^* *is the* j^{th} *element of* Y^**.*

Distance-weighted kNN is the special case of SLaPkNN where each prototype is assigned a hard label.

5.3.2 Probabilistic Prototypes and SLaPkNN with k=2

We first derive and analyze several methods of configuring soft-label prototypes to separate a space into N partitions using M points in the restricted case where prototypes must be probabilistic, and the number of considered neighbors (k) is set to two.

Theorem 8 ***(Learning Three Classes From Two Samples)*** *Assume that two points,* x_1 *and* x_2*, are positioned 3 units apart in two-dimensional Euclidean space and have probabilistic labels* y_1 *and* y_2*, respectively. We denote the* i^{th} *element of each label by* $y_{1,i}$ *and* $y_{2,i}$ *for* $i = 1, 2, 3$*. There exist* y_1 *and* y_2 *such that decision boundaries created by SLaPkNN with* $k = 2$ *separate the space into three classes when fitted on* (x_1, y_1) *and* (x_2, y_2)*.*

Assuming that we want symmetrical labels for the two prototypes (i.e. $y_{1,i} = y_{2,(3-(i-1))}$), the resulting system of linear equations is quite simple.

$$\begin{cases} \frac{2}{3} > y_{1,1} > \frac{1}{2} > y_{1,2} > \frac{1}{3} > y_{1,3} \geq 0 \\ 4y_{1,2} = 1 + y_{1,1} \\ 5y_{1,2} = 2 - y_{1,3} \end{cases} \tag{5.3}$$

Since we have a system of linear equations with one free variable, infinite solutions exist. We set $y_{1,3}$ to zero in order to get a single solution and simplify the resulting label.

$$y_{1,1} = y_{2,3} = \frac{3}{5}$$
$$y_{1,2} = y_{2,2} = \frac{2}{5}$$
$$y_{1,3} = y_{2,1} = \frac{0}{5}$$

We visualize the results of fitting a SLaPkNN classifier with $k = 2$ to a set of two points with these labels in Figure 5.9.

Corollary 9 *Assume that the distance between two points (in two-dimensional Euclidean space), x_1 and x_2, is c, and they have probabilistic labels y_1 and y_2, respectively. We denote the i^{th} element of each label by y_{1i} and y_{2i}. There exist values of y_1 and y_2 such that the decision boundaries created by SLaPkNN with $k = 2$ can separate the space into three classes when fitted on (x_1, y_1) and (x_2, y_2).*

'Every Pair' Methods

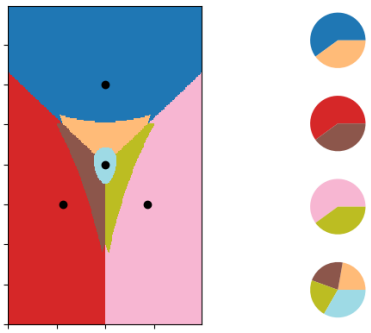
We have shown that a third class can be induced between a pair of points. We now focus on the case where we consider more than two points. We first show that it is possible for a single point to simultaneously belong to multiple pairs, each creating an additional class.

Theorem 10 *Suppose we have M soft-label prototypes $(x_0, y_0), (x_1, y_1), ...(x_{M-1}, y_{M-1})$ with the x_i arranged such that each pair of the form $\{(x_0, x_i)|i = 1, .., M-1\}$ is unique and the other terms are all equidistant from x_0. There exist values of $y_0, y_1, ..., y_{M-1}$ such that SLaPkNN with $k = 2$ can can partition the space into $2M - 1$ classes.*

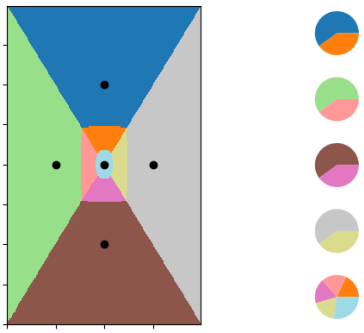
One way to select such labels is to use the same labels for each pair as in Theorem 8, This results in $y_1, ..., y_{M-1}$ each having a label distribution containing two non-zero values: $\frac{3}{5}$ (associated with its main class) and $\frac{2}{5}$ (associated with the class created between itself and x_0). Meanwhile, y_0 contains one element with value $\frac{3}{5}$ (associated with its own class) and M-1 elements with value $\frac{2}{5}$ (each associated with a unique class created between x_0 and each one of the other points). To get probabilistic labels, we can normalize y_0 to instead have values $\frac{3}{2M+1}$ and $\frac{2}{2M+1}$. The resulting decision landscape is visualized in Figure 5.10. The local decision boundaries in the neighbourhood between (x_0, y_0) and any one of the surrounding prototypes then takes the following form.

$$\text{Predicted Class} = \begin{cases} a \text{ if } d < \frac{5p}{4M+7} \\ b \text{ if } d > \frac{10p}{2M+11} \\ c \text{ if } \frac{5p}{4M+7} < d < \frac{10p}{2M+11} \end{cases} \tag{5.4}$$

Examining the asymptotic behavior as the total number of classes increases, we notice a potentially undesirable property of this configuration. Increases in M 'dilute' classes a and c, which results in them shrinking towards x_0. In the limit, only class b remains. It is possible to find a configuration of probabilistic labels that results in asymptotically stable classes but this would require either lifting our previous restriction that the third label value be zero when separating three classes with two points, or changing the underlying geometrical arrangement of our prototypes.



(a) Seven classes using four soft-label prototypes



(b) Nine classes using five soft-label prototypes

Figure 5.10: SLaPkNN can separate $2M-1$ classes using M soft-label prototypes

Proposition 11 *Suppose M soft-label prototypes are arranged as the vertices of an M-sided regular polygon. There exist soft labels $(Y_1, ..., Y_M)$ such that fitting SLaPkNN with $k = 2$ will divide the space into $2M$ classes.*

In this configuration, it is possible to decouple the system from the number of pairs that each prototype participates in. It can then be shown that the local decision boundaries, in the neighbourhood of any pair of adjacent prototypes, do not depend on M.

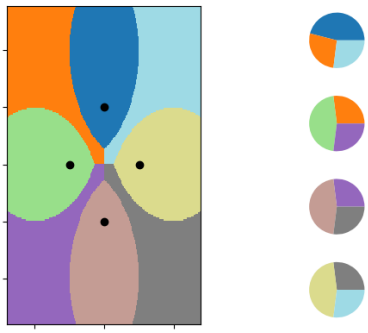
$$\text{Predicted Class} = \begin{cases} a \text{ if } d < \frac{p}{3} \\ b \text{ if } d > \frac{2p}{3} \\ c \text{ if } \frac{p}{3} < d < \frac{2p}{3} \end{cases} \tag{5.5}$$

We visualize the resulting decision landscape in Figure 5.11. By combining these two results, we can produce configurations that further increase the number of classes that can be separated by M soft-label prototypes.

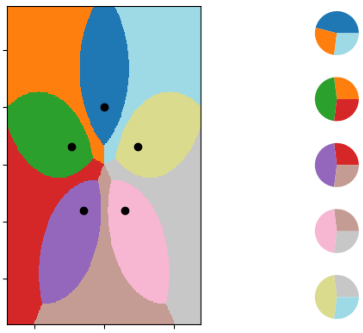
Theorem 12 *Suppose M soft-label prototypes are arranged as the vertices and center of an $(M-1)$-sided regular polygon. There exist soft labels $(Y_1, ..., Y_M)$ such that fitting SLaPkNN with $k = M$ will divide the space into $3M-2$ classes.*

Interval Partitioning Methods

We now aim to show that two points can even induce multiple classes between them.



(a) Eight classes using four soft-label prototypes



(b) Ten classes using five soft-label prototypes

Figure 5.11: SLaPkNN can separate $2M$ classes using M soft-label prototypes

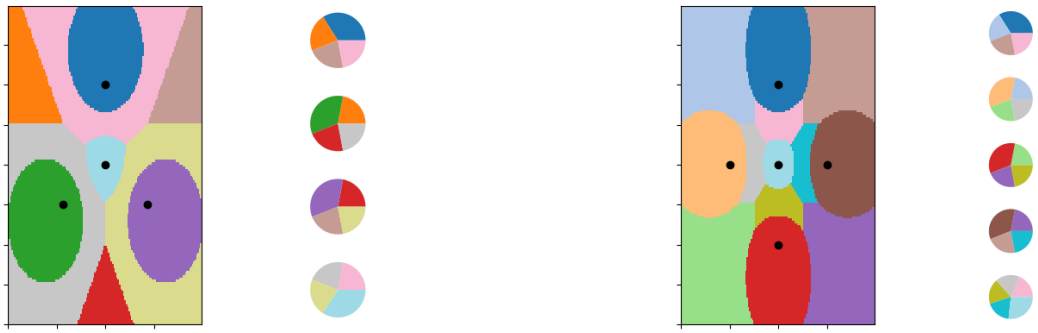
Lemma 13 *Assume that two points are positioned 4 units apart in two-dimensional Euclidean space. Without loss of generality, suppose that point* $x_1 = (0, 0)$ *and point* $x_2 = (4, 0)$ *have probabilistic labels* y_1 *and* y_2 *respectively. We denote the* i^{th} *element of each label by* $y_{1,i}$ *and* $y_{2,i}$*. There exist values of* y_1 *and* y_2 *such that SLaPkNN with* $k = 2$ *can separate four classes when fitted on* x_1 *and* x_2*.*

We can again find a solution to this system that has symmetrical labels and also sets an element of the label to zero.

$$y_{1,1} = y_{2,4} = \frac{6}{14}$$
$$y_{1,2} = y_{2,3} = \frac{5}{14}$$
$$y_{1,3} = y_{2,2} = \frac{3}{14}$$
$$y_{1,4} = y_{2,1} = \frac{0}{14}$$

We visualize the results of fitting a SLaPkNN classifier with $k = 2$ to a set of two points with these labels in Figure 5.13.

We can further extend this line of reasoning to produce the main theorem of this section.



(a) Ten classes using four soft-label prototypes

(b) Thirteen classes using five soft-label prototypes

Figure 5.12: SLaPkNN can separate $3M - 2$ classes using M soft-label prototypes

Theorem 14 ***(Main Theorem)*** *SLaPkNN with $k = 2$ can separate $n \in [1, \infty)$ classes using two soft-label prototypes.*

This unexpected result is crucial for enabling extreme LO-shot learning as it shows that in some cases we can completely disconnect the number of classes from the number of prototypes required to separate them. The full proof can be found in the supplemental materials, but we provide a system of equations describing soft labels that result in two soft-label prototypes separating $n \in [1, \infty)$ classes.

$$y_{1,i} = y_{2,(n-i)} = \frac{\sum_{j=i}^{n-1} j}{\sum_{j=1}^{n-1} j^2} = \frac{n(n-1) - i(i-1)}{2\sum_{j=1}^{n-1} j^2},$$
$$i = 1, 2, ..., n$$

Other Results

The diverse decision landscapes we have already seen were generated using probabilistic labels and $k = 2$. Using unrestricted soft labels and modifying k enables us to explore a much larger space of decision landscapes. Due to space constraints we present only a few of the results from our experiments with modifying these parameters, but more are included in the accompanying supplemental material. One unexpected result is that SLaPkNN can generate any number of concentric elliptical decision bounds using a fixed number of

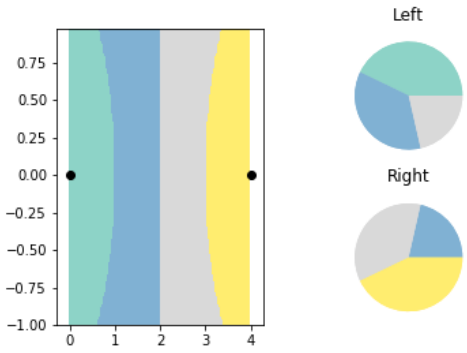


Figure 5.13: A SLaPkNN classifier is fitted on two points and used to partition the space into four classes. The probabilistic soft labels of each point are illustrated by the pie charts.

prototypes as seen in Figures 5.14a and 5.14b. In Figure 5.14d, we show that variations in k can cause large changes in the decision landscape, even producing non-contiguous classes.

5.3.3 Robustness

Several studies have been conducted on the robustness and stability of kNN classifiers. El Gayar et al. [2006] found that that using soft labels with kNN resulted in classifiers that were more robust to noise in the training data. Sun et al. [2016] proposed a nearest neighbor classifier with optimal stability based on their extensive study of the stability of hard-label kNN classifiers for binary classification. Our robustness and stability analyses focus specifically on the LO-shot learning setting which has never previously been explored.

In order to understand the robustness of LO-shot learning landscapes to noise, we aim to analyze which regions are at highest risk of changing their assigned class if the prototype positions or labels are perturbed. We use the difference between the largest predicted label value and second-largest predicted label value as a measure of confidence in our prediction for a given point. The risk of a given point being re-classified is inversely proportional to the confidence. Since our aim is to inspect the entire landscape at once, rather than individual points, we visualize risk as a gradient from black (high confidence/low risk) to white (low confidence/high risk) over the space. We find that due to the inverse distance weighting mechanism, there are often extremely high confidence regions directly surrounding the prototypes, resulting in a very long-tailed, right-skewed distribution of confidence values. As a result, we need to either clip values or convert them to a logarithmic scale in order to be

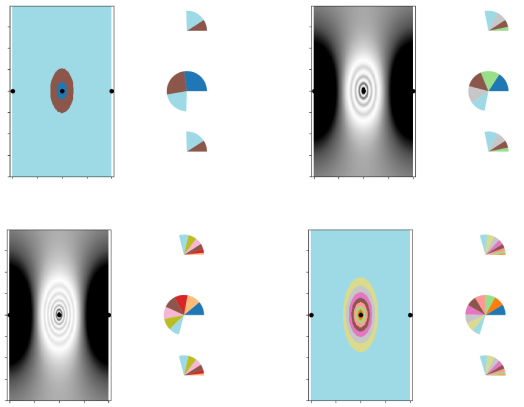
able to properly visualize them. We generally find that clipping is helpful for understanding intra-class changes in risk, while the log scale is more suited for visualizing inter-class changes in risk that occur at decision boundaries.

Figure 5.14 visualizes the risk gradient for many of the decision landscapes derived above. Overall, we find that LO-shot learning landscapes have lower risk in regions that are distant from decision boundaries and lowest risk near the prototypes themselves. Changes in k can have a large effect on the inter-class risk behaviour which can be seen as changes in the decision boundaries. However, they tend to have a smaller effect on intra-class behavior in regions close to the prototypes. The most noticeable changes occur when increasing k from 1 to 2, as this causes a large effect on the decision landscape itself, and from 2 to 3, which tends to not affect the decision landscape but causes a sharp increase in risk at decision boundaries. Meanwhile, increasing the number of classes (M) can in some cases cause large changes in the intra-class risk behaviour, but we can design prototype configurations (such as the ellipse generating configuration) that prevent this behaviour. In general, it appears that by tuning the prototype positions and labels we can simultaneously elicit desired properties both in the decision landscape and risk gradient. This suggests that LO-shot learning is not only feasible, but can also be made at least partially robust to noisy training data.

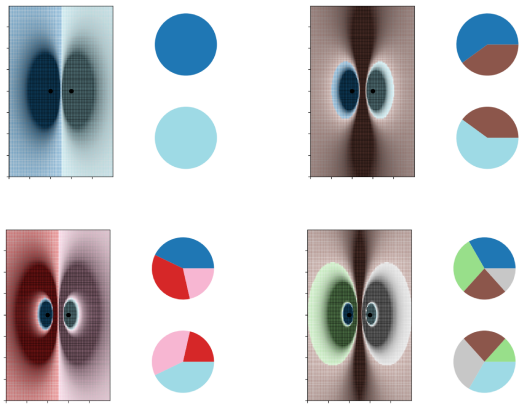
5.3.4 Case Study: Prototype Generation for Circles

We consider the pathological circle dataset from the previous section as it adeptly captures the difference between hard and soft-label prototype generation. As mentioned, we tested several hard-label prototype methods and found their performance to be poor on this simulated data. Many of the methods produced prototypes that did not achieve separation between classes thus preventing us from performing a fair comparison between hard and soft-label prototypes. In order to allow such a comparison, we use the analytically derived near-optimal hard-label prototype configuration for fitting a 1NN classifier to the circle data.

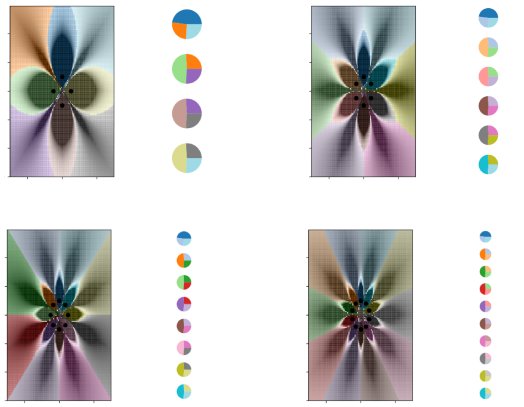
As previously shown, the number of required hard-label prototypes is quadratic in the number of classes. However, SLaPkNN requires only a constant number of prototypes to generate concentric ellipses. In Figure 5.15, SLaPkNN fitted on five soft-label prototypes is shown separating the same six circles that were used with hard-label prototypes and 1NN in Figure 5.1. The decision boundaries created by SLaPkNN match the underlying geometry of the data much more accurately than those created by 1NN. This is because 1NN can only create piecewise-linear decision boundaries.



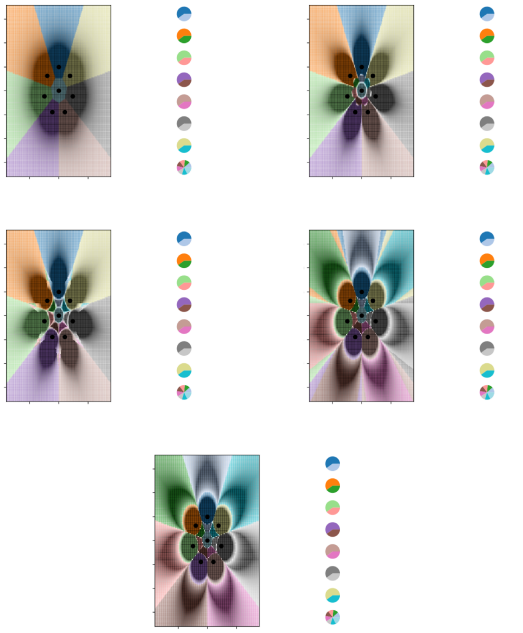
(a) SLaPkNN with $k = 3$ is shown separating $N = 3, 5, 7, 9$ classes (left to right, top to bottom) with elliptical decision boundaries after being fitted on 3 prototypes. The pie charts represent unrestricted labels.



(b) SLaPkNN with $k = 2$ is shown separating $N = 2, 3, 4, 5$ classes (left to right, top to bottom) after being fitted on 2 prototypes. The pie charts represent probabilistic labels.



(c) SLaPkNN with $k = 2$ is shown separating $2M$ classes after being fitted on $M = 4, 6, 8, 10$ prototypes (left to right, top to bottom). The pie charts represent probabilistic labels.



(d) SLaPkNN with $k = 1, 2, 3, 4, 5$ (left to right, top to bottom) is shown separating 15 classes after being fitted on 8 prototypes. The pie charts represent unrestricted labels.

Figure 5.14: Various LO-shot learning decision landscapes and risk gradients are presented. Each color represents a different class. Gray-scale is used to visualize the risk gradient, with darker shadows corresponding to lower risk. In (a), the two colorful charts show decision landscapes and the two gray-scale charts show risk landscapes. In (b)-(d), the risk gradient is laid directly over the decision landscape.

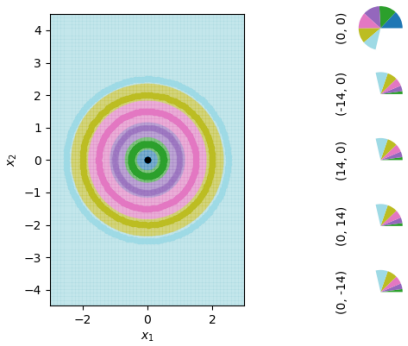


Figure 5.15: SLaPkNN can separate 6 circles using 5 soft-label prototypes. Each pie chart represents the soft label of one prototype, and is labeled with its location. 4 of the prototypes are located outside of the visible range of the chart.

In this case study, enabling soft labels reduced the minimal number of prototypes required to perfectly separate N classes from $\mathcal{O}(N^2)$ down to $\mathcal{O}(1)$. We note that the number of required hard-label prototypes may be reduced by carefully tuning k as well as the neighbor weighting mechanism (e.g. uniform or distance-based). However, even in the best case, the number of required hard-label prototypes is at the very least linear in the number of classes.

5.3.5 Conclusion

We have presented a number of results that we believe can be used to create powerful and efficient dataset condensation and prototype generation techniques. More generally, our contributions lay the theoretical foundations necessary to establish 'less than one'-shot learning as a viable new direction in machine learning research. We have shown that even a simple classifier like SLaPkNN can perform LO-shot learning, and we have proposed a way to analyze the robustness of the decision landscapes produced in this setting. We believe that creating a soft-label prototype generation algorithm that specifically optimizes prototypes for LO-shot learning is an important next step in exploring this area. Such an algorithm will also be helpful for empirically analyzing LO-shot learning in high-dimensional spaces where manually designing soft-label prototypes is not feasible.

Additionally, we are working on showing that LO-shot learning is compatible with a large variety of machine learning models. Improving prototype design is critical for

speeding up instance-based, or lazy learning, algorithms like kNN by reducing the size of their training sets. However, eager learning models like deep neural networks would benefit more from the ability to learn directly from a small number of real samples to enable their usage in settings where little training data is available. This remains a major open challenge in LO-shot learning.

5.4 Algorithms for 'Less Than One'-Shot Learning with kNN

5.4.1 Introduction

While the results above have shown that LO-shot learning is theoretically plausible, a practical algorithm for harnessing its potential is yet to be developed. Our main theorem from the previous section showed that with just two carefully-designed soft-label prototypes it is possible to separate any finite number of classes. However, this particular result requires that the classes being separated lie roughly on a 1-dimensional manifold so that a soft-label prototype could be assigned to each end of the manifold and used to separate the classes in between. Thus, we propose the first method for generating prototypical lines and a new kNN classification algorithm that can use them to perform LO-shot learning in practice. Our key contributions in this section can be summarized as follows:

- We develop three methods for finding co-linear classes.
- We develop a method for producing 'prototypical lines' by optimizing the two soft-label prototypes assigned to each set of approximately co-linear classes.
- We develop a novel classification algorithm, the Hierarchical Soft-Label Prototype kNN (HSLaPkNN), that can be fitted on the prototypical lines produced by these two components.
- We show that HSLaPkNN can perform LO-shot learning with prototypical lines and determine the tradeoff between dataset size reduction and classification accuracy. In particular, our method can retain over 90% of the classification accuracy of 1NN while reducing the required number of prototypes (prototypical lines) by up to 80%.

Our modular approach allows newly-developed algorithms for each component to be swapped in without interfering with the remaining components. This implies that our approach accommodates a continuous performance improvement through component-wise innovations.

This is indeed an attractive feature for researchers, as the progress on individual components can by combined into the improvement of overall process. The remainder of this section is divided into four subsections. In Section 5.4.2 we detail our method, each of its components, and the theory behind them. In Section 5.4.3 we describe our experimental setup and results. In Section 5.4.4 we analyze the impact of our method and suggest promising directions for future work.

5.4.2 LO-Shot Prototype Generation Algorithm

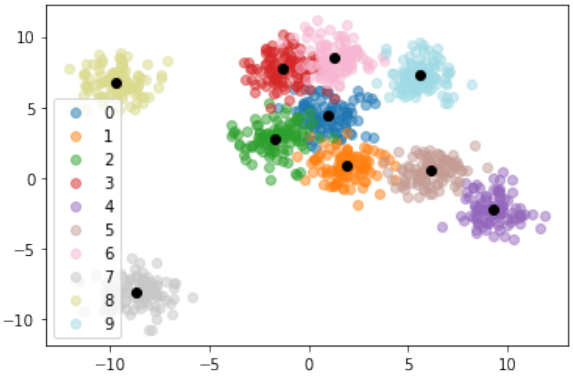
Our method primarily consists of three modular components. In the first component, we find a small set of prototypical line segments that cover all classes (i.e. each class is associated with a line). The second component generates two soft-label prototypes for each line, one at each endpoint, that are optimized based on the subset of classes assigned to that line. Finally, the HSLaPkNN algorithm uses the lines and prototypes to classify the dataset. We present a visual illustration of each component in Figure 5.16.

Component 1: Finding Lines

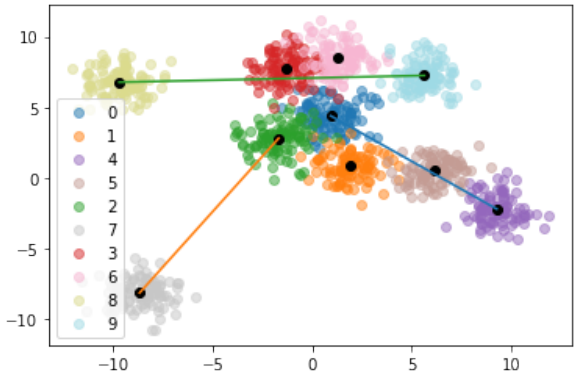
The objective of the first component of our prototype generation method is to find subsets of classes that lie along the same 1-dimensional manifold. We can find subsets of this type by grouping together classes that are approximately co-linear. In other words, our objective is roughly to find the smallest possible set of lines that cover (pass through) all classes. We propose three methods for finding such lines.

The search for observations lying on a line can be dated back to the analysis of multicollinearity in linear regression Belsley et al. [2005]. Conventionally, multicollinearity is a topic of concern in modelling due to it resulting in a verbose model. However, identifying co-linear observations could be useful in finding an efficient representation. If the classes in a dataset could be grouped by various co-linear structures, then a representation of arbitrarily many classes using a much smaller number of lines may be possible. There has been significant past work on covering points with various geometrical objects Langerman and Morin [2001, 2005], Grantson and Levcopoulos [2006], Genç et al. [2011], Dumitrescu and Jiang [2015], Carmi et al. [2007], Mahapatra et al. [2007], Ahn et al. [2011]. Our approach draws contrast from these methods in that we seek to use lines as a satisfactory approximation of a multi-class dataset, instead of a precise covering of all points.

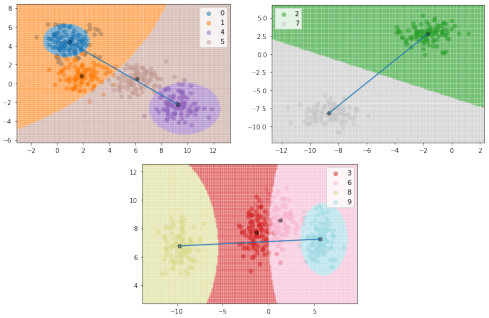
(a) Select a dataset



(b) Find 'good' covering lines



(c) Solve system to find two soft-label prototypes for each line



(d) Classify points based on nearest line and its two prototypes

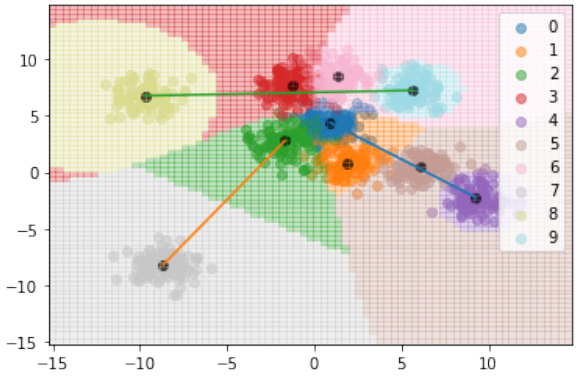


Figure 5.16: Prototype generation and classification process Hierarchical Soft-Label Prototype k-Nearest Neighbors (HSLaPkNN)

Brute Force One straightforward approach for finding the best combination of M lines that pass through/near all the classes is to generate all such combinations of lines, score each one, and finally pick the best one. To reduce interactions between different lines in Component 3, we first filter out all combinations of lines where any of the lines intersect. For our experiments with this approach, we score each line by first finding all the classes that are closest to that line, and then taking the sum of the absolute values of the shortest distances from every point in those classes to the line. The scores are summed across each combination of lines, and the combination with the lowest score is chosen. This process is detailed in Algorithm 5. Unfortunately, the complexity of this algorithm is $O(n^{2*l})$ where n is the number of classes and l is the number of lines. While computationally expensive, it is guaranteed to find near-optimal lines. As a result, this method is best used when there are either relatively few classes or only a small number of lines is required to cover all of them. For datasets with a large number of classes that cannot be covered with a small number of lines, we instead propose an approximate method.

Recursive Regression The exhaustive enumeration like the brute force approach can quickly lose its viability as the number of classes increases. Instead, a preliminary clustering of the classes can produce a more elegant algorithm for line-finding. Let N be the number of classes present, and denote by $c_1, \ldots, c_N$ the class-wise centroids. We want to find M lines. The set $\{c_1, \ldots, c_N\}$ is partitioned into M clusters such that each cluster contains at least two centroids. We use hierarchical clustering with the single linkage Everitt et al. [2011], but a different method could be deployed based on the investigator's judgement. For the purpose of regression fit, we selected the last feature of the dataset to be the 'response' and the rest of the features to be the 'covariate'. This means that, for a centroid c, its last entry could function as a response variate and the remaining entries would be the covariates. However, the response-covariate configuration may be different based on the context or other external information. As an illustration, consider one of the clusters G_i with size n_i, $G_i = \{c_{i1}, c_{i2}, \ldots, c_{in_i}\}$, where each element in G_i is a centroid. We partition G_i into $A_i = \{c_{i1}, c_{i2}\}$ and $B_i = G_i \setminus A_i$ where c_{i1} and c_{i2} are of maximum pairwise Euclidean norm between the centroids in G_i. We fit a regression line β_i on c_{i1} and c_{i2}. Then, for each $c \in B_i$, we estimate a line $\hat{\beta}$ on $A_i \cup \{c\}$. Then, if $||\beta - \hat{\beta}||_2$ is less than a pre-determined tolerance $\epsilon > 0$, then c is added to A_i and B_i is updated to $B_i \setminus \{c\}$. Otherwise, c is not added to A_i and is discarded from B_i. This forward selection process is repeated until $B_i = \emptyset$. This procedure is applied to all G_i for $i = 1, 2, \ldots, M$. The results are $\{A_1, \ldots, A_M\}$ and $\{\beta_1, \ldots, \beta_M\}$, where each line β_j $(j = 1, \ldots, M)$ is segmented to have endpoints at the furthest-apart pair of centroids that generated it. This method will be called Recursive Regression (RR) hereafter.

Algorithm 5: Brute-force line-finding algorithm

```
Result: Best set of M non-intersecting lines (as pairs of endpoints) for covering
        all classes
M = desired number of lines;
C = centroids of each class;
Lines = all combinations of two elements of C;
M_Lines = all combinations of M elements of Lines;
best_lines = None;
min_dist = -1;
for cur_lines in M_Lines do
    if no_intersections(cur_lines) then
        cur_dist=0;
        for c in C do
            nearest = nearest_line(c, cur_lines);
            cur_dist += dist_to_line(c, nearest);
        end
        if cur_dist ¡ min_dist or min_dist==-1 then
            min_dist = cur_dist;
            best_lines = cur_lines;
        end
    end
end
return best_lines;
```

Algorithm 6: Recursive Regression (RR) for line-finding

Result: A set of M lines and classes assigned to each line
$\mathbf{C}$ = centroids of each class;
M = number of preliminary clusters that contain at least two different centroids from $\mathbf{C}$;
$\mathbf{G}_i$ (i = 1, 2, . . . , M) = i^{th} preliminary cluster;
ϵ = pre-determined positive maximum tolerance;
best_lines = $\emptyset$;
captured_groups = $\emptyset$;
for *i* ***in*** *1, 2, . . . , M* **do**
    $\mathbf{A}$ = set of two furthest centroids in $\mathbf{G}_i$ in terms of Euclidean distance;
    $\mathbf{B} = \mathbf{G}_i \setminus \mathbf{A}$;
    β = regression line fitted on two furthest centroids in $\mathbf{G}_i$;
    while $\boldsymbol{B} \neq \emptyset$ **do**
        all_dist = $\{||\beta - \beta_{\mathbf{A}\cup\{c\}}||_2\}_{c\in\mathbf{B}}$ where $\beta_{\mathbf{A}\cup\{c\}}$ is the regression line fitted on $\mathbf{A} \cup \{c\}$;
        if *all(all_dist ¿ ϵ)* **then**
            stop
        end
        c^* = minimizer among $\mathbf{B}$ of all_dist ;
        $\mathbf{A} = \mathbf{A} \cup c^*$;
        $\mathbf{B} = \mathbf{B} \setminus c^*$;
    end
    best_lines = best_lines $\cup$ β;
    captured_groups = captured_groups $\cup$ {classes_associated_with_$\mathbf{A}$};
end
return best_lines, captured_groups;

Distance-based Attraction While the Recursive Regression is careful when adding classes to the distilling lines, a possible shortfall is its dependence on the initial clustering result, because the cluster-wise regression lines are not altered. Moreover, the tolerance threshold ϵ can influence the number of classes that are left out. To mitigate these issues, we propose another clustering-based algorithm called Distance-based Attraction (DA). The initial clustering stage is similar to that of RR, and we obtain M line segments $\beta_1, \ldots, \beta_M$. Then, for each centroid c_i $(i = 1, \ldots, N)$, we compute the shortest distance from it to each line segment, denoted by $d_{\beta_1}, \ldots, d_{\beta_M}$. Then, c_i is assigned to the line $\text{argmin}_{\beta_1,\ldots,\beta_M}\{d_{\beta_1}, \ldots, d_{\beta_M}\}$. Notice that we do not require a tolerance threshold, and that every class is guaranteed to be assigned to a line. Once the assignment is completed, the line segment for each cluster is re-calculated, as there may be a new furthest-apart pair of centroids.

Algorithm 7: Distance-based Attraction (DA) for line-finding

Result: A set of M lines and classes assigned to each line
$\mathbf{C} = \{c_1, \ldots, c_N\}$ = centroids of each class;
M = number of preliminary clusters that contain at least two different centroids from $\mathbf{C}$;
$\mathbf{G}_i$ (i = 1, 2, ..., M) = i^{th} preliminary cluster;
pre_lines = line segments generated from furthest-apart pair of centroids from each $\mathbf{G}_i$;
best_lines = $\emptyset$;
group_assignment = $\emptyset$;
for *i* ***in*** *1, 2, ..., N* **do**
    index = $\underset{j=1,\ldots,M}{\text{argmin}}(\text{shortest\_dist}(c_i, \beta_j))$;
    group_assignment = group_assignment $\cup$ {index};
end
for *i in unique_elements(group_assignment)* **do**
    set = $\{c_j : \text{group\_assignment}[j-1] = i\}$;
    β_i^* = line_on_furthest_pair(set);
    best_lines = best_lines $\cup$ β_i^*;
end
return best_lines, group_assignment;

Component 2: Finding Optimal Prototypes for a Single Line

Once a suitable line segment is found, along with the classes assigned to it, we can use the idea of the main theorem from Sucholutsky and Schonlau [2021a] to design two soft-label prototypes that will be placed at each endpoint of the line segment. The soft labels of these two prototypes must be designed in such a way that a SLaPkNN classifier fitted on them would accurately separate the classes lying along the line segment. Similarly to Sucholutsky and Schonlau [2021a] we can formulate this as an optimization problem where we want to maximize each class's influence over its interval of the line segment. We approximate each class's interval of the line segment as starting from the midpoint between the class centroid and the preceding class's centroid, and ending at the midpoint between the class centroid and the next class's centroid. We approximate a class's influence over its interval by its influence at the centroid of that interval. A class's influence at a given point is equal to the sum of the associated soft-label value divided by distance from the prototype, for each prototype. For a point to be assigned to a particular class, that class's influence must be higher than all the other classes' influences at that point. To enforce this, at each interval centroid we not only add a constraint forcing the desired class to have the highest influence, but we also actually maximize the *difference* between the influence of the desired classes and the sum of the influences of all the other classes. A key difference between the system we aim to solve and the one solved in Sucholutsky and Schonlau [2021a] is that we do not assume that classes will be distributed symmetrically along the line segment. As a result, we cannot use the simplifying constraints that the soft labels of the two prototypes are symmetrical. Instead, we add the additional constraint that the influence of neghboring classes must be equal at the midpoint of their centroids. In order to solve the resulting optimization problem we use the CVXPY library Diamond and Boyd [2016], Agrawal et al. [2018]. The full algorithm for generating and solving this optimization problem is specified in Algorithm 8.

Component 3: Classifying with Multiple Lines

The two prototypes assigned to the endpoints of a single line are near-optimal for fitting a SLaP2NN classifier if that line and its associated classes are isolated from all other classes. However, in practice, lines could pass fairly close to each other. As a result, the two nearest prototypes to a particular point on a line may not end up being the two prototypes assigned to the endpoints of that line. In order to rectify this problem, we propose the Hierarchical Soft-Label Prototype kNN (HSLaPkNN) classification rule. This classifier performs two main steps when determining how to classify a target point. First, it finds

Algorithm 8: Generating system of equations and constraints for two soft-label prototypes

Result: Two lists containing the soft labels corresponding to the two prototypes
p1 = location of first prototype, p2 = location of second prototype;
N = number of classes assigned to line, centroids = list of class centroids;
x = length 2N array of variables to optimize;
$\epsilon = 0.01$;
projections = [], dists = [], middists=[], A=[];
constraints=[x $\geq$ 0, x $\leq$ 1, sum(x[0:N])==1, sum(x[N:2N])==1];
for *centroid* ***in*** *centroids* **do**
    projections.append(project(centroid, [p1, p2]));
    dists.append(dist(p1, projection));
    middists.append(dist(p1, projection)/2);
end
for *i* ***in*** *0, 1, 2, . . . , N-1* **do**
    vector = zeros(2*N) ;
    vector[i] += 1/(dists[i]+ϵ-p1) ;
    vector[N+i] += 1/(p2-dists[i]+ϵ) ;
    q1 = x[i]/(dists[i]+ϵ-p1) + x[N+i]/(p2-dists[i]+ϵ);
    for *j* ***in*** *0, 1, 2, . . . , N-1* **do**
        if $i \neq j$ **then**
            vector[j] -= 1/(dists[i]+ϵ-p1);
            vector[N+j] -= 1/(p2-dists[i]+ϵ);
            q2=x[j]/(dists[i]+ϵ-p1) + x[N+j]/(p2-dists[i]+ϵ);
            constraints.append(q1 $\geq$ q2+ϵ^2);
        end
        A.append(vector);
    end
    if $i <$ *N-1* **then**
        q1 = x[i]/(mid_dists[i+1]-p1) + x[N+i]/(p2-mid_dists[i+1]);
        q2 = x[i+1]/(mid_dists[i+1]-p1) + x[N+i+1]/(p2-mid_dists[i+1]);
        constraints.append(q1==q2);
    end
end
objective = Maximize(sum($A'x$)+sum_smallest($A'x$,2));
solve(objective, constraints);
return x.value[0:N], x.value[N:2N];

the nearest prototype line to the target point. Second, it fits a SLaP2NN classifier on the two endpoint prototypes assigned to that line. These steps are detailed in Algorithm 9. We note that the algorithm is intentionally designed to allow more than the minimum required two prototypes per line in case the user wishes to improve accuracy by adding additional prototypes.

Algorithm 9: Hierarchical Soft-Label Prototype k-Nearest Neighbor (HSLaP-kNN) classification rule

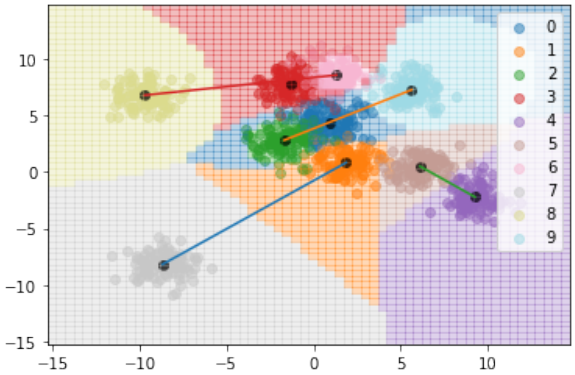
```
Result: Predicted classes of every target point in P
P = list of target points for classification;
lines = list of M pairs of prototypes;
preds = [];
for p in P do
    nearest = nearest_line(p, lines);
    soft_pred = [0]*N;
    for prototype in nearest do
        proto_loc = prototype[0];
        proto_lab = prototype[1];
        dist = dist(proto_loc, p);
        soft_pred += proto_lab/dist;
    end
    hard_pred = argmax(soft_pred);
    preds.append(hard_pred);
end
return preds;
```

5.4.3 Experiments

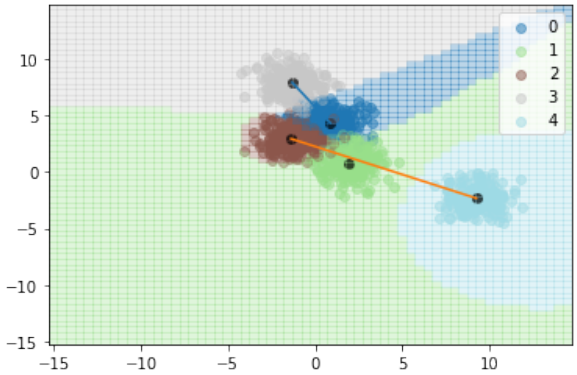
We perform a variety of experiments to determine the tradeoff between classification accuracy and dataset size reduction offered by our prototyping algorithm and HSLaPkNN. For every experiment, we also fit a normal 1NN classifier as a baseline. Each type of experiment is summarized below and some examples of the resulting classification decision boundaries are visualized in Figures 5.17 and 5.18.

- Regular1: Brute force line-finding is used to find three lines in a 10-class dataset consisting of 1000 points with two feature dimensions. Each class consists of 100

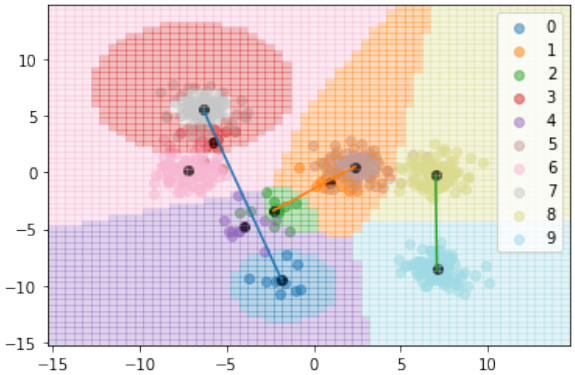
(a) Example result of Regular2 experiment



(b) Example result of Regular (5) experiment



(c) Example result of Imbalanced1 experiment



(d) Example result of Imbalanced2 experiment

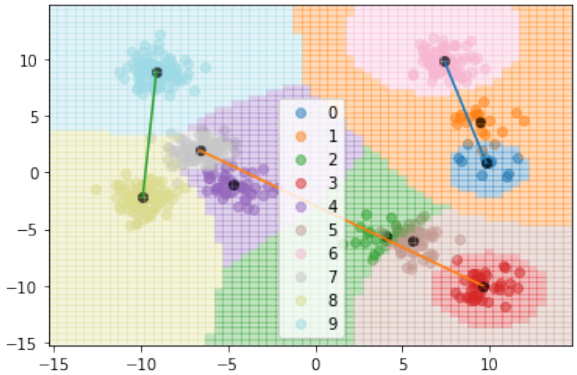


Figure 5.17: Examples of resulting HSLaPkNN decision landscapes.

points.

- Regular2: Brute force line-finding is used to find four lines in a 10-class dataset consisting of 1000 points with two feature dimensions. Each class consists of 100 points.

- Regular (5): Brute force line-finding is used to find two lines in a 5-class dataset consisting of 1000 points with two feature dimensions. Each class consists of 200 points.

- Giant: Distance-based attraction is used to find some number of lines that cover all classes in a 100-class dataset consisting of 2000 points with two feature dimensions. Each class consists of 20 points. For this experiment we also record the average number of lines found along with the other metrics.

- Imbalanced1: Brute force line-finding is used to find three lines in a 10-class dataset consisting of 550 points with two feature dimensions. Five classes consist of 10 points each, and five classes consist of 100 points each.

- Imbalanced2: Brute force line-finding is used to find three lines in a 10-class dataset consisting of 550 points with two feature dimensions. Class i consists of $10i$ points for $i = 1, 2, ..., 10$.

- Small: Brute force line-finding is used to find three lines in a 10-class dataset consisting of 100 points with two feature dimensions. Each class consists of 10 points.

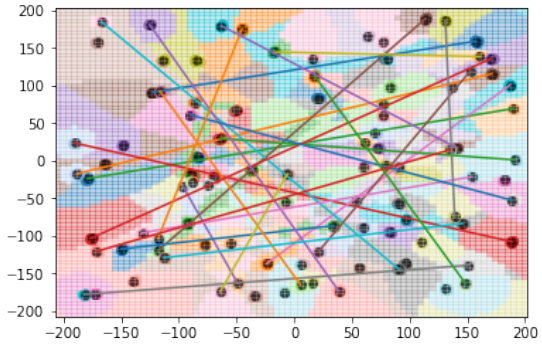
- Penguins: Distance-based attraction is used to find some number of lines that cover all classes in the 5-class version of the Palmer Penguins dataset Horst et al. [2020]. We use the four continuous explanatory variables (bill length, bill depth, flipper length, body mass) as the features, and the combination of 'species' and 'island' as the class.

- EColi: Distance-based attraction is used to find some number of lines that cover all classes in the 5-class version of the E. Coli dataset ('ecoli') from OpenML Vanschoren et al. [2013]. We use the six continuous explanatory variables (mcg, gvh, lip, aac, alm1, alm2) as the features.

We repeat each experiment involving simulated data 100 times with a different random seed and record the mean and standard deviation of the classification accuracy in Table 5.3, along with other details about each experiment. In order to understand the tradeoff between dataset size reduction and classification accuracy, we also calculate the

ratio of the number of prototypical lines used by HSLaPkNN compared to 1NN, as well as the ratio of their classification accuracies when fitted on these prototypes. We summarize these results in Table 5.4. Notably, our method retains upwards of 90% of the classification accuracy of 1NN while reducing the number of nearest prototypes (prototypical lines) that must be considered at inference time by as much as 80%.

(a) Example result of Giant experiment with 25 lines found



(b) Example result of Giant experiment with 30 lines found

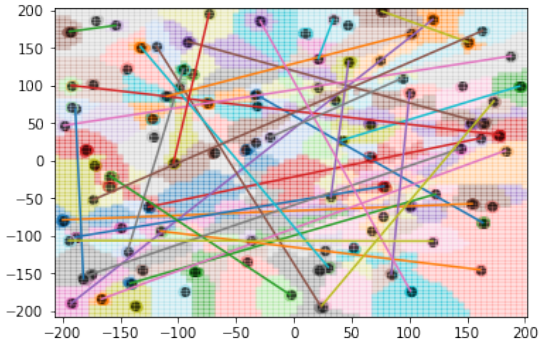


Figure 5.18: Examples of resulting HSLaPkNN decision landscapes.

Our method is most interpretable when working with datasets that have two-dimensional feature sets due to the ease of visualizing the resulting decision landscapes. However, our method can also work with higher-dimensional datasets as seen by the results with the Palmer Penguins dataset and E. Coli dataset. In order to better understand the effect of data dimensionality on the performance of our method, we perform an additional set of experiments where we hold all other hyperparameters of the algorithm and data-generation constant but increase the dimensionality of the datasets. Table 5.5 compares the mean and standard deviations of the classification accuracy achieved by HSLaPkNN against those achieved by vanilla 1NN on these experiments with synthetic datasets containing 80 classes. Table 5.6 summarizes the associated prototype and accuracy ratios. Each experiment uses the distance-based attraction line-finding method and is repeated 100 times with a different random seed used to generate the data each time. Because we hold all hyperparameters constant, our method exhibits lower classification accuracy on this set of experiments. Curiously, we notice that our method exhibits greater variability in classification accuracy when used with higher-dimensional datasets. We believe that this may be caused by the sparsity introduced at higher dimensions (i.e. the curse of dimensionality) but further investigation is required to confirm this.

Experiment	Points	Lines	HSLaPkNN ($\mu \pm \sigma$)	1NN ($\mu \pm \sigma$)
Regular	1000	3	0.814 ± 0.063	0.895 ± 0.045
Regular	1000	4	0.837 ± 0.06	0.894 ± 0.045
Regular (5)	1000	2	0.909 ± 0.077	0.951 ± 0.05
Small	100	3	0.82 ± 0.06	0.90 ± 0.05
Imbalanced1	550	3	0.815 ± 0.098	0.896 ± 0.059
Imbalanced2	550	3	0.813 ± 0.081	0.895 ± 0.054
Giant	2000	23.82	0.835 ± 0.042	0.996 ± 0.004
Penguins	342	1	0.4327	0.5029
EColi	327	2	0.8135	0.8654

Table 5.3: Experimental results on a variety of simulated datasets comparing the performance of HSLaPkNN fitted on our soft-label prototypes to vanilla 1NN fitted on class centroids. Lines refers to the average number of lines found for the dataset. Experiments involving synthetic data (all except Penguins and EColi) are repeated 100 times with different random seeds during data generation to produce the standard deviations.

Experiment	Points	Lines	Prototypes ratio	Accuracy ratio
Regular	1000	3	0.3	0.91
Regular	1000	4	0.4	0.936
Regular (5)	1000	2	0.2	0.948
Small	100	3	0.3	0.911
Imbalanced1	550	3	0.3	0.909
Imbalanced2	550	3	0.3	0.909
Giant (100)	2000	23.82	0.238	0.838
Penguins	342	1	0.2	0.86
EColi	327	2	0.4	0.94

Table 5.4: Experimental results on a variety of simulated datasets comparing the performance of HSLaPkNN fitted on our soft-label prototypes to vanilla 1NN fitted on class centroids. Prototypes ratio refers to the ratio of the number of prototypical lines used by HSLaPkNN to the number of prototypes used by 1NN. Accuracy ratio refers to the ratio of the mean classification accuracy of HSLaPkNN to the mean classification accuracy of 1NN.

Dimension	Lines	HSLaPkNN ($\mu \pm \sigma$)	1NN ($\mu \pm \sigma$)
2	23.3	0.497 ± 0.034	0.782 ± 0.021
3	22.7	0.692 ± 0.046	0.974 ± 0.01
4	22.6	0.717 ± 0.055	0.997 ± 0.002
5	22.0	0.701 ± 0.062	0.999 ± 0.001
6	22.3	0.692 ± 0.068	1 ± 0
7	21.8	0.67 ± 0.067	1 ± 0.0
8	21.4	0.653 ± 0.062	1 ± 0.0
9	21.5	0.637 ± 0.073	1 ± 0.0
10	21.5	0.63 ± 0.07	1 ± 0.0

Table 5.5: Experimental results on simulated datasets of different dimensionalities comparing the performance of HSLaPkNN fitted on our soft-label prototypes to vanilla 1NN fitted on class centroids. Each dataset contains 2000 points across 80 classes. Experiments are repeated 100 times with different random seeds during data generation to produce the standard deviations.

Dimension	Lines	Prototypes ratio	Accuracy ratio
2	23.3	0.291	0.635
3	22.7	0.284	0.71
4	22.6	0.283	0.719
5	22.0	0.275	0.701
6	22.3	0.279	0.692
7	21.8	0.273	0.67
8	21.4	0.268	0.653
9	21.5	0.269	0.637
10	21.5	0.269	0.63

Table 5.6: Experimental results on simulated datasets with increasing feature dimensionalities comparing the performance of HSLaPkNN fitted on our soft-label prototypes to vanilla 1NN fitted on class centroids. Each dataset contains 2000 points across 80 classes. Prototypes ratio refers to the ratio of the number of prototypical lines used by HSLaPkNN to the number of prototypes used by 1NN. Accuracy ratio refers to the ratio of the mean classification accuracy of HSLaPkNN to the mean classification accuracy of 1NN.

5.4.4 Conclusion

We have proposed an algorithm for finding LO-shot prototypes in practice. The algorithm is intentionally designed to be modular so that each component can be improved independently. Next steps include finding better algorithms for detecting co-linear classes in datasets, improving the formulation of the soft-label optimization problem, and generalizing the method to work with a greater variety of classifiers.

Our proposed algorithm currently makes distributional assumptions about the datasets and classes to which it is applied. In particular, it assumes that each class is fairly contiguous and disjoint. When these assumptions are violated, even existing hard-label prototype methods need to increase the number of prototypes they produce in order to maintain classification accuracy. We believe an interesting direction would be to relax these assumptions for our soft-label prototype generation algorithm, perhaps by treating clusters in the data as sub-classes and optimizing for them separately rather than treating the entire class in a monolithic way. While this would likely increase the average number of classes assigned to each line, and may even increase the total number of lines required to achieve good coverage, it would likely result in higher performance on a larger variety of datasets.

We note that our proposed algorithm builds directly on the result from the previous section regarding separation of classes lying on a 1-dimensional manifold. However, it is possible that the underlying theory could be extended to classes lying on higher dimensional manifolds. In particular, we conjecture that, given some distributional assumptions, if a finite set of classes lies on an M-dimensional manifold, only M+1 soft-label prototypes are required to separate them. If this conjecture holds, then our proposed algorithm could be extended to work with M-dimensional manifolds. However, the key problem that would need to be solved is how to automatically detect subsets of the training dataset that lie on various manifolds with differing dimensionalities, and then optimize the selection of these subsets so as to minimize the total number of soft-label prototypes required to represent the dataset. When optimizing soft labels for these higher-dimensional manifolds, maintaining stability and robustness to noise may become increasingly challenging as there may be many more potentially unstable solutions than in the 1D case. Thus the optimization problem may require either a secondary objective that rewards stability or additional constraints that try to enforce it directly.

Chapter 6

Conclusion

The contributions described in the technical chapters of this thesis flow from the framework we established in Chapter 2. The loss restoration experiments in Chapter 3 provided an example of a typical applied machine learning workflow starting from analyzing a custom real-world dataset and ending with experimenting with different architectures to see if performance on a specific task could be improved. Information sharing (between observations) is what enables us to impute missing values based on the remaining ones, and information repackaging suggests that a reconstructed dataset contains no more information than the one affected by loss, yet 'imperfect' models may still work better with the repackaged data. The soft-label extension of dataset distillation in Chapter 4 enabled us to study near-optimal training examples and probe the limits of sample-efficiency in neural networks by repackaging information into increasingly small datasets. Information sharing between classes (via soft labels) helped us push past the previous best of one example per class and led to the first empirical evidence of LO-Shot Learning. Finally, the analytical results with SLaPkNN in Chapter 5 provided a theoretical basis for LO-Shot Learning.

More generally, if there is a single message that a reader should take away from this thesis, it is that training giant models on big data is not the only valid way to do machine learning. Users, practitioners, companies, and researchers should not be locked out of benefiting from the latest breakthroughs in machine learning and deep learning just because they face restrictions in terms of data availability or compute power. Small data problems are just as important as their big data counterparts and can often be solved to the same level of performance though this may sometimes require some careful and creative thought about how to best leverage information repackaging and injection techniques. For readers seeking inspiration, in Appendix A we provide a practical example related to medical imaging of the type of creative solutions that small data problems can require.

References

Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

Nur Ahmed and Muntasir Wahed. The de-democratization of ai: Deep learning and the compute divide in artificial intelligence research. *arXiv preprint arXiv:2010.15581*, 2020.

Hee-Kap Ahn, Sang Won Bae, Erik D Demaine, Martin L Demaine, Sang-Sub Kim, Matias Korman, Iris Reinbacher, and Wanbin Son. Covering points by disjoint boxes with outliers. *Computational Geometry*, 44(3):178–190, 2011.

Tahani Aljuaid and Sreela Sasi. Proper imputation techniques for missing values in data sets. In *International Conference on Data Science and Engineering (ICDSE), 2016*, pages 1–5. IEEE, 2016.

Fazil Altinel, Mete Ozay, and Takayuki Okatani. Deep structured energy-based image inpainting. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 423–428. IEEE, 2018.

Mohammad Mohammadi Amiri, Deniz Gunduz, Sanjeev R Kulkarni, and H Vincent Poor. Federated learning with quantized global model updates. *arXiv preprint arXiv:2006.10672*, 2020.

Rebecca R Andridge and Roderick JA Little. A review of hot deck imputation for survey non-response. *International Statistical Review*, 78(1):40–64, 2010.

Anelia Angelova, Yaser Abu-Mostafam, and Pietro Perona. Pruning training sets for learning of object categories. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 494–501. IEEE, 2005.

Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017.

Olivier Bachem, Mario Lucic, and Andreas Krause. Practical coreset constructions for machine learning. *arXiv preprint arXiv:1703.06476*, 2017.

Brett K. Beaulieu-Jones and Jason H. Moore. Missing data imputation in the electronic health record using deeply learned autoencoders. *Biocomputing*, 2017. doi: 10.1142/9789813207813_0021.

David A Belsley, Edwin Kuh, and Roy E Welsch. *Regression diagnostics: Identifying influential data and sources of collinearity*, volume 571. John Wiley & Sons, 2005.

James C Bezdek and Ludmila I Kuncheva. Nearest prototype classifier designs: An experimental study. *International Journal of Intelligent Systems*, 16(12):1445–1473, 2001.

Jacob Bien and Robert Tibshirani. Prototype selection for interpretable classification. *The Annals of Applied Statistics*, 5(4):2403–2424, Dec 2011. ISSN 1932-6157. doi: 10.1214/11-aoas495. URL http://dx.doi.org/10.1214/11-AOAS495.

Darren Blend and Tshilidzi Marwala. Comparison of data imputation techniques and their impact. *arXiv preprint arXiv:0812.1539*, 2008.

Ondrej Bohdal, Yongxin Yang, and Timothy Hospedales. Flexible dataset distillation: learn labels instead of images. *arXiv:2006.08572*, 2020.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 535–541. ACM, 2006.

Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, pages 8224–8234, 2018.

S van Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, pages 1–68, 2010.

Paz Carmi, Matthew J Katz, and Nissan Lev-Tov. Covering points by unit disks of fixed location. In *International Symposium on Algorithms and Computation*, pages 644–655. Springer, 2007.

Chin-Liang Chang. Finding prototypes for nearest neighbor classifiers. *IEEE Transactions on Computers*, 100(11):1179–1184, 1974.

Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8789–8797. IEEE, 2018.

Francois Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1251–1258, 2017.

David A Cohn, Zoubin Ghahramani, and Michael I Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1996.

Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

Thomas M Cover and Peter E Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

Greta Cutulenco, Yogi Joshi, Apurva Narayan, and Sebastian Fischmeister. Mining timed regular expressions from system traces. In *Proceedings of the 5th International Workshop on Software Mining*, SoftwareMining 2016, pages 3–10, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4511-8. doi: 10.1145/2975961.2975962. URL http://doi.acm.org/10.1145/2975961.2975962.

Valentin Dallmeier, Nikolai Knopp, Christoph Mallon, Sebastian Hack, and Andreas Zeller. Generating test cases for specification mining. In *Proceedings of the 19th International Symposium on Software Testing and Analysis*, pages 85–96. ACM, 2010.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North*, 2019. doi: 10.18653/v1/n19-1423. URL http://dx.doi.org/10.18653/v1/N19-1423.

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

A Rogier T Donders, Geert JMG Van Der Heijden, Theo Stijnen, and Karel GM Moons. A gentle introduction to imputation of missing values. *Journal of Clinical Epidemiology*, 59(10):1087–1091, 2006.

Yanjie Duan, Yisheng Lv, Wenwen Kang, and Yifei Zhao. A deep learning based approach for traffic data imputation. *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, 2014. doi: 10.1109/itsc.2014.6957805.

Sahibsingh A Dudani. The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, (4):325–327, 1976.

Adrian Dumitrescu and Minghui Jiang. On the approximability of covering points by lines and related problems. *Computational Geometry*, 48(9):703–717, 2015.

Matthew B Dwyer, George S Avrunin, and James C Corbett. Patterns in Property Specifications for Finite-State Verification. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 411–420. IEEE, 1999.

Bradley Efron. Missing data, imputation, and the bootstrap. *Journal of the American Statistical Association*, 89(426):463–475, 1994.

Neamat El Gayar, Friedhelm Schwenker, and Günther Palm. A study of the robustness of knn classifiers trained using soft labels. In *IAPR Workshop on Artificial Neural Networks in Pattern Recognition*, pages 67–80. Springer, 2006.

Mohamed M El-Zahhar and Neamat F El-Gayar. A semi-supervised learning approach for soft labeled data. In *2010 10th International Conference on Intelligent Systems Design and Applications*, pages 1136–1141. IEEE, 2010.

Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019.

Oded Maler Eugene Asarin, Paul Caspi. Timed Regular Expressions. *J. ACM*, 49(2): 172–206, March 2002. ISSN 0004-5411. doi: 10.1145/506147.506151. URL http://doi.acm.org/10.1145/506147.506151.

Brian S Everitt, Sabine Landau, Morven Leese, and Daniel Stahl. Cluster analysis 5th ed, 2011.

Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, 2006.

Katja Filippova, Enrique Alfonseca, Carlos A Colmenares, Lukasz Kaiser, and Oriol Vinyals. Sentence compression by deletion with lstms. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 360–368, 2015.

R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936. doi: 10.1111/j.1469-1809.1936.tb02137.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-1809.1936.tb02137.x.

Salvador Garcia, Joaquin Derrac, Jose Cano, and Francisco Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *IEEE transactions on pattern analysis and machine intelligence*, 34(3):417–435, 2012.

Salvador García, Julián Luengo, and Francisco Herrera. *Data preprocessing in data mining*. Springer, 2015.

Burkay Genç, Cem Evrendilek, and Brahim Hnich. Covering points with orthogonally convex polygons. *Computational Geometry*, 44(5):249–264, 2011.

F. A. Gers and E. Schmidhuber. LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, Nov 2001. ISSN 1045-9227. doi: 10.1109/72.963769.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256. PMLR, 2010.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 513–520, 2011.

Jack Goetz and Ambuj Tewari. Federated learning via synthetic data. *arXiv preprint arXiv:2008.04489*, 2020.

Lovedeep Gondara and Ke Wang. Multiple imputation using deep denoising autoencoders. *CoRR*, abs/1705.02737, 2017. URL http://arxiv.org/abs/1705.02737.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances*

in Neural Information Processing Systems, pages 2672–2680. Curran Associates, Inc., 2014.

Jianping Gou, Lan Du, Yuhong Zhang, Taisong Xiong, et al. A new distance-weighted k-nearest neighbor classifier. *Journal of Information and Computational Science*, 9(6): 1429–1436, 2012.

Magdalene Grantson and Christos Levcopoulos. Covering a set of points with a minimum number of lines. In *Italian Conference on Algorithms and Complexity*, pages 6–17. Springer, 2006.

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015. URL http://arxiv.org/abs/1503.04069.

Hyukjun Gweon, Matthias Schonlau, and Stefan H Steiner. The k conditional nearest neighbor algorithm for classification and class probability estimation. *PeerJ Computer Science*, 5:e194, 2019.

Peter Hart. The condensed nearest neighbor rule (corresp.). *IEEE Transactions on Information Theory*, 14(3):515–516, 1968.

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

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Sepp Hochreiter, Yoshua Bengio, and Paolo Frasconi. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In J. Kolen and S. Kremer, editors, *Field Guide to Dynamical Recurrent Networks*. IEEE Press, 2001.

Allison Marie Horst, Alison Presmanes Hill, and Kristen B Gorman. *palmerpenguins: Palmer Archipelago (Antarctica) penguin data*, 2020. URL https://allisonhorst.github.io/palmerpenguins/. R package version 0.1.0.

Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

Orakanya Kanjanatarakul, Siwarat Kuson, and Thierry Denoeux. An evidential k-nearest neighbor classifier based on contextual discounting and likelihood maximization. In *International Conference on Belief Functions*, pages 155–162. Springer, 2018.

Yoon Kim and Alexander M. Rush. Sequence-level knowledge distillation. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016. doi: 10.18653/v1/d16-1139. URL http://dx.doi.org/10.18653/v1/D16-1139.

Zachary Kincaid and Andreas Podelski. Automated Program Verification. In *Language and Automata Theory and Applications: 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*, volume 8977, page 25. Springer, 2015.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

SB Kotsiantis, Dimitris Kanellopoulos, and PE Pintelas. Data preprocessing for supervised leaning. *International Journal of Computer Science*, 1(2):111–117, 2006.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

M Kubat. Addressing the curse of imbalanced training sets: One-sided selection. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML), 1997*, pages 179–186, 1997.

Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

Matt Kusner, Stephen Tyree, Kilian Weinberger, and Kunal Agrawal. Stochastic neighbor compression. In *International Conference on Machine Learning*, pages 622–630, 2014.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Kamakshi Lakshminarayan, Steven A Harp, Robert P Goldman, and Tari Samad. Imputation of missing data using machine learning techniques. In *KDD*, pages 140–145, 1996.

Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 1978.

Stefan Langerman and Pat Morin. Covering points with lines. In *11th Fall Workshop on Compu-tational Geometry*, 2001.

Stefan Langerman and Pat Morin. Covering things with things. *Discrete & Computational Geometry*, 33(4):717–729, 2005.

Jorma Laurikkala. Improving identification of difficult small classes by balancing class distribution. In *Conference on Artificial Intelligence in Medicine in Europe*, pages 63–66. Springer, 2001.

Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. *Object Recognition with Gradient-Based Learning*, pages 319–345. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. ISBN 978-3-540-46805-9. doi: 10.1007/3-540-46805-6_19. URL https://doi.org/10.1007/3-540-46805-6_19.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4681–4690, 2017.

Jaakko Lehtinen, Jacob Munkberg, Jon Hasselgren, Samuli Laine, Tero Karras, Miika Aittala, and Timo Aila. Noise2noise: Learning image restoration without clean data. *arXiv preprint arXiv:1803.04189*, 2018.

J. Lei Ba, J. R. Kiros, and G. E. Hinton. Layer Normalization. *ArXiv e-prints*, July 2016.

Collins Leke, Tshilidzi Marwala, and Satyakama Paul. Proposition of a theoretical model for missing data imputation using deep learning and evolutionary algorithms. *CoRR*, abs/1512.01362, 2015. URL http://arxiv.org/abs/1512.01362.

Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017. URL http://jmlr.org/papers/v18/16-365.html.

Caroline Lemieux, Dennis Park, and Ivan Beschastnikh. General LTL Specification Mining. In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 81–92. IEEE, 2015.

Gregory W Lesher, Bryan J Moulton, and D Jeffery Higginbotham. Effects of ngram order and training text size on word prediction. In *Proceedings of the RESNA'99 Annual Conference*, pages 52–54. Citeseer, 1999.

Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838*, 2018.

Guang Li, Ren Togo, Takahiro Ogawa, and Miki Haseyama. Soft-label anonymous gastric x-ray image distillation. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 305–309. IEEE, 2020a.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3): 50–60, 2020b.

Yuri Lin, Jean-Baptiste Michel, Erez Lieberman Aiden, Jon Orwant, Will Brockman, and Slav Petrov. Syntactic annotations for the google books ngram corpus. In *Proceedings of the ACL 2012 system demonstrations*, pages 169–174. Association for Computational Linguistics, 2012.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

Lsdefine. Lsdefine/attention-is-all-you-need-keras, 2018. URL https://github.com/Lsdefine/attention-is-all-you-need-keras.

Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, 2015.

Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016. doi: 10.18653/v1/p16-1101. URL http://dx.doi.org/10.18653/v1/P16-1101.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

James ES Macleod, Andrew Luk, and D Michael Titterington. A re-examination of the distance-weighted k-nearest neighbor classification rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 17(4):689–696, 1987.

P Mahapatra, R Sinha, Partha P Goswami, and Sandip Das. Covering points by isothetic units squares. 2007.

Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, page 89. Presses universitaires de Louvain, 2015.

Inderjeet Mani and I Zhang. knn approach to unbalanced data distributions: a case study involving information extraction. In *Proceedings of Workshop on Learning from Imbalanced Datasets*, volume 126, 2003.

Pascal Mettes, Elise van der Pol, and Cees GM Snoek. Hyperspherical prototype networks. *arXiv:1901.10514*, 2019.

John Miller and Moritz Hardt. When recurrent models don't need to be recurrent. *arXiv preprint arXiv:1805.10369*, 2018.

HB Mitchell and PA Schaefer. A "soft" k-nearest neighbor voting scheme. *International Journal of Intelligent Systems*, 16(4):459–468, 2001.

Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

Teresa A Myers. Goodbye, listwise deletion: Presenting hot deck imputation as an easy and effective tool for handling missing data. *Communication Methods and Measures*, 5 (4):297–310, 2011.

Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3303–3313, 2018.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292*, 2019.

Preetum Nakkiran, Behnam Neyshabur, and Hanie Sedghi. The deep bootstrap: Good online learners are good offline generalizers. *arXiv preprint arXiv:2010.08127*, 2020.

Loris Nanni and Alessandra Lumini. Particle swarm optimization for prototype reduction. *Neurocomputing*, 72(4-6):1092–1097, 2009.

Apurva Narayan, Greta Cutulenco, Yogi Joshi, and Sebastian Fischmeister. Mining timed regular specifications from system traces. *ACM Trans. Embed. Comput. Syst.*, 17(2): 46:1–46:21, January 2018. ISSN 1539-9087. doi: 10.1145/3147660. URL http://doi.acm.org/10.1145/3147660.

Simon Niklaus, Long Mai, and Feng Liu. Video frame interpolation via adaptive convolution. *CoRR*, abs/1703.07514, 2017. URL http://arxiv.org/abs/1703.07514.

Maximilian Panzner and Philipp Cimiano. Comparing hidden markov models and long short term memory neural networks for learning action representations. In *International Workshop on Machine Learning, Optimization and Big Data*, pages 94–105. Springer, 2016.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, 2018. doi: 10.18653/v1/n18-1202. URL http://dx.doi.org/10.18653/v1/N18-1202.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*, 2018.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. *arXiv preprint arXiv:2103.00020*, 2021.

Trivellore E Raghunathan, James M Lepkowski, John Van Hoewyk, and Peter Solenberger. A multivariate technique for multiply imputing missing values using a sequence of regression models. *Survey Methodology*, 27(1):85–96, 2001.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=rJY0-Kcll.

Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

Bernardino Romera-Paredes and Philip Torr. An embarrassingly simple approach to zero-shot learning. In *International conference on machine learning*, pages 2152–2161. PMLR, 2015.

Dymitr Ruta. Dynamic data condensation for classification. In *International Conference on Artificial Intelligence and Soft Computing*, pages 672–681. Springer, 2006.

S Rasoul Safavian and David Landgrebe. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674, 1991.

Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *CoRR*, abs/1402.1128, 2014. URL http://arxiv.org/abs/1402.1128.

José Salvador Sánchez. High training set size reduction by space partitioning and prototype abstraction. *Pattern Recognition*, 37(7):1561–1564, 2004.

Teven Le Scao and Alexander M Rush. How many data points is a prompt worth? *arXiv preprint arXiv:2103.08493*, 2021.

Joseph L Schafer and John W Graham. Missing data: our view of the state of the art. *Psychological Methods*, 7(2):147, 2002.

Lukas Schmidt, Apurva Narayan, and Sebastian Fischmeister. Trem: A tool for mining timed regular specifications from system traces. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pages 901–906, Piscataway, NJ, USA, 2017. IEEE Press. ISBN 978-1-5386-2684-9. URL http://dl.acm.org/citation.cfm?id=3155562.3155675.

Peter Schmitt, Jonas Mandel, and Mickael Guedj. A comparison of six methods for missing data imputation. *Journal of Biometrics & Biostatistics*, 6(1):1, 2015.

Matthias Schonlau, Nick Guenther, and Ilia Sucholutsky. Text mining with n-gram variables. *Stata Journal*, 17(4):866–881, 2017.

Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. Recurrent dropout without memory loss. *CoRR*, abs/1603.05118, 2016. URL http://arxiv.org/abs/1603.05118.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

Tao Shen, Jie Zhang, Xinkang Jia, Fengda Zhang, Gang Huang, Pan Zhou, Fei Wu, and Chao Wu. Federated mutual learning. *arXiv preprint arXiv:2006.16765*, 2020.

Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM National Conference*, pages 517–524, 1968.

Sam Shleifer and Eric Prokop. Using small proxy datasets to accelerate hyperparameter search. *arXiv:1906.04887*, 2019.

Michael R Smith, Tony Martinez, and Christophe Giraud-Carrier. An instance level analysis of data complexity. *Machine Learning*, 95(2):225–256, 2014.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087, 2017.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, 2013.

Shaoming Song, Yunfeng Shao, and Jian Li. Loosely coupled federated learning over generative models. *arXiv preprint arXiv:2009.12999*, 2020.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019. doi: 10.18653/v1/p19-1355. URL http://dx.doi.org/10.18653/v1/P19-1355.

Ilia Sucholutsky and Matthias Schonlau. Soft-label dataset distillation and text dataset distillation. *2021 International Joint Conference on Neural Networks (IJCNN)*, 2019. Forthcoming. Pre-print available at arXiv:1910.02551.

Ilia Sucholutsky and Matthias Schonlau. 'Less than one'-shot learning: Learning N classes from M< N samples. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021a. Forthcoming. Pre-print available at arXiv:2009.08449.

Ilia Sucholutsky and Matthias Schonlau. Optimal 1-NN prototypes for pathological geometries. *PeerJ Computer Science*, 7, 2021b. doi: 10.7717/peerj-cs.464.

Ilia Sucholutsky and Matthias Schonlau. SecDD: Efficient and secure method for remotely training neural networks (Student Abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021c. Forthcoming. Pre-print available at arXiv:2009.09155.

Ilia Sucholutsky, Apurva Narayan, Matthias Schonlau, and Sebastian Fischmeister. Deep learning for system trace restoration. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, July 2019a. doi: 10.1109/IJCNN.2019.8852116. Pre-print at arXiv:1904.05411.

Ilia Sucholutsky, Apurva Narayan, Matthias Schonlau, and Sebastian Fischmeister. Pay attention and you won't lose it: a deep learning approach to sequence imputation. *PeerJ Computer Science*, 5:e210, August 2019b.

Ilia Sucholutsky, Nam-Hwui Kim, Ryan P Browne, and Matthias Schonlau. One line to rule them all: Generating LO-shot soft-label prototypes. *2021 International Joint Conference on Neural Networks (IJCNN)*, 2021. Forthcoming. Pre-print available at arXiv:2102.07834.

Will Wei Sun, Xingye Qiao, and Guang Cheng. Stabilized nearest neighbor classifier and its statistical properties. *Journal of the American Statistical Association*, 111(515):1254–1265, 2016.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL http://arxiv.org/abs/1409.3215.

Christian Thiel. Classification on soft labels is robust against label noise. In *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*, pages 65–73. Springer, 2008.

Ivan Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(6):448–452, 1976a. doi: 10.1109/TSMC.1976.4309523.

Ivan Tomek. Two modifications of cnn. *IEEE Trans. Systems, Man and Cybernetics*, 6: 769–772, 1976b.

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

Isaac Triguero, Joaquín Derrac, Salvador Garcia, and Francisco Herrera. A taxonomy and experimental study on prototype generation for nearest neighbor classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42 (1):86–100, 2011a.

Isaac Triguero, Salvador García, and Francisco Herrera. Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification. *Pattern Recognition*, 44(4):901–916, 2011b.

Ivor W Tsang, James T Kwok, and Pak-Ming Cheung. Core vector machines: Fast SVM training on very large data sets. *Journal of Machine Learning Research*, 6(Apr):363–392, 2005.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL http://doi.acm.org/10.1145/2641190.2641198.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010, 2017.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

Ellen M Voorhees et al. The TREC-8 question answering track report. In *the Proceedings of the Eighth Text Retrieval Conference (TREC-8)*, volume 99, pages 77–82, 1999.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

Shanshan Wang and Lei Zhang. Regularized deep transfer learning: When cnn meets knn. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2019.

Tongzhou Wang, Jun-Yan Zhu, Antonio Torralba, and Alexei A Efros. Dataset distillation. *arXiv preprint arXiv:1811.10959*, 2018.

Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys (CSUR)*, 53(3): 1–34, 2020.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

Dennis L Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, (3):408–421, 1972.

Yongqin Xian, Christoph H Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. *IEEE transactions on pattern analysis and machine intelligence*, 41(9):2251–2265, 2018.

Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In *Advances in neural information processing systems*, pages 341–349, 2012.

Halil Yigit. Abc-based distance-weighted knn algorithm. *Journal of Experimental & Theoretical Artificial Intelligence*, 27(2):189–198, 2015.

L Yu, W Zhang, J Wang, and Y Yu. SeqGAN: sequence generative adversarial nets with policy gradient. In *AAAI-17: Thirty-First AAAI Conference on Artificial Intelligence*, volume 31, pages 2852–2858. Association for the Advancement of Artificial Intelligence (AAAI), 2017.

Guoping Zeng. A unified definition of mutual information with applications in machine learning. *Mathematical Problems in Engineering*, 2015, 2015.

Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

Yuhao Zhang, Hang Jiang, Yasuhide Miura, Christopher D Manning, and Curtis P Langlotz. Contrastive learning of medical visual representations from paired images and text. *arXiv preprint arXiv:2010.00747*, 2020.

Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. Dataset condensation with gradient matching. *arXiv:2006.05929*, 2020.

Jingguang Zhou and Zili Huang. Recover missing sensor data with iterative imputing network. *CoRR*, abs/1711.07878, 2017. URL http://arxiv.org/abs/1711.07878.

Yanlin Zhou, George Pu, Xiyao Ma, Xiaolin Li, and Dapeng Wu. Distilled one-shot federated learning. *arXiv preprint arXiv:2009.07999*, 2020.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

APPENDICES

Appendix A

Additional Practical Setting - Expensive Data and Annotations

In the extreme case, both data and annotations are expensive or hard to come by. A perfect motivating example is applications of AI to medical imaging tasks. Data are notoriously expensive in the medical space; depending on the disease/topic being studied, there may only be a handful of patients available for imaging and observation. In addition, expert annotators in the medical space (e.g. pathologists) are busy and their time is expensive. Even if a large amount of imaging data is somehow collected, annotating it may not be possible.

Unfortunately, this annotation cost problem is further exacerbated when multiple tasks need to be solved with the same dataset. The current process for developing AI applications for the medical space is as follows:

1. Define the task (e.g. classification - diagnose a patient's chest infection based on imaging of the lungs).
2. Send the medical images to the expert annotator (e.g. pathologist).
3. The annotator expends valuable time carefully analyzing every image.
4. The annotator returns roughly a single byte of information per image (e.g. a class for each image).
5. Pick a new task and repeat from step 1.

As a result, each time that a new task is defined, the expert analyzes the same images and likely performs mostly the same analysis before generating a (roughly one-byte) annotation for the specific task currently being targeted.

A much more efficient process would enable the experts to provide more information about their full analysis, that can then be used to solve multiple tasks without requiring that experts create new annotations for each one. Perhaps the most natural way for human experts to provide this kind of rich information is through open-ended natural language annotations. If information is shared between different tasks for one dataset, then we can still use the same natural language annotations to guide model training regardless of which task we are solving. Powerful, modern language models could be used to extract intermediate features from these annotations that can in turn be used to guide model training by learning a mapping from images to intermediate features and from intermediate features to the target variable associated with each task. If such pairs of mappings can be successfully found, then natural language annotations are only needed at training time, and the pair of mappings can be used to predict the target variable from the image at inference time. In the medical space, such a system aligns perfectly with the regular workflow that pathologists have since they already produce pathology reports for every patient. For example, Zhang et al. [2020] train medical image encoders by aligning representations of medical images with representations of their associated pathology reports and show that these encoders can be transferred to multiple different medical imaging tasks using far fewer labeled samples than other methods.

Unfortunately, natural language comments of the quality and information-density of pathology reports are usually hard to come by aside from in the medical industry. For example, it is difficult for an annotator to produce a large amount of descriptive text about samples from image datasets like ImageNet. In the medical setting, the incentives between the AI researcher and the annotator are aligned since the pathologist naturally wants to write detailed, accurate, informative pathology reports that describe anything that they find relevant in the images they analyze. In other settings, it may be difficult to properly align incentives between the AI researcher and the annotator.

A similar but more general solution to natural language annotations may be to leverage recent advances in multi-modal modelling, such as joint text and image modelling [Radford et al., 2021], and use natural language prompts as a mechanism for guiding the training process. Scao and Rush [2021] recently showed that natural language prompts can reduce the number of training examples required for fine-tuning large language models to a particular task by two to three orders of magnitude. Developing sample-efficient, prompt-based, multi-modal systems remains an important direction for future research.

Appendix B

Proofs for Chapter 5

B.1 Proof of Theorem 1

Proof. One way to partition the space into 3 partitions, is to have a different class for every unit of space between the two points. Points lying on the x-axis within the interval $(i-1, i)$ should be assigned to the i^{th} class. For example, points on the x-axis within $(0, 1)$ should be assigned to class 1. Succinctly put, we are trying to find y_1 and y_2 such that

$$\arg\max_i \left(\frac{y_{1i}}{d} + \frac{y_{2i}}{3-d} \right) = \lceil d \rceil \ \forall d \in (0, 3) \tag{B.1}$$

It may also be desirable to have a class's 'influence' decrease monotonically as distance increases along the x-axis away from the center of its corresponding interval. Combining these two objectives results in the following system of inequalities.

$$\begin{cases} \frac{y_{11}}{d} + \frac{y_{21}}{3-d} > \frac{y_{12}}{d} + \frac{y_{22}}{3-d} > \frac{y_{13}}{d} + \frac{y_{23}}{3-d} & 0 < d < 1 \\ \frac{y_{12}}{d} + \frac{y_{22}}{3-d} = \frac{y_{11}}{d} + \frac{y_{21}}{3-d} > \frac{y_{13}}{d} + \frac{y_{23}}{3-d} & d = 1 \\ \frac{y_{12}}{d} + \frac{y_{22}}{3-d} > \frac{y_{11}}{d} + \frac{y_{21}}{3-d} > \frac{y_{13}}{d} + \frac{y_{23}}{3-d} & 1 < d < \frac{3}{2} \\ \frac{y_{12}}{d} + \frac{y_{22}}{3-d} > \frac{y_{11}}{d} + \frac{y_{21}}{3-d} = \frac{y_{13}}{d} + \frac{y_{23}}{3-d} & d = \frac{3}{2} \\ \frac{y_{12}}{d} + \frac{y_{22}}{3-d} > \frac{y_{13}}{d} + \frac{y_{23}}{3-d} > \frac{y_{11}}{d} + \frac{y_{21}}{3-d} & \frac{3}{2} < d < 2 \\ \frac{y_{13}}{d} + \frac{y_{23}}{3-d} = \frac{y_{12}}{d} + \frac{y_{22}}{3-d} > \frac{y_{11}}{d} + \frac{y_{21}}{3-d} & d = 2 \\ \frac{y_{13}}{d} + \frac{y_{23}}{3-d} > \frac{y_{12}}{d} + \frac{y_{22}}{3-d} > \frac{y_{11}}{d} + \frac{y_{21}}{3-d} & 2 < d < 3 \end{cases} \tag{B.2}$$

Since the labels are probabilistic, the sum of their elements must be one and each element must have a value greater than or equal to 0.

$$\sum_{i=1}^{3} y_{1i} = \sum_{i=1}^{3} y_{2i} = 1$$
$$y_{1i} \geq 0,\ y_{2i} \geq 0 \text{ for } i = 1, 2, 3$$

To simplify this system of equations, we can also require that the label values for the two classes be symmetric with $y_{1i} = y_{2(n-i)}$ for $i = 1, 2, 3$.

$$y_{11} = y_{23}$$
$$y_{12} = y_{22}$$
$$y_{13} = y_{21}$$

$$\begin{cases} \frac{y_{11}}{d} + \frac{y_{13}}{3-d} > \frac{y_{12}}{d} + \frac{y_{12}}{3-d} > \frac{y_{13}}{d} + \frac{y_{11}}{3-d} & 0 < d < 1 \\ \frac{y_{12}}{d} + \frac{y_{12}}{3-d} = \frac{y_{11}}{d} + \frac{y_{13}}{3-d} > \frac{y_{13}}{d} + \frac{y_{11}}{3-d} & d = 1 \\ \frac{y_{12}}{d} + \frac{y_{12}}{3-d} > \frac{y_{11}}{d} + \frac{y_{13}}{3-d} > \frac{y_{13}}{d} + \frac{y_{11}}{3-d} & 1 < d < \frac{3}{2} \\ \frac{y_{12}}{d} + \frac{y_{12}}{3-d} > \frac{y_{11}}{d} + \frac{y_{13}}{3-d} = \frac{y_{13}}{d} + \frac{y_{11}}{3-d} & d = \frac{3}{2} \\ \frac{y_{12}}{d} + \frac{y_{12}}{3-d} > \frac{y_{13}}{d} + \frac{y_{11}}{3-d} > \frac{y_{11}}{d} + \frac{y_{13}}{3-d} & \frac{3}{2} < d < 2 \\ \frac{y_{13}}{d} + \frac{y_{11}}{3-d} = \frac{y_{12}}{d} + \frac{y_{12}}{3-d} > \frac{y_{11}}{d} + \frac{y_{13}}{3-d} & d = 2 \\ \frac{y_{13}}{d} + \frac{y_{11}}{3-d} > \frac{y_{12}}{d} + \frac{y_{12}}{3-d} > \frac{y_{11}}{d} + \frac{y_{13}}{3-d} & 2 < d < 3 \\ y_{11} + y_{12} + y_{13} = 1 & \\ y_{1i} \geq 0, \text{ for } i = 1, 2, 3 & \end{cases} \tag{B.3}$$

Remark *The inequalities in Equation B.3 are symmetric about $d = \frac{3}{2}$. We can reduce them by considering only the cases where $d \leq \frac{3}{2}$.*

$$\begin{cases} \frac{y_{11}}{d} + \frac{y_{13}}{3-d} > \frac{y_{12}}{d} + \frac{y_{12}}{3-d} > \frac{y_{13}}{d} + \frac{y_{11}}{3-d} & 0 < d < 1 \\ \frac{y_{12}}{d} + \frac{y_{12}}{3-d} = \frac{y_{11}}{d} + \frac{y_{13}}{3-d} > \frac{y_{13}}{d} + \frac{y_{11}}{3-d} & d = 1 \\ \frac{y_{12}}{d} + \frac{y_{12}}{3-d} > \frac{y_{11}}{d} + \frac{y_{13}}{3-d} > \frac{y_{13}}{d} + \frac{y_{11}}{3-d} & 1 < d < \frac{3}{2} \\ \frac{y_{12}}{d} + \frac{y_{12}}{3-d} > \frac{y_{11}}{d} + \frac{y_{13}}{3-d} & d = \frac{3}{2} \\ y_{11} + y_{12} + y_{13} = 1 & \\ y_{1i} \geq 0, \textit{ for } i = 1, 2, 3 & \end{cases} \tag{B.4}$$

We can further simplify this system.

$$\begin{cases} (3-d)y_{11} + dy_{13} > 3y_{12} > (3-d)y_{13} + dy_{11} & 0 < d < 1 \\ 3y_{12} = 2y_{11} + y_{13} > 2y_{13} + y_{11} & d = 1 \\ 3y_{12} > (3-d)y_{11} + dy_{13} > (3-d)y_{13} + dy_{11} & 1 < d < \frac{3}{2} \\ 4y_{12} > 2y_{11} + 2y_{13} & d = \frac{3}{2} \\ y_{11} + y_{12} + y_{13} = 1 & \\ y_{1i} \geq 0, \text{ for } i = 1, 2, 3 & \end{cases}$$

$$\begin{cases} 3y_{12} = 2y_{11} + y_{13} > 1 \\ y_{11} > y_{12} > y_{13} \geq 0 \\ y_{11} + y_{12} + y_{13} = 1 \end{cases}$$

□

B.2 Proof of Corollary 2

Proof. We proceed similarly to the previous proof. One way to partition the space into 3 partitions, is to have a different class for every unit of space between the two points; each interval should have length $\frac{c}{3}$. Points lying on the x-axis within the interval $((i-1)\frac{c}{3}, (i)\frac{c}{3})$ should be assigned to the i^{th} class. For example, points on the x-axis within $(0, \frac{c}{3})$ should be assigned to class 1. Succinctly put, we are trying to find y_1 and y_2 such that

$$\arg\max_i \left(\frac{y_{1i}}{(d)\frac{c}{3}} + \frac{y_{2i}}{(3-d)\frac{c}{3}} \right) = \lceil (d) \rceil \frac{c}{3} \; \forall d \in (0, 3) \tag{B.5}$$

However, this is exactly equal to Equation B.1 and the rest of the proof follows as above. □

B.3 Proof of Theorem 3

Proof. One way to select such labels is to use the same labels for each pair as in Theorem 1. This results in $y_1, ..., y_{M-1}$ each having a label distribution containing two non-zero values:

$\frac{3}{5}$ (associated with its main class) and $\frac{2}{5}$ (associated with the class created between itself and x_0). Meanwhile, y_0 contains one element with value $\frac{3}{5}$ (associated with its own class) and M-1 elements with value $\frac{2}{5}$ (each associated with a unique class created between x_0 and each one of the other points). To get all probabilistic labels, we can normalize y_0 to instead have values $\frac{3}{2M+1}$ and $\frac{2}{2M+1}$.

It can be shown that this configuration of labels leads to $2M-1$ classes as each point has its own class and every pair creates one additional one. Without loss of generality, assume X_0 has position $(0,0)$ and one of the remaining points $x_i, i>0$ has position $(p,0)$. Since all the points are equidistant to x_0, the nearest 2 points to any location between $(0,0)$ and $(\frac{p}{2},0)$ must be x_0 and x_i. Let d be the distance from x_0 to some arbitrary point $q=(d,0)$ within this interval (then $p-d$ is the distance from q to x_i). Let a denote the index of the main class of x_0 (i.e. the class with the highest label value in y_0), b denote the index of the main class of x_i, and c denote the index of the class they share (i.e. the only other non-zero class in y_1). Using the distance-weighted SLaPkNN rule, q would be classified as follows.

$$\begin{cases} a \text{ if } \frac{y_{0,a}}{d} > \frac{y_{i,b}}{p-d}, \frac{y_{0,c}}{d} + \frac{y_{i,c}}{p-d} \\ b \text{ if } \frac{y_{i,b}}{p-d} > \frac{y_{0,a}}{d}, \frac{y_{0,c}}{d} + \frac{y_{i,c}}{p-d} \\ c \text{ if } \frac{y_{0,c}}{d} + \frac{y_{i,c}}{p-d} > \frac{y_{0,a}}{d}, \frac{y_{i,b}}{p-d} \end{cases} \tag{B.6}$$

Plugging in the label values described above, we get the following system.

$$\begin{cases} a \text{ if } \frac{3}{(2M+1)d} > \frac{3}{5(p-d)}, \frac{2}{(2M+1)d} + \frac{2}{5(p-d)} \\ b \text{ if } \frac{3}{5(p-d)} > \frac{3}{(2M+1)d}, \frac{2}{(2M+1)d} + \frac{2}{5(p-d)} \\ c \text{ if } \frac{2}{(2M+1)d} + \frac{2}{5(p-d)} > \frac{3}{(2M+1)d}, \frac{3}{5(p-d)} \end{cases} \tag{B.7}$$

This can be further simplified.

$$\begin{cases} a \text{ if } \frac{1}{(2M+1)d} > \frac{2}{5(p-d)} \\ b \text{ if } \frac{1}{5(p-d)} > \frac{2}{(2M+1)d} \\ c \text{ if } \frac{1}{(2M+1)d} < \frac{2}{5(p-d)} < \frac{4}{(2M+1)d} \end{cases} \tag{B.8}$$

$$\begin{cases} a \text{ if } d < \frac{5p}{4M+7} \\ b \text{ if } d > \frac{10p}{2M+11} \\ c \text{ if } \frac{5p}{4M+7} < d < \frac{10p}{2M+11} \end{cases} \tag{B.9}$$

Clearly, there are valid values of d that can result in each of these classifications. □

B.4 Proof of Proposition 4

Proof. Theorem 1 shows that a pair of neighboring prototypes can define their own respective classes as well as induce a third class between them. Theorem 3 shows that a single point can belong to multiple such pairs that each generate a unique 'third' class. If this is the case, then by taking the pair of vertices corresponding to each edge in an M-sided regular polygon we can create one class for every vertex and one for every edge of the polygon for a total of $2M$ classes.

In this case, every point belongs to two pairs, so the system will be somewhat different from the one found in the proof of Theorem 3 since every point will now need to have its label normalized. We use T to denote the number of pairs each point participates in, which in this case would just be two.

$$\begin{cases} a \text{ if } \frac{3}{(2T+3)d} > \frac{3}{(2T+3)(p-d)}, \frac{2}{(2T+1)d} + \frac{2}{(2T+3)(p-d)} \\ b \text{ if } \frac{3}{(2T+3)(p-d)} > \frac{3}{(2T+3)d}, \frac{2}{(2T+3)d} + \frac{2}{(2T+3)(p-d)} \\ c \text{ if } \frac{2}{(2T+3)d} + \frac{2}{(2T+3)(p-d)} > \frac{3}{(2T+3)d}, \frac{3}{(2T+3)(p-d)} \end{cases} \tag{B.10}$$

This can be further simplified.

$$\begin{cases} a \text{ if } \frac{1}{d} > \frac{2}{(p-d)} \\ b \text{ if } \frac{1}{(p-d)} > \frac{2}{d} \\ c \text{ if } \frac{1}{d} < \frac{2}{(p-d)} < \frac{4}{d} \end{cases} \tag{B.11}$$

$$\begin{cases} a \text{ if } d < \frac{p}{3} \\ b \text{ if } d > \frac{2p}{3} \\ c \text{ if } \frac{p}{3} < d < \frac{2p}{3} \end{cases} \tag{B.12}$$

□

B.5 Proof of Theorem 5

Proof. We have already shown that every pair of neighboring prototypes can define their own respective classes as well as induce a third class between them. However, we now arrange M soft-label prototypes to be the vertices and centroid of an $(M-1)$-sided regular

polygon. Using our previous method, we now have $2(M-1)$ classes from the perimeter. In addition, the prototype in the middle induces its own class as well as another class with each of the $M-1$ other points. Thus this configuration allows us to divide the space into $3M-2$ partitions. □

B.6 Proof of Lemma 6

Proof. We proceed in the same manner as in Theorem 1 .
One way to partition the space into 4 partitions, is to have a different class for every unit of space between the two prototypes. Points lying on the x-axis within the interval $(i-1, i)$ should be assigned to the i^{th} class. For example, points on the x-axis within $(0, 1)$ should be assigned to class 1. Succinctly put, we are trying to find y_1 and y_2 such that

$$\arg\max_i \left(\frac{y_{1i}}{d} + \frac{y_{2i}}{4-d} \right) = \lceil d \rceil \ \forall d \in (0, 4) \tag{B.13}$$

It may also be desirable to have a class's 'influence' decrease monotonically as distance increases along the x-axis away from the center of its corresponding interval. Combining these two objectives results in a system of inequalities that are symmetric about $d = 2$.

We can reduce them by considering only the cases where $d \leq 2$.

$$
\begin{cases}
\frac{y_{11}}{d} + \frac{y_{14}}{4-d} > \frac{y_{12}}{d} + \frac{y_{13}}{4-d} > \frac{y_{13}}{d} + \frac{y_{12}}{4-d} > \frac{y_{14}}{d} + \frac{y_{11}}{4-d} & 0 < d < 1 \\
\frac{y_{11}}{d} + \frac{y_{14}}{4-d} = \frac{y_{12}}{d} + \frac{y_{13}}{4-d} > \frac{y_{13}}{d} + \frac{y_{12}}{4-d} > \frac{y_{14}}{d} + \frac{y_{11}}{4-d} & d = 1 \\
\frac{y_{12}}{d} + \frac{y_{13}}{4-d} > \frac{y_{11}}{d} + \frac{y_{14}}{4-d} > \frac{y_{13}}{d} + \frac{y_{12}}{4-d} > \frac{y_{14}}{d} + \frac{y_{11}}{4-d} & 1 < d < 1.5 \\
\frac{y_{12}}{d} + \frac{y_{13}}{4-d} > \frac{y_{11}}{d} + \frac{y_{14}}{4-d} = \frac{y_{13}}{d} + \frac{y_{12}}{4-d} > \frac{y_{14}}{d} + \frac{y_{11}}{4-d} & d = 1.5 \\
\frac{y_{12}}{d} + \frac{y_{13}}{4-d} > \frac{y_{13}}{d} + \frac{y_{12}}{4-d} > \frac{y_{11}}{d} + \frac{y_{14}}{4-d} > \frac{y_{14}}{d} + \frac{y_{11}}{4-d} & 1.5 < d < 2 \\
\frac{y_{12}}{d} + \frac{y_{13}}{4-d} = \frac{y_{13}}{d} + \frac{y_{12}}{4-d} > \frac{y_{11}}{d} + \frac{y_{14}}{4-d} = \frac{y_{14}}{d} + \frac{y_{11}}{4-d} & d = 2 \\
y_{11} + y_{12} + y_{13} + y_{14} = 1 & \\
y_{1i} \geq 0, \text{ for } i = 1, 2, 3, 4 &
\end{cases}
$$

$$
\begin{cases}
\frac{y_{11}}{d} + \frac{y_{14}}{4-d} > \frac{y_{12}}{d} + \frac{y_{13}}{4-d} > \frac{y_{13}}{d} + \frac{y_{12}}{4-d} > \frac{y_{14}}{d} + \frac{y_{11}}{4-d} & 0 < d < 1 \\
3y_{11} + y_{14} = 3y_{12} + y_{13} > 3y_{13} + y_{12} > 3y_{14} + y_{11} & d = 1 \\
\frac{y_{12}}{d} + \frac{y_{13}}{4-d} > \frac{y_{11}}{d} + \frac{y_{14}}{4-d} > \frac{y_{13}}{d} + \frac{y_{12}}{4-d} > \frac{y_{14}}{d} + \frac{y_{11}}{4-d} & 1 < d < 1.5 \\
5y_{12} + 3y_{13} > 5y_{11} + 3y_{14} = 5y_{13} + 3y_{12} > 5y_{14} + 3y_{11} & d = 1.5 \\
\frac{y_{12}}{d} + \frac{y_{13}}{4-d} > \frac{y_{13}}{d} + \frac{y_{12}}{4-d} > \frac{y_{11}}{d} + \frac{y_{14}}{4-d} > \frac{y_{14}}{d} + \frac{y_{11}}{4-d} & 1.5 < d < 2 \\
y_{12} + y_{13} > y_{11} + y_{14} & d = 2 \\
y_{11} + y_{12} + y_{13} + y_{14} = 1 & \\
y_{1i} \geq 0, \text{ for } i = 1, 2, 3, 4 &
\end{cases}
$$

$$
\begin{cases}
y_{11} + y_{14} = 2y_{13} \\
2y_{11} + 10y_{13} = 3 \\
y_{12} + 3y_{13} = 1 \\
\frac{1}{2} > y_{11} > \frac{5}{12} > y_{12} > \frac{1}{4} > y_{13} > \frac{1}{6} > y_{14} \geq 0
\end{cases}
\tag{B.14}
$$

□

B.7 Proof of Theorem 7 (Main Theorem)

Proof. The $n = 1$ and $n = 2$ cases are trivial. We have already shown that this holds for $n = 3$ and $n = 4$ in Theorem 1 and Lemma 6 . In order to show the statement holds for $n \in [5, \infty)$ we proceed similarly. Assume that two prototypes are positioned distance n apart in a two-dimensional Euclidean space. Without loss of generality, suppose that point $x_1 = (0, 0)$ and point $x_2 = (n, 0)$ have probabilistic labels y_1 and y_2 respectively. We

denote the i^{th} element of each label by y_{1i} and y_{2i}. One way to partition the space into n partitions, is to have a different class for every unit of space between the two points. Points lying on the x-axis within the interval $[i-1, i)$ belong to the i^{th} class. The general objective is to find y_1 and y_2 satisfying the following equation.

$$\arg\max_i \left(\frac{y_{1i}}{d} + \frac{y_{2i}}{n-d} \right) = \lceil d \rceil$$

We require that the label values for the two classes be symmetric with $y_{1i} = y_{2(n-i)}$ for $i = 1, 2, ..., n$. We also require that a class's 'influence' decrease monotonically as distance away from the center of its interval increases. Just as before, we can equivalently consider only the equations where $d \leq \frac{n}{2}$. The resulting system is described in Equation B.15.

$$\begin{aligned} \forall i = 1, 2, ..., \lceil \frac{n}{2} \rceil - 1 \\ \forall j = 2i, 2i+1, ..., n-1 \\ \frac{y_{1i}}{\frac{j}{2}} + \frac{y_{1(n-i+1)}}{n-\frac{j}{2}} = \frac{y_{1(j-i+1)}}{\frac{j}{2}} + \frac{y_{1(n-j+i)}}{n-\frac{j}{2}} \end{aligned} \tag{B.15}$$

Unfortunately, this system is overdetermined for $n > 6$. In fact, the number of equations in this system for a given n is described in Equation B.16.

$$\sum_{i=1}^{\lceil \frac{n}{2} \rceil - 1} \sum_{j=2i}^{n-1} 1 = \sum_{i=1}^{\lceil \frac{n}{2} \rceil - 1} n - 2i = (\lceil \frac{n}{2} \rceil - 1)(n - \lceil \frac{n}{2} \rceil) \tag{B.16}$$

Lemma 15 *The system described in Equation B.15 has only $n-2$ linearly independent equations.*

Proof. First, we rewrite the system slightly to remove the denominators.

$$\begin{aligned} \forall i = 1, 2, ..., \lceil \frac{n}{2} \rceil - 1 \\ \forall j = 2i, 2i+1, ..., n-1 \\ (n - \frac{j}{2}) y_{1i} + (\frac{j}{2}) y_{1(n-i+1)} = (n - \frac{j}{2}) y_{1(j-i+1)} + (\frac{j}{2}) y_{1(n-j+i)} \end{aligned}$$

It may be helpful to visualize this system as the equations arranged onto a grid corresponding to different values of i and j. Note that for $i = 1$, there are $n-2$ equations and they

are clearly linearly independent. In addition, for $i > 1$ each equation can be rewritten as a linear combination of these first $n-2$ equations. We denote the equation associated with $i = a$, $j = b$ by $eqn_{a,b}$.

$$eqn_{i,j} = -\frac{n+i-j-1}{n-1}(eqn_{1,i}) + \frac{n-i}{n-1}(eqn_{1,(j-i+1)}) + \frac{i}{n-1}(eqn_{1,(j-i)}) - \frac{j-i}{n-1}(eqn_{1,(n-i)})$$

Thus, the system has exactly $n-2$ linearly independent equations. □

Finally, we add our familiar restrictions: making the labels probabilistic and setting the last label value to zero. These additional equations increase the rank of the system to n.

$$\sum_{k=1}^{n} y_{1k} = 1,\ y_{1n} = 0 \tag{B.17}$$

The resulting system always has one solution, and this solution allows us to separate n classes using two points. □

Remark *We can find the exact solution to the system in Equation B.15 combined with the equations from Equation B.17 for a given n.*

$$y_{1i} = y_{2(n-i)} = \frac{\sum_{j=i}^{n-1} j}{\sum_{j=1}^{n-1} j^2} = \frac{n(n-1) - i(i-1)}{2\sum_{j=1}^{n-1} j^2},\ i = 1, 2, ..., n$$

Remark *The two points do not have to be n units apart. The same result can be shown to hold for two points any distance part by dividing that distance into n intervals of equal length and scaling the associated constants accordingly.*

B.8 Proof of Theorem 8

Proof. We can rephrase our objective as the following. Find the minimum number of prototypes required on each circle, such that the nearest prototype to any point along the circumference of the i^{th} circle, is also on the i^{th} circle.

Let the number of prototypes on the i^{th} circle (with radius $i * c$ for some positive constant c) be denoted by n_i. Since the prototypes must be evenly spread out along the

circumference of each circle, the furthest point on circle i from any prototype of circle i, must be the point located at the arc midpoint of two neighboring prototypes. In that case, a sufficient condition is that the shortest distance between two neighboring circles (c), be longer than the distance between neighboring prototypes and arc midpoints on the i^{th} circle. Since c is constant, we can only modify the distance between prototypes and arc midpoints. We can do so by changing the number of prototypes found on the i^{th} circle. The Euclidean distance between a prototype and its neighboring arc midpoint is $d_i = r_i\sqrt{2 - 2cos(\frac{l_i}{r_i})}$ where $r_i = i * c$ is the radius of the i^{th} circle, and $l_i = \frac{2\pi r}{2n}$ is the arclength between neighboring prototypes and arc midpoints. Thus $d_i = ic\sqrt{2 - 2cos(\frac{\pi}{n_i})}$ and we want $d_i < c \; \forall i = 1, 2, ...$ Solving for n_i we get the following inequality.

$$n_i > \frac{\pi}{cos^{-1}(1 - \frac{1}{2i^2})} \tag{B.18}$$

We can use the approximation $cos^{-1}(1-y) \approx \sqrt{2y}$ to get that $\frac{\pi}{cos^{-1}(1-\frac{1}{2i^2})} \approx \frac{\pi}{\frac{1}{i})} = i\pi$. $\square$

Appendix C

SecDD: Efficient and Secure Method for Remotely Training Neural Networks

We leverage what are typically considered the worst qualities of deep learning algorithms - high computational cost, requirement for large data, no explainability, high dependence on hyper-parameter choice, overfitting, and vulnerability to adversarial perturbations - in order to create a method for the secure and efficient training of remotely deployed neural networks over unsecured channels.

C.1 Introduction

We consider the situation where a neural network must be trained using proprietary or confidential data, but only an unsecured channel is available for providing data to the network. We assume that any data transmitted over this channel can be accessed by other parties. Our objective is to transmit data that will train the target network to desired accuracy, but will be unusable by other networks, and will also not reveal any information through qualitative inspection. A secondary objective is to improve efficiency by minimizing the size of our transmission.

We propose using dataset distillation, the process of representing the knowledge of a large dataset using a smaller number of synthetic samples [Wang et al., 2018], as a

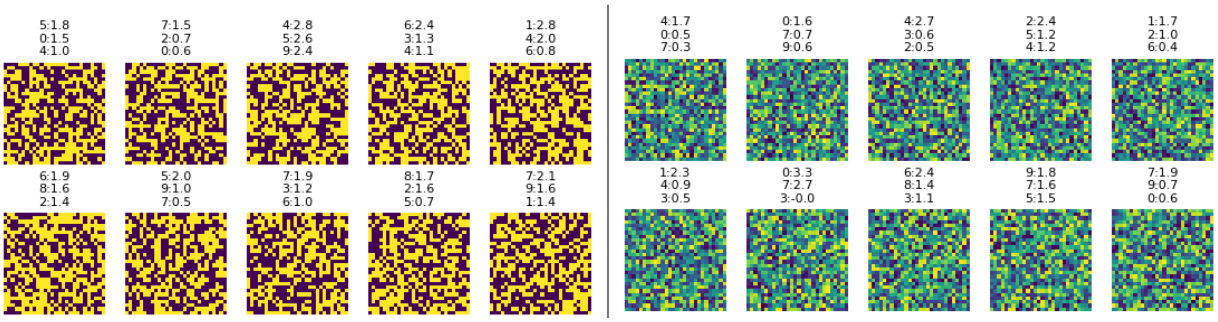


Figure C.1: SecDD can create various sets of 10 synthetic MNIST images that train target networks to over 95% accuracy while visually appearing to consist almost entirely of noise. Each image is labeled with its top 3 classes and their associated logits.

method for efficiently and securely training neural networks. Soft-Label Dataset Distillation (SLDD) is an extension to the dataset distillation algorithm that achieves better performance by learning distillation labels along with the distillation images [Sucholutsky and Schonlau, 2019]. We propose Secure Dataset Distillation (SecDD) as an extension of SLDD that intentionally overfits samples to a target network in order to create tiny privacy-preserving training sets that reduce transmission size by several orders of magnitude. These synthetic samples can only train a network with the same architecture and random initialization as the target network. These synthetic training samples can also be designed to be qualitatively dissimilar to real samples; even appearing to belong to completely unrelated datasets. In order to retrieve private information from the synthetic samples, an attacker would need to discover both the architecture and the random initialization of the target network. To do so, an attacker would have to perform Neural Architecture Search (NAS) on the synthetic training set. Fortunately, NAS methods are computationally intensive and data-hungry [Strubell et al., 2019]. In particular, NAS has been shown to be ineffective when using small distilled datasets as proxies for the full training set [Shleifer and Prokop, 2019]. In addition, the search space for the NAS algorithm grows rapidly as the size of the target network increases. If the target network contains unusual components, it may even be impossible for NAS to find it as the search space is often constrained to popular network components. A good analogy for this is the process for creating a strong password: having a long password with special characters greatly increases the search space making it difficult for a brute-force attack to succeed.

C.2 Related Work

Prototypes have long been studied in the context of algorithms like k-nearest neighbours [Chang, 1974, Sánchez, 2004]. Generally speaking, prototype methods aim to approximate datasets using a smaller number of samples. Prototype selection methods aim to choose prototypes from the actual dataset [Garcia et al., 2012]. Prototype generation methods, like the k-means algorithm, instead create synthetic samples [Nanni and Lumini, 2009, Triguero et al., 2011b]. Most prototype methods use hard labels, but some propose more complex prototypes that aim to increase efficiency [Mettes et al., 2019, Sucholutsky and Schonlau, 2021a].

Dataset distillation can be described as a family of prototype generation methods intended for use with neural networks [Wang et al., 2018, Sucholutsky and Schonlau, 2019, Bohdal et al., 2020]. Flexible Dataset Distillation (LD) is a recently proposed extension of dataset distillation that learns unrestricted labels as in SLDD but for a small fixed set of real images taken from the training dataset [Bohdal et al., 2020].

C.3 Secure Dataset Distillation

When using DD, and especially SLDD, with fixed initialization, the distilled images qualitatively look mostly like random noise, yet they train the target network to impressive accuracies. Several studies criticized this behavior and proposed algorithms that result in clearer patterns [Zhao et al., 2020, Bohdal et al., 2020]. However, we instead utilize the lack of interpretability to preserve privacy by transmitting samples that do not resemble the ones in the original dataset. We modify the SLDD algorithm to encourage aggressive overfitting to the target network. While SLDD generally uses one-hot encoding to initialize the distilled labels, we experiment with alternative initializations that encourage more class mixing, and result in less identifiable features in the distilled images.

Flexible Dataset Distillation uses fixed distilled images and instead only learns the associated soft labels. In fact, Bohdal et al. [2020] showed that the frozen images can come from a different dataset and still train the model to high accuracies on the target dataset. We propose two SecDD modes that leverage this idea in order to mask transmissions. In the first mode, fixed distilled images are initialized as random noise to ensure that attackers would not be able to discern qualitative features by observing transmissions. In the second mode, fixed distilled images are taken from a different, completely unrelated dataset.

For both modes, the soft labels for the images are learned through backpropagation. While aiding with privacy preservation, these modes may require larger distilled datasets

to train models to the same accuracies than when using regular SLDD. Two example sets of fixed, random-noise samples used for training a target network to achieve high accuracy on MNIST are shown in Figure C.1. The two sets used different initializations which resulted in visually different images, but both initializations still result in high accuracy of around 95% for the target network.

C.4 Conclusion and Future Work

We have proposed a method for producing synthetic data that can be used to securely and efficiently train remotely deployed neural networks over unsecured channels. These transmissions can even appear to contain random noise or completely unrelated data while still training target neural networks to high accuracies.

We have so far only conducted exploratory experiments to validate our claims and are working on conducting a comprehensive set of experiments that would quantify the improvements in privacy preservation and efficiency that SecDD can provide.