Deep Learning in Unconventional Domains

Thesis by
Milan Cvitkovic

In Partial Fulfillment of the Requirements for the
Degree of
Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2020
Defended March 10, 2020

ACKNOWLEDGEMENTS

To my parents, Kate Turpin and Michael Cvitkovic, and my sister, Adriana Cvitkovic: thank you for your boundless love and support. And all the teasing.

To my thesis committee, Prof. Anima Anandkumar, Prof. Yisong Yue, Prof. Adam Wierman, and Prof. Thomas Vidick: thank you for your invaluable guidance and encouragement throughout my time at Caltech, both in your roles as committee members and beyond.

To Prof. Anima Anandkumar, Prof. Pietro Perona, and Prof. Stefano Soatto: thank you for facilitating my affiliation with Amazon Web Services, without which this dissertation would never have been possible.

To all my coauthors who helped with the work in this thesis — Badal Singh, Prof. Anima Anandkumar, Dr. Günther Koliander, and Dr. Zohar Karnin — and to all my other collaborators during my PhD — Joe Marino, Dr. Grant Van Horn, Dr. Hamed Alemohammad, Prof. Yisong Yue, Keng Wah Loon, Laura Graesser, Jiahao Su, and Prof. Furong Huang: thank you for your generosity and your brilliance.

To Lee, Carmen, Kirsten, Leslie, James, Alex, Josh, Claudia, and Patricia: thank you for being cheerful stewards of daily rituals.

And most of all to my advisor, Prof. Thomas Vidick: thank you for your unwavering support of a non-traditional advisee.

ABSTRACT

Machine learning methods have dramatically improved in recent years thanks to advances in deep learning (LeCun, Bengio, and Hinton, 2015), a set of methods for training high-dimensional, highly-parameterized, nonlinear functions. Yet deep learning progress has been concentrated in the domains of computer vision, vision-based reinforcement learning, and natural language processing. This dissertation is an attempt to extend deep learning into domains where it has thus far had little impact or has never been applied. It presents new deep learning algorithms and state-of-the-art results on tasks in the domains of source-code analysis, relational databases, and tabular data.

PUBLISHED CONTENT AND CONTRIBUTIONS

Cvitkovic, Milan (2020). "Supervised Learning on Relational Databases with Graph Neural Networks". In: *Current version in submission; previous version in RLGM workshop ICLR 2019*. URL: `https://arxiv.org/abs/2002.02046`.
M.C. conceived of and designed the project, developed and implemented the method, ran the experiments, and wrote the manuscript.

Cvitkovic, Milan and Zohar Karnin (2020). "Subset-Restricted Finetuning on Tabular Data with Transformers". In: *In preparation*.
M.C. participated in designing the project and developing the method; he also implemented and ran the experiments and wrote the manuscript.

Cvitkovic, Milan (2019). "Some Requests for Machine Learning Research from the East African Tech Scene". In: *Proceedings of the Conference on Computing & Sustainable Societies, ACM COMPASS 2019, Accra, Ghana, July 3-5, 2019*, pp. 37–40. DOI: `10.1145/3314344.3332492`. URL: `https://doi.org/10.1145/3314344.3332492`.
M.C. designed the project, performed the interviews, and wrote the manuscript.

Cvitkovic, Milan and Günther Koliander (2019). "Minimal Achievable Sufficient Statistic Learning". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 1465–1474. URL: `http://proceedings.mlr.press/v97/cvitkovic19a.html`.
M.C. participated in designing the project, developing the theory and method, running the experiments, and writing the manuscript.

Cvitkovic, Milan, Badal Singh, and Anima Anandkumar (2019). "Open Vocabulary Learning on Source Code with a Graph-Structured Cache". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 1475–1485. URL: `http://proceedings.mlr.press/v97/cvitkovic19b.html`.
M.C. participated in designing the project, developing the method, running the experiments, and writing the manuscript.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

Over the past eight years deep learning (LeCun, Bengio, and Hinton, 2015) has become the dominant approach to machine learning, and machine learning has become the dominant approach to artificial intelligence.

Yet progress in deep learning has been concentrated in the domains of computer vision, vision-based reinforcement learning, and natural language processing. Upon seeing the revolution deep learning has sparked in these fields, one must ask: to what other problem domains can deep learning be applied, and to what effect?

This dissertation is an attempt to help answer this question.

The brief history of modern deep learning suggests that necessary conditions for deep learning efficacy include having lots of high-dimensional data and appropriate computer hardware to process it. So why have domains like database analysis, genomics, tabular machine learning, chemistry, industrial control systems, information security, formal methods, or neuroscience, which meet these criteria, not had deep learning revolutions of their own (yet)?

One reason is that deep learning research moves at the pace of deep learning software, and support for domains beyond images, reinforcement learning simulators, and text is slow in coming. This is a major reason why all code written in service of this dissertation has been open-sourced.

Another reason, in parallel to software, is that machine learning research requires datasets and benchmarks. For many less-well-studied domains data are inaccessible or difficult to use, and there is no consensus on benchmarks. The work presented in this dissertation includes the curation and release of three new, large-scale datasets and benchmarks in hopes of addressing this issue.

But perhaps the most important reason is the dependence of deep learning on model architecture. Convolutional networks for vision and transformer models for natural language are not successful merely because they are deep, but because their structure matches their domain of application. Much is not understood about why this is the case, but it is clear that new domains require new model designs. Most of this dissertation is devoted to developing them.

Besides the benefits to other fields of science, engineering, and technology, bringing deep learning to other domains is important for machine learning itself. Machine learning is a discipline defined by benchmarks, and if all benchmarks are computer vision, reinforcement learning, or natural language benchmarks, the field risks unhelpfully narrowing its scope. In particular, it risks narrowing its scope to the problems most immediately visible to the types of people who do deep learning research.

This leads us to the deeper motivation for this dissertation, which is to push deep learning into domains that matter for those with fewer resources, computational and otherwise. To that end, this dissertation proceeds as follows:

Chapter 2 presents the results of extensive interviews with scientists, engineers, and executives in East Africa on what problems they face and how machine learning might help solve them. This project, though nontechnical, was a significant factor in orienting the research direction taken in this dissertation.

Chapter 3 delves into the domain of deep learning on computer source code. We present an extension to graph neural network-based models that addresses the unique open-vocabulary problem faced in learning tasks on computer source code and thereby improves performance.

Chapter 4 presents a new application of deep learning to supervised learning problems on relational databases. We introduce new datasets and a promising graph neural network-based system for this domain of considerable practical importance.

Chapter 5 explores the performance of deep learning models on tabular data, often considered an area where deep learning has little to offer. We introduce a new benchmark that mostly confirms this notion, but also explore a new regime called subset-restricted finetuning for which deep learning proves useful.

Chapter 6 examines issues with information-theoretic regularization methods used in Chapter 5, and presents a method called Minimal Achievable Sufficient Statistic Learning for circumvents them.

Finally, **Chapter 7** takes the lessons learned in Chapters 5 and 6 and applies them to the problems of Chapter 4, leading to noticeable performance improvements.

References

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). “Deep learning”. In: *Nature* 521.7553, pp. 436–444.

Chapter 2

SOME REQUESTS FOR MACHINE LEARNING RESEARCH FROM THE EAST AFRICAN TECH SCENE

2.1 Introduction

Based on 46 in-depth interviews with scientists, engineers, and executives, this chapter presents a list of concrete machine research problems, progress on which would directly benefit technology ventures in East Africa.[1]

The goal of this chapter is to give machine learning researchers a fuller picture of where and how their efforts as scientists can be useful. The goal is *not* to highlight research problems that are *unique* to East Africa — indeed many of the problems listed below are of general interest in machine learning. The problems on the list are united solely by the fact that technology practitioners and organizations in East Africa reported a pressing need for their solution.

We are aware that listing machine learning problems without also providing data for them is not a recipe for getting those problems solved. If the reader is interested in any of the problems below, we will gladly introduce them to the organizations or people with access to data for those problems. But to protect privacy and intellectual property, we have not attributed problems to specific organizations or people.

2.2 Research Problems

Natural Language Processing

Mobile phone ownership and use, particularly of feature phones, is widespread in East Africa. SMS and voice interactions are one of the few big data sources in the region. Moreover, since literacy, technological and otherwise, remains low, natural language interfaces and conversational agents have huge potential for impact.

A few organizations in East Africa are trying to leverage NLP methods, but they face many challenges, including the following:

Handling Rapid Code-Switching with Models trained on Single Language Corpora: In SMS and voice communication, many East Africans rapidly code-switch (switch

[1]This chapter's focus on East Africa is based on where the author had work experience and connections. But many of the problems listed are likely relevant to other regions.

between languages). This is usually done multiple times per sentence, throughout an interaction, and usually between English and another language.

Despite perhaps striking some readers as a fringe linguistic phenomenon, every engineer interviewed herein who had worked with NLP models reported that this was a significant issue for them. It makes using models trained on single-language corpora — the most widely available corpora — difficult. Additionally, the number of possible combinations of local languages plus English makes training language models for each combination infeasible.

Named Entity Recognition with Multiple-Use Words: NER is an important part of NLP pipelines in East Africa. However, the entity detection step of NER is complicated by the fact that English words are commonly used as names in East Africa, e.g. Hope, Wednesday, Silver, Editor, Angel, Constant. Capitalization is not used regularly enough in SMS to help.

Location Extraction from Natural Language: Despite the proliferation of mobile phones, GPS availability and accuracy is limited in East Africa. Extraction of locations from natural language is therefore critical for numerous applications, from localization of survey respondents to building speech-only navigation apps (for use with feature phones).

This task is complicated by the fact that most rural locales, and many urban ones, lack usable addressing schemes. Most people specify locations and directions contextually, e.g. “My house is in Kasenge; it’s the yellow one two minutes down the dirt road across from the Medical Center.”

Even approximate or probabilistic localization based on such location information from natural language would be invaluable. Combining satellite imagery or user interaction would be particularly impressive.

Priors for Autocorrect and Low-Literacy SMS Use: SMS text contains many language misuses due to a combination of autocorrection and low literacy, e.g. “poultry farmer” becoming “poetry farmer”. Such mistakes are bound to occur in any written language corpus, but engineers working with rural populations in East Africa report that this is a prevalent issue for them, confounding the use of pretrained language models. This problem also exists to some degree in voice data with respect to English spoken in different accents. Priors over autocorrect substitution rules, or custom, per-dialect confusion matrices between phonetically similar words could potentially help.

Disambiguating Similar Languages: Numerous languages are spoken in East Africa, many of which are quite distinct in meaning but can be difficult to identify in small inputs or when rendered into text (especially when combined with typographical errors). Even when data are available for tasks like sentiment analysis in multiple similar languages, performing tasks when the input language is ambiguous and from a set of similar languages remains an open problem.

Data Gaps: Specific domains for which data and pretrained models are limited include East African languages; non-Latin-character text; non-Western English.

Computer Vision

Satellite data and mobile phone data (including mobile money data) are the primary sources of big data in East Africa. But satellite data are the more abundant and open of the two, which has led to widespread use of computer vision models for satellite data in the region. Mobile phone cameras also have enabled the use of computer vision for applications ranging from disease identification to stock management. Yet many research problems remain to be solved to maximize the utility of computer vision in East Africa.

Specialized Models for Satellite Imagery:[2] Excellent work has been done using satellite data in East Africa, and its importance to the region cannot be overstated. But satellite data, as a subset of general image data, have many unique properties. Little work has been done to develop specialized image models to exploit/compensate for these properties, some of which include:

- The presence of reliable image metadata, such as precise geolocation of images on the Earth's surface, camera position and orientation, image acquisition time, and pixel resolution.
- The inherent time-series nature of satellite imagery, which consists of repeated images of the same location across time.
- Imagery that is captured at different wavelengths and modes, each with unique properties, e.g. optical vs. near-infrared or passive vs. active/radar.
- The presence of cloud occlusion (particularly in passive measurements), cloud shadows, and the ensuing illumination variability these both cause.

[2]A more detailed explanation of this topic, written with the help of Dr. Hamed Alemohammad, can be found at `https://milan.cvitkovic.net/assets/documents/Satellite_Imagery_2018.pdf`.

- Multiple resolutions of imagery for the same ground truth, and the frequent need to transfer models trained on one resolution to another.

Map Generation: Generating road maps and identifying homes in rural areas are critical tasks for many East African organizations. But road networks and buildings change rapidly in East Africa due to construction and weather, and road identification is challenging when roads are unpaved. Rapid, frequent, satellite-imagery-based map making is thus a high value use of computer vision in East Africa.

While the general task of generating maps and road-networks from satellite images is not new (Bastani et al., 2018), the scarcity of labeled map data in East Africa means structured prediction models for map making in the region need to be developed that can better leverage prior knowledge about road network structure.

Document Understanding and OCR: Many East African government agencies, organizations, and businesses have all their records on paper. Given the general scarcity of data in the region, solutions for automated information extraction from such documents is greatly desired. These documents are usually handwritten, however, so existing OCR extraction pipelines have not proven usable.

Data Gaps: Specific domains for which data and pretrained models are limited include: dark-skinned faces; high-resolution satellite imagery of East African geography; general ground-level imagery of East Africa (models pretrained on, e.g., ImageNet have trouble identifying East African consumer goods, vehicles, buildings, flora, etc.); low-resolution imagery of documents (bank statements, government records, IDs); images taken with feature phone cameras.

Data *Au Naturel*

Data from East Africa are scarce and expensive to obtain, and they almost never come to practitioners as perfectly i.i.d. real-valued vector pairs. Research into models better suited for data-as-they-are was the most common request heard from interviewees.

Learning Directly on Relational Databases: Many data in East Africa (and the rest of the world) are stored in relational, usually SQL, databases, including perhaps the most abundant, though least open, source of data in the region: private business data. Extracting data from a relational database and converting them to real-valued vectors is one of the largest time expenditures of working data scientists and engineers in East Africa. Moreover, converting the inherently relational data in a SQL database into vectors that a machine learning model can pretend are i.i.d. is, even when done by the best data scientist, a fraught process.

A system that could build a structured model based on a relational database's particular schema and train it without requiring manual ETL and flattening of data into vectors would be beneficial for many of the organizations interviewed. We develop such a system in Chapter 4.

Merging datasets: The most common way interviewees handled data scarcity was by merging datasets. Versions of this tactic included:

- Combining surveys containing differently-phrased versions of essentially the same question.
- Combining customer or surveyee interaction histories gathered over different but overlapping times.
- Augmenting survey data with satellite data, without accurate location information in the surveys.

No interviewee was familiar with any techniques or models well-suited for these scenarios. The obvious choice given recent advances in natural language processing is some form of transfer learning. We are not aware of any work on transfer learning for general tabular data, though we explore this briefly in Chapter 5.

Adversarial Inputs (no, not that kind): East African economies are low-trust relative to those of wealthier regions (Zak and Knack, 2001). Interviewees who use machine learning with surveys or customer interaction data reported spending significant effort fighting fraud or dishonesty.

This issue is pervasive not only with companies that offer products, e.g. loans, based on survey responses. It is present even in surveys where the surveyee stands to gain nothing. Prof. Tim Brown suggests distinguishing such adversarial inputs into two categories: “defensive”, where surveyees do not trust your intent and thus obfuscate or misrepresent themselves in responses, and “offensive”, where surveyees are searching for the answer you want to hear.

Handling fraud is not exclusively a machine learning problem by any means. But there are interesting open research questions around building models, especially interactive systems, that are wary of being gamed, e.g. safe reinforcement learning models for when the danger is not extreme variance in rewards, but rather adversarial corruption of observations.

Resource-Limited Machine Learning

Cellular access is widespread in East Africa, but it remains patchy in many rural areas, and electrical power for devices is scarce. Additionally, some countries like Kenya have laws that prevent some data from leaving the country. These issues limit the utility of existing machine learning technologies.

Models for small, low-powered devices: The utility of reducing the computation and memory requirements of modern machine learning models is well known (Cheng et al., 2017). It is listed here simply to reiterate its importance to East African organizations trying to use deep learning.

Communication-Limited Active Learning and Decision Making: Survey collection is an expensive, slow process, usually done by sending enumerators (human data collectors) across large regions to conduct interviews. But it is also one of the only sources of data about East Africa, especially about poorer and rural regions. Maximizing the information collected in such surveys is thus critical.

One potential strategy to do this is active learning. Survey enumerators typically have access to smartphones with data-collection software like ODK,[3] making it potentially viable to employ machine learning techniques like active learning in surveying. However, this active learning scenario does not fit neatly into the standard settings of query synthesis, stream-based, or pool-based. It is a multi-agent, cost-sensitive active learning task, where the agents cannot reliably communicate with one another, and where the agent's decision is not just whom to survey but also which subset of a large set of questions to ask each surveyee.

In a similar vein, some organizations are piloting automated money or food distribution programs in rural areas based on satellite, weather, survey, and other data. This is a distributed decision-making task with the same complications as the surveying case described above.

Other

Reinforcement Learning: No interviewee reported using any reinforcement learning methods. However interest was expressed in it, particularly regarding machine teaching and using reinforcement learning in simulations, e.g. using reinforcement learning in epidemiological simulations to find worst case scenarios in outbreak planning.

[3]`https://opendatakit.org/software/`

Machine Teaching: There is a shortage of good educational resources and teachers in East Africa. Several initiatives exist that use mobile phones as an education platform. Practitioners were interested in using ideas from machine teaching in their work to personalize content delivered. However, we did not encounter anyone who had employed any results from the machine teaching literature at this point.

Uncertainty Quantification: An important factor that keeps the wealth of rich regions from moving into poorer regions like East Africa, despite the fact that it should earn greater returns there, is risk (Alfaro, Kalemli-Ozcan, and Volosovych, 2008). Not all risk can be machine-learned away by any means. But (accurate) predictive models are risk-reduction tools.

Machine learning models are most useful for risk-reduction when they can (accurately) quantify their uncertainty. This is particularly true when data are scarce, as they usually are in East Africa. UQ is hardly a new problem in machine learning, but it is listed here to reiterate its importance to the organizations interviewed. Importantly, when used in East Africa, UQ is typically much more concerned with conservatively quantifying overall downside risk, with respect to some quantity of interest, than characterizing overall model uncertainty around point predictions.

2.3 Interviews Conducted

- Chris Albon, Devoted Health
- Hamed Alemohammad, Radiant Earth Foundation
- Elvis Bando, Independent Data Science Consultant
- Joanna Bichsel, Kasha
- Prof. Tim Brown, Carnegie Mellon University Africa
- Ben Cline, Apollo Agriculture
- Johannes Ebert, Gravity.Earth
- Dylan Fried, Lendable
- Sam Floy, The East Africa Business Podcast
- Lukas Lukoschek, MeshPower
- Daniel Maison, Sky.Garden
- Lauren Nkuranga, GET IT
- Mehdi Oulmakki, African Leadership University
- Jim Savage, Lendable
- Rob Stanley, Wefarm
- Linda Dounia Rebeiz, Eneza Education
- Kamande Wambui, mSurvey
- Muthoni Wanyoike, Instadeep
- Anonymous individuals with the following affiliations:
  Fenix International, Kasha, GET IT, Carnegie Mellon University Africa,

kLab, Safaricom, Give Directly (2), Sankofa.africa, Rwanda Online, We Effect, Andela, Moringa School (2), Nelson Mandela African Institute of Science and Technology, IBM Research Africa, Sky.Garden (2), Medic Mobile, Engineering and data science students (9), independent data science consultant, anonymous company

References

Alfaro, Laura, Sebnem Kalemli-Ozcan, and Vadym Volosovych (2008). "Why Doesn't Capital Flow from Rich to Poor Countries? An Empirical Investigation". In: *The Review of Economics and Statistics* 90.2, pp. 347–368. DOI: 10.1162/rest.90.2.347. URL: https://doi.org/10.1162/rest.90.2.347.

Bastani, Favyen et al. (2018). "RoadTracer: Automatic Extraction of Road Networks from Aerial Images". In:

Cheng, Yu et al. (2017). "A Survey of Model Compression and Acceleration for Deep Neural Networks". In: *CoRR* abs/1710.09282. URL: http://arxiv.org/abs/1710.09282.

Zak, Paul J. and Stephen Knack (2001). "Trust and Growth". In: *The Economic Journal* 111.470, pp. 295–321. DOI: 10.1111/1468-0297.00609. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/1468-0297.00609.

Chapter 3

OPEN VOCABULARY LEARNING ON SOURCE CODE WITH A GRAPH–STRUCTURED CACHE

3.1 Introduction

Computer program source code is an abundant and accessible form of data from which machine learning algorithms could learn to perform many useful software development tasks, including variable name suggestion, code completion, bug finding, security vulnerability identification, code quality assessment, or automated test generation. But despite the similarities between natural language and source code, deep learning methods for Natural Language Processing (NLP) have not been straightforward to apply to learning problems on source code (Allamanis, Barr, et al., 2017).

There are many reasons for this, but two central ones are:

1. *Code's syntactic structure is unlike natural language.* While code contains natural language words and phrases in order to be human-readable, code is not meant to be read like a natural language text. Code is written in a rigid syntax with delimiters that may open and close dozens of lines apart; it consists in great part of references to faraway lines and different files; and it describes computations that proceed in an order often quite distinct from its written order.

2. *Code is written using an open vocabulary.* Natural language is composed of words from a large but mostly unchanging vocabulary. Standard NLP methods can thus perform well by fixing a large vocabulary of words before training, and labeling the few words they encounter outside this vocabulary as "unknown". But in code every new variable, class, or method declared requires a name, and this abundance of names leads to the use of many obscure words: abbreviations, brand names, technical terms, etc.[1] A model must be able to

[1] We use the terminology that a *name* in source code is a sequence of *words*, split on `CamelCase` or `snake_case`, e.g. the method name `addItemToList` is composed of the words `add`, `item`, `to`, and `list`.
We also use the term *variable* in its slightly broader sense to refer to any user-named language construct, including function parameter, method, class, and field names, in addition to declared variable names.

reason about these newly-coined words to understand code.

The second of these issues is significant. To give one indication: 28% of variable names contain out-of-vocabulary words in the test set we use in our experiments below. But more broadly, the open vocabulary issue in code is an acute example of a fundamental challenge in machine learning: how to build models that can reason over unbounded domains of entities, sometimes called "open-set" learning. Despite this, the open vocabulary issue in source code has received relatively little attention in prior work.

The first issue, however, has received attention. A common strategy in prior work is to represent source code as an Abstract Syntax Tree (AST) rather than as linear text. Once in this graph-structured format, code can be passed as input to models like Recursive Neural Networks or Graph Neural Networks (GNNs) that can, in principle, exploit the relational structure of their inputs and avoid the difficulties of reading code in linear order (Allamanis, Brockschmidt, and Khademi, 2018).

In this chapter we present a method for extending such AST-based models for source code in order to address the open vocabulary issue. We do so by introducing a Graph-Structured Cache (GSC) to handle out-of-vocabulary words. The GSC represents vocabulary words as additional nodes in the AST as they are encountered and connects them with the edges to where they are used in the code. We then process the AST+GSC with a GNN to produce outputs. See Figure 3.1.

We empirically evaluated the utility of a Graph-Structured Cache on two tasks: a code completion (a.k.a. fill-in-the-blank) task and a variable naming task. We found that using a GSC improved performance on both tasks at the cost of an approximately 30% increase in training time. More precisely: even when using hyperparameters optimized for the baseline model, adding a GSC to a baseline model improved its accuracy by at least 7% on the fill-in-the-blank task and 103% on the variable naming task. We also report a number of ablation results in which we carefully demonstrate the necessity of each part of the GSC to a model's performance.

3.2 Prior Work

Representing Code as a Graph

Given their prominence in the study of programming languages, Abstract Syntax Trees (ASTs) and parse trees are a natural choice for representing code and have been used extensively. Often models that operate on source code consume ASTs by linearizing them, usually with a depth-first traversal, as in (Amodio, Chaudhuri, and Reps, 2017), (Liu et al., 2017), or (J. Li et al., 2017) or by using AST paths as input features as in (Alon et al., 2018), but they can also be processed by deep learning models that take graphs as input, as in (White et al., 2016) and (Chen, Liu, and Song, 2018) who use Recursive Neural Networks (RveNNs) (Goller and Kuchler, 1996) on ASTs. RveNNs are models that operate on tree-topology graphs, and have been used extensively for language modeling (Socher et al., 2013) and on domains similar to source code, like mathematical expressions (Zaremba, Kurach, and Fergus, 2014; Arabshahi, Singh, and Anandkumar, 2018). They can be considered a special case of Message Passing Neural Networks (MPNNs) in the framework of (Gilmer et al., 2017): in this analogy RveNNs are to Belief Propagation as MPNNs are to Loopy Belief Propagation. They can also be considered a special case of Graph Networks in the framework of (Battaglia et al., 2018). ASTs also serve as a natural basis for models that generate code as output, as in (Maddison and Tarlow, 2014), (Yin and Neubig, 2017), (Rabinovich, Stern, and Klein, 2017), (Chen, Liu, and Song, 2018), and (Brockschmidt et al., 2019).

Data-flow graphs are another type of graphical representation of source code with a long history (Krinke, 2001), and they have occasionally been used to featurize source code for machine learning (Chae et al., 2017).

Most closely related to our work is the work of (Allamanis, Brockschmidt, and Khademi, 2018), on which our model is heavily based. (Allamanis, Brockschmidt, and Khademi, 2018) combine the data-flow graph and AST representation strategies for source code by representing code as an AST augmented with extra labeled edges indicating semantic information like data- and control-flow between variables. These augmentations yield a directed multigraph rather than just a tree,[2] so in (Allamanis, Brockschmidt, and Khademi, 2018) a variety of MPNN called a Gated Graph Neural Network (GGNN) (Y. Li et al., 2016) is used to consume the Augmented AST and produce an output for a supervised learning task.

Graph-based models that are not based on ASTs are also sometimes used for analyzing source code, like Conditional Random Fields for joint variable name prediction in (Raychev, Vechev, and Krause, 2015).

[2]This multigraph was referred to as a Program Graph in (Allamanis, Barr, et al., 2017) and is called an Augmented AST herein.

Reasoning about Open Sets

The question of how to gracefully reason over an open vocabulary is longstanding in NLP. Character-level embeddings are a typical way deep learning models handle this issue, whether used on their own as in (Kim et al., 2016), or in conjunction with word-level embedding Recurrent Neural Networks (RNNs) as in (Luong and Manning, 2016), or in conjunction with an *n*-gram model as in (Bojanowski et al., 2017). Another approach is to learn new word embeddings on-the-fly from context (Kobayashi et al., 2016). Caching novel words, as we do in our model, is yet another strategy (Grave, Cissé, and Joulin, 2017) and has been used to augment N-gram models for analyzing source code (Hellendoorn and Devanbu, 2017).

In terms of producing outputs over variable-sized input and outputs, also known as open-set learning, attention-based pointer mechanisms were introduced in (Vinyals, Fortunato, and Jaitly, 2015) and have been used for tasks on code, e.g. in (Bhoopchand et al., 2016) and (Vasic et al., 2019). Such methods have been used to great effect in NLP in e.g. (Gulcehre et al., 2016) and (Merity et al., 2017). The latter's pointer sentinel mixture model is the direct inspiration for the readout function we use in the Variable Naming task below.

Using graphs to represent arbitrary collections of entities and their relationships for processing by deep networks has been widely used (Johnson, 2017; Bansal, Neelakantan, and McCallum, 2017; Pham, Tran, and Venkatesh, 2018; Lu et al., 2017), but to our knowledge we are the first to use a graph-building strategy for reasoning at train *and* test time about an open vocabulary of words.

3.3 Preliminaries

Abstract Syntax Trees

An Abstract Syntax Tree (AST) is a graph — specifically an ordered tree with labeled nodes — that is a representation of some written computer source code. There is a 1-to-1 relationship between source code and an AST of that source code, modulo comments and whitespace in the written source code.

Typically the leaves of an AST correspond to the tokens written in the source code, like variable and method names, while the non-leaf nodes represent syntactic language constructs like function calls or class definitions. The specific node labels and construction rules of ASTs can differ between or within languages. The first step in Figure 3.1 shows an example.

Graph Neural Networks

A Graph Neural Network (GNN) is any differentiable, parameterized function that takes a graph as input and produces an output by computing successively refined representations of the input. Many GNNs have been presented in the literature, and several nomenclatures have been proposed for describing the computations they perform, in particular in (Gilmer et al., 2017) and (Battaglia et al., 2018). Here we give a brief recapitulation of supervised learning with GNNs using the Message Passing Neural Network framework of (Gilmer et al., 2017).

A GNN is trained using pairs (G, y) where $G = (V, E)$ is a graph defined by its vertices V and edges E, and y is a label. y can be any sort of mathematical object: scalar, vector, another graph, etc. In the most general case, each graph in the dataset can be a directed multigraph, each with a different number of nodes and different connectivity. In each graph, each vertex $v \in V$ has associated features $\boldsymbol{x}_v$, and each edge $(v, w) \in E$ has features $\boldsymbol{e}_{vw}$.

A GNN produces a prediction $\hat{y}$ for the label y of a graph $G = (V, E)$ by the following procedure:

1. A function S is used to initialize a hidden state vector $\boldsymbol{h}_v^0$ for each vertex $v \in V$ as a function of the vertex's features (e.g. if the $\boldsymbol{x}_v$ are words, S could be a word embedding function):
$$\boldsymbol{h}_v^0 = S(\boldsymbol{x}_v)$$
2. For each round t from 1 to T:
    a) Each vertex $v \in V$ receives the vector $\boldsymbol{m}_v^t$, which is the sum of "messages" from its neighbors, each produced by a function M_t:
    $$\boldsymbol{m}_v^t = \sum_{w \in \text{neighbors of } v} M_t(\boldsymbol{h}_v^{t-1}, \boldsymbol{h}_w^{t-1}, \boldsymbol{e}_{vw}).$$
    b) Each vertex $v \in V$ updates its hidden state based on the message it received via a function U_t:
    $$\boldsymbol{h}_v^t = U_t(\boldsymbol{h}_v^{t-1}, \boldsymbol{m}_v^t).$$
3. A function R, the "readout function", produces a prediction based on the hidden states generated during the message passing (usually just those at from time T):
$$\hat{y} = R(\{\boldsymbol{h}_v^t | v \in V, t \in \{1, \ldots, T\}\}).$$

GNNs differ in how they implement S, M_t, U_t, and R. But all these functions are differentiable and most are parameterized, so the model is trainable via stochastic gradient descent of a loss function on y and $\hat{y}$.

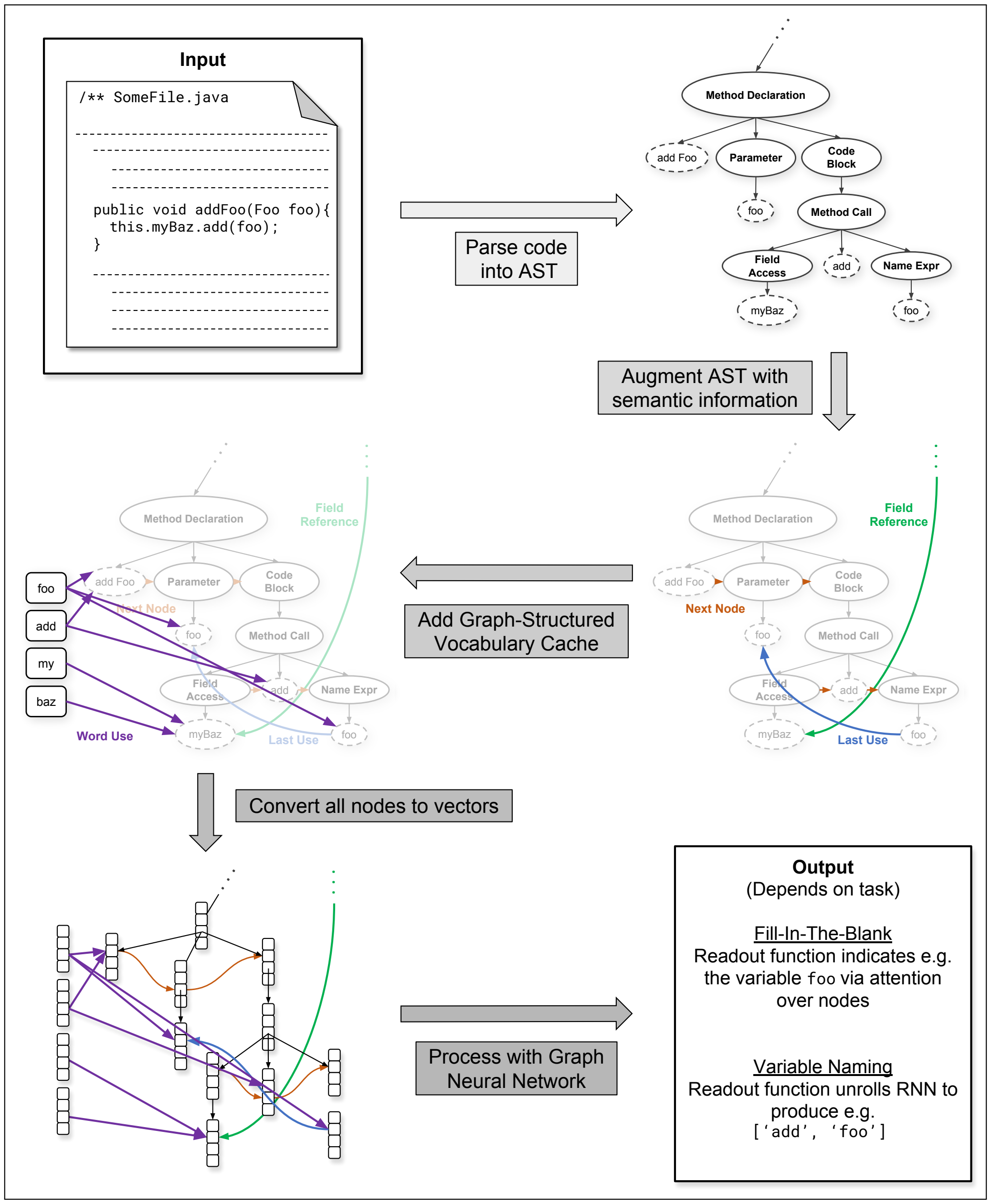

Input
/** SomeFile.java
public void addFoo(Foo foo){
this.myBaz.add(foo);
}
Parse code into AST
Method Declaration
add Foo
Parameter
Code Block
foo
Method Call
Field Access
add
Name Expr
myBaz
foo
Augment AST with semantic information
Field Reference
Next Node
Last Use
Add Graph-Structured Vocabulary Cache
foo
add
my
baz
Word Use
Convert all nodes to vectors
Process with Graph Neural Network
Output
(Depends on task)
Fill-In-The-Blank
Readout function indicates e.g. the variable foo via attention over nodes
Variable Naming
Readout function unrolls RNN to produce e.g. ['add', 'foo']

Figure 3.1: Our model's procedure for consuming a single input instance of source code and producing an output for a supervised learning task.

3.4 Model

Our model consumes an input instance of source code and produces an output for a supervised learning task via the following five steps, sketched in Figure 3.1:

1. Parse the source code (snippet, file, repository, version control history, etc.) into an Abstract Syntax Tree.

2. Add edges of varying types (details in Table 3.3) to this AST that represent semantic information like data- and control-flow, in the spirit of (Allamanis, Brockschmidt, and Khademi, 2018). Also add the reversed version of all edges with their own edge type. This results in a directed multigraph called an Augmented AST.

3. Further augment the Augmented AST by adding a Graph-Structured Cache. That is, add a node to the Augmented AST for each vocabulary word encountered in the input instance. Then connect each such "cache node" with an edge (of edge type `WORD_USE`) to all variables whose names contain its word.

4. Vectorize the Augmented AST + GSC graph into a form suitable for a GNN. (That is, perform Step 1 from Section 3.3.) Each AST node that doesn't represent a variable is vectorized as a learned embedding of the language construct it represents, e.g. `Parameter`, `Method Declaration`, etc. Each cache node and each node that represents a variable is vectorized as a learned linear map of the concatenation of a type embedding and a name embedding. The name embedding is a Character-Level Convolutional Neural Network (CharCNN) (Zhang, Zhao, and LeCun, 2015) embedding of the word/name the node contains. The type embedding is a learned embedding of the name of the Java type of the token it contains, e.g. `int`, a user-defined class, etc., with cache nodes having their own unique `Cache Node` type.

5. Process the graph with a GNN, as per Section 3.3. (That is, perform Steps 2 and 3 from Section 3.3.) The readout functions differ depending on the task and are described in the Experiments section below.

Our main contribution to previous works is the addition of Step 3, the Graph-Structured Cache step. The combination of relational information from the cache nodes' connections and lexical information from these nodes' CharCNN embeddings allows the model to, in principle, flexibly reason about words it never saw during training, but also recognize words it did. For example, it could potentially see a class named "`getGuavaDictionary`" and a variable named "`guava_dict`" and both (a) utilize the fact that the word "guava" is common to both names despite having never seen this word before, and (b) exploit learned representations for words like "get", "dictionary", and "dict" that it has seen during training.

3.5 Experiments

We evaluated our model, described in Section 3.4, on two supervised tasks: a Fill-In-The-Blank task and a Variable Naming task. For each task, we compare our model to others that differ in how they parse the code and how they treat the words they encounter. Table 3.1 details the different variations of the procedure in Section 3.4 against which we compare our model.

Code to reproduce all experiments is available online[3] and in the code accompanying this dissertation.

Data and Implementation Details

We used Java source code as the data for our experiments as it is among the most popular programming languages in use today (TIOBE, 2018; Github, 2017). To construct our dataset, we randomly selected 18 of the 100 most popular Java repos from the Maven repository[4] to serve as training data. See Table 3.2 for the list. Together these repositories contain about 500,000 non-empty, non-comment lines of code. We checked for excessive code duplication in our dataset (Lopes et al., 2017) using CPD[5] and found only about 7% of the lines to be contiguous, duplicated code blocks containing more than 150 tokens.

We randomly chose 3 of these repositories to sequester as an "Unseen Repos" test set. We then separated out 15% of the files in the remaining 15 repositories to serve as our "Seen Repos" test set. The remaining files served as our training set, from which we separated 15% of the datapoints to act as a validation set.

Our data preprocessor builds on top of the open-source Javaparser[6] library to generate ASTs of our source code and then augment the ASTs with the edges described in Table 3.3. We used Apache MXNet[7] as our deep learning framework. All hidden states

[3]https://github.com/mwcvitkovic/Deep_Learning_On_Code_With_A_Graph_Vocabulary

[4]https://mvnrepository.com/

[5]https://pmd.github.io/latest/pmd_userdocs_cpd.html

[6]https://javaparser.org/

[7]https://mxnet.apache.org/

Table 3.1: Nomenclature used in Experiments section. Each abbreviation describes a tweak/ablation to our full model as presented in Section 3.4. Using this nomenclature, our full model as described in Section 3.4 and shown in Figure 3.1 would be an "AugAST-GSC" model.

	Abbreviation	**Meaning**
Code Representation	AST	Skips Step 2 in Section 3.4.
	AugAST	Performs Step 2 in Section 3.4.
Vocab Strategies	Closed Vocab	Skips Step 3 in Section 3.4, and instead maintains word-embedding vectors for words in a closed vocabulary. In Step 4, name embeddings for nodes representing variables are produced by taking the mean of the embeddings of the words in the variable's name. Words outside this model's closed vocabulary are labeled as `<UNK>`. This is the strategy used in (Allamanis, Brockschmidt, and Khademi, 2018).
	CharCNN	Skips Step 3 in Section 3.4.
	Pointer Sentinel	Follows Steps 3 and 4 as described in Section 3.4, except it doesn't add edges connecting cache nodes to the nodes where their word is used. In the Variable Naming task, this is equivalent to using the Pointer Sentinel Mixture Model of (Merity et al., 2017) to produce outputs.
	GSC	Follows Steps 3 and 4 as described in Section 3.4.
Graph Neural Network	GGNN	Performs Step 5 in Section 3.4 using the Gated Graph Neural Network of (Y. Li et al., 2016).
	DTNN	Performs Step 5 in Section 3.4 using the Deep Tensor Neural Network of (Schütt et al., 2017).
	RGCN	Performs Step 5 in Section 3.4 using the Relational Graph Convolutional Network of (Schlichtkrull et al., 2017).

in the GNN contained 64 units; all GNNs ran for 8 rounds of message passing; all models used a 2-layer CharCNN with max-pooling to perform the name embedding; all models were optimized using the Adam optimizer (Kingma and Ba, 2015); all inputs to the GNNs were truncated to a maximum size of 500 nodes centered on the `<FILL-IN-THE-BLANK>` or `<NAME-ME>` tokens, as in (Allamanis, Brockschmidt, and Khademi, 2018). About 53% of input graphs were larger than 500 nodes before truncation. The only regularization we used was early stopping — early in our experiments we briefly tried L_2 and dropout regularization, but saw no improvements.

We performed only a moderate amount of hyperparameter optimization, but all of it was done on the baseline models to avoid biasing our results in favor of our model. Specifically, we tuned all hyperparameters on the Closed Vocab baseline model, and also did a small amount of extra learning rate exploration for the Pointer Sentinel baseline model to try to maximize its performance.

The Fill-In-The-Blank Task

In this task we randomly selected a single usage of a variable in some source code, replaced it with a `<FILL-IN-THE-BLANK>` token, and then asked the model to predict what variable should have been there. An example instance from our dataset is shown in Figure 3.2. This task is a simplified formulation of the VarMisuse task from (Allamanis, Barr, et al., 2017) that accomplishes the same goal of finding misused variables in code.

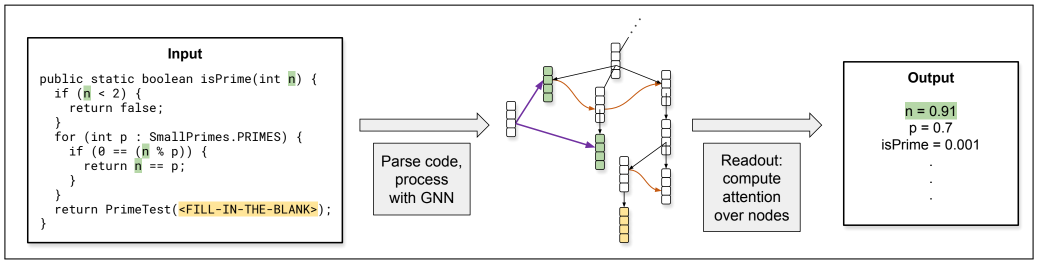
The models indicate their prediction for what variable should go in the blank by pointing with neural attention over all the nodes in the AugAST. This means all training and test instances only considered cases where the obfuscated variable appears somewhere else in the code. Single uses are rare however, since in Java variables must be declared before they are used. It also means there are sometimes multiple usages of the same, correct variable to which a model can point to get the right answer. In our dataset 78% of variables were used more two times, and 33% were used more than four times.

The models compute the attention weightings y_i for each Augmented AST node i differently depending on the readout function of the GNN they use. Models using a GGNN as their GNN component, as all those in Table 3.4 do, compute the attention weightings as per (Y. Li et al., 2016):

$$\hat{y}_i = \sigma\left(f_1(\boldsymbol{h}_v^T, \boldsymbol{h}_v^0)\right) \odot f_2(\boldsymbol{h}_v^T),$$

Table 3.2: Repositories used in experiments. All were taken from the Maven repository (https://mvnrepository.com/). Entries are in the form “group/repository name/version”.

Seen Repos
com.fasterxml.jackson.core/jackson-core/2.9.5
com.h2database/h2/1.4.195
javax.enterprise/cdi-api/2.0
junit/junit/4.12
mysql/mysql-connector-java/6.0.6
org.apache.commons/commons-collections4/4.1
org.apache.commons/commons-math3/3.6.1
org.apache.commons/commons-pool2/2.5.0
org.apache.maven/maven-project/2.2.1
org.codehaus.plexus/plexus-utils/3.1.0
org.eclipse.jetty/jetty-server/9.4.9.v20180320
org.reflections/reflections/0.9.11
org.scalacheck/scalacheck_2.12/1.13.5
org.slf4j/slf4j-api/1.7.25
org.slf4j/slf4j-log4j12/1.7.25
Unseen Repos
org.javassist/javassist/3.22.0-GA
joda-time/joda-time/2.9.9
org.mockito/mockito-core/2.17.0



```
public static boolean isPrime(int n) {
  if (n < 2) {
    return false;
  }
  for (int p : SmallPrimes.PRIMES) {
    if (0 == (n % p)) {
      return n == p;
    }
  }
  return PrimeTest(<FILL-IN-THE-BLANK>);
}
```

Parse code, process with GNN
Readout: compute attention over nodes
Output
n = 0.91
p = 0.7
isPrime = 0.001

Figure 3.2: Example of a model’s procedure for completing the Fill-In-The-Blank task. Each Fill-In-The-Blank instance is created by replacing a single usage of a variable (`n`, in this example) with the special token `<FILL-IN-THE-BLANK>`. The model then processes the code as depicted in Figure 3.1. To produce outputs, the model’s readout function computes a soft-attention weighting over all nodes in the graph; the model’s output is the variable at the node on which it places maximal attention. In this example, if the model put maximal attention weighting on any of the green-highlighted variables, this would be a correct output. If maximal attention is placed on any other node, it would be an incorrect output. Only in-scope usages of a variable are counted as correct.

Table 3.3: Edge types used in Augmented ASTs. The initial AST is constructed using the `AST` and `NEXT_TOKEN` edges, and then the remaining edges are added. The reversed version of every edge is also added as its own type (e.g. `reverse_AST`, `reverse_LAST_READ`) to let the GNN message passing occur in both directions.

Edge Name	Description
`AST`	The edges used to construct the original AST.
`NEXT_TOKEN`	Edges added to the original AST that specify the left-to-right ordering of the children of a node in the AST. These edges are necessary since ASTs have ordered children, but we are representing the AST as a directed multigraph.
`COMPUTED_FROM`	Connects a node representing a variable on the left of an equality to those on the right. (E.g. edges from `y` to `x` and `z` to `x` in `x = y + z`.) The same as in (Allamanis, Brockschmidt, and Khademi, 2018).
`LAST_READ`	Connects a node representing a usage of a variable to all nodes in the AST at which that variable's value could have been last read from memory. The same as in (Allamanis, Brockschmidt, and Khademi, 2018).
`LAST_WRITE`	Connects a node representing a usage of a variable to all nodes in the AST at which that variable's value could have been last written to memory. The same as in (Allamanis, Brockschmidt, and Khademi, 2018).
`RETURNS_TO`	Points a node in a return statement to the node containing the return type of the method. (E.g. `x` in `return x` gets an edge pointing to `int` in `public static int getX(x)`.)
`LAST_SCOPE_USE`	Connects a node representing a variable to the node representing the last time this variable's name was used in the text of the code (i.e. capturing information about the text, not the control flow), but only within lexical scope. This edge exists to try and give the non-GSC models as much lexical information as possible to make them as comparable with the GSC model.
`LAST_FIELD_LEX`	Connects a field access (e.g. `this.whatever` or `Foo.whatever`) node to the last use of this.whatever (or to the variable's initialization, if it's the first use). This is not lexical-scope aware (and, in fact, can't be in Java, in general).
`FIELD`	Points each node representing a field access (e.g. `this.whatever`) to the node where that field was declared.
`WORD_USE`	Points cache nodes to nodes representing variables in which the vocab word was used in the variable's name.

where the fs are MLPs, $\boldsymbol{h}_v^t$ is the hidden state of node v after t message passing iterations, σ is the sigmoid function, and $\odot$ is elementwise multiplication. The DTNN and RGCN GNNs compute the attention weightings as per (Schütt et al., 2017):

$$\hat{y}_i = f(\boldsymbol{h}_v^T),$$

where f is a single hidden layer MLP. The models were trained using a binary cross entropy loss computed across the nodes in the graph.

The performance of models using our GSC versus those using other methods is reported in Table 3.4. For context, a baseline strategy of random guessing among all variable nodes within an edge radius of 8 of the `<FILL-IN-THE-BLANK>` token achieves an accuracy of 0.22. We also compare the performance of different GNNs in Table 3.5.

The Variable Naming Task

In this task we replaced all usages of a name of a particular variable, method, class, or parameter in the code with the special token `<NAME-ME>`, and asked the model to produce the obfuscated name (in the form of the sequence of words that compose the name). An example instance from our dataset is shown in Figure 3.3. This task is the same as the VarNaming task from (Allamanis, Barr, et al., 2017).

To produce a name from the output of the GNN, our models used the readout function of (Allamanis, Brockschmidt, and Khademi, 2018). This readout function computes the mean of the hidden states of the `<NAME-ME>` nodes and passing it as the initial hidden state to a 1-layer Gated Recurrent Unit (GRU) RNN (Cho et al., 2014). This GRU is then unrolled to produce words in its predicted name, in the style of a traditional NLP decoder. We used a fixed length unrolling of 8 words, as 99.8% of names in our training set were 8 or fewer words long. The models were trained by cross entropy loss over the sequence of words in the name.

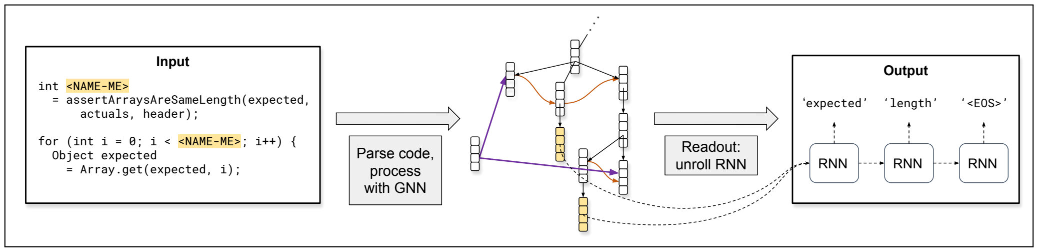
To decode each hidden state output of the GRU $\boldsymbol{h}$ into a probability distribution $P_{\text{vocab}}(\mathbf{w}|\boldsymbol{h})$ over words $\mathbf{w}$, the Closed Vocab and CharCNN models pass $\boldsymbol{h}$ through a linear layer and a softmax layer with output dimension equal to the number of words in their closed vocabularies (i.e. a traditional decoder output for NLP). In contrast, the GSC model not only has access to a fixed-size vocabulary but can also produce words by pointing to cache nodes in its Graph-Structured Cache. Specifically, it uses a decoder architecture inspired by the Pointer Sentinel Mixture Model of (Merity et al., 2017): the probability of a word $\mathbf{w}$ being the GSC decoder's output given that

Table 3.4: Accuracy on the Fill-In-The-Blank task. Our model is the AugAST-GSC. The first number in each cell is the accuracy of the model (1.0 is perfect accuracy), where a correct prediction is one in which the graph node that received the maximum attention weighting by the model contained the variable that was originally in the <FILL-IN-THE-BLANK> spot. The second, parenthetical numbers are the top-5 accuracies, i.e. whether the correct node was among those that received the 5 largest attentions weightings from the model. See Table 3.1 for explanations of the abbreviations. All models use Gated Graph Neural Networks as their GNN component.

		Closed Vocab	CharCNN	GSC
Seen repos	AST	0.57 (0.83)	0.60 (0.84)	0.89 (0.96)
	AugAST	0.80 (0.90)	0.90 (0.94)	**0.97** (0.99)
Unseen repos	AST	0.36 (0.68)	0.48 (0.80)	0.80 (0.93)
	AugAST	0.59 (0.78)	0.84 (0.92)	**0.92** (0.96)

Table 3.5: Accuracy (and top-5 accuracy) on the Fill-In-The-Blank task, depending on which type of GNN the model uses. See Table 3.1 for explanations of the abbreviations. All models use AugAST as their code representation.

		GGNN	DTNN	RGCN
Seen repos	Closed Vocab	0.80 (0.90)	0.72 (0.84)	0.80 (0.90)
	GSC	**0.97** (0.99)	0.89 (0.95)	0.95 (0.98)
Unseen repos	Closed Vocab	0.59 (0.78)	0.46 (0.68)	0.62 (0.79)
	GSC	**0.92** (0.96)	0.80 (0.89)	0.88 (0.95)



```
int <NAME-ME>
  = assertArraysAreSameLength(expected,
      actuals, header);

for (int i = 0; i < <NAME-ME>; i++) {
  Object expected
    = Array.get(expected, i);
```

Figure 3.3: Example of a model's procedure for completing the Variable Naming task. Each Variable Naming instance is created by replacing all uses of some variable (`expectedLength`, in this example) with a special <NAME-ME> token. The model then processes the code as depicted in Figure 1. To produce outputs, the model takes the mean of the <NAME-ME> nodes' hidden states (depicted here in orange), uses them as the initial hidden state of a Recurrent Neural Network, and unrolls this RNN to produce a name as a sequence of words.

the GRU's hidden state was $\boldsymbol{h}$ is

$$\begin{aligned} P(\mathbf{w}|\boldsymbol{h}) = P_{\text{graph}}(\mathbf{s}|\boldsymbol{h})P_{\text{graph}}(\mathbf{w}|\boldsymbol{h}) + \\ (1 - P_{\text{graph}}(\mathbf{s}|\boldsymbol{h}))P_{\text{vocab}}(\mathbf{w}|\boldsymbol{h}) \end{aligned}$$

where $P_{\text{graph}}(\cdot|\boldsymbol{h})$ is a conditional probability distribution over cache nodes in the GSC and the sentinel $\mathbf{s}$, and $P_{\text{vocab}}(\cdot|\boldsymbol{h})$ is a conditional probability distribution over words in a closed vocabulary. $P_{\text{graph}}(\cdot|\boldsymbol{h})$ is computed by passing the hidden states of all cache nodes and the sentinel node through a single linear layer and then computing the softmax dot-product attention of these values with $\boldsymbol{h}$. $P_{\text{vocab}}(\cdot|\boldsymbol{h})$ is computed as the softmax of a linear mapping of $\boldsymbol{h}$ to indices in a closed vocabulary, as in the Closed Vocab and CharCNN models. If there is no cache node for $\mathbf{w}$ in the Augmented AST or if $\mathbf{w}$ is not in the model's closed dictionary then $P_{\text{graph}}(\mathbf{w}|\boldsymbol{h})$ and $P_{\text{vocab}}(\mathbf{w}|\boldsymbol{h})$ are 0, respectively.

The performance of our GSC versus other methods is reported in Table 3.6. More granular performance statistics are reported in Table 3.7. We also compare the performance of different GNNs in Table 3.8.

3.6 Discussion

As can be seen in Tables 3.4 and 3.6, the addition of a GSC improved performance on all tasks. Our full model, the AugAST-GSC model, outperforms the other models tested and does comparatively well at maintaining accuracy between the Seen and Unseen test repos on the Variable Naming task.

To some degree the improved performance from adding the GSC is unsurprising: its addition to a graph-based model is adding extra features and does not remove any information or flexibility. Under a satisfactory training regime in which overfitting is avoided, a model could simply learn to ignore it if it is unhelpful, so its inclusion should never hurt performance. The degree to which it helps, though, especially on the Variable Naming task, suggests that a GSC is well worth using for some tasks, whether on source code or in NLP tasks in general. Moreover, the fact that the Pointer Sentinel approach shown in Table 3.6 performs noticeably less well than the full GSC approach suggests that the relational aspect of the GSC is key. Simply having the ability to output out-of-vocabulary words without relational information about their usage, as in the Pointer Sentinal model, is insufficient for our task.

The downside of using a GSC is the computational cost. Our GSC models ran about 30% slower than the Closed Vocab models. Since we capped the graph size at 500

Table 3.6: Accuracy on the Variable Naming task. Our model is the AugAST-GSC. The first number in each cell is the accuracy of the model (1.0 is perfect accuracy), where we consider a correct output to be exact reproduction of the full name of the obfuscated variable (i.e. all the words in the name and then an `EOS` token). The second, parenthetical numbers are the top-5 accuracies, i.e. whether the correct full name was among the 5 most probable sequences output by the model. See Table 3.1 for explanations of the abbreviations. All models use Gated Graph Neural Networks as their GNN component.

		Closed Vocab	CharCNN	Pointer Sentinel	GSC
Seen repos	AST	0.23 (0.31)	0.22 (0.28)	0.19 (0.33)	0.49 (0.67)
	AugAST	0.19 (0.26)	0.20 (0.27)	0.26 (0.40)	**0.53** (0.69)
Unseen repos	AST	0.05 (0.07)	0.06 (0.09)	0.06 (0.11)	0.38 (0.53)
	AugAST	0.04 (0.07)	0.06 (0.08)	0.08 (0.14)	**0.41** (0.57)

Table 3.7: Extra information about performance on the Variable Naming task. Entries in this table are of the form “subword accuracy, edit distance divided by real name length”. The edit distance is the mean of the character-wise Levenshtein distance between the produced name and the real name.

		Closed Vocab	CharCNN	Pointer Sentinel	GSC (ours)
Seen repos	AST	0.30, 0.94	0.28, 1.08	0.32, 1.07	0.56, 0.39
	AugAST	0.26, 0.94	0.28, 0.99	0.35, 0.77	0.60, 0.37
Unseen repos	AST	0.09, 1.23	0.10, 1.12	0.13, 1.37	0.42, 0.59
	AugAST	0.09, 1.14	0.10, 1.12	0.14, 1.07	0.48, 0.49

Table 3.8: Accuracy (and top-5 accuracy) on the Variable Naming task, depending on which type of GNN the model uses. See Table 3.1 for explanations of the abbreviations. All models use AugAST as their code representation.

		GGNN	DTNN	RGCN
Seen repos	Closed Vocab	0.19 (0.26)	0.23 (0.31)	0.27 (0.34)
	GSC	**0.53** (0.69)	0.33 (0.48)	0.46 (0.63)
Unseen repos	Closed Vocab	0.04 (0.07)	0.06 (0.08)	0.06 (0.09)
	GSC	**0.41** (0.57)	0.25 (0.40)	0.35 (0.49)

nodes, the slowdown is presumably due to the large number of edges to and from the graph cache nodes. Better support for sparse operations on GPU in deep learning frameworks would be useful for alleviating this downside.

There remain a number of design choices to explore regarding AST- and GNN-models for processing source code. Adding information about word order to the GSC might improve performance, as might constructing the vocabulary out of subwords rather than words. It also might help to treat variable types as the GSC treats words: storing them in a GSC and connecting them with edges to the variables of those types; this could be particularly useful when working with code snippets rather than fully compilable code. For the Variable Naming task, there are also many architecture choices to be explored in how to produce a sequence of words for a name: how to unroll the RNN, what to use as the initial hidden state, etc.

Given that all results above show that augmenting ASTs with data- and control-flow edges improves performance, it would be worth exploring other static analysis concepts from the Programming Language and Software Verification literatures and seeing whether they could be usefully incorporated into Augmented ASTs. Better understanding of how Graph Neural Networks learn is also crucial, since they are central to the performance of our model and many others. Additionally, the entire domain of machine learning on source code faces the practical issue that many of the best data for supervised learning on source code — things like high-quality code reviews, integration test results, code with high test coverage, etc. — are not available outside private organizations.

3.7 Acknowledgements

Many thanks to Miltos Allamanis and Hyokun Yun for their advice and useful conversations.

References

Allamanis, Miltiadis, Earl T. Barr, et al. (2017). "A Survey of Machine Learning for Big Code and Naturalness". In: *arXiv:1709.06182 [cs]*. URL: `https://arxiv.org/abs/1709.06182`.

Allamanis, Miltiadis, Marc Brockschmidt, and Mahmoud Khademi (2018). "Learning to Represent Programs with Graphs". In: *International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=BJOFETxR-`.

Alon, Uri et al. (2018). "A General Path-based Representation for Predicting Program Properties". In: *Proceedings of the 39th ACM SIGPLAN Conference on*

Programming Language Design and Implementation. PLDI 2018. Philadelphia, PA, USA: ACM, pp. 404–419. ISBN: 978-1-4503-5698-5. DOI: 10.1145/3192366.3192412. URL: http://doi.acm.org/10.1145/3192366.3192412.

Amodio, Matthew, Swarat Chaudhuri, and Thomas Reps (2017). "Neural Attribute Machines for Program Generation". In: *arXiv:1705.09231 [cs]*. URL: http://arxiv.org/abs/1705.09231 (visited on 04/02/2018).

Arabshahi, Forough, Sameer Singh, and Animashree Anandkumar (2018). "Combining Symbolic Expressions and Black-box Function Evaluations for Training Neural Programs". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Hksj2WWAW.

Bansal, Trapit, Arvind Neelakantan, and Andrew McCallum (2017). "RelNet: End-to-End Modeling of Entities & Relations". In: *arXiv:1706.07179 [cs]*. URL: http://arxiv.org/abs/1706.07179.

Battaglia, Peter W. et al. (2018). "Relational inductive biases, deep learning, and graph networks". In: abs/1806.01261. URL: https://arxiv.org/abs/1806.01261.

Bhoopchand, Avishkar et al. (2016). "Learning Python Code Suggestion with a Sparse Pointer Network". In: abs/1611.08307. URL: https://arxiv.org/abs/1611.08307.

Bojanowski, Piotr et al. (2017). "Enriching Word Vectors with Subword Information". In: *TACL* 5, pp. 135–146.

Brockschmidt, Marc et al. (2019). "Generative Code Modeling with Graphs". In: *International Conference on Learning Representations* abs/1805.08490. URL: https://arxiv.org/abs/1805.08490.

Chae, Kwonsoo et al. (2017). "Automatically Generating Features for Learning Program Analysis Heuristics for C-like Languages". In: *Proc. ACM Program. Lang.* 1.OOPSLA, 101:1–101:25. ISSN: 2475-1421. DOI: 10.1145/3133925. URL: http://doi.acm.org/10.1145/3133925.

Chen, Xinyun, Chang Liu, and Dawn Song (2018). *Tree-to-tree Neural Networks for Program Translation*. URL: https://openreview.net/forum?id=rkxY-sl0W.

Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1724–1734. DOI: 10.3115/v1/D14-1179. URL: http://www.aclweb.org/anthology/D14-1179.

Gilmer, Justin et al. (2017). "Neural Message Passing for Quantum Chemistry". In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning

Research. International Convention Centre, Sydney, Australia: PMLR, pp. 1263–1272. URL: `http://proceedings.mlr.press/v70/gilmer17a.html`.

Github (2017). *The State of the Octoverse 2017.* URL: `https://octoverse.github.com/`.

Goller, C. and A. Kuchler (1996). “Learning task-dependent distributed representations by backpropagation through structure”. In: *Neural Networks, 1996., IEEE International Conference on*. Vol. 1, 347–352 vol.1. DOI: `10.1109/ICNN.1996.548916`.

Grave, Edouard, Moustapha Cissé, and Armand Joulin (2017). “Unbounded cache model for online language modeling with open vocabulary”. In: *NIPS*.

Gulcehre, Caglar et al. (2016). “Pointing the Unknown Words”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 140–149. DOI: `10.18653/v1/P16-1014`. URL: `http://www.aclweb.org/anthology/P16-1014`.

Hellendoorn, Vincent J. and Premkumar T. Devanbu (2017). “Are deep neural networks the best choice for modeling source code?” In: *ESEC/SIGSOFT FSE*.

Johnson, Daniel D. (2017). “Learning Graphical State Transitions”. In: *International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=HJ0NvFzxl`.

Kim, Yoon et al. (2016). “Character-aware Neural Language Models”. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI’16. Phoenix, Arizona: AAAI Press, pp. 2741–2749. URL: `http://dl.acm.org/citation.cfm?id=3016100.3016285`.

Kingma, Diederik P. and Jimmy Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations*. URL: `https://arxiv.org/abs/1412.6980`.

Kobayashi, Sosuke et al. (2016). “Dynamic Entity Representation with Max-pooling Improves Machine Reading”. In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics. URL: `http://www.aclweb.org/anthology/N16-1099` (visited on 04/03/2018).

Krinke, Jens (2001). “Identifying Similar Code with Program Dependence Graphs”. In: *Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE’01)*. WCRE ’01. Washington, DC, USA: IEEE Computer Society, pp. 301–. ISBN: 0-7695-1303-4. URL: `http://dl.acm.org/citation.cfm?id=832308.837142`.

Li, Jian et al. (2017). “Code Completion with Neural Attention and Pointer Networks”. In: *arXiv:1711.09573 [cs]*. URL: `http://arxiv.org/abs/1711.09573` (visited on 01/15/2018).

Li, Yujia et al. (2016). “Gated Graph Sequence Neural Networks”. In: *International Conference on Learning Representations*. URL: `https://arxiv.org/abs/1511.05493`.

Liu, Chang et al. (2017). “Neural Code Completion”. In: URL: `https://openreview.net/forum?id=rJbPBt9lg&noteId=rJbPBt9lg`.

Lopes, Cristina V. et al. (2017). “DéjàVu: a map of code duplicates on GitHub”. In: *PACMPL* 1, 84:1–84:28.

Lu, Zhengdong et al. (2017). “Object-oriented Neural Programming (OONP) for Document Understanding”. In: *arXiv:1709.08853 [cs]*. arXiv: 1709.08853. URL: `http://arxiv.org/abs/1709.08853` (visited on 04/14/2018).

Luong, Minh-Thang and Christopher D. Manning (2016). “Achieving Open Vocabulary Neural Machine Translation with Hybrid Word-Character Models”. In: *arXiv:1604.00788 [cs]*. URL: `http://arxiv.org/abs/1604.00788` (visited on 02/13/2018).

Maddison, Chris J. and Daniel Tarlow (2014). “Structured Generative Models of Natural Source Code”. In: ICML’14, pp. II-649–II-657. URL: `http://dl.acm.org/citation.cfm?id=3044805.3044965`.

Merity, Stephen et al. (2017). “Pointer Sentinel Mixture Models”. In: *International Conference on Learning Representations*. URL: `https://openreview.net/pdf?id=Byj72udxe`.

Pham, Trang, Truyen Tran, and Svetha Venkatesh (2018). “Graph Memory Networks for Molecular Activity Prediction”. In: *arXiv:1801.02622 [cs]*. URL: `http://arxiv.org/abs/1801.02622` (visited on 04/13/2018).

Rabinovich, Maxim, Mitchell Stern, and Dan Klein (2017). “Abstract Syntax Networks for Code Generation and Semantic Parsing”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 1139–1149. DOI: `10.18653/v1/P17-1105`. URL: `http://www.aclweb.org/anthology/P17-1105`.

Raychev, Veselin, Martin Vechev, and Andreas Krause (2015). “Predicting Program Properties from "Big Code"”. In: *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL ’15. Mumbai, India: ACM, pp. 111–124. ISBN: 978-1-4503-3300-9. DOI: `10.1145/2676726.2677009`. URL: `http://doi.acm.org/10.1145/2676726.2677009`.

Schlichtkrull, Michael et al. (2017). “Modeling Relational Data with Graph Convolutional Networks”. In: *arXiv:1703.06103 [cs, stat]*. URL: `http://arxiv.org/abs/1703.06103` (visited on 04/09/2018).

Schütt, Kristof T. et al. (2017). “Quantum-chemical insights from deep tensor neural networks”. In: *Nature communications*.

Socher, Richard et al. (2013). “Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank”. In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, pp. 1631–1642. URL: `http://www.aclweb.org/anthology/D13-1170`.

TIOBE (2018). *TIOBE Index for September 2018*. URL: `https://www.tiobe.com/tiobe-index/`.

Vasic, Marko et al. (2019). “Neural Program Repair by Jointly Learning to Localize and Repair”. en. In: URL: `https://arxiv.org/abs/1904.01720v1` (visited on 05/07/2019).

Vinyals, Oriol, Meire Fortunato, and Navdeep Jaitly (2015). “Pointer Networks”. In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., pp. 2692–2700. URL: `http://papers.nips.cc/paper/5866-pointer-networks.pdf`.

White, M. et al. (2016). “Deep learning code fragments for code clone detection”. In: *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 87–98. URL: `http://www.cs.wm.edu/~mtufano/publications/C5.pdf`.

Yin, Pengcheng and Graham Neubig (2017). “A Syntactic Neural Model for General-Purpose Code Generation”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 440–450. DOI: `10.18653/v1/P17-1041`. URL: `http://www.aclweb.org/anthology/P17-1041`.

Zaremba, Wojciech, Karol Kurach, and Rob Fergus (2014). “Learning to Discover Efficient Mathematical Identities”. In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., pp. 1278–1286.

Zhang, Xiang, Junbo Zhao, and Yann LeCun (2015). “Character-level Convolutional Networks for Text Classification”. In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., pp. 649–657. URL: `http://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf`.

Chapter 4

SUPERVISED LEARNING ON RELATIONAL DATABASES WITH GRAPH NEURAL NETWORKS, PART 1

4.1 Introduction

Relational data is the most widely used type of data across all industries (Kaggle, Inc., 2017). Besides HTML/CSS/Javascript, relational databases (RDBs) are the most popular technology among developers (Stack Exchange, Inc., 2018). The market merely for hosting RDBs is over \$45 billion USD (Asay, 2016), which is to say nothing of the societal value of the data they contain.

Yet learning on data in relational databases has received relatively little attention from the deep learning community. The standard strategy for working with RDBs in machine learning is to "flatten" the relational data they contain into tabular form, since most popular supervised learning methods expect their inputs to be fixed–size vectors. This flattening process not only destroys potentially useful relational information present in the data, but the feature engineering required to flatten relational data is often the most arduous and time-consuming part of a machine learning practitioner's work.

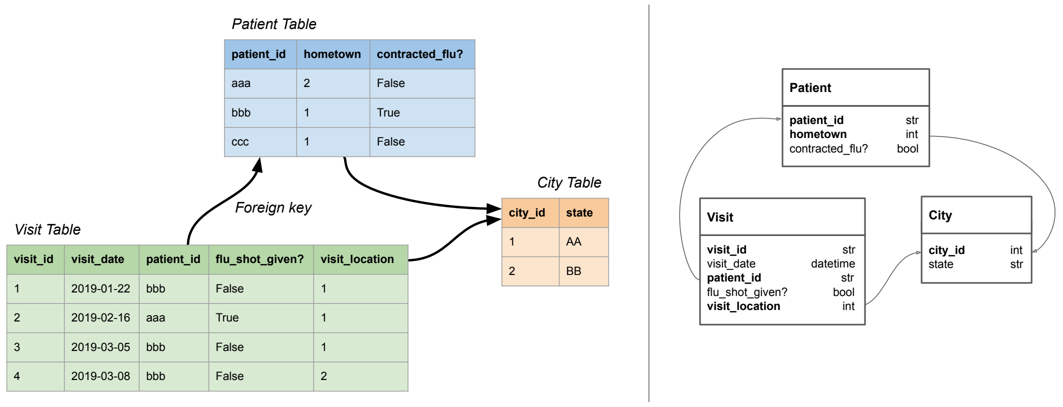
In this chapter, we introduce a method based on Graph Neural Networks (GNNs) that operates on RDB data in its relational form without the need for manual feature engineering or flattening. (For an introduction to GNNs, see the previous chapter's Section 3.3.)

4.2 Relational Databases

A relational database[1] (RDB) is a set of tables, $\{\boldsymbol{T}^1, \boldsymbol{T}^2, \ldots, \boldsymbol{T}^T\}$. An example is shown in Figure 4.1. Each table represents a different type of entity: its rows correspond to different instances of that type of entity and its columns correspond to the features possessed by each entity. Each column in a table may contain a different type of data, like integers, real numbers, text, images, date, times, geolocations, etc. Unless otherwise specified, any entry in a table can potentially be empty, i.e. contain a `null` value. We denote row i of table t as $\boldsymbol{T}^t_{i,:}$ and column j of table t as $\boldsymbol{T}^t_{:,j}$.

What makes RDBs "relational" is that the values in a column in one table can refer

[1]RDBs are sometimes informally called "SQL databases".



patient_id	hometown	contracted_flu?
aaa	2	False
bbb	1	True
ccc	1	False

visit_id	visit_date	patient_id	flu_shot_given?	visit_location
1	2019-01-22	bbb	False	1
2	2019-02-16	aaa	True	1
3	2019-03-05	bbb	False	1
4	2019-03-08	bbb	False	2

city_id	state
1	AA
2	BB

Figure 4.1: (Left) Toy example of a relational database (RDB). (Right) Schema diagram of the same RDB.

to rows of another table. For example, in Figure 4.1 the column $\boldsymbol{T}^{\text{Visit}}_{:,\text{patient\_id}}$ refers to rows in $\boldsymbol{T}^{\text{Patient}}$ based on the values in $\boldsymbol{T}^{\text{Patient}}_{:,\text{patient\_id}}$. The value in $\boldsymbol{T}^{\text{Visit}}_{i,\text{patient\_id}}$ indicates which patient came for Visit i. A column like this that refers to another table is called a *foreign key*.

The specification of an RDB's tables, their columns, and the foreign key relationships between them is called the *database schema*. It is usually depicted diagrammatically, as on the right side of Figure 4.1.

Readers familiar with object-oriented programming may find it helpful to think of each table as an object class. In this analogy, the table's columns are the class's attributes, and each of the table's rows is an instance of that class. A foreign key is an attribute that refers to an instance of a different class.

There are many software systems for creating and managing RDBs, including MySQL, PostgreSQL, and SQLite. But effectively all RDB systems adhere closely to the same technical standard (International Organization for Standardization, 2016), which defines how they are structured and what data types they can contain. Thanks to this nonproliferation of standards, the ideas we present in this chapter apply to supervised learning problems in nearly every RDB in use today.

4.3 Supervised Learning on Relational Databases

A broad class of learning problems on data in a relational database $D = \{\boldsymbol{T}^1, \boldsymbol{T}^2, \ldots, \boldsymbol{T}^K\}$ can be formulated as follows: predict the values in a target column $\boldsymbol{T}^k_{:,\text{target}}$ of $\boldsymbol{T}^k \in D$ given all other relevant information in the database.[2] In this chapter, *supervised learning on relational databases* refers to this problem formulation.

This formulation encompasses all classification and regression problems in which we wish to predict the values in a particular column in D, or predict any aggregations or combinations of values in D. This includes time series forecasting or predicting relationships between entities. Traditional supervised learning problems like image classification or tabular regression are trivial cases of this formulation where D contains one table.

There are several approaches for predicting values in $\boldsymbol{T}^k_{:,\text{target}}$, including first-order logic- and graphical model inference-based approaches (Getoor and Taskar, 2007). In this chapter we consider the empirical risk minimization (ERM) approach. The ERM approach is commonly used, though not always mentioned by name: it is the approach being implicitly used whenever a machine learning practitioner "flattens" or "extracts" or "feature engineers" data from an RDB into tabular form for use with a tabular, supervised, machine learning model.

More precisely, ERM assumes the entries of $\boldsymbol{T}^k_{:,\text{target}}$ are sampled i.i.d. from some distribution, and we are interested in finding a function f that minimizes the empirical risk

$$\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(f\left(\boldsymbol{T}^1, \boldsymbol{T}^2, \ldots, \boldsymbol{T}^k_{:,:\backslash\text{target}}, \ldots, \boldsymbol{T}^K\right), \boldsymbol{T}^k_{i,\text{target}}\right) \tag{4.1}$$

for a real-valued loss function $\mathcal{L}$ and a suitable function class $\mathcal{F}$, where $\boldsymbol{T}^k_{:,:\backslash\text{target}}$ denotes table k with the target column removed, and we assume that rows $1, \ldots, N$ of $\boldsymbol{T}^k$ contain the training samples.

To solve, or approximately solve, Equation 4.1, we must choose a hypothesis class $\mathcal{F}$ and an optimization procedure over it. We will accomplish this by framing supervised learning on RDBs in terms of learning tasks on graphs.

Connection to Learning Problems on Graphs

A relational database can be interpreted as a directed graph — specifically, a directed multigraph where each node has a type and has associated features that depend on its type. The analogy is laid out in Table 4.1.

[2] We ignore issues of target leakage for the purposes of this chapter, but in practice care is needed.

Table 4.1: Corresponding terms when interpreting a relational database as a graph.

Relational Database	**Graph**
Row	Node
Table	Node type
Foreign key column	Edge type
Non-foreign-key column	Node feature
Foreign key from $\boldsymbol{T}^A_{u,i}$ to $\boldsymbol{T}^B_{v,j}$	Edge from node u of type A to node v of type B
ith target value in table k, $\boldsymbol{T}^k_{i,\text{target}}$	Target feature on node i of type k

Note that the RDB's schema diagram (an example of which is in Figure 4.1) is not the same as this interpretation of the entire RDB as a graph. The former, which has tables as nodes, is a diagrammatic description of the properties of the latter, which has rows as nodes.

Note also that the RDB-as-graph interpretation is not bijective: directed multigraphs cannot in general be stored in RDBs by following the correspondence in Table 4.1.[3] An RDB's schema places restrictions on which types of nodes can be connected by which types of edges, and on which features are associated with each node type, that general directed multigraphs may not satisfy.

The interpretation of an RDB as a graph shows that supervised learning on RDBs reduces to a node classification problem (Atwood and Towsley, 2016). (For concision we only refer to classification problems, but our discussion applies equally to regression problems.) In addition, the interpretation suggests that GNN methods are applicable to learning tasks on RDBs.

Learning on RDBs with GNNs

The first challenge in defining a hypothesis class $\mathcal{F}$ for use in Equation 4.1 is specifying how functions in $\mathcal{F}$ will interact with the RDB. Equation 4.1 is written with $f \in \mathcal{F}$ taking the entire database D (excluding the target values) as input. But it is so written only for completeness. In reality processing the entire RDB to produce a single output is intractable and inadvisable from a modeling perspective. The algorithms in the hypothesis class $\mathcal{F}$ must retrieve and process only the information in the RDB that is relevant for making their prediction.

Stated in the language of node classification: we must choose how to select a subgraph

[3]In principle one could use many-to-many tables to store any directed multigraph in an RDB, but this would not follow the correspondence in Table 4.1.

of the full graph to use as input when predicting the label of a target node. How best to do this in general in node classification is an active topic of research (Hamilton, Ying, and Leskovec, 2017). Models that can learn a strategy for selecting a subgraph from an RDB graph are an interesting prospect for future research. In lieu of this, we present Algorithm 1, or RDBToGraph, a deterministic heuristic for selecting a subgraph. RDBToGraph simply selects every ancestor of the target node, then selects every descendant of the target node and its ancestors.

Algorithm 1 RDBToGraph: Produce a graph of related entries in an RDB D useful for classifying a target entry in D. Runs in $O(|V| + |E|)$ time.

```
Require: RDB $D$ with target values removed, target row $i$, target table $k$.
  function RDBToGraph($D$, $i$, $k$)
      Let $G = (V, E)$ be $D$ encoded as a directed multigraph as per Table 4.1.
      Let $u \in V$ be the node corresponding to target entry $\boldsymbol{T}^k_{i,\text{target}} \in D$
      $V_S \leftarrow \{u\}$
      repeat
          $A \leftarrow \{(v, w) \mid v \notin V_S \wedge w \in V_S\}$ // $A$ is for ancestors
          $V_S \leftarrow V_S \cup \{v \mid (v, w) \in A\}$
      until $V_S$ stops changing
      repeat
          $D \leftarrow \{(v, w) \mid v \in V_S \wedge w \notin V_S\}$ // $D$ is for descendants
          $V_S \leftarrow V_S \cup \{w \mid (v, w) \in D\}$
      until $V_S$ stops changing
      $E_S \leftarrow \{(v, w) \mid v \in V_S \wedge w \in V_S\}$
      return $(V_S, E_S)$
  end function
```

RDBToGraph is motivated by the ERM assumption and the semantics of RDBs. ERM assumes all target nodes were sampled i.i.d. Since the set of ancestors of a target node u refer uniquely to u through a chain of foreign keys, the ancestors of u can be thought of as having been sampled along with u. This is why RDBToGraph includes the ancestors of u in the datapoint containing u. The descendants of u and its ancestors are included because, in the semantics of RDBs, a foreign key reference is effectively a type of feature, and we want to capture all potentially relevant features of u (and its ancestors).

RDBToGraph misses potential gains that could be achieved by joining tables. For example, in the KDD Cup 2014 dataset below, the Project table, which contains the target column, has a reverse foreign key to the Essay table. In principle this means there could be multiple Essays for each Project. But it so happens that there is one Essay for each Project in the dataset. Knowing this, one could join the Project and

Essay tables together into one table, reducing the number of nodes and edges in the graph datapoint.

RDBToGraph may also be a bad heuristic when a table in an RDB has foreign keys to itself. Imagine a foreign key column in a table of employees that points to an employee's manager. Applying RDBToGraph to make a prediction about someone at the bottom of the corporate hierarchy would result in selecting the entire database. For such an RDB, we could modify RDBToGraph to avoid selecting too many nodes, for example by adding a restriction to follow each edge type only one time. Nevertheless, RDBToGraph is a good starting heuristic and performs well for all datasets we present in this chapter.

RDBToGraph followed by a GNN gives us a hypothesis class $\mathcal{F}$ suitable for optimizing Equation 4.1. In other words, for a GNN g_θ with parameters θ, our optimization problem for performing supervised learning on relational databases is

$$\min_\theta \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(g_\theta\left(\text{RDBToGraph}\left(\left\{\boldsymbol{T}^1, \boldsymbol{T}^2, \ldots, \boldsymbol{T}^k_{:,:\setminus\text{target}}, \ldots, \boldsymbol{T}^K\right\}, i, k\right)\right), \boldsymbol{T}^k_{i,\text{target}}\right), \tag{4.2}$$

which we can perform using stochastic gradient descent. Figure 4.2 shows an example of the entire procedure with the Acquire Valued Shoppers Challenge RDB introduced in Section 4.5.

4.4 Related Work

The only work we are aware of that studies supervised learning tasks on relational databases of the sort considered above comes from the feature engineering literature.

(Kanter and Veeramachaneni, 2015) present a method called Deep Feature Synthesis (DFS) that automatically aggregates features from related tables to help predict the value in a user-selected target column in an RDB. These engineered features can then be used as inputs to any standard tabular machine learning algorithm. DFS performs feature aggregation by recursively applying functions like `MAX` or `SUM` to rows connected to the target column by foreign key relationships. This is somewhat analogous to the multi-view learning approach of (Guo and Viktor, 2008).

The main advantages of DFS over a GNN-based approach like the one we are proposing are (1) it produces more interpretable features, and (2) by converting RDB data into tabular features, it lets the user apply powerful tabular learning methods like gradient boosted decision trees (Ke et al., 2017) to their problem, which have been observed to generally perform better on tabular data than existing deep models

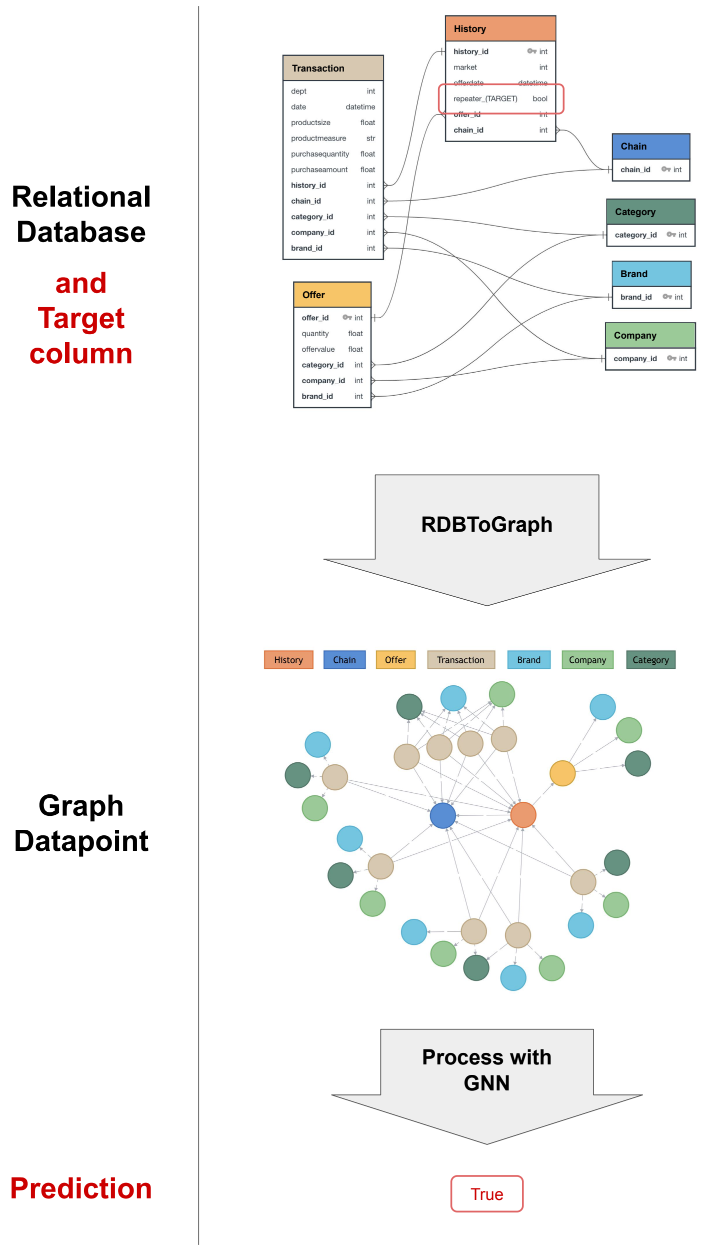


Figure 4.2: Example of our procedure for performing supervised learning on relational databases. This example uses the Acquire Valued Shoppers Challenge RDB introduced in Section 4.5. Given an RDB and a target column, we first use RDBToGraph to select the entries from the RDB that are relevant for predicting the target and assemble them into a directed multigraph. Then we use a graph neural network to make a prediction of the target value. Computing a differentiable loss function of the target and the output of g_θ lets us train g_θ by SGD.

(Arik and Pfister, 2019). The disadvantages include the combinatorial explosion of possible aggregated features that must be searched over and the lack of ability to learn these features end-to-end. These disadvantages mean users must either be judicious about how many features they wish to engineer or pay a large computational cost. In addition, a system based on DFS may miss out on potential gains from transfer learning that are possible with deep methods like GNNs.

(Lam, Thiebaut, et al., 2017) and (Lam, Minh, et al., 2018) both extend (Kanter and Veeramachaneni, 2015), the former by expanding the types of feature quantizations and embeddings used, and the latter by using Recurrent Neural Networks as the aggregation functions rather than functions like `MAX` and `SUM`.

A more loosely related area of prior work is Statistical Relational Learning (Getoor and Taskar, 2007), specifically Probabilistic Relational Models (PRMs) (Koller et al., 2007). PRMs define a joint probability distribution over the entities in an RDB schema. Learning a PRM amounts to estimating parameters for this joint model from the contents of a particular RDB. Supervised learning on RDBs is somewhat similar to learning in PRMs, except we are only interested in learning one particular conditional distribution (that of the target conditional on other entries in the RDB) rather than learning a joint model of all the entities in the RDB schema.

Work on modeling heterogeneous networks (Shi et al., 2016) is also relevant. Heterogeneous networks are examples of directed multigraphs, and thus techniques applicable to modeling them, including recent GNN-based techniques (X. Wang et al., 2019), may prove useful for supervised learning on RDBs, though we do not explore them in this chapter.

4.5 Datasets

Despite the ubiquity of supervised learning problems on data from relational databases, there are few public RDB datasets available to the machine learning research community. This is a barrier to research into this important area.

Included with this dissertation and online[4] we provide code for converting data from three public Kaggle[5] competitions into relational databases. The datasets are the Acquire Valued Shoppers Challenge, the KDD Cup 2014, and the Home Credit Default Risk Challenge. All are binary classification tasks with competition

[4]`https://github.com/mwcvitkovic/Supervised-Learning-on-Relational-Databases-with-GNNs`

[5]`http://www.kaggle.com/`

performance measured by the area under the receiver operating characteristic curve (AUROC).

Basic information about the datasets is given in Table 4.2. Detailed information about each dataset is given in Appendix 4.10.

Table 4.2: Dataset summary information

	Acquire Valued Shoppers Challenge	Home Credit Default Risk	KDD Cup 2014
Train datapoints	160,057	307,511	619,326
Tables/Node types	7	7	4
Foreign keys	10	9	4
Feature types	Categorical, Scalar, Datetime	Categorical, Scalar	Categorical, Geospatial, Scalar, Textual, Datetime

4.6 Experiments

We compare the performance of GNN-based methods of the type proposed in Section 4.3 to state-of-the-art tabular models and feature engineering approaches. Tables 4.3 and 4.4 give AUROC and accuracy results, respectively.

The tabular models we use in our baselines are logistic regression (LogReg), a 2-hidden-layer multilayer perceptron (MLP), and the `LightGBM`[6] gradient boosted tree algorithm (Ke et al., 2017) (GBDT). "Single-table" means the tabular model was trained only on data from the RDB table that contains the target column, ignoring the other tables in the RDB. "DFS" means that the Deep Feature Synthesis method of (Kanter and Veeramachaneni, 2015), as implemented in the `featuretools`[7] library, was used to engineer features from all the tables in the RDB for use by the tabular model.

The standard GNN models we use as the learned part g_θ of Equation 4.2 are the Graph Convolutional Network of (Kipf and Welling, 2016) (GCN), the Graph Isomorphism Network of (Xu et al., 2019) (GIN), and the Graph Attention Network of (Veličković et al., 2018) (GAT). Additionally, inspired by (Luzhnica, Day, and Liò, 2019), we compare against a baseline called PoolMLP, which does no message passing and

[6]`https://lightgbm.readthedocs.io`

[7]`https://docs.featuretools.com/`

computes the output merely by taking the mean of all node hidden states ($\boldsymbol{h}_v^0$ in the notation of Section 3.3) and passing this through a 1-hidden-layer MLP.

Thorough details about model and experiment implementation are in Appendix 4.9.

Table 4.3: AUROC of baseline (above the line) and our GNN-based (below the line) learning algorithms on three supervised learning problems on RDBs. Values are the mean over 5 cross-validation splits, plus or minus the standard deviation. Bold values are those within one standard deviation of the maximum in the column. Larger values are better.

	Acquire Valued Shoppers Challenge	Home Credit Default Risk	KDD Cup 2014
Single-table LogReg	0.686 ± 0.002	0.748 ± 0.004	0.774 ± 0.002
Single-table MLP	0.685 ± 0.002	0.750 ± 0.004	0.788 ± 0.002
Single-table GBDT	0.690 ± 0.002	0.754 ± 0.004	**0.801 ± 0.002**
DFS + LogReg	0.696 ± 0.001	0.771 ± 0.005	0.778 ± 0.002
DFS + MLP	0.694 ± 0.001	0.764 ± 0.005	0.781 ± 0.002
DFS + GBDT	0.689 ± 0.003	**0.777 ± 0.004**	**0.801 ± 0.003**
PoolMLP	0.678 ± 0.004	**0.773 ± 0.005**	0.736 ± 0.002
GCN	0.684 ± 0.002	0.768 ± 0.005	0.745 ± 0.002
GIN	**0.655 ± 0.079**	0.767 ± 0.003	0.754 ± 0.002
GAT	**0.717 ± 0.008**	**0.772 ± 0.005**	0.759 ± 0.004

4.7 Discussion

The results in Tables 4.3 and 4.4 bear out that GNNs are a viable and potentially valuable modeling strategy for supervised learning on RDBs. While they do not indicate that GNNs are a surefire replacement for baseline methods, they suggest that GNN-based strategies merit further research. The GNN models we used in this chapter were originally developed for modeling things like networks and molecular structures, and it is likely that GNNs tailored to RDBs could achieve improved performance. We also note that the GNN methods are faster than the DFS baseline methods provided one has access to a GPU.

Interestingly, as one can see from the RDB schema diagrams of the datasets in Appendix 4.10, the more tables and foreign key relationships an RDB has, the better GNN-based methods seem to perform on it, with the KDD Cup 2014 dataset being the least "relational" of the datasets and the one on which GNNs seem to offer no benefit.

Table 4.4: Percent accuracy of baseline (above the line) and our GNN-based (below the line) learning algorithms on three supervised learning problems on RDBs. Values are the mean over 5 cross-validation splits, plus or minus the standard deviation. Bold values are those within one standard deviation of the maximum in the column. Larger values are better.

	Acquire Valued Shoppers Challenge	Home Credit Default Risk	KDD Cup 2014
Guess Majority Class	72.9	**91.9**	**94.07**
Single-table LogReg	73.1 ± 0.2	**91.9 ± 0.1**	**94.07 ± 0.07**
Single-table MLP	73.2 ± 0.2	**91.9 ± 0.1**	**94.07 ± 0.07**
Single-table GBDT	73.3 ± 0.2	**91.9 ± 0.1**	**94.06 ± 0.07**
DFS + LogReg	73.5 ± 0.2	**91.9 ± 0.1**	**94.07 ± 0.07**
DFS + MLP	73.6 ± 0.3	**91.9 ± 0.1**	**94.07 ± 0.07**
DFS + GBDT	73.4 ± 0.3	**92.0 ± 0.1**	**94.06 ± 0.07**
PoolMLP	73.2 ± 0.3	**91.9 ± 0.1**	**94.04 ± 0.05**
GCN	73.4 ± 0.3	**91.9 ± 0.2**	**94.04 ± 0.05**
GIN	73.8 ± 0.2	**91.9 ± 0.1**	**94.07 ± 0.07**
GAT	**75.2 ± 0.3**	**91.9 ± 0.1**	**94.03 ± 0.06**

The strong performance of the Single-Table models in the Acquire Valued Shoppers Challenge and the KDD Cup 2014 suggests that a key determinant of the performance of our GNN-based models is how they process the tabular data from each node in the graph. Inspired by this, the next chapter focuses on deep models for tabular data.

4.8 Acknowledgments

We thank Da Zheng, and Minjie Wang, and Zohar Karnin for helpful discussions.

4.9 Appendix: Experiment Details

Software and Hardware

The `LightGBM`[8] library (Ke et al., 2017) was used to implement the GBDT models. All other models were implemented using the `PyTorch`[9] library (Paszke et al., 2019). GNNs were implemented using the `DGL`[10] library (M. Wang et al., 2019). All experiments were run on an Ubuntu Linux machine with 8 CPUs and 60GB memory, with all models except for the GBDTs trained using a single NVIDIA V100 Tensor Core GPU. Creating the DFS features was done on an Ubuntu Linux machine with

[8]`https://lightgbm.readthedocs.io`
[9]`https://pytorch.org/`
[10]`https://www.dgl.ai/`

48 CPUs and 185GB memory.

GNN Implementation

Adopting the nomenclature from Section 3.3, once the input graph has been assembled by `RDBToGraph`, the initialization function S for converting the features $\boldsymbol{x}_v$ of each vertex v into a real–valued vector in $\mathbb{R}^d$ proceeded as follows: (1) each of the vertex's features was encoded as real-valued vector, either by scaling to zero mean and unit variance in the case of scalar features or one-hot encoding in the case of categorical features, (2) these vectors were concatenated, (3) the concatenated vector was passed through a linear mapping with output dimension $\mathbb{R}^d$.

After obtaining $\boldsymbol{h}_v^0 = S(\boldsymbol{x}_v)$ for all vertices v, all models ran 2 rounds of message passing ($T = 2$), except for the GCN which ran 1 round. Dropout regularization of probability 0.3 was used in all models, applied at the layers specified in the paper that originally introduced the model. Most models used a hidden state size of $d = 256$, except for a few exceptions when necessary to fit things onto the GPU. Full details are in the scripts in the code included with this dissertation.

The readout function R for all GNNs was mean pooling all hidden states followed by a linear transform to obtain logits followed by a softmax layer. The cross entropy loss was used for training, and the AdamW optimizer (Loshchilov and Hutter, 2017) was used to update the model parameters. All models used early stopping based on the performance on a validation set.

Other Model Implementation

The Single-table LogReg and MLP models used the same steps (1) and (2) for initializing their input data as the GNN implementation in the previous section. We did not normalize the inputs to the GBDT model, as the `LightGBM` library handles this automatically. The MLP model contained 2 hidden layers, the first with dimension 4x the number of inputs, the second with dimension 2x the number of inputs. LogReg and MLP were trained with weight decay of 0.01, and MLP also used dropout regularization with probability 0.3. The cross entropy loss was used for training, and for LogReg and MLP the AdamW optimizer (Loshchilov and Hutter, 2017) was used to update the model parameters. All models used early stopping based on the performance on a validation set.

DFS features were generated using as many possible aggregation primitives and transformation primitives as offered in the `featuretools` library, except for the

Home Credit Default Risk dataset, which had too many features to make this possible with our hardware. For that dataset we used the same DFS settings that the `featuretools` authors did when demonstrating their system on that dataset.[11]

Hyperparameter Optimization and Evaluation

No automatic hyperparameter optimization was used for any models. We manually looked for reasonable values of the following hyperparameters by comparing, one-at-a-time, their effect on the model's performance on the validation set of the first cross-validation split:

- Number of leaves (in a GBDT tree)
- Minimum number of datapoints contained in a leaf (in a GBDT tree)
- Weight decay
- Dropout probability
- Whether to one-hot encode embeddings
- Whether to oversample the minority class during SGD
- Readout function
- Whether to apply Batch Normalization, Layer Normalization, or no normalization
- Number of layers and message passing rounds

Additionally, to find a learning rate for the models trained with SGD, we swept through one epoch of training using a range of learning rates to find a reasonable value, in the style of the FastAI (Howard et al., 2018) learning rate finder.

Full hyperparameter specifications for every model and experiment can be found in the experiment scripts in the code released with this dissertation.

Every model was trained and tested on the same five cross-validation splits. 80% of each split was used for training, and 15% of that 80% was used as a validation set.

Table 4.5: Acquire Valued Shoppers Challenge dataset summary information. (www.kaggle.com/c/acquire-valued-shoppers-challenge)

Train datapoints	160057
Test datapoints	151484
Total datapoints	311541
Dataset size (uncompressed, compressed)	47 GB, 15 GB
Node types	7
Edge types (not including self edges)	10
Output classes	2
Class balance in training set	116619 negative, 43438 positive
Types of features in dataset	Categorical, Scalar, Datetime

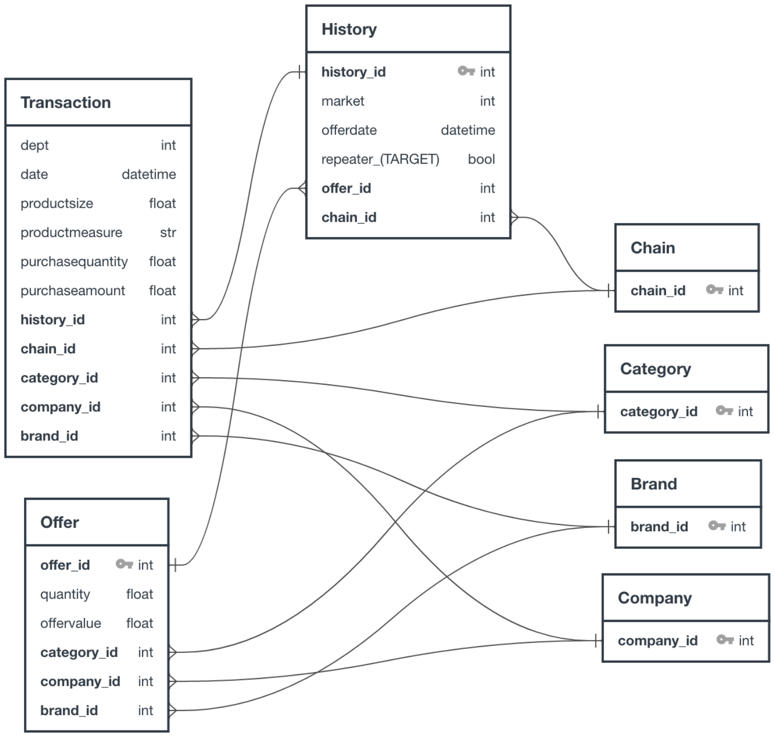


Figure 4.3: Acquire Valued Shoppers Challenge database schema.

4.10 Appendix: Dataset Information

References

Arik, Sercan O and Tomas Pfister (2019). "TabNet: Attentive Interpretable Tabular Learning". In: *arXiv preprint arXiv:1908.07442*. URL: https://arxiv.org/abs/1908.07442.

[11] https://www.kaggle.com/willkoehrsen/home-credit-default-risk-feature-tools

Table 4.6: Home Credit Default Risk dataset summary information. (www.kaggle.com/c/home-credit-default-risk)

Train datapoints	307511
Test datapoints	48744
Total datapoints	356255
Dataset size (uncompressed, compressed)	6.6 GB, 1.6 GB
Node types	7
Edge types (not including self edges)	9
Output classes	2
Class balance in training set	282686 negative, 24825 positive
Types of features in dataset	Categorical, Scalar, Geospatial (indirectly), Datetime (indirectly)

Table 4.7: KDD Cup 2014 dataset summary information. (www.kaggle.com/c/kdd-cup-2014-predicting-excitement-at-donors-choose)

Train datapoints	619326
Test datapoints	44772
Total datapoints	664098
Dataset size (uncompressed, compressed)	3.9 GB, 0.9 GB
Node types	4
Edge types (not including self edges)	4
Output classes	2
Class balance in training set	582616 negative, 36710 positive
Types of features in dataset	Categorical, Geospatial, Scalar, Textual, Datetime

Asay, Matt (2016). *NoSQL keeps rising, but relational databases still dominate big data*. en. URL: https://www.techrepublic.com/article/nosql-keeps-rising-but-relational-databases-still-dominate-big-data/ (visited on 03/22/2019).

Atwood, James and Don Towsley (2016). "Diffusion-convolutional neural networks". In: *Advances in Neural Information Processing Systems*, pp. 1993–2001.

Getoor, Lise and Ben Taskar (2007). *Introduction to Statistical Relational Learning*. en. MIT Press. ISBN: 978-0-262-07288-5.

Guo, Hongyu and Herna L Viktor (2008). "Multirelational classification: a multiple view approach". In: *Knowledge and Information Systems* 17.3, pp. 287–312.

Hamilton, Will, Zhitao Ying, and Jure Leskovec (2017). "Inductive representation learning on large graphs". In: *Advances in Neural Information Processing Systems*, pp. 1024–1034. URL: https://arxiv.org/abs/1706.02216.

Howard, Jeremy et al. (2018). *FastAI*. `https://github.com/fastai/fastai`.

International Organization for Standardization (2016). *ISO/IEC 9075-1:2016: Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework)*. en. URL: `http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/06/35/63555.html` (visited on 03/21/2019).

Kaggle, Inc. (2017). *The State of ML and Data Science 2017*. URL: `https://www.kaggle.com/surveys/2017` (visited on 03/22/2019).

Kanter, James Max and Kalyan Veeramachaneni (2015). "Deep feature synthesis: Towards automating data science endeavors". In: *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 1–10. DOI: `10.1109/DSAA.2015.7344858`. URL: `https://www.jmaxkanter.com/static/papers/DSAA_DSM_2015.pdf`.

Ke, Guolin et al. (2017). "LightGBM: A highly efficient gradient boosting decision tree". In: *Advances in Neural Information Processing Systems*, pp. 3146–3154. URL: `https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf`.

Kipf, Thomas N and Max Welling (2016). "Semi-supervised classification with graph convolutional networks". In: *International Conference on Learning Representations*. URL: `https://arxiv.org/abs/1609.02907`.

Koller, Daphne et al. (2007). "Introduction to statistical relational learning". In: MIT press. Chap. 2, pp. 129–174.

Lam, Hoang Thanh, Tran Ngoc Minh, et al. (2018). "Neural Feature Learning From Relational Database". In: *arXiv:1801.05372 [cs]*. arXiv: 1801.05372. URL: `http://arxiv.org/abs/1801.05372` (visited on 02/22/2019).

Lam, Hoang Thanh, Johann-Michael Thiebaut, et al. (2017). "One button machine for automating feature engineering in relational databases". In: *arXiv:1706.00327 [cs]*. arXiv: 1706.00327. URL: `http://arxiv.org/abs/1706.00327` (visited on 02/23/2019).

Loshchilov, Ilya and Frank Hutter (2017). "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations*. URL: `https://arxiv.org/abs/1711.05101`.

Luzhnica, Enxhell, Ben Day, and Pietro Liò (2019). "On Graph Classification Networks, Datasets and Baselines". In: *ArXiv* abs/1905.04682. URL: `https://arxiv.org/abs/1905.04682`.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Shi, Chuan et al. (2016). “A survey of heterogeneous information network analysis”. In: *IEEE Transactions on Knowledge and Data Engineering* 29.1, pp. 17–37. URL: `https://arxiv.org/pdf/1511.04854.pdf`.

Stack Exchange, Inc. (2018). *Stack Overflow Developer Survey 2018*. URL: `https://insights.stackoverflow.com/survey/2018/` (visited on 03/22/2019).

Veličković, Petar et al. (2018). “Graph Attention Networks”. In: *International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=rJXMpikCZ`.

Wang, Minjie et al. (2019). “Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. URL: `https://arxiv.org/abs/1909.01315`.

Wang, Xiao et al. (2019). “Heterogeneous Graph Attention Network”. In: *The World Wide Web Conference*. ACM, pp. 2022–2032. URL: `https://arxiv.org/abs/1903.07293`.

Xu, Keyulu et al. (2019). “How Powerful are Graph Neural Networks?” In: *International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=ryGs6iA5Km`.

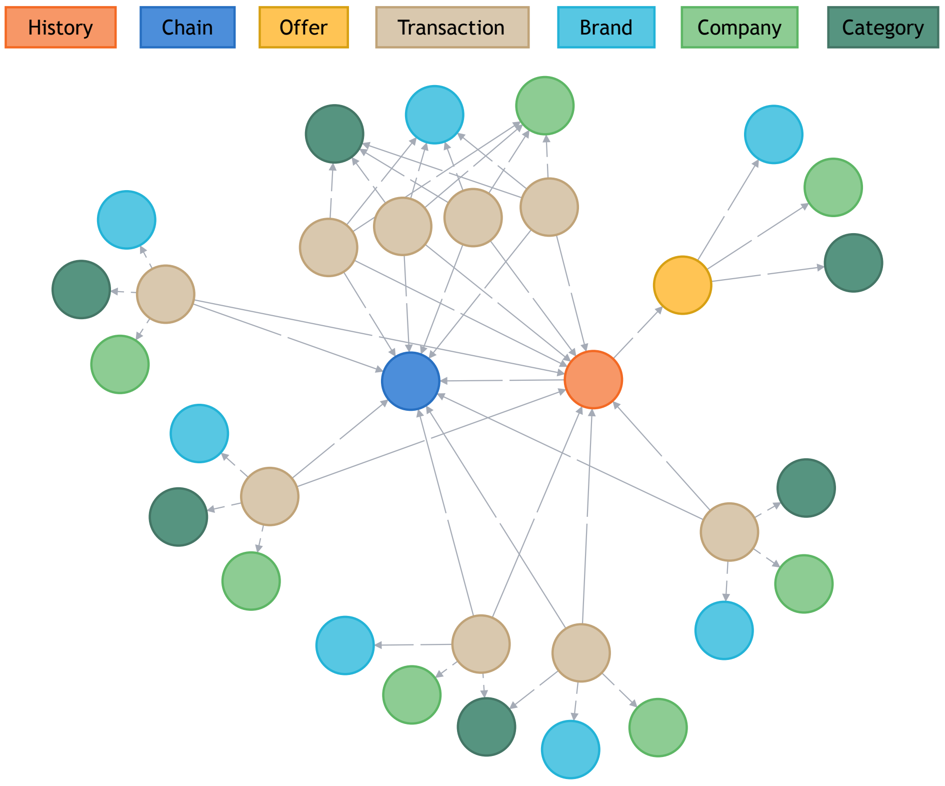


Figure 4.4: Acquire Valued Shoppers Challenge example datapoint.

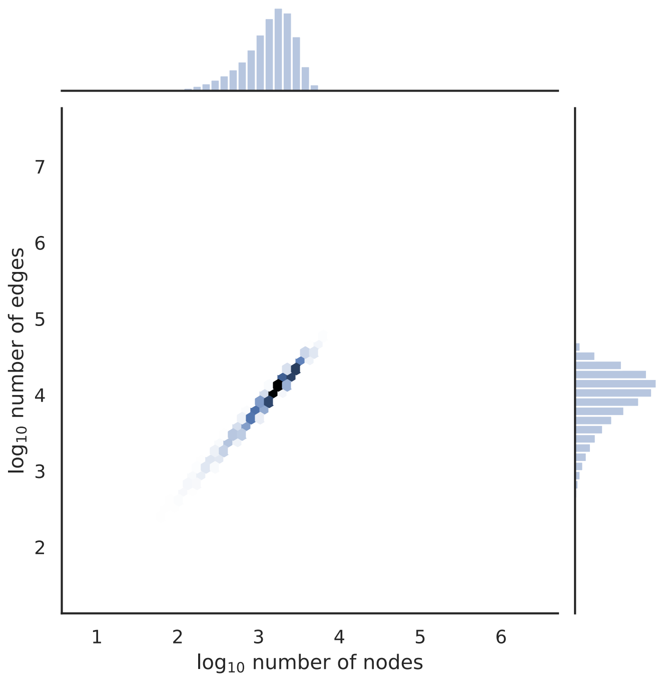


Figure 4.5: Acquire Valued Shoppers Challenge dataset graph size histogram. The horizontal axis ranges from the minimum number of nodes in any datapoint to the maximum; likewise with number of edges for the vertical axis. Thus the distribution of graph sizes in the dataset varies over six orders of magnitude, even if the histogram bins are too small to see in the plot.

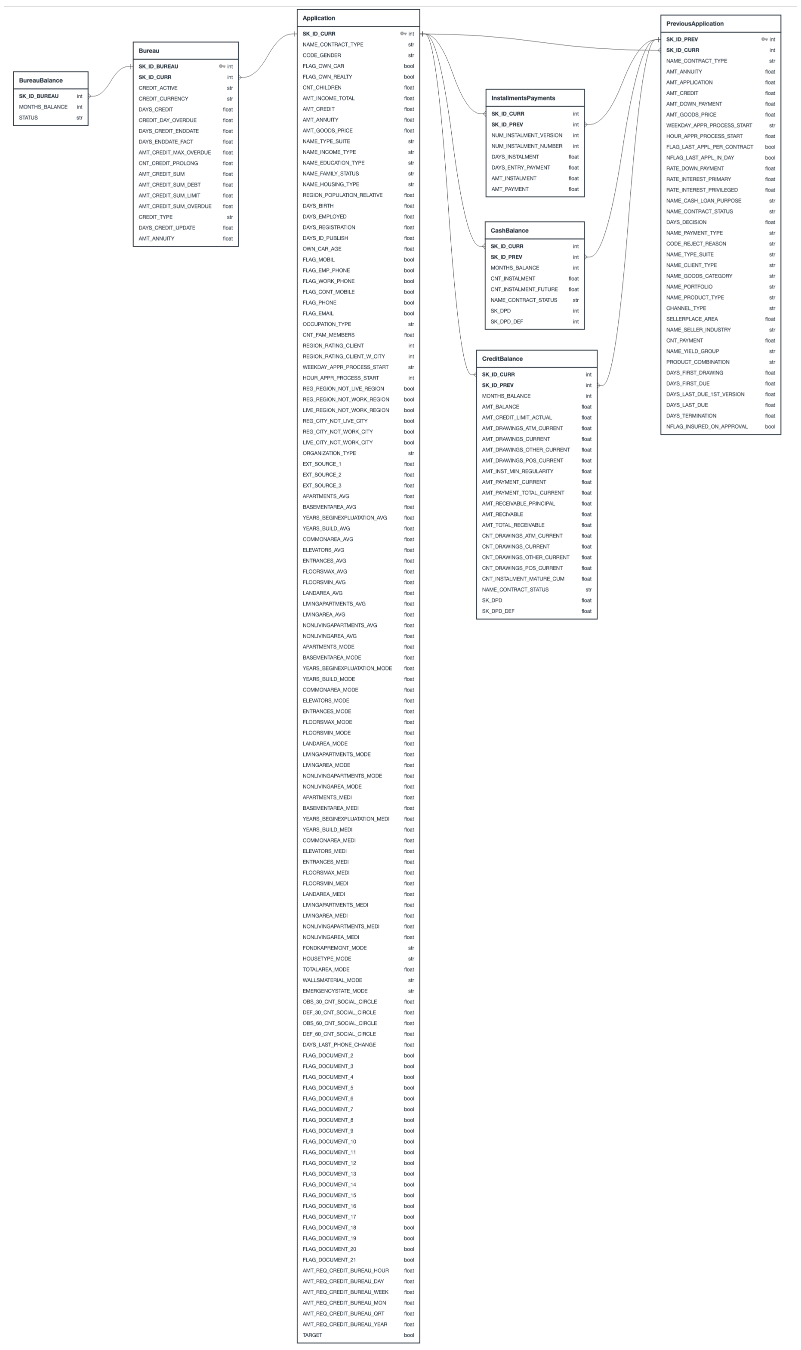


Figure 4.6: Home Credit Default Risk database schema.

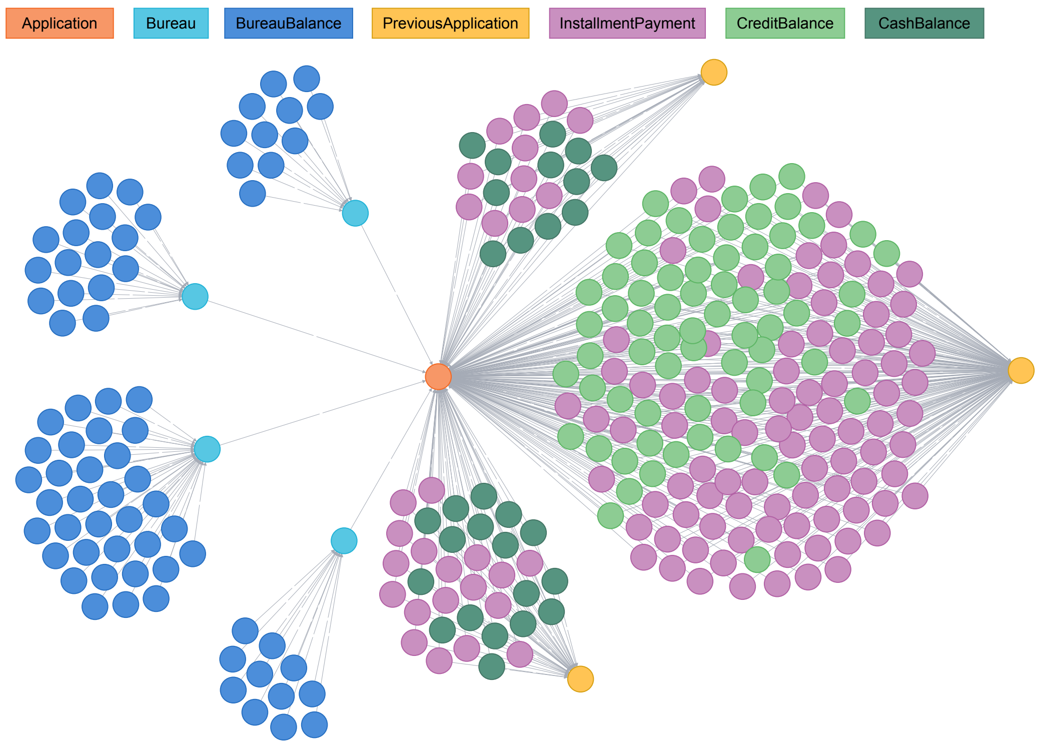


Figure 4.7: Home Credit Default Risk example datapoint.

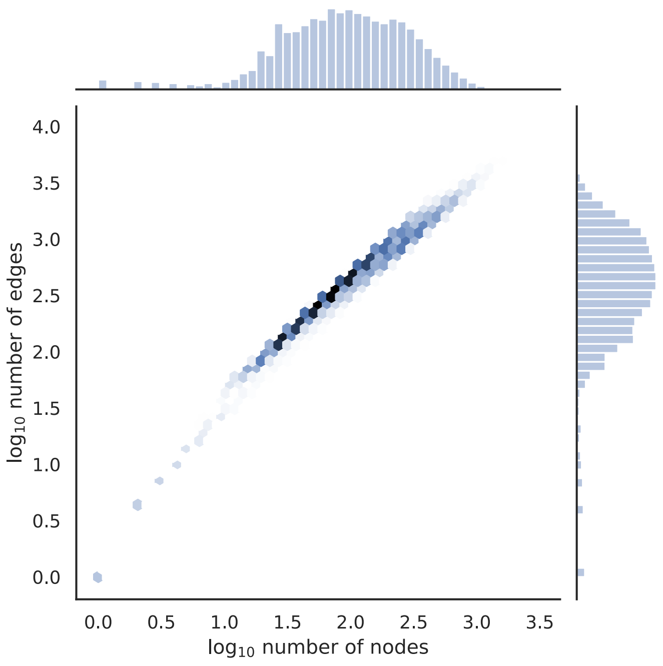


Figure 4.8: Home Credit Default Risk dataset graph size histogram. The horizontal axis ranges from the minimum number of nodes in any datapoint to the maximum; likewise with number of edges for the vertical axis. Thus the distribution of graph sizes in the dataset varies hugely, even if the histogram bins are too small to see in the plot.

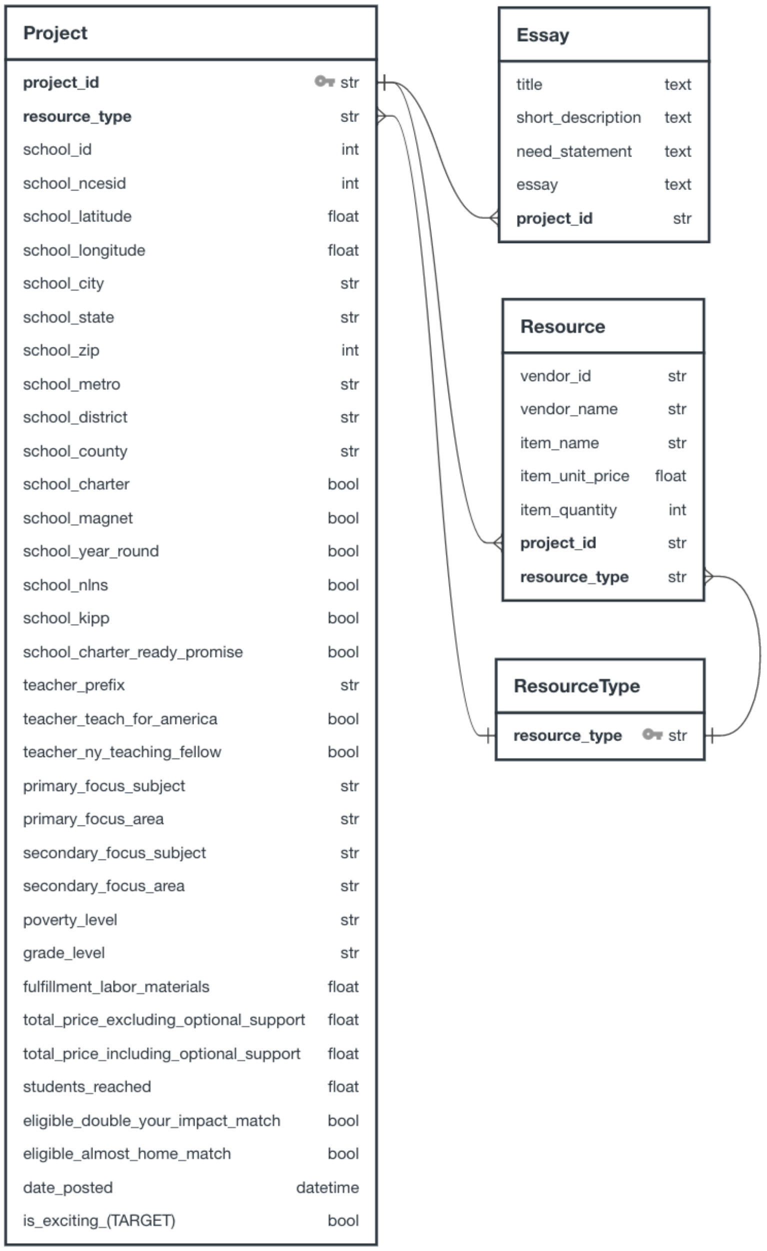


Figure 4.9: KDD Cup 2014 database schema.

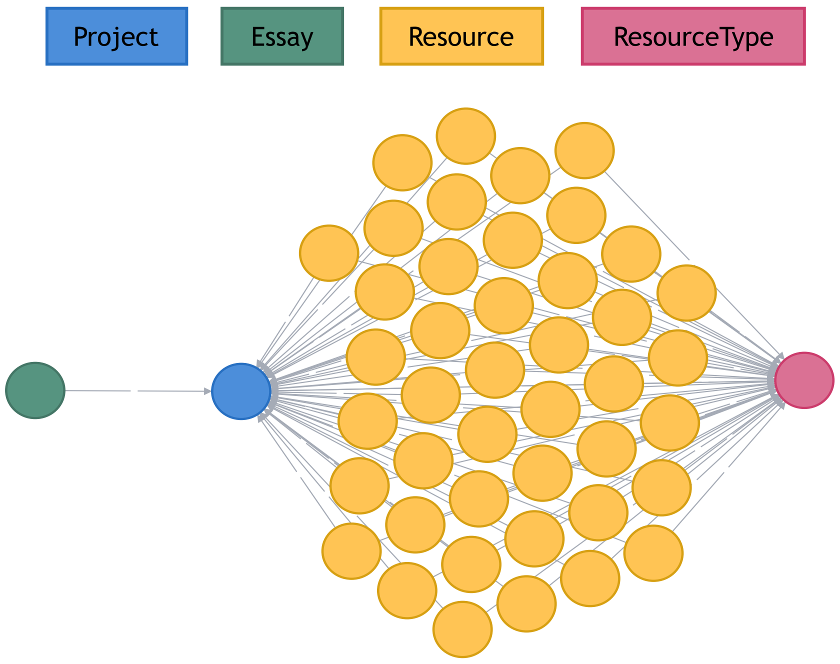


Figure 4.10: KDD Cup 2014 example datapoint.

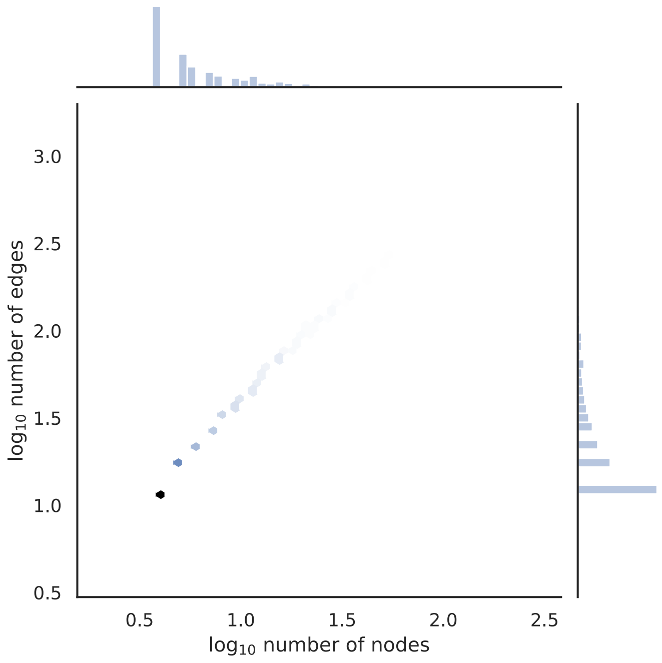


Figure 4.11: KDD Cup 2014 dataset graph size histogram. The horizontal axis ranges from the minimum number of nodes in any datapoint to the maximum; likewise with number of edges for the vertical axis. Thus the distribution of graph sizes in the dataset varies hugely, even if the histogram bins are too small to see in the plot.

Chapter 5

STRENGTHS AND WEAKNESSES OF DEEP LEARNING FOR TABULAR DATA

5.1 Introduction

Tabular data refers to data about which we have no prior knowledge regarding the structure or relationships between their features.[1] Typically stored in a table, hence their name, with datapoints as rows and features as columns, tabular data are widely used in practical machine learning and data science, as evidenced by their abundance on sites like Kaggle.[2]

The established wisdom among data scientists is that decision tree-based models, especially gradient boosted decision trees (GBDTs), are the best choice for supervised learning on tabular data (Goldbloom, 2017). While it is surprising that a category as broad as tabular data would be dominated by a single type of model, reasonable explanations have been given, including that GBDTs are built to find low-order feature interactions and avoid overfitting (Arik and Pfister, 2019), which would make them well suited to data defined by the lack of known relationships between their features.

Despite the dominance of GBDTs for tabular data, much effort has gone into developing deep models for tabular data. Indeed, most papers presenting new deep models for tabular data give evidence that they outperform GBDTs, or do not bother to compare to them at all; see the Related Work below. Yet it is notable that none has gone on to supplant GBDTs as the method of choice for tabular data.

This gap between published results and practice raises the question of whether machine learning researchers are assessing the performance of deep methods for tabular a useful way. In a benchmark-driven field like machine learning, it is notable that there is no standard benchmark for supervised learning on tabular data.

To address this issue, in Section 5.3 below we introduce a new benchmark consisting of 20 binary classification tasks on tabular data. We use this benchmark to present the first, as far as we are aware, large-scale, evenly-matched comparison of deep

[1]Confusingly, given this definition, tabular data are sometimes known as “structured data” in the literature.

[2]`https://www.kaggle.com/`

models for tabular data. Giving the same hyperparameter tuning budget to each of six types of deep models for tabular data, along with GBDTs and logistic regression, we find that while deep models can often match the performance of GBDTs, they almost never outperform it, and simple MLPs often perform as well as models designed for tabular data.

Deep models do have some comparative advantages over tree-based models, however. In the context of this dissertation, deep models for tabular data are of interest as a way to improve the performance of the GNN-based methods presented in the previous chapter. The initial features from each table in an RDB can be passed through a deep model before passing through the GNN. So improvements to deep models tabular data might improve performance on supervised learning on RDBs. (Q.v. Chapter 7.)

Of more general interest, though, is that deep models allow us to leverage unlabeled data and data from other sources to improve performance on our dataset of interest, a feat which has proved difficult for tree-based models (Tanha, Someren, and Afsarmanesh, 2017). Variants of this idea include unsupervised pretraining (Caron et al., 2019), semi-supervised learning (Stretcu et al., 2019), transductive learning (Elezi et al., 2018), and transfer learning (Devlin et al., 2019).

As a demonstration of the utility that deep learning methods can have for tabular data, in Section 5.4 below we introduce a variant of semi-supervised learning called *subset-restricted finetuning* and show that training techniques adapted from natural language processing lead deep models to generally outperform GBDTs in this regime.

5.2 Related Work

Standard multilayer perceptrons (MLPs) have been applied to tabular data for many years, sometimes to good effect (De Brébisson et al., 2015). More modern deep learning techniques like batch normalization and learnable embeddings for categorical features can significantly improve the performance of MLPs for tabular data (Howard et al., 2018).

The major problem when using MLPs for tabular data is overfitting. A number of methods for combating overfitting in MLPs have been tried at least partly on tabular data, notably sparsification and pruning methods (Hassibi et al., 1993; Mocanu et al., 2018) and information bottleneck-based methods (Tishby, Pereira, and Bialek, 2000; Strouse and Schwab, 2015).

In terms of deep models designed specifically for tabular data, many "deep versions" of non-deep-learning-based machine learning algorithms have been proposed. These include deep versions of factorization machines (Guo et al., 2018; Xiao et al., 2017), extensions of these based on transformers and multi-head attention (Song et al., 2019; Li et al., 2020; Sun et al., 2019), and deep versions of decision-tree-based algorithms (Ke, Zhang, et al., 2019; Yang, Morillo, and Hospedales, 2018).

Other models have been designed around the purported properties of tabular data such as low-order and sparse feature interactions. These include Deep & Cross Networks (Wang et al., 2017), Wide & Deep Networks (Cheng et al., 2016), and TabNets (Arik and Pfister, 2019).

5.3 Benchmarking Deep Models for Tabular Data

We assembled 20 publicly available tabular, binary classification datasets from the UCI repository (Dua and Graff, 2017), the AutoML Challenge (Guyon et al., 2019), and Kaggle.com. As shown in Table 5.1, the number of datapoints and number of features in the datasets vary over 3 orders of magnitude, and include a mix of scalar and categorical features along with some missing values. We annotated each column in each dataset as to whether it contains a scalar or categorical variable, though it is up to the model how to use this information in preprocessing. Full details on each dataset is in Appendix Tables 5.2 and 5.3. In order to encourage adoption of the benchmark, we provide code with this dissertation to automatically obtain all datasets and convert them into a common format.

For the benchmark, each dataset is divided into 5 cross-validation splits, each with 80% of the datapoints available for training and 20% for testing. 15% of this 80% is then split off as a validation set. All these splits are obtained with a fixed random seed so that every model tested receives exactly the same training and testing conditions.

The diversity of datasets in the benchmark makes it critical to perform hyperparameter optimization (HPO) for each model. Equally critical is giving each model the same HPO budget to make the comparison fair. In our experiments, each model was given 10 HPO rounds on each of the 5 cross-validation splits for each dataset, with the best-performing model among these 10 HPO rounds then evaluated on the test set. All models used early stopping on the validation set to identify the best set of parameters in each training run.

The models tested are logistic regression, GBDT as implemented in the `LightGBM` library (Ke, Meng, et al., 2017), multilayer perceptron (MLP), a transformer-style model (Tfmr) (Vaswani et al., 2017) modified for use on tabular data in the style of (Song et al., 2019; Li et al., 2020; Sun et al., 2019), a novel, heterogeneous transformer-style model using different parameters for each feature in the tabular dataset (HetTfmr), a sparsifying/pruning MLP (Prune) in the style of (Morcos et al., 2019), the TabNet model of (Arik and Pfister, 2019), and the Variational Information Bottleneck model (VIB) of (Alemi, Fischer, Dillon, and Murphy, 2017). We reimplemented all algorithms for consistency of preprocessing. In cases where there exist published results for a model we test on a dataset used in our benchmark, our results are close to the published results. More details on all these models and the hyperparameters tested are in Appendix 5.7.

Table 5.1: Benchmark datasets. All datasets are binary classification problems. Positive:Total is the fraction of datapoints that are the positive class.

Dataset Name	N Datapts	N Feats	Positive:Total
1995_income	32561	14	0.241
adult	34190	25	0.854
albert	425240	79	0.5
bank_marketing	45211	16	0.117
blastchar	7043	20	0.265
dota2games	92650	117	0.527
fabert	8237	801	0.113
hcdr_main	307511	120	0.081
htru2	17898	8	0.092
insurance_co	5822	85	0.06
jannis	83733	55	0.02
jasmine	2984	145	0.5
online_shoppers	12330	17	0.155
philippine	5832	309	0.5
qsar_bio	1055	41	0.337
seismicbumps	2583	18	0.066
shrutime	10000	11	0.204
spambase	4601	57	0.394
sylvine	5124	20	0.5
volkert	58310	181	0.127

Results

Results in terms of accuracy and AUROC are presented in Figures 5.1 and 5.2, respectively. Since each dataset in the benchmark has a different class balance and (unknown) maximum possible performance, Figures 5.1 and 5.2 show performance relative to logistic regression, and in terms of $\log_2$ improvement to the metric. In other words, if a model achieves 0.95 accuracy on a dataset, and logistic regression achieves 0.9 accuracy on that dataset, Figure 5.1 will show this as equivalent to if a model achieves 0.75 accuracy on a dataset for which logistic regression achieves 0.5 accuracy. Raw results are given in Appendix Tables 5.4 and 5.5. The comparisons between each model and logistic regression are done within each cross-validation split and then aggregated, since for some datasets the variance in performance between cross-validation splits is larger than the variance of performance between algorithms.

The results in Figures 5.1 and 5.2 show deep models performing as well as GBDTs on about half of datasets, but significantly outperforming them on only two. This confirms the common wisdom among practitioners that GBDTs are the model of choice for tabular data, but also shows that deep models are not inherently ill-suited for tabular data. (In terms of computational complexity, however, GBDTs are clearly preferable.) Among deep models, simple MLPs perform comparably to complex models. Notably, on four out of twenty datasets no model significantly outperforms logistic regression. This is an important reminder of the diversity of tabular problems and why simple baseline algorithms must always be included in comparisons.

5.4 Subset-Restricted Finetuning with Deep Models

Having tempered our expectations for deep models on standard supervised learning problems on tabular data given the benchmark results above, we turn to a domain of problems on tabular data for which deep models are better suited.

In *semi-supervised learning* we are given some labeled data and some unlabeled data, and our goal is to train a model that performs well on an unseen test set. Most semi-supervised learning methods require access to the full partially-labeled dataset during training (Chapelle, Scholkopf, and Zien, 2009; Oliver et al., 2018; Stretcu et al., 2019). But in many practical scenarios such as edge computing (Murshed et al., 2019) or mass model personalization (Schneider and Vlachos, 2019), labels are obtained after the model has already had access to the unlabeled data, and it is impractical or impossible to train the model from scratch using the now-partially-labeled dataset.

To distinguish this scenario from semi-supervised learning and general unsupervised pretraining, we introduce *subset-restricted finetuning* (SRF). In SRF, a model can perform unrestricted, unsupervised pretraining on a (large) set of unlabeled data, but at finetuning time can only access the (small) labeled subset of the full dataset.

Most methods for semi-supervised learning are not applicable to SRF, while any method for unsupervised pretraining is potentially applicable to SRF. In what follows,

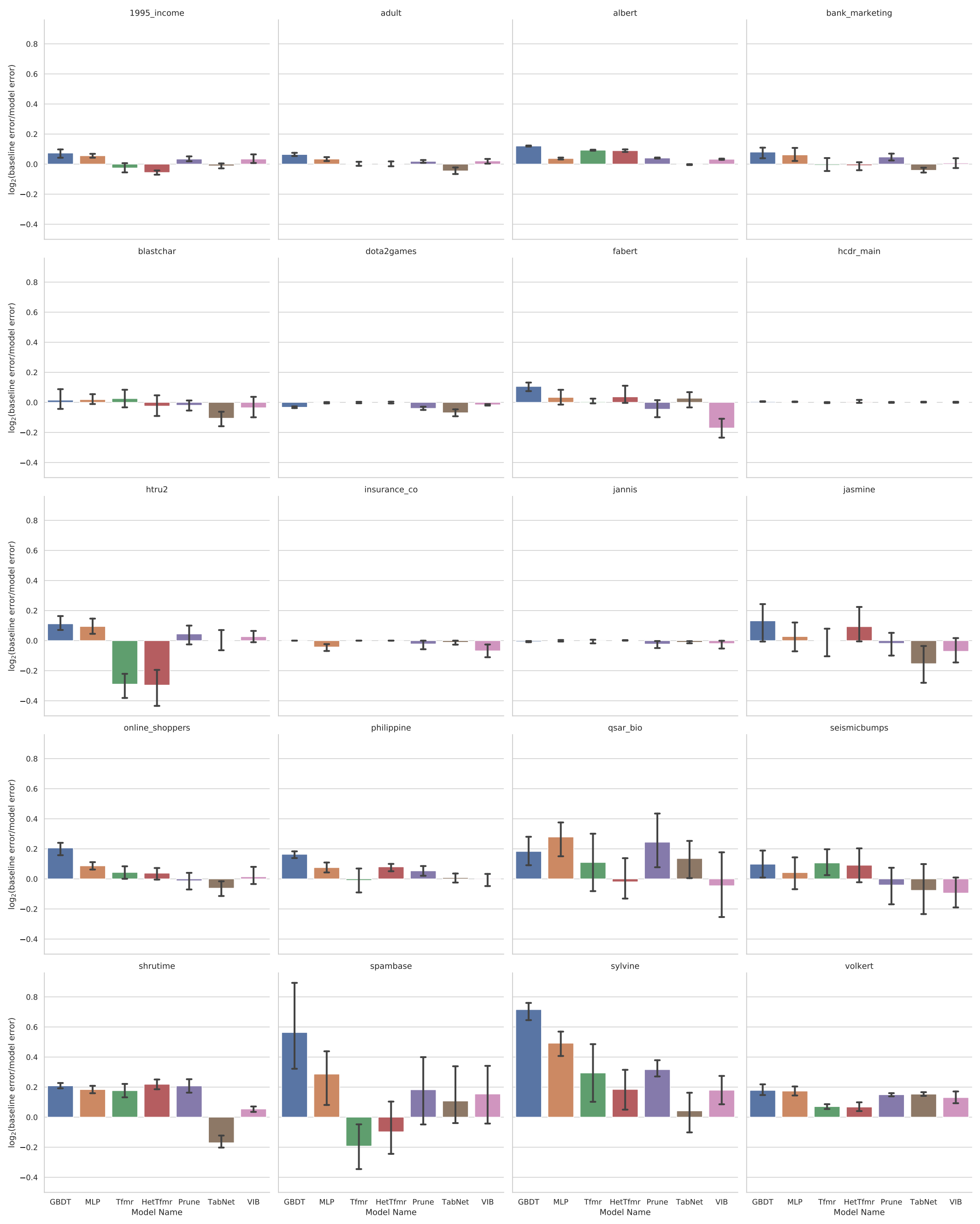


Figure 5.1: $\log_2$ improvement in prediction error relative to logistic regression. For example, a value of 1.0 means the algorithm's test set error was half that of logistic regression. Higher is better.

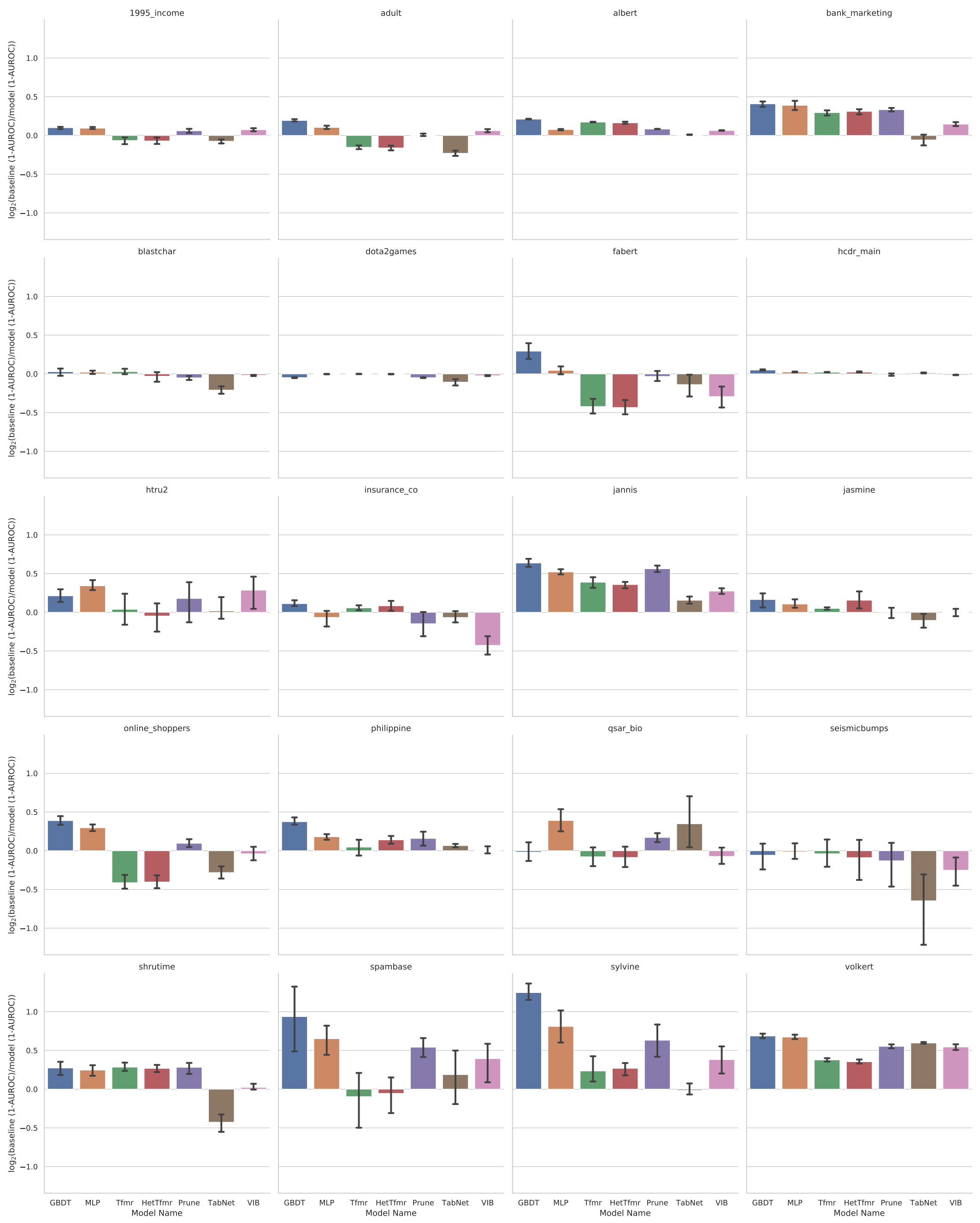


Figure 5.2: $\log_2$ improvement in AUROC relative to logistic regression. For example, a value of 1.0 means the algorithm's test set error was half that of logistic regression. Higher is better.

we compare the performance of several methods for unsupervised pretraining on SRF.

We are aware of only one unsupervised pretraining method explicitly designed for deep models on tabular data: the swap noise denoising autoencoder (DAE) method for pretraining MLPs of (Jahrer, 2018). However the recent application of transformer-style models to tabular data (Song et al., 2019; Li et al., 2020; Sun et al., 2019) suggests that porting unsupervised pretraining ideas from NLP to the SRF regime may be fruitful. We compare the masked language modeling (TfrMLM) method (Devlin et al., 2019) and the related replaced token detection (TfrRTD) method (Clark et al., 2020). As MLM and RTD are methods originally developed for transfer learning, we also attempt to see whether pretraining a transformer model on all 20 benchmark datasets with MLM or RTD (XfrMLM and XfrRTD, respectively) offers an advantage over the SRF task of pretraining only on the dataset of interest.

More details on all these models and the hyperparameters tested are in Appendix 5.7.

Results

Figure 5.3 presents the test set AUROC for models trained in the SRF regime on the 8 datasets in the benchmark introduced above that contain more than 30,000 samples. The finetuning subset for all datasets contained 500 labeled datapoints.

DAE pretrained MLPs never outperformed untrained MLPs — in other words, DAE pretraining did not allow the MLP model to effectively exploit information from the unlabeled pretraining dataset. However, MLM and RTD pretraining for transformer models was highly effective. And the larger the unsupervised training dataset, the larger the effect (and the largest savings by using SRF instead of full semi-supervised learning).

Transfer learning from other datasets, the Xfr models, does not seem to offer any advantages, contrary to what is observed in natural language processing (Devlin et al., 2019). This is unsurprising given the diverse nature of the datasets in the benchmark, though it does not rule out transfer learning for tabular data that might exploit properties of the column contents that are not present in the transformer architecture we used.

5.5 Discussion

Despite the suggestions in much of the literature on the topic, the large-scale benchmark experiments in this chapter suggest that deep models do not offer a

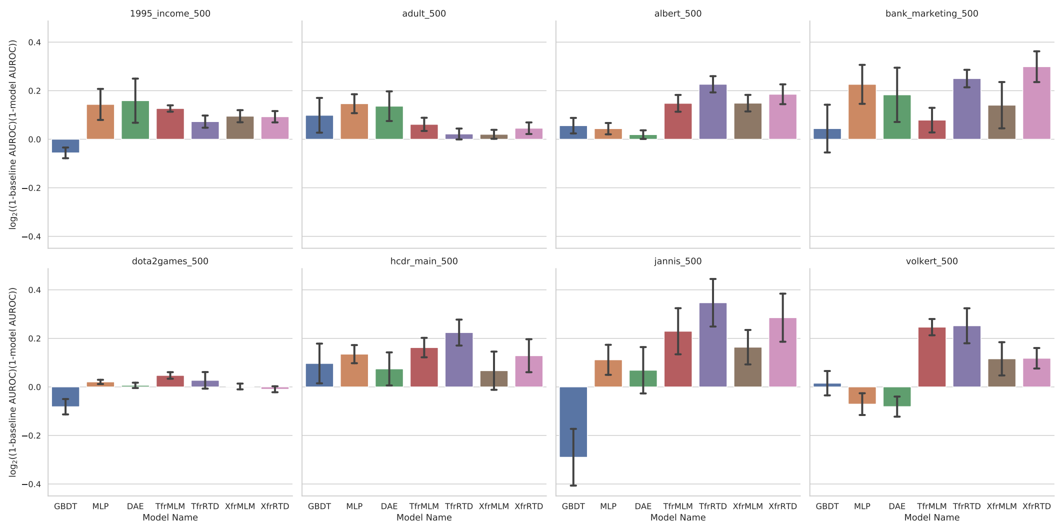


Figure 5.3: $\log_2$ improvement in AUROC relative to a baseline model (a non-pretrained transformer) on 8 datasets in the SRF regime. Each dataset contained between 30,000 and 450,000 unlabeled datapoints (see Appendix for precise numbers), and each finetuning subset contained 500 labeled datapoints. Error bars show the standard deviation across 5 cross-validation splits. Higher is better.

meaningful performance benefit over GBDT models for standard supervised learning on tabular data. This is particularly disappointing for strongly theoretically motivated methods like those based on the information bottleneck. We explore this disconnect between theory and practice further in the next chapter.

However there are regimes, like subset-restricted finetuning, that let deep learning play to its strengths. As the variance in performance between the MLM and RTD pretraining methods in Figure 5.3 suggests, there remains much to explore regarding unsupervised methods for deep models on tabular data.

5.6 Acknowledgments

We thank Xin Huang, Ashish Khetan, and Jonas Mueller for helpful discussions.

5.7 Appendix: Experiment and Model Details

The `LightGBM`[3] library (Ke, Meng, et al., 2017) was used to implement the GBDT models. All other models were implemented using the `PyTorch`[4] library (Paszke et al., 2019). All experiments were run on an Ubuntu Linux machine with 8 CPUs

[3]`https://lightgbm.readthedocs.io`
[4]`https://pytorch.org/`

and 60GB memory, with all models except for the GBDTs trained using a single NVIDIA V100 Tensor Core GPU.

All scalar features were normalized to zero median and unit interquartile range[5] and had a binary flag appended for missing values.

Exact values for all hyperparameter searches can be found in the scripts in the code included with this dissertation.

Benchmark Experiments

The cross entropy loss was used for training, and for all deep models the AdamW optimizer (Loshchilov and Hutter, 2017) was used to update the model parameters. All models used early stopping based on the performance on the validation set.

The hyperparameters tuned for the GBDT model were the number of leaves in the trees, the minimum number of datapoints required to split a leaf in the trees, the boosting learning rate, and the number of trees used for boosting.

The MLP had varying numbers of hidden units and layers, but all used SELU activations (Klambauer et al., 2017) followed by batch normalization followed by dropout in each layer. The hyperparameters tuned were the weight decay factor, the learning rate, the dropout probability, whether to one-hot encode categorical variables or train learnable embeddings for each, and the number of layers and hidden unit sizes (relative to the input size).

The transformer models we used (Tfmr and HetTfmr) were simplified versions of the various flavors of multi-head attention-based models described in the Related Work section above. The model was implemented as described in (Vaswani et al., 2017), though the position encoding was not used and instead separate embeddings were learned for each feature in the dataset. In order to convert scalar features into the categorical embeddings required by a transformer, scalar features were discretized into 8 quantiles and an embedding learned for each quantile. The HetTfmr model was identical to the transformer model except it maintained different parameters for the multihead attention and linear mappings for each column. This increased the parameter count of the model significantly, but not its computational complexity.

The hyperparameters tuned were the weight decay factor, the dropout probability, whether to learn an additional embedding for each column's embeddings, the number

[5]`https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html`

of attention heads, the hidden dimension, the number of layers, and what type of readout to use for producing a prediction from the hidden state output of the transformer (taking the mean, using a readout embedding, or the concat pool method of (Howard and Ruder, 2018)).

The sparsifying MLP (Prune) was the same as the MLP except at every k epochs during training the fraction p of weights with the smallest magnitude were permanently set to zero The hyperparameters tuned were the weight decay factor, the learning rate, the number of layers and hidden unit sizes (relative to the input size), k and p.

The TabNet model was implemented exactly as described in (Arik and Pfister, 2019), though we also added the option to use a softmax attention instead of a sparsemax attention, and did not include the sparsification term in the loss function. The hyperparameters tuned were the weight decay factor, the learning rate, the number of layers, the hidden dimension, and the attention function.

The VIB model was implemented as described in (Alemi, Fischer, and Dillon, 2018). We used a diagonal covariance, with 10 samples from the variational distribution during training and 20 during testing. The hyperparameters tuned were the learning rate, the number of layers and hidden unit sizes (relative to the input size), the β coefficient, and the number of mixture components in the mixture of gaussians used in the marginal distribution.

Subset-Restricted Finetuning

All models and hyperparameter searches were implemented as described above, except each model was given 20 rounds of HPO instead of 10, and for pretrained models HPO included searching over two learning rates: one for the new last fully connected layer, and one for the rest of the model parameters.

DAE pretraining used swap noise as described in (Jahrer, 2018). MLM pretraining used the protocol described in (Devlin et al., 2019), though separate parameters were used to make predictions from each final hidden state since each hidden state had a different categorical input. RTD pretraining was carried out as described in (Clark et al., 2020), except using random noise instead of a generator model.

5.8 Appendix: Benchmark Dataset Information

References

Alemi, Alexander A., Ian Fischer, and Joshua V. Dillon (2018). “Uncertainty in the Variational Information Bottleneck”. In: *arXiv:1807.00906 [cs, stat]*.

Table 5.2: Additional benchmark dataset information. All datasets are binary classification problems. Total Values is N Datapts $\times$ N Feats. Positive:Total is the fraction of datapoints that are the positive class. N Minority Class is the number of positive or negative datapoints in the dataset, whichever is smaller.

Dataset Name	N Datapts	N Feats	N Cat Feats	N Cont Feats	Total Values	Positive:Total	N Minority Class Datapts
1995_income	32561	14	8	6	455854	0.241	7842
adult	34190	25	0	25	854750	0.854	4982
albert	425240	79	0	79	33593960	0.5	212620
bank_marketing	45211	16	11	5	723376	0.117	5289
blastchar	7043	20	17	3	140860	0.265	1869
dota2games	92650	117	117	0	10840050	0.527	43868
fabert	8237	801	0	801	6597837	0.113	933
hcdr_main	307511	120	51	69	36901320	0.081	24825
htru2	17898	8	0	8	143184	0.092	1639
insurance_co	5822	85	82	3	494870	0.06	348
jannis	83733	55	0	55	4605315	0.02	1687
jasmine	2984	145	137	8	432680	0.5	1492
online_shoppers	12330	17	11	6	209610	0.155	1908
philippine	5832	309	35	274	1802088	0.5	2916
qsar_bio	1055	41	0	41	43255	0.337	356
seismicbumps	2583	18	4	14	46494	0.066	170
shrutime	10000	11	5	6	110000	0.204	2037
spambase	4601	57	0	57	262257	0.394	1813
sylvine	5124	20	0	20	102480	0.5	2562
volkert	58310	181	33	148	10554110	0.127	7377

Table 5.3: Benchmark dataset links.

Dataset Name	URL
1995_income	https://www.kaggle.com/lodetomasi1995/income-classification
adult	http://automl.chalearn.org/data
albert	http://automl.chalearn.org/data
bank_marketing	https://archive.ics.uci.edu/ml/datasets/bank+marketing
blastchar	https://www.kaggle.com/blastchar/telco-customer-churn
dota2games	https://archive.ics.uci.edu/ml/datasets/Dota2+Games+Results
fabert	http://automl.chalearn.org/data
hcdr_main	https://www.kaggle.com/c/home-credit-default-risk
htru2	https://archive.ics.uci.edu/ml/datasets/HTRU2
insurance_co	https://archive.ics.uci.edu/ml/datasets/Insurance+Company+Benchmark+%28COIL+2000%29
jannis	http://automl.chalearn.org/data
jasmine	http://automl.chalearn.org/data
online_shoppers	https://archive.ics.uci.edu/ml/datasets/Online+Shoppers+Purchasing+Intention+Dataset
philippine	http://automl.chalearn.org/data
qsar_bio	https://archive.ics.uci.edu/ml/datasets/QSAR+biodegradation
seismicbumps	https://archive.ics.uci.edu/ml/datasets/seismic-bumps
shrutime	https://www.kaggle.com/shrutimechlearn/churn-modelling
spambase	https://archive.ics.uci.edu/ml/datasets/Spambase
sylvine	http://automl.chalearn.org/data
volkert	http://automl.chalearn.org/data

Table 5.4: Prediction accuracy relative to logistic regression on benchmark datasets. Values are the mean over 5 cross-validation splits, plus or minus the standard deviation. Bold values are those within one standard deviation of the maximum in the row. Larger values better.

Dataset	GBDT	HetTfmr	MLP	PruneMLP	TabNet	Tfmr	VIB
1995_income	0.07 ± 0.04	−0.06 ± 0.02	0.06 ± 0.02	0.03 ± 0.02	−0.01 ± 0.02	−0.03 ± 0.04	0.03 ± 0.04
adult	0.06 ± 0.01	0.00 ± 0.02	0.03 ± 0.02	0.02 ± 0.01	−0.05 ± 0.03	0.00 ± 0.02	0.02 ± 0.02
albert	0.120 ± 0.005	0.09 ± 0.01	0.038 ± 0.008	0.041 ± 0.004	−0.003 ± 0.004	0.093 ± 0.004	0.032 ± 0.005
bank_marketing	0.08 ± 0.05	−0.01 ± 0.03	0.06 ± 0.06	0.05 ± 0.03	−0.04 ± 0.02	−0.01 ± 0.05	0.01 ± 0.04
blastchar	0.02 ± 0.08	−0.02 ± 0.09	0.02 ± 0.04	−0.02 ± 0.04	−0.11 ± 0.06	0.03 ± 0.08	−0.04 ± 0.09
dota2games	−0.03 ± 0.01	−0.01 ± 0.01	−0.002 ± 0.006	−0.04 ± 0.01	−0.07 ± 0.03	−0.01 ± 0.01	−0.02 ± 0.01
fabert	0.11 ± 0.04	0.04 ± 0.08	0.03 ± 0.06	−0.05 ± 0.07	0.03 ± 0.07	0.01 ± 0.02	−0.17 ± 0.09
hcdr_main	0.006 ± 0.003	0.00 ± 0.01	0.004 ± 0.002	0.000 ± 0.004	0.002 ± 0.003	−0.01 ± 0.01	0.001 ± 0.004
htru2	0.11 ± 0.06	−0.3 ± 0.2	0.09 ± 0.06	0.04 ± 0.08	0.00 ± 0.09	−0.3 ± 0.1	0.03 ± 0.05
insurance_co	0.0 ± 0.0	0 ± 0	−0.04 ± 0.03	−0.02 ± 0.04	−0.01 ± 0.02	0 ± 0	−0.07 ± 0.05
jannis	−0.01 ± 0.01	0.002 ± 0.002	−0.01 ± 0.01	−0.02 ± 0.03	−0.01 ± 0.01	0.00 ± 0.02	−0.02 ± 0.04
jasmine	0.1 ± 0.2	0.1 ± 0.1	0.0 ± 0.1	−0.02 ± 0.10	−0.2 ± 0.2	0.0 ± 0.1	−0.1 ± 0.1
online_shoppers	0.21 ± 0.06	0.04 ± 0.05	0.09 ± 0.03	−0.01 ± 0.07	−0.06 ± 0.06	0.04 ± 0.05	0.01 ± 0.07
philippine	0.16 ± 0.03	0.08 ± 0.03	0.08 ± 0.05	0.05 ± 0.04	0.01 ± 0.04	−0.01 ± 0.10	−0.01 ± 0.05
qsar_bio	0.2 ± 0.1	0.0 ± 0.2	0.3 ± 0.1	0.2 ± 0.2	0.1 ± 0.2	0.1 ± 0.3	0.0 ± 0.3
seismicbumps	0.1 ± 0.1	0.1 ± 0.1	0.0 ± 0.1	0.0 ± 0.2	−0.1 ± 0.2	0.1 ± 0.1	−0.1 ± 0.1
shrutime	0.21 ± 0.02	0.22 ± 0.04	0.18 ± 0.03	0.21 ± 0.06	−0.17 ± 0.05	0.18 ± 0.06	0.05 ± 0.02
spambase	0.6 ± 0.4	−0.1 ± 0.2	0.3 ± 0.2	0.2 ± 0.3	0.1 ± 0.3	−0.2 ± 0.2	0.2 ± 0.3
sylvine	0.72 ± 0.08	0.2 ± 0.2	0.5 ± 0.1	0.32 ± 0.07	0.0 ± 0.2	0.3 ± 0.2	0.2 ± 0.1
volkert	0.18 ± 0.05	0.07 ± 0.04	0.17 ± 0.04	0.15 ± 0.01	0.15 ± 0.02	0.07 ± 0.02	0.13 ± 0.05

Table 5.5: AUROC relative to logistic regression on benchmark datasets. Values are the mean over 5 cross-validation splits, plus or minus the standard deviation. Bold values are those within one standard deviation of the maximum in the row. Larger values better.

Dataset	GBDT	HetTfmr	MLP	PruneMLP	TabNet	Tfmr	VIB
1995_income	0.10 ± 0.02	−0.07 ± 0.06	0.09 ± 0.02	0.06 ± 0.03	−0.07 ± 0.03	−0.07 ± 0.05	0.07 ± 0.02
adult	0.19 ± 0.02	−0.16 ± 0.04	0.10 ± 0.03	0.01 ± 0.02	−0.23 ± 0.05	−0.15 ± 0.03	0.06 ± 0.03
albert	0.211 ± 0.005	0.16 ± 0.02	0.08 ± 0.01	0.083 ± 0.003	0.009 ± 0.004	0.173 ± 0.006	0.06 ± 0.01
bank_marketing	0.41 ± 0.05	0.31 ± 0.04	0.39 ± 0.08	0.33 ± 0.03	−0.06 ± 0.09	0.30 ± 0.04	0.15 ± 0.03
blastchar	0.03 ± 0.06	−0.03 ± 0.08	0.02 ± 0.03	−0.05 ± 0.03	−0.21 ± 0.06	0.03 ± 0.04	−0.02 ± 0.01
dota2games	−0.05 ± 0.01	−0.003 ± 0.004	−0.003 ± 0.003	−0.05 ± 0.01	−0.11 ± 0.05	0.00 ± 0.01	−0.02 ± 0.01
fabert	0.3 ± 0.1	−0.4 ± 0.1	0.05 ± 0.07	−0.03 ± 0.09	−0.1 ± 0.2	−0.4 ± 0.1	−0.3 ± 0.2
hcdr_main	0.05 ± 0.01	0.02 ± 0.01	0.024 ± 0.006	−0.01 ± 0.02	0.013 ± 0.007	0.021 ± 0.004	−0.01 ± 0.01
htru2	0.2 ± 0.1	0.0 ± 0.2	0.34 ± 0.09	0.2 ± 0.4	0.0 ± 0.2	0.0 ± 0.2	0.3 ± 0.3
insurance_co	0.11 ± 0.05	0.08 ± 0.08	−0.1 ± 0.1	−0.1 ± 0.2	−0.07 ± 0.10	0.06 ± 0.04	−0.4 ± 0.1
jannis	0.64 ± 0.07	0.36 ± 0.05	0.52 ± 0.04	0.56 ± 0.05	0.16 ± 0.06	0.39 ± 0.09	0.27 ± 0.05
jasmine	0.2 ± 0.1	0.2 ± 0.1	0.11 ± 0.07	−0.01 ± 0.08	−0.1 ± 0.1	0.05 ± 0.02	−0.01 ± 0.06
online_shoppers	0.39 ± 0.07	−0.4 ± 0.1	0.30 ± 0.06	0.10 ± 0.07	−0.3 ± 0.1	−0.4 ± 0.1	0.0 ± 0.1
philippine	0.38 ± 0.06	0.14 ± 0.07	0.18 ± 0.05	0.2 ± 0.1	0.07 ± 0.03	0.0 ± 0.1	0.00 ± 0.06
qsar_bio	0.0 ± 0.2	−0.1 ± 0.2	0.4 ± 0.2	0.17 ± 0.08	0.3 ± 0.4	−0.1 ± 0.1	−0.1 ± 0.1
seismicbumps	−0.1 ± 0.2	−0.1 ± 0.3	0.0 ± 0.1	−0.1 ± 0.4	−0.6 ± 0.6	0.0 ± 0.2	−0.3 ± 0.2
shrutime	0.3 ± 0.1	0.27 ± 0.06	0.25 ± 0.08	0.28 ± 0.09	−0.4 ± 0.1	0.28 ± 0.07	0.02 ± 0.05
spambase	0.9 ± 0.5	−0.1 ± 0.3	0.7 ± 0.2	0.5 ± 0.2	0.2 ± 0.4	−0.1 ± 0.5	0.4 ± 0.3
sylvine	1.2 ± 0.1	0.27 ± 0.10	0.8 ± 0.3	0.6 ± 0.3	−0.02 ± 0.10	0.2 ± 0.2	0.4 ± 0.2
volkert	0.69 ± 0.04	0.35 ± 0.03	0.67 ± 0.04	0.55 ± 0.03	0.60 ± 0.01	0.38 ± 0.03	0.54 ± 0.04

arXiv: 1807.00906. URL: http://arxiv.org/abs/1807.00906 (visited on 01/09/2019).

Alemi, Alexander A., Ian Fischer, Joshua V. Dillon, and Kevin Murphy (2017). "Deep Variational Information Bottleneck". In: *International Conference on Learning Representations* abs/1612.00410. URL: https://arxiv.org/abs/1612.00410.

Arik, Sercan O and Tomas Pfister (2019). "TabNet: Attentive Interpretable Tabular Learning". In: *arXiv preprint arXiv:1908.07442*. URL: https://arxiv.org/abs/1908.07442.

Caron, Mathilde et al. (2019). "Unsupervised Pre-Training of Image Features on Non-Curated Data". In: *arXiv:1905.01278 [cs]*. arXiv: 1905.01278. URL: http://arxiv.org/abs/1905.01278 (visited on 02/23/2020).

Chapelle, Olivier, Bernhard Scholkopf, and Alexander Zien (2009). "Semi-supervised learning)". In: *IEEE Transactions on Neural Networks* 20.3, pp. 542–542.

Cheng, Heng-Tze et al. (2016). "Wide & deep learning for recommender systems". In: *Proceedings of the 1st workshop on deep learning for recommender systems*, pp. 7–10.

Clark, Kevin et al. (2020). "ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=r1xMH1BtvB.

De Brébisson, Alexandre et al. (2015). "Artificial Neural Networks Applied to Taxi Destination Prediction". In: *Proceedings of the 2015th International Conference on ECML PKDD Discovery Challenge - Volume 1526*. ECMLPKDDDC'15. Porto, Portugal: CEUR-WS.org, pp. 40–51.

Devlin, Jacob et al. (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *NAACL-HLT*.

Dua, Dheeru and Casey Graff (2017). *UCI Machine Learning Repository*. URL: http://archive.ics.uci.edu/ml.

Elezi, Ismail et al. (2018). "Transductive label augmentation for improved deep network learning". In: *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, pp. 1432–1437.

Goldbloom, Anthony (2017). *What Kaggle has learned from almost a million data scientists - Anthony Goldbloom (Kaggle)*. URL: https://www.youtube.com/watch?v=jmHbS8z57yI&feature=youtu.be.

Guo, Huifeng et al. (2018). "DeepFM: An End-to-End Wide & Deep Learning Framework for CTR Prediction". en. In: *arXiv:1804.04950 [cs, stat]*. arXiv: 1804.04950. URL: http://arxiv.org/abs/1804.04950 (visited on 02/08/2020).

Guyon, Isabelle et al. (2019). “Analysis of the AutoML Challenge series 2015-2018”. In: *AutoML*. Springer series on Challenges in Machine Learning. URL: `https://www.automl.org/wp-content/uploads/2018/09/chapter10-challenge.pdf`.

Hassibi, Babak et al. (1993). “Optimal Brain Surgeon: Extensions and Performance Comparisons”. In: *Proceedings of the 6th International Conference on Neural Information Processing Systems*. NIPS’93. Denver, Colorado: Morgan Kaufmann Publishers Inc., pp. 263–270.

Howard, Jeremy et al. (2018). *FastAI*. `https://github.com/fastai/fastai`.

Howard, Jeremy and Sebastian Ruder (2018). “Universal language model fine-tuning for text classification”. In: *arXiv preprint arXiv:1801.06146*.

Jahrer, Michael (2018). *Porto Seguro’s Safe Driver Prediction*. en. URL: `https://kaggle.com/c/porto-seguro-safe-driver-prediction` (visited on 02/24/2020).

Ke, Guolin, Qi Meng, et al. (2017). “LightGBM: A highly efficient gradient boosting decision tree”. In: *Advances in Neural Information Processing Systems*, pp. 3146–3154. URL: `https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf`.

Ke, Guolin, Jia Zhang, et al. (2019). *TabNN: A Universal Neural Network Solution for Tabular Data*. URL: `https://openreview.net/forum?id=r1eJssCqY7`.

Klambauer, Günter et al. (2017). “Self-normalizing neural networks”. In: *Advances in neural information processing systems*, pp. 971–980.

Li, Zeyu et al. (2020). “Interpretable Click-Through Rate Prediction through Hierarchical Attention”. en. In: *Proceedings of the 13th International Conference on Web Search and Data Mining*. Houston TX USA: ACM, pp. 313–321. ISBN: 978-1-4503-6822-3. DOI: `10.1145/3336191.3371785`. URL: `http://dl.acm.org/doi/10.1145/3336191.3371785` (visited on 02/08/2020).

Loshchilov, Ilya and Frank Hutter (2017). “Decoupled Weight Decay Regularization”. In: *International Conference on Learning Representations*. URL: `https://arxiv.org/abs/1711.05101`.

Mocanu, Decebal Constantin et al. (2018). “Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science”. In: *Nature Communications*.

Morcos, Ari S. et al. (2019). “One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers”. en. In: *arXiv:1906.02773 [cs, stat]*. arXiv: 1906.02773. URL: `http://arxiv.org/abs/1906.02773` (visited on 01/15/2020).

Murshed, M. G. Sarwar et al. (2019). “Machine Learning at the Network Edge: A Survey”. In: URL: `https://arxiv.org/abs/1908.00080v2` (visited on 02/24/2020).

Oliver, Avital et al. (2018). “Realistic evaluation of deep semi-supervised learning algorithms”. In: *Advances in Neural Information Processing Systems*, pp. 3235–3246.

Paszke, Adam et al. (2019). “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Schneider, Johannes and Michail Vlachos (2019). “Mass Personalization of Deep Learning”. en. In: URL: `https://arxiv.org/abs/1909.02803v2` (visited on 02/24/2020).

Song, Weiping et al. (2019). “AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks”. en. In: *Proceedings of the 28th ACM International Conference on Information and Knowledge Management - CIKM ’19*. arXiv: 1810.11921, pp. 1161–1170. DOI: `10.1145/3357384.3357925`. URL: `http://arxiv.org/abs/1810.11921` (visited on 02/08/2020).

Stretcu, Otilia et al. (2019). “Graph Agreement Models for Semi-Supervised Learning”. In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp. 8713–8723. URL: `http://papers.nips.cc/paper/9076-graph-agreement-models-for-semi-supervised-learning.pdf`.

Strouse, Daniel and David J. Schwab (2015). “The Deterministic Information Bottleneck”. In: *Neural Computation* 29, pp. 1611–1630. URL: `https://arxiv.org/abs/1604.00268`.

Sun, Qiang et al. (2019). “DeepEnFM: Deep neural networks with Encoder enhanced Factorization Machine”. In: URL: `https://openreview.net/forum?id=SJlyta4YPS` (visited on 02/15/2020).

Tanha, Jafar, Maarten van Someren, and Hamideh Afsarmanesh (2017). “Semi-supervised self-training for decision tree classifiers”. In: *International Journal of Machine Learning and Cybernetics* 8, pp. 355–370.

Tishby, Naftali, Fernando C. Pereira, and William Bialek (2000). “The information bottleneck method”. In: *arXiv:physics/0004057*. arXiv: physics/0004057. URL: `http://arxiv.org/abs/physics/0004057` (visited on 12/11/2018).

Vaswani, Ashish et al. (2017). “Attention is all you need”. In: *Advances in neural information processing systems*, pp. 5998–6008.

Wang, Ruoxi et al. (2017). “Deep & Cross Network for Ad Click Predictions”. In: *ADKDD@KDD*.

Xiao, Jun et al. (2017). “Attentional Factorization Machines: Learning the Weight of Feature Interactions via Attention Networks”. en. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. Melbourne, Australia: International Joint Conferences on Artificial Intelligence Organization, pp. 3119–3125. ISBN: 978-0-9992411-0-3. DOI: `10.24963/ijcai.2017/435`. URL: `https://www.ijcai.org/proceedings/2017/435` (visited on 02/08/2020).

Yang, Yongxin, Irene Garcia Morillo, and Timothy M Hospedales (2018). "Deep neural decision trees". In: *arXiv preprint arXiv:1806.06988*.

Chapter 6

MINIMAL ACHIEVABLE SUFFICIENT STATISTIC LEARNING: ADDRESSING ISSUES IN THE INFORMATION BOTTLENECK

6.1 Introduction

The *representation learning* approach to machine learning focuses on finding a representation Z of an input random variable X that is useful for predicting a random variable Y (Goodfellow, Y. Bengio, and Courville, 2016).

What makes a representation Z "useful" is much debated, but a common assertion is that Z should be a *minimal sufficient statistic* of X for Y(Adragni, Kofi P. and Cook, R. Dennis, 2009; Shamir, Sabato, and Tishby, 2010; James, Mahoney, and Crutchfield, 2017; Achille and Soatto, 2018b). That is:

1. Z should be a *statistic* of X. This means $Z = f(X)$ for some function f.
2. Z should be *sufficient* for Y. This means $p(X|Z,Y) = p(X|Z)$.
3. Given that Z is a sufficient statistic, it should be *minimal* with respect to X. This means for any measurable, non-invertible function g, $g(Z)$ is no longer sufficient for Y.[1]

In other words: a minimal sufficient statistic is a random variable Z that tells you everything about Y you could ever care about, but if you do any irreversible processing to Z, you are guaranteed to lose some information about Y. Such a representation would be extremely desirable in machine learning, as it would (essentially by definition) preclude overfitting by excluding any information not relevant to the task.

Minimal sufficient statistics have a long history in the field of statistics (Lehmann and Scheffe, 1950; Dynkin, 1951). But the minimality condition (3, above) is perhaps too strong to be useful in machine learning, since it is a statement about *any* function g, rather than about functions in a practical hypothesis class like the class of deep neural networks. As we will discuss below, this strictness erodes the mathematical

[1]This is not the most common phrasing of statistical minimality, but we feel it is more understandable. For the equivalence of this phrasing and the standard definition see Section 6.8.

justification for the popular Information Bottleneck method (Tishby, Pereira, and Bialek, 2000).

Instead, in this chapter we consider *minimal achievable sufficient statistics*: sufficient statistics that are minimal among some particular set of functions.

Definition 1 (Minimal Achievable Sufficient Statistic)**.** Let $Z = f(X)$ be a sufficient statistic of X for Y. Z is *minimal achievable* with respect to a set of functions $\mathcal{F}$ if $f \in \mathcal{F}$ and for any Lipschitz continuous, non-invertible function g where $g \circ f \in \mathcal{F}$, $g(Z)$ is no longer sufficient for Y.

In this chapter we introduce Conserved Differential Information (CDI), an information-theoretic quantity that, unlike mutual information, is meaningful for deterministically-dependent continuous random variables, such as the input and output of a deep network. We also introduce Minimal Achievable Sufficient Statistic Learning (MASS Learning), a training objective based on CDI for finding minimal achievable sufficient statistics that achieves competitive performance on supervised learning and uncertainty quantification benchmarks.

6.2 Conserved Differential Information

Before we present MASS Learning, we need to introduce Conserved Differential Information (CDI), on which MASS Learning is based.

CDI is an information-theoretic quantity that addresses an oft-cited issue in machine learning (Bell and Sejnowski, 1995; Amjad and Geiger, 2018; Saxe et al., 2018; Nash, Kushman, and Williams, 2018; Goldfeld et al., 2018), which is that for a continuous random variable X and a continuous, non-constant function f, the mutual information $I(X, f(X))$ is infinite. (See Section 6.8 for details.) This makes $I(X, f(X))$ unsuitable for use in a learning objective when f is, for example, a standard deep network.

The infinitude of $I(X, f(X))$ has been circumvented in prior works by two strategies. One is to discretize X and $f(X)$ (Tishby and Zaslavsky, 2015; Shwartz-Ziv and Tishby, 2017), though this is controversial (Saxe et al., 2018). Another is to use a random variable Z with distribution $p(Z|X)$ as the representation of X rather than using $f(X)$ itself as the representation (Alemi, Fischer, Dillon, and Murphy, 2017; Kolchinsky, Tracey, and Wolpert, 2017; Achille and Soatto, 2018b). In this latter approach, $p(Z|X)$ is usually implemented by adding noise to a deep network that takes X as input.

These are both reasonable strategies for avoiding the infinitude of $I(X, f(X))$. But another approach would be to derive a new information-theoretic quantity that is better suited to this situation. To that end we present Conserved Differential Information:

Definition 2. For a continuous random variable X taking values in $\mathbb{R}^d$ and a Lipschitz continuous function $f\colon \mathbb{R}^d \to \mathbb{R}^r$, the **Conserved Differential Information** (CDI) is

$$\boxed{C(X, f(X)) := H(f(X)) - \mathbb{E}_X\left[\log\left(J_f(X)\right)\right]} \tag{6.1}$$

where H denotes the differential entropy

$$H(Z) = -\int p(z) \log p(z) \, \mathrm{d}z$$

and J_f is the Jacobian determinant of f

$$J_f(x) = \sqrt{\det\left(\frac{\partial f(x)}{\partial x^{\mathrm{T}}}\left(\frac{\partial f(x)}{\partial x^{\mathrm{T}}}\right)^{\mathrm{T}}\right)}$$

with $\frac{\partial f(x)}{\partial x^{\mathrm{T}}} \in \mathbb{R}^{r\times d}$ the Jacobian matrix of f at x.

Readers familiar with normalizing flows (Rezende and Mohamed, 2015) or Real NVP (Dinh, Sohl-Dickstein, and S. Bengio, 2017) will note that the Jacobian determinant used in those methods is a special case of the Jacobian determinant in the definition of CDI. This is because normalizing flows and Real NVP are based on the change of variables formula for invertible mappings, while CDI is based in part on the more general change of variables formula for non-invertible mappings. More details on this connection are given in Section 6.8. The mathematical motivation for CDI based on the recent work of (Koliander et al., 2016) is provided in Section 6.8. Figure 6.1 gives a visual example of what CDI measures about a function.

The conserved differential information $C(X, f(X))$ between deterministically-dependent, continuous random variables behaves much like mutual information does between discrete random variables. For example, when f is invertible, $C(X, f(X)) = H(X)$, just like with the mutual information between discrete random variables. Most importantly for our purposes, though, $C(X, f(X))$ obeys the following data processing inequality:

Theorem 1 (CDI Data Processing Inequality)**.** *For Lipschitz continuous functions f and g with the same output space,*

$$C\left(X, f(X)\right) \geq C\left(X, g(f(X))\right)$$

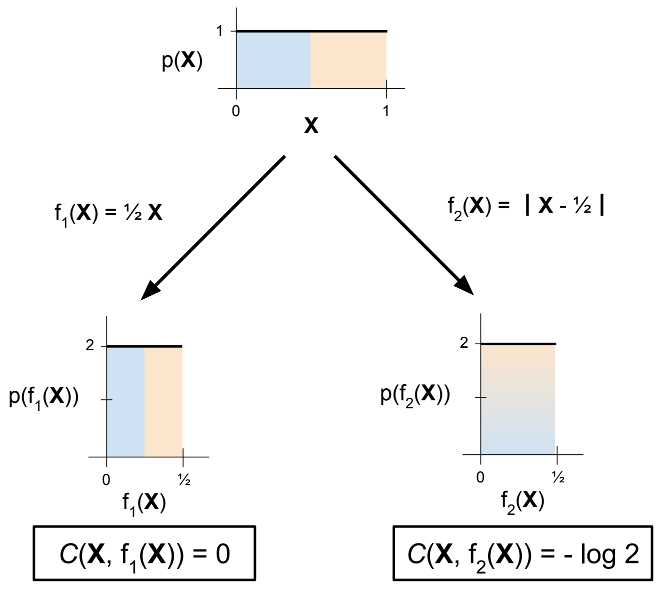


Figure 6.1: CDI of two functions f_1 and f_2 of the random variable X. Even though the random variables $f_1(X)$ and $f_2(X)$ have the same distribution, $C(X, f_1(X))$ is different from $C(X, f_2(X))$. This is because f_1 is an invertible function, while f_2 is not. CDI quantifies, roughly speaking, "how non-invertible" f_2 is.

with equality if and only if g is invertible almost everywhere.

The proof is in Section 6.8.

6.3 MASS Learning

With CDI and its data processing inequality in hand, we can give the following optimization-based characterization of minimal achievable sufficient statistics:

Theorem 2. *Let X be a continuous random variable, Y be a discrete random variable, and $\mathcal{F}$ be any set of Lipschitz continuous functions with a common output space (e.g. different parameter settings of a deep network). If*

$$\begin{aligned} f \in \arg\min_{S\in\mathcal{F}} \; & C(X, S(X)) \\ s.t. \; & I(S(X), Y) = \max_{S'} I(S'(X), Y) \end{aligned}$$

then $f(X)$ is a minimal achievable sufficient statistic of X for Y with respect to $\mathcal{F}$.

Proof. First note the following lemma (Cover and Thomas, 2006).

Lemma 1. *$Z = f(X)$ is a sufficient statistic for a discrete random variable Y if and only if $I(Z, Y) = \max_{S'} I(S'(X), Y)$.*

Lemma 1 guarantees that any f satisfying the conditions in Theorem 2 is sufficient. Suppose such an f was not minimal achievable. Then by Definition 1 there would exist a non-invertible, Lipschitz continuous g such that $g(f(X))$ was sufficient. But by Theorem 1, it would then also be the case that $C(X, g(f(X))) < C(X, f(X))$, which would contradict f minimizing $C(X, S(X))$. □

We can turn Theorem 2 into a learning objective over functions f by relaxing the strict constraint into a Lagrangian formulation with Lagrange multiplier $1/\beta$ for $\beta > 0$:

$$C(X, f(X)) - \frac{1}{\beta} I(f(X), Y)$$

The larger the value of β, the more our objective will encourage minimality over sufficiency. We can then simplify this formulation using the identity $I(f(X), Y) = H(Y) - H(Y|f(X))$, which gives us the following optimization objective:

$$\boxed{\begin{aligned}\mathcal{L}_{MASS}(f) := H(Y|f(X)) + \beta H(f(X)) \\ - \beta \mathbb{E}_X[\log J_f(X)].\end{aligned}} \tag{6.2}$$

We refer to minimizing this objective as **MASS Learning**.

Practical implementation

In practice, we are interested in using MASS Learning to train a deep network f_θ with parameters θ using a finite dataset $\{(x_i, y_i)\}_{i=1}^N$ of N datapoints sampled from the joint distribution $p(x, y)$ of X and Y. To do this, we introduce a parameterized variational approximation $q_\phi(f_\theta(x)|y) \approx p(f_\theta(x)|y)$. Using q_ϕ, we minimize the following empirical upper bound to $\mathcal{L}_{MASS}$:

$$\begin{aligned}\mathcal{L}_{MASS} \leq \widehat{\mathcal{L}}_{MASS}(\theta, \phi) := \frac{1}{N} \sum_{i=1}^{N} & - \log q_\phi(y_i | f_\theta(x_i)) \\ & - \beta \log q_\phi(f_\theta(x_i)) \\ & - \beta \log J_{f_\theta}(x_i),\end{aligned}$$

where the quantity $q_\phi(f_\theta(x_i))$ is computed as $\sum_y q_\phi(f_\theta(x_i)|y)p(y)$ and the quantity $q_\phi(y_i|f_\theta(x_i))$ is computed with Bayes rule as $\frac{q_\phi(f_\theta(x_i)|y_i)p(y_i)}{\sum_y q_\phi(f_\theta(x_i)|y)p(y)}$. When Y is discrete and takes on finitely many values, as in classification problems, and when we choose a variational distribution q_ϕ that is differentiable with respect to ϕ (e.g. a multivariate Gaussian), then we can minimize $\widehat{\mathcal{L}}_{MASS}(\theta, \phi)$ using stochastic gradient descent (SGD).

To perform classification using our trained network, we use the learned variational distribution q_ϕ and Bayes rule:

$$p(y_i|x_i) \approx p(y_i|f_\theta(x_i)) \approx \frac{q_\phi(f_\theta(x_i)|y_i)p(y_i)}{\sum_y q_\phi(f_\theta(x_i)|y)p(y)}.$$

Computing the J_{f_θ} term in $\widehat{\mathcal{L}}_{MASS}$ for every sample in an SGD minibatch is too expensive to be practical. For $f_\theta \colon \mathbb{R}^d \to \mathbb{R}^r$, doing so would require on the order of r times more operations than in standard training of deep networks by, since computing the J_{f_θ} term involves computing the full Jacobian matrix of the network, which, in our implementation, involves performing r backpropagations. Thus to make training tractable, we use a subsampling strategy: we estimate the J_{f_θ} term using only a $1/r$ fraction of the datapoints in a minibatch. In practice, we have found this subsampling strategy to not noticeably alter the numerical value of the J_{f_θ} term during training.

Subsampling for the J_{f_θ} term results in a significant training speedup, but it must nevertheless be emphasized that, even with subsampling, our implementation of MASS Learning is roughly eight times as slow as standard deep network training. (Unless $\beta = 0$, in which case the speed is the same.) This is by far the most significant drawback of (our implementation of) MASS Learning. There are many easier-to-compute upper bounds or estimates of J_{f_θ} that one could use to make MASS Learning faster, and one could also potentially find non-invertible network architectures which admit more efficiently computable Jacobians, but we do not explore these options.

6.4 Related Work

Connection to the Information Bottleneck

The well-studied Information Bottleneck learning method (Tishby, Pereira, and Bialek, 2000; Tishby and Zaslavsky, 2015; Strouse and Schwab, 2015; Alemi, Fischer, Dillon, and Murphy, 2017; Saxe et al., 2018; Amjad and Geiger, 2018; Goldfeld et al., 2018; Kolchinsky, Tracey, and Van Kuyk, 2019; Achille and Soatto, 2018b; Achille and Soatto, 2018a) is based on minimizing the Information Bottleneck Lagrangian

$$\mathcal{L}_{IB}(Z) := \beta I(X, Z) - I(Y, Z)$$

for $\beta > 0$, where Z is the representation whose conditional distribution $p(Z|X)$ one is trying to learn.

The $\mathcal{L}_{IB}$ learning objective can be motivated based on pure information-theoretic elegance. But some works like (Shamir, Sabato, and Tishby, 2010) also point out the connection between the $\mathcal{L}_{IB}$ objective and minimal sufficient statistics, which is based on the following theorem:

Theorem 3. *Let X be a discrete random variable drawn according to a distribution $p(X|Y)$ determined by the discrete random variable Y. Let $\mathcal{F}$ be the set of deterministic functions of X to any target space. Then $f(X)$ is a minimal sufficient statistic of X for Y if and only if*

$$\begin{aligned} f \in \arg\min_{S\in\mathcal{F}} \; & I(X, S(X)) \\ s.t. \;\; & I(S(X), Y) = \max_{S'\in\mathcal{F}} I(S'(X), Y). \end{aligned}$$

The $\mathcal{L}_{IB}$ objective can then be thought of as a Lagrangian relaxation of the optimization problem in this theorem.

Theorem 3 only holds for discrete random variables. For continuous X it holds only in the reverse direction, so minimizing $\mathcal{L}_{IB}$ for continuous X has no formal connection to finding minimal sufficient statistics, not to mention minimal achievable sufficient statistics. See Section 6.8 for details.

Nevertheless, the optimization problems in Theorem 2 and Theorem 3 are extremely similar, relying as they both do on Lemma 1 for their proofs. And the idea of relaxing the optimization problem in Theorem 2 into a Lagrangian formulation to get $\mathcal{L}_{MASS}$ is directly inspired by the Information Bottleneck. So while MASS Learning and Information Bottleneck learning entail different network architectures and loss functions, there is an Information Bottleneck flavor to MASS Learning.

Jacobian Regularization

The presence of the J_{f_θ} term in $\widehat{\mathcal{L}}_{MASS}$ is reminiscent of contrastive autoencoders (Rifai et al., 2011) and Jacobian Regularization (Sokolic et al., 2017; Ross and Doshi-Velez, 2018; Varga, Csiszárik, and Zombori, 2017; "Sensitivity and Generalization in Neural Networks" 2018; Jakubovitz and Giryes, 2018). Both these techniques are based on the notion that minimizing $\mathbb{E}_X[||D_f(X)||_F]$, where $D_f(x) = \frac{\partial f(x)}{\partial x^{\mathrm{T}}} \in \mathbb{R}^{r\times d}$ is the Jacobian matrix, improves generalization and adversarial robustness.

This may seem paradoxical at first, since by applying the AM-GM inequality to the eigenvalues of $D_f(x)D_f(x)^{\mathrm{T}}$ we have

$$\begin{aligned}
\mathbb{E}_X[\|D_f(X)\|_F^{2r}] &= \mathbb{E}_X[\mathrm{Tr}(D_f(X)D_f(X)^{\mathrm{T}})^r] \\
&\geq \mathbb{E}_X[r^r \det(D_f(X)D_f(X)^{\mathrm{T}})] \\
&= \mathbb{E}_X[r^r J_f(X)^2] \\
&\geq \log \mathbb{E}_X[r^r J_f(X)^2] \\
&\geq 2\mathbb{E}_X[\log J_f(X)] + r \log r
\end{aligned}$$

and $\mathbb{E}_X[\log J_f(X)]$ is being *maximized* by $\widehat{\mathcal{L}}_{MASS}$. So $\widehat{\mathcal{L}}_{MASS}$ might seem to be optimizing for worse generalization according to the Jacobian regularization literature. However, the entropy term in $\widehat{\mathcal{L}}_{MASS}$ strongly encourages minimizing $\mathbb{E}_X[\|D_f(X)\|_F]$. So overall $\widehat{\mathcal{L}}_{MASS}$ seems to be seeking the right balance of sensitivity (dependent on the value of β) in the network to its inputs, which is precisely in alignment with what the Jacobian regularization literature suggests.

6.5 Experiments

In this section we compare MASS Learning to other approaches for training deep networks. Code to reproduce all experiments is available online.[2] Full details on all experiments is in Section 6.8.

We use the abbreviation “SoftmaxCE” to refer to the standard approach of training deep networks for classification problems by minimizing the softmax cross entropy loss

$$\widehat{\mathcal{L}}_{SoftmaxCE}(\theta) := -\frac{1}{N}\sum_{i=1}^{N}\Big(\log \mathtt{softmax}(f_\theta(x_i))_{y_i}\Big)$$

where $\mathtt{softmax}(f_\theta(x_i))_{y_i}$ is the y_ith element of the softmax function applied to the outputs $f_\theta(x_i)$ of the network's last linear layer. As usual, $\mathtt{softmax}(f_\theta(x_i))_{y_i}$ is taken to be the network's estimate of $p(y_i|x_i)$.

We also compare against the Variational Information Bottleneck method (Alemi, Fischer, Dillon, and Murphy, 2017) for representation learning, which we abbreviate as “VIB”.

We use two networks in our experiments. “SmallMLP” is a feedforward network with two fully-connected layers of 400 and 200 hidden units, respectively, both with `elu` nonlinearities (Clevert, Unterthiner, and Hochreiter, 2015). “ResNet20” is the 20-layer residual network of (He et al., 2016).

[2] `https://github.com/mwcvitkovic/MASS-Learning`

We performed all experiments on the CIFAR-10 dataset (Krizhevsky, 2009) to ensure consistency with previously published results and implemented all experiments using PyTorch (Paszke et al., 2017).

Classification Accuracy and Regularization

We first confirm that networks trained by MASS Learning can make accurate predictions in supervised learning tasks. We compare the classification accuracy of networks trained on varying amounts of data to see the extent to which MASS Learning regularizes networks.

Classification accuracies for the SmallMLP network are shown in Table 6.1, and for the ResNet20 network in Table 6.2. For the SmallMLP network, MASS Learning performs slightly worse than SoftmaxCE and VIB training. For the larger ResNet20 network, MASS Learning performs equivalently to the other methods. It is notable that with the ResNet20 network VIB and MASS Learning both perform well when $\beta = 0$, and neither perform significantly better than SoftmaxCE. This may be because the hyperparameters used in training the ResNet20 network, which were taken directly from the original paper (He et al., 2016), are specifically tuned for SoftmaxCE training and are more sensitive to the specifics of the network architecture than to the loss function.

Uncertainty Quantification

We also evaluate the ability of networks trained by MASS Learning to properly quantify their uncertainty about their predictions. We assess uncertainty quantification in two ways: using proper scoring rules (Lakshminarayanan, Pritzel, and Blundell, 2017), which are scalar measures of how well a network's predictive distribution $p(y|f_\theta(x))$ is calibrated, and by assessing performance on an out-of-distribution (OOD) detection task.

Tables 6.3 through 6.8 show the uncertainty quantification performance of networks according to two proper scoring rules: the Negative Log Likelihood (NLL) and the Brier Score. The entropy and test accuracy of the predictive distributions are also given, for reference.

For the SmallMLP network in Tables 6.3, 6.4, and 6.5, VIB provides the best combination of high accuracy and low NLL and Brier score across all sizes of training set, despite SoftmaxCE with weight decay achieving the best scoring rule values. For the larger ResNet20 network in Tables 6.6 and 6.7, MASS Learning

Table 6.1: Test-set classification accuracy (percent) on CIFAR-10 dataset using the SmallMLP network trained by various methods. Full experiment details are in Section 6.8. Values are the mean classification accuracy over 4 training runs with different random seeds, plus or minus the standard deviation. Emboldened accuracies are those for which the maximum observed mean accuracy in the column was within one standard deviation. WD is weight decay; D is dropout.

Method	Training Set Size		
	2500	10,000	40,000
SoftmaxCE	34.2 ± 0.8	**44.6 ± 0.6**	52.7 ± 0.4
SoftmaxCE, WD	23.9 ± 0.9	36.4 ± 0.9	48.1 ± 0.1
SoftmaxCE, D	33.7 ± 1.1	44.1 ± 0.6	53.7 ± 0.3
VIB, β=1e−1	32.2 ± 0.6	40.6 ± 0.4	46.1 ± 0.5
VIB, β=1e−2	34.6 ± 0.4	43.8 ± 0.8	51.9 ± 0.8
VIB, β=1e−3	**35.6 ± 0.5**	**44.6 ± 0.6**	51.8 ± 0.8
VIB, β=1e−1, D	29.0 ± 0.6	40.1 ± 0.5	49.5 ± 0.5
VIB, β=1e−2, D	32.5 ± 0.9	43.9 ± 0.3	53.6 ± 0.3
VIB, β=1e−3, D	34.5 ± 1.0	**44.4 ± 0.4**	**54.3 ± 0.2**
MASS, β=1e−2	29.6 ± 0.4	39.9 ± 1.2	46.3 ± 1.2
MASS, β=1e−3	32.7 ± 0.8	41.5 ± 0.7	47.8 ± 0.8
MASS, β=1e−4	34.0 ± 0.3	41.5 ± 1.1	47.9 ± 0.8
MASS, β=0	34.1 ± 0.6	42.0 ± 0.6	48.2 ± 0.9
MASS, β=1e−2, D	29.3 ± 1.2	41.7 ± 0.4	52.0 ± 0.6
MASS, β=1e−3, D	31.5 ± 0.6	43.7 ± 0.2	53.1 ± 0.4
MASS, β=1e−4, D	32.7 ± 0.8	43.4 ± 0.5	53.2 ± 0.1
MASS, β=0, D	32.2 ± 1.1	43.9 ± 0.4	52.7 ± 0.0

provides the best combination of accuracy and proper scoring rule performance, though its performance falters when trained on only 2,500 datapoints in Table and 6.8. These ResNet20 UQ results also show the trend that MASS Learning with larger β leads to better calibrated network predictions. Thus, as measured by proper scoring rules, MASS Learning can significantly improve the calibration of a network's predictions while maintaining the same accuracy.

Tables 6.9 through 6.14 show metrics for performance on an OOD detection task where the network predicts not just the class of the input image, but whether the image is from its training distribution (CIFAR-10 images) or from another distribution (SVHN images (Netzer et al., 2011)). Following (Hendrycks and Gimpel, 2017) and (Alemi, Fischer, and Dillon, 2018), the metrics we report for this task are the Area under the ROC curve (AUROC) and Average Precision score (APR). APR depends on whether the network is tasked with identifying in-distribution or out-of-distribution

Table 6.2: Test-set classification accuracy (percent) on CIFAR-10 dataset using the ResNet20 network trained by various methods. No data augmentation was used — full details in Section 6.8. Values are the mean classification accuracy over 4 training runs with different random seeds, plus or minus the standard deviation. Emboldened accuracies are those for which the maximum observed mean accuracy in the column was within one standard deviation.

Method	Training Set Size		
	2500	10,000	40,000
SoftmaxCE	**50.0 ± 0.7**	**67.5 ± 0.8**	**81.7 ± 0.3**
VIB, β=1e−3	**49.5 ± 1.1**	**66.9 ± 1.0**	81.0 ± 0.3
VIB, β=1e−4	49.4 ± 1.0	66.4 ± 0.5	81.2 ± 0.4
VIB, β=1e−5	**50.0 ± 1.1**	**67.9 ± 0.8**	80.9 ± 0.5
VIB, β=0	**50.6 ± 0.8**	**67.1 ± 1.0**	**81.5 ± 0.2**
MASS, β=1e−3	38.2 ± 0.7	59.6 ± 0.8	75.8 ± 0.5
MASS, β=1e−4	**49.9 ± 1.0**	66.6 ± 0.4	80.6 ± 0.5
MASS, β=1e−5	**50.1 ± 0.5**	**67.4 ± 1.0**	**81.6 ± 0.4**
MASS, β=0	**50.2 ± 1.0**	**67.4 ± 0.3**	**81.5 ± 0.2**

images; we report values for both cases as APR In and APR Out, respectively.

There are different detection methods that networks can use to identify OOD inputs. One way, applicable to all training methods, is to use the entropy of the predictive distribution $p(y|f_\theta(x))$: larger entropy suggests the input is OOD. For networks trained by MASS Learning, the variational distribution $q_\phi(f_\theta(x)|y)$ is a natural OOD detector: a small value of $\max_i q_\phi(f_\theta(x)|y_i)$ suggests the input is OOD. For networks trained by SoftmaxCE, a distribution $q_\phi(f_\theta(x)|y)$ can be learned by MLE on the training set and used to detect OOD inputs in the same way.

For both the SmallMLP network in Tables 6.9, 6.10, and 6.11 and the ResNet20 network in Tables 6.12, 6.13, and 6.14, MASS Learning performs comparably or better than SoftmaxCE and VIB. However, one should note that MASS Learning with $\beta = 0$ gives performance not significantly different to MASS Learning with $\beta \neq 0$ on these OOD tasks, which suggests that the good performance of MASS Learning may be due to its use of a variational distribution to produce predictions, rather than to the overall MASS Learning training scheme.

Table 6.3: Uncertainty quantification metrics (proper scoring rules) on CIFAR-10 using the SmallMLP network trained on 40,000 datapoints. Test Accuracy and Entropy of the network's predictive distribution are given for reference. Full experiment details are in Section 6.8. Values are the mean over 4 training runs with different random seeds, plus or minus the standard deviation. Emboldened values are those for which the minimum observed mean value in the column was within one standard deviation. WD is weight decay; D is dropout. Lower values are better.

Method	Test Accuracy	Entropy	NLL	Brier Score
SoftmaxCE	52.7 ± 0.4	0.211 ± 0.003	4.56 ± 0.07	0.0840 ± 0.0005
SoftmaxCE, WD	48.1 ± 0.1	1.500 ± 0.009	**1.47 ± 0.01**	**0.0660 ± 0.0003**
SoftmaxCE, D	53.7 ± 0.3	0.606 ± 0.005	1.79 ± 0.02	0.0681 ± 0.0005
VIB, β=1e−1	46.1 ± 0.5	0.258 ± 0.005	5.35 ± 0.15	0.0944 ± 0.0009
VIB, β=1e−2	51.9 ± 0.8	0.193 ± 0.004	5.03 ± 0.19	0.0861 ± 0.0015
VIB, β=1e−3	51.8 ± 0.8	0.174 ± 0.003	5.49 ± 0.20	0.0866 ± 0.0015
VIB, β=1e−1, D	49.5 ± 0.5	0.957 ± 0.005	1.62 ± 0.01	**0.0660 ± 0.0003**
VIB, β=1e−2, D	53.6 ± 0.3	0.672 ± 0.014	1.69 ± 0.01	0.0668 ± 0.0006
VIB, β=1e−3, D	54.3 ± 0.2	0.617 ± 0.007	1.75 ± 0.02	0.0677 ± 0.0005
MASS, β=1e−2	46.3 ± 1.2	0.203 ± 0.005	6.89 ± 0.16	0.0968 ± 0.0024
MASS, β=1e−3	47.8 ± 0.8	0.207 ± 0.004	5.89 ± 0.21	0.0935 ± 0.0017
MASS, β=1e−4	47.9 ± 0.8	0.212 ± 0.003	5.71 ± 0.16	0.0934 ± 0.0017
MASS, β=0	48.2 ± 0.9	0.208 ± 0.004	5.74 ± 0.20	0.0927 ± 0.0017
MASS, β=1e−2, D	52.0 ± 0.6	0.690 ± 0.013	1.85 ± 0.03	0.0694 ± 0.0005
MASS, β=1e−3, D	53.1 ± 0.4	0.649 ± 0.010	1.82 ± 0.04	0.0684 ± 0.0007
MASS, β=1e−4, D	53.2 ± 0.1	0.664 ± 0.020	1.79 ± 0.02	0.0680 ± 0.0002
MASS, β=0, D	52.7 ± 0.0	0.662 ± 0.003	1.82 ± 0.02	0.0690 ± 0.0003

Does MASS Learning finally solve the mystery of why stochastic gradient descent with the cross entropy loss works so well in deep learning?

We do not believe so. Figure 6.2 shows how the values of the three terms in $\widehat{\mathcal{L}}_{MASS}$ change as the SmallMLP network trains on the CIFAR-10 dataset using either the SoftmaxCE training or MASS Learning. Despite achieving similar accuracies, the SoftmaxCE training method does not seem to be implicitly performing MASS Learning, based on the differing values of the entropy (orange) and Jacobian (green) terms between the two methods as training progresses.

6.6 Discussion

MASS Learning is a new approach to representation learning that performs well on classification accuracy, regularization, and uncertainty quantification benchmarks, despite not being directly formulated for any of these tasks. It shows particularly strong performance in improving uncertainty quantification.

Table 6.4: Uncertainty quantification metrics (proper scoring rules) on CIFAR-10 using the SmallMLP network trained on 10,000 datapoints. Test Accuracy and Entropy of the network's predictive distribution are given for reference. Full experiment details are in Section 6.8. Values are the mean over 4 training runs with different random seeds, plus or minus the standard deviation. Emboldened values are those for which the minimum observed mean value in the column was within one standard deviation. WD is weight decay; D is dropout. Lower values are better.

Method	Test Accuracy	Entropy	NLL	Brier Score
SoftmaxCE	44.6 ± 0.6	0.250 ± 0.004	5.33 ± 0.06	0.0974 ± 0.0011
SoftmaxCE, WD	36.4 ± 0.9	0.897 ± 0.033	**2.44 ± 0.11**	**0.0905 ± 0.0019**
SoftmaxCE, D	44.1 ± 0.6	0.379 ± 0.007	3.76 ± 0.04	0.0935 ± 0.0012
VIB, β=1e−1	40.6 ± 0.4	0.339 ± 0.011	4.86 ± 0.23	0.1017 ± 0.0016
VIB, β=1e−2	43.8 ± 0.8	0.274 ± 0.004	4.83 ± 0.16	0.0983 ± 0.0017
VIB, β=1e−3	44.6 ± 0.6	0.241 ± 0.004	5.50 ± 0.11	0.0983 ± 0.0005
VIB, β=1e−1, D	40.1 ± 0.5	0.541 ± 0.015	3.22 ± 0.09	0.0945 ± 0.0012
VIB, β=1e−2, D	43.9 ± 0.3	0.413 ± 0.009	3.43 ± 0.09	0.0927 ± 0.0011
VIB, β=1e−3, D	44.4 ± 0.4	0.389 ± 0.004	3.61 ± 0.06	0.0927 ± 0.0004
MASS, β=1e−2	39.9 ± 1.2	0.172 ± 0.008	10.06 ± 0.37	0.1109 ± 0.0020
MASS, β=1e−3	41.5 ± 0.7	0.197 ± 0.005	8.03 ± 0.28	0.1069 ± 0.0016
MASS, β=1e−4	41.5 ± 1.1	0.208 ± 0.008	7.55 ± 0.44	0.1054 ± 0.0023
MASS, β=0	42.0 ± 0.6	0.215 ± 0.009	7.21 ± 0.28	0.1043 ± 0.0015
MASS, β=1e−2, D	41.7 ± 0.4	0.399 ± 0.017	4.21 ± 0.17	0.0974 ± 0.0013
MASS, β=1e−3, D	43.7 ± 0.2	0.412 ± 0.010	3.71 ± 0.07	0.0930 ± 0.0006
MASS, β=1e−4, D	43.4 ± 0.5	0.435 ± 0.011	3.50 ± 0.05	0.0923 ± 0.0005
MASS, β=0, D	43.9 ± 0.4	0.447 ± 0.009	3.40 ± 0.03	**0.0913 ± 0.0008**

There are several potential ways to improve MASS Learning. Starting at the lowest level: it is likely that we did not manage to minimize $\widehat{\mathcal{L}}_{MASS}$ anywhere close to the extent possible in our experiments, given the minimal hyperparameter tuning we performed. In particular, we noticed that the initialization of the variational distribution played a large role in performance, but we were not able to fully explore it.

Moving a level higher, it may be that we are effectively minimizing $\widehat{\mathcal{L}}_{MASS}$, but that $\widehat{\mathcal{L}}_{MASS}$ is not a useful empirical approximation or upper bound to $\mathcal{L}_{MASS}$. This could be due to an insufficiently expressive variational distribution, or simply that the quantities in $\widehat{\mathcal{L}}_{MASS}$ require more data to approximate well than our datasets contained.

At higher levels still, it may be the case that the Lagrangian formulation of Theorem 2 as $\mathcal{L}_{MASS}$ is impractical for finding minimal achievable sufficient statistics. Or

Table 6.5: Uncertainty quantification metrics (proper scoring rules) on CIFAR-10 using the SmallMLP network trained on 2,500 datapoints. Test Accuracy and Entropy of the network's predictive distribution are given for reference. Full experiment details are in Section 6.8. Values are the mean over 4 training runs with different random seeds, plus or minus the standard deviation. Emboldened values are those for which the minimum observed mean value in the column was within one standard deviation. WD is weight decay; D is dropout. Lower values are better.

Method	Test Accuracy	Entropy	NLL	Brier Score
SoftmaxCE	34.2 ± 0.8	0.236 ± 0.025	8.14 ± 0.84	0.1199 ± 0.0024
SoftmaxCE, WD	23.9 ± 0.9	0.954 ± 0.017	**3.41 ± 0.07**	**0.1114 ± 0.0013**
SoftmaxCE, D	33.7 ± 1.1	0.203 ± 0.006	9.68 ± 0.06	0.1219 ± 0.0013
VIB, β=1e−1	32.2 ± 0.6	0.247 ± 0.007	8.33 ± 0.50	0.1219 ± 0.0013
VIB, β=1e−2	34.6 ± 0.4	0.249 ± 0.004	7.36 ± 0.18	0.1175 ± 0.0005
VIB, β=1e−3	35.6 ± 0.5	0.217 ± 0.008	8.03 ± 0.37	0.1175 ± 0.0012
VIB, β=1e−1, D	29.0 ± 0.6	0.383 ± 0.011	6.32 ± 0.16	0.1219 ± 0.0010
VIB, β=1e−2, D	32.5 ± 0.9	0.260 ± 0.006	7.41 ± 0.25	0.1211 ± 0.0019
VIB, β=1e−3, D	34.5 ± 1.0	0.200 ± 0.002	9.44 ± 0.16	0.1203 ± 0.0020
MASS, β=1e−2	29.6 ± 0.4	0.047 ± 0.002	57.13 ± 1.60	0.1381 ± 0.0007
MASS, β=1e−3	32.7 ± 0.8	0.048 ± 0.004	46.40 ± 3.81	0.1322 ± 0.0018
MASS, β=1e−4	34.0 ± 0.3	0.052 ± 0.002	39.10 ± 1.96	0.1293 ± 0.0009
MASS, β=0	34.1 ± 0.6	0.061 ± 0.003	33.60 ± 1.34	0.1285 ± 0.0012
MASS, β=1e−2, D	29.3 ± 1.2	0.118 ± 0.008	20.51 ± 0.83	0.1349 ± 0.0018
MASS, β=1e−3, D	31.5 ± 0.6	0.145 ± 0.004	15.65 ± 0.71	0.1289 ± 0.0010
MASS, β=1e−4, D	32.7 ± 0.8	0.185 ± 0.010	11.21 ± 0.66	0.1245 ± 0.0011
MASS, β=0, D	32.2 ± 1.1	0.217 ± 0.008	9.70 ± 0.29	0.1236 ± 0.0021

it may be that the difference between minimal and minimal achievable sufficient statistics is relevant for performance on machine learning tasks. Or it may simply be that framing machine learning as a problem of finding minimal sufficient statistics is not productive. The results of the VIB model in Chapter 5 suggest this in the case of tabular data.

Finally, while we again note that more work is needed to reduce the computational cost of our implementation of MASS Learning, we believe the concept of MASS learning, and the concepts of minimal achievability and Conserved Differential Information we introduce along with it, are beneficial to the theoretical understanding of representation learning.

Table 6.6: Uncertainty quantification metrics (proper scoring rules) on CIFAR-10 using the ResNet20 network trained on 40,000 datapoints. Test Accuracy and Entropy of the network's predictive distribution are given for reference. Full experiment details are in Section 6.8. Values are the mean over 4 training runs with different random seeds, plus or minus the standard deviation. Emboldened values are those for which the minimum observed mean value in the column was within one standard deviation. Lower values are better.

Method	Test Accuracy	Entropy	NLL	Brier Score
SoftmaxCE	81.7 ± 0.3	0.087 ± 0.002	1.45 ± 0.04	**0.0324 ± 0.0005**
VIB, β=1e−3	81.0 ± 0.3	0.089 ± 0.003	1.51 ± 0.04	0.0334 ± 0.0005
VIB, β=1e−4	81.2 ± 0.4	0.092 ± 0.002	1.46 ± 0.05	0.0331 ± 0.0007
VIB, β=1e−5	80.9 ± 0.5	0.087 ± 0.005	1.58 ± 0.08	0.0339 ± 0.0008
VIB, β=0	81.5 ± 0.2	0.079 ± 0.001	1.70 ± 0.06	0.0331 ± 0.0007
MASS, β=1e−3	75.8 ± 0.5	0.139 ± 0.003	1.66 ± 0.07	0.0417 ± 0.0011
MASS, β=1e−4	80.6 ± 0.5	0.109 ± 0.002	**1.33 ± 0.02**	0.0337 ± 0.0008
MASS, β=1e−5	81.6 ± 0.4	0.095 ± 0.003	**1.36 ± 0.03**	**0.0320 ± 0.0005**
MASS, β=0	81.5 ± 0.2	0.092 ± 0.000	1.43 ± 0.04	0.0325 ± 0.0004

Table 6.7: Uncertainty quantification metrics (proper scoring rules) on CIFAR-10 using the ResNet20 network trained on 10,000 datapoints. Test Accuracy and Entropy of the network's predictive distribution are given for reference. Full experiment details are in Section 6.8. Values are the mean over 4 training runs with different random seeds, plus or minus the standard deviation. Emboldened values are those for which the minimum observed mean value in the column was within one standard deviation. Lower values are better.

Method	Test Accuracy	Entropy	NLL	Brier Score
SoftmaxCE	67.5 ± 0.8	0.195 ± 0.011	**2.19 ± 0.06**	**0.0557 ± 0.0012**
VIB, β=1e−3	66.9 ± 1.0	0.193 ± 0.008	**2.26 ± 0.13**	0.0570 ± 0.0017
VIB, β=1e−4	66.4 ± 0.5	0.197 ± 0.009	2.30 ± 0.02	0.0577 ± 0.0007
VIB, β=1e−5	67.9 ± 0.8	0.166 ± 0.010	2.49 ± 0.13	**0.0561 ± 0.0011**
VIB, β=0	67.1 ± 1.0	0.162 ± 0.009	2.64 ± 0.11	0.0578 ± 0.0016
MASS, β=1e−3	59.6 ± 0.8	0.252 ± 0.007	2.61 ± 0.11	0.0688 ± 0.0014
MASS, β=1e−4	66.6 ± 0.4	0.209 ± 0.009	**2.18 ± 0.05**	0.0570 ± 0.0005
MASS, β=1e−5	67.4 ± 1.0	0.192 ± 0.007	**2.22 ± 0.07**	**0.0561 ± 0.0017**
MASS, β=0	67.4 ± 0.3	0.189 ± 0.004	2.30 ± 0.08	**0.0562 ± 0.0007**

Table 6.8: Uncertainty quantification metrics (proper scoring rules) on CIFAR-10 using the ResNet20 network trained on 2,500 datapoints. Test Accuracy and Entropy of the network's predictive distribution are given for reference. Full experiment details are in Section 6.8. Values are the mean over 4 training runs with different random seeds, plus or minus the standard deviation. Emboldened values are those for which the minimum observed mean value in the column was within one standard deviation. Lower values are better.

Method	Test Accuracy	Entropy	NLL	Brier Score
SoftmaxCE	50.0 ± 0.7	0.349 ± 0.005	**2.98 ± 0.06**	**0.0833 ± 0.0012**
VIB, β=1e−3	49.5 ± 1.1	0.363 ± 0.005	3.10 ± 0.11	**0.0836 ± 0.0020**
VIB, β=1e−4	49.4 ± 1.0	0.372 ± 0.016	**3.02 ± 0.10**	**0.0833 ± 0.0016**
VIB, β=1e−5	50.0 ± 1.1	0.306 ± 0.021	3.48 ± 0.15	0.0849 ± 0.0013
VIB, β=0	50.6 ± 0.8	0.271 ± 0.019	3.80 ± 0.15	0.0850 ± 0.0007
MASS, β=1e−3	38.2 ± 0.7	0.469 ± 0.012	3.75 ± 0.08	0.1010 ± 0.0017
MASS, β=1e−4	49.9 ± 1.0	0.344 ± 0.001	3.24 ± 0.08	**0.0837 ± 0.0017**
MASS, β=1e−5	50.1 ± 0.5	0.277 ± 0.008	3.81 ± 0.11	0.0859 ± 0.0005
MASS, β=0	50.2 ± 1.0	0.265 ± 0.009	3.96 ± 0.15	0.0861 ± 0.0020

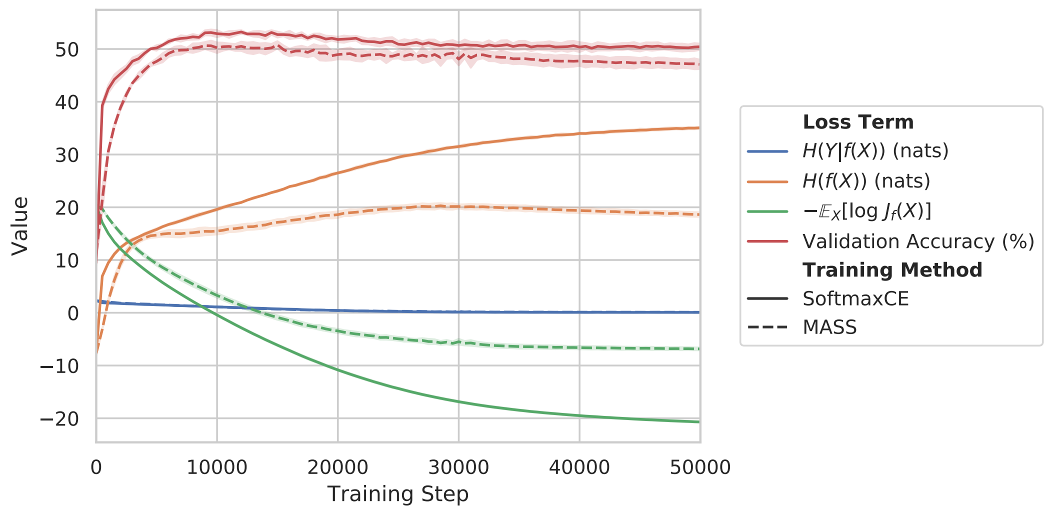


Figure 6.2: Estimated value of each term in the MASS Learning loss function, $\mathcal{L}_{MASS}(f) = H(Y|f(X)) + \beta H(f(X)) - \beta\mathbb{E}_X[\log J_f(X)]$, during training of the SmallMLP network on the CIFAR-10 dataset. The MASS training was performed with $\beta = 0.001$, though the plotted values are for the terms without being multiplied by the β coefficients. The values of these terms for SoftmaxCE training are estimated using a distribution $q_\phi(f_\theta(x)|y)$, with the distribution parameters ϕ being estimated at each training step by MLE over the training data.

6.7 Acknowledgements

We would like to thank Georg Pichler, Thomas Vidick, Alex Alemi, Alessandro Achille, and Joseph Marino for useful discussions.

6.8 Appendix: Omitted Proofs

Standard Definition of Minimal Sufficient Statistics

The most common phrasing of the definition of *minimal sufficient statistic* is:

Definition 3 (Minimal Sufficient Statistic)**.** A sufficient statistic $f(X)$ for Y is *minimal* if for any other sufficient statistic $h(X)$ there exists a measurable function g such that $f = g \circ h$ almost everywhere.

Some references do not explicitly mention the "measurability" and "almost everywhere" conditions on g, but since we are in the probabilistic setting it is this definition of $f = g \circ h$ that is meaningful.

Our preferred phrasing of the definition of *minimal sufficient statistic*, which we use in our Introduction, is:

Definition 4 (Minimal Sufficient Statistic)**.** A sufficient statistic $f(X)$ for Y is *minimal* if for any measurable function g, $g(f(X))$ is no longer sufficient for Y unless g is invertible almost everywhere (i.e. there exist a measurable function g^{-1} and a set $\mathcal{A}$ such that $g^{-1}(g(x)) = x$ for all $x \in \mathcal{A}$ and the event $\{X \in \mathcal{A}^c\}$ has probability zero).

The equivalence of Definition 3 and Definition 4 is given by the following lemma:

Lemma 2. *Assume that there exists a minimal sufficient statistic $h(X)$ for Y by Definition 3. Then a sufficient statistic $f(X)$ is minimal in the sense of Definition 3 if and only if it is minimal in the sense of Definition 4.*

Proof. We first assume that $f(X)$ is minimal in the sense of Definition 3. Let g be any measurable function such that $g(f(X))$ is sufficient for Y. By the minimality (Def. 3) of f there must exist a measurable function $\tilde{g}$ such that $\tilde{g}(g(f(x))) = f(x)$ almost everywhere. This proves that f is minimal in the sense of Definition 4.

Now assume that $f(X)$ is minimal in the sense of Definition 4 and let $\tilde{f}(X)$ be another sufficient statistic. Because h is minimal (Def. 3), there exist g_1 such that $h = g_1 \circ \tilde{f}$ almost everywhere and g_2 such that $h = g_2 \circ f$ almost everywhere. Because f is minimal (Def. 4), g_2 must be one-to-one almost everywhere, i.e. there exists a $\tilde{g}_2$ such that $\tilde{g}_2 \circ h = \tilde{g}_2 \circ g_2 \circ f = f$ almost everywhere. In turn, we obtain that $\tilde{g}_2 \circ g_1 \circ \tilde{f} = f$ almost everywhere, and since $\tilde{f}$ was arbitrary this proves the minimality of f in the sense of Definition 3. $\square$

The Mutual Information Between the Input and Output of a Deep Network is Infinite

Typically the mutual information between continuous random variables X and Y is given by

$$I(X, Y) = \int p(x, y) \log \frac{p(x, y)}{p(x)p(y)} \mathrm{d}x\mathrm{d}y,$$

but this quantity is only defined when the joint density $p(x, y)$ is integrable, which it is not in the case that $Y = f(X)$. (The technical term for $p(x, y)$ in this case is a "singular distribution".) Instead, to compute $I(X, f(X))$ we must refer to the "master definition" of mutual information (Cover and Thomas, 2006), which is

$$I(X, Y) = \sup_{\mathcal{P},\mathcal{Q}} I([X]_{\mathcal{P}}, [Y]_{\mathcal{Q}}), \tag{6.3}$$

where $\mathcal{P}$ and $\mathcal{Q}$ are finite partitions of the range of X and Y, respectively, and $[X]_{\mathcal{P}}$ is the random variable obtained by quantizing X using partition $\mathcal{P}$, and analogously for $[Y]_{\mathcal{Q}}$.

From this definition, we can prove the following Lemma:

Lemma 3. *If X and Y are continuous random variables, and there are open sets O_X and O_Y in the support of X and Y, respectively, such that $y = f(x)$ for $x \in O_X$ and $y \in O_Y$, then $I(X, Y) = \infty$.*

This includes all X and Y where $Y = f(X)$ for an f that is continuous somewhere on its domain, e.g. any deep network (considered as a function from an input vector to an output vector).

Proof. Suppose X and Y satisfy the conditions of the lemma. Let O_X and O_Y be open sets with $f(O_X) = O_Y$ and $\mathbb{P}[X \in O_X] =: \delta > 0$, which exist by the lemma's assumptions. Then let $\mathcal{P}^n_{O_Y}$ be a partition of O_Y into n disjoint sets. Because Y is continuous and hence does not have any atoms, we may assume that the probability of Y belonging to each element of $\mathcal{P}^n_{O_Y}$ is equal to the same nonzero value δ/n. Denote by $\mathcal{P}^n_{O_X}$ the partition of O_X into n disjoint sets, where each set in $\mathcal{P}^n_{O_X}$ is the preimage of one of the sets in $\mathcal{P}^n_{O_Y}$. We can construct partitions of the whole domains of X and

Y as $\mathcal{P}^n_{O_X} \cup O^c_X$ and $\mathcal{P}^n_{O_Y} \cup O^c_Y$, respectively. Using these partitions in equation 6.3, we obtain

$$\begin{aligned}
&I(X,Y)\\
&\quad \geq (1-\delta)\log(1-\delta) + \sum_{A \in [X]_{\mathcal{P}^n_{O_X}}} \mathbb{P}[X \in A, Y \in f(A)] \log \frac{\mathbb{P}[X \in A, Y \in f(A)]}{\mathbb{P}[X \in A]\mathbb{P}[Y \in f(A)]}\\
&\quad = (1-\delta)\log(1-\delta) + n\frac{\delta}{n}\log\frac{\frac{\delta}{n}}{\frac{\delta}{n}\frac{\delta}{n}}\\
&\quad = (1-\delta)\log(1-\delta) + \delta\log\frac{n}{\delta}.
\end{aligned}$$

By letting n go to infinity, we can see that the supremum in Eq. 6.3 is infinity. □

The Change of Variables Formula for Non-invertible Mappings

The change of variables formula is widely used in machine learning and is key to recent results in density estimation and generative modeling like normalizing flows (Rezende and Mohamed, 2015), NICE (Dinh, Krueger, and Y. Bengio, 2014), and Real NVP (Dinh, Sohl-Dickstein, and S. Bengio, 2017). But all uses of the change of variables formula in the machine learning literature that we are aware of use it with respect to bijective mappings between random variables, despite the formula also being applicable to non-invertible mappings between random variables. To address this gap, we offer the following brief tutorial.

The familiar form of the change of variables formula for a random variable X with density $p(x)$ and a bijective, differentiable function $f\colon \mathbb{R}^d \to \mathbb{R}^d$ is

$$\int_{\mathbb{R}^d} p(x) J_f(x)\,\mathrm{d}x = \int_{\mathbb{R}^d} p(f^{-1}(y))\,\mathrm{d}y. \tag{6.4}$$

where $J_f(x) = \left|\det \frac{\partial f(x)}{\partial x^{\mathrm{T}}}\right|$.

A slightly more general phrasing of Equation 6.4 is

$$\int_{f^{-1}(\mathcal{B})} g(x) J_f(x)\,\mathrm{d}x = \int_{\mathcal{B}} g(f^{-1}(y))\,\mathrm{d}y. \tag{6.5}$$

where $g\colon \mathbb{R}^d \to \mathbb{R}$ is any non-negative measurable function, and $\mathcal{B} \subseteq \mathbb{R}^d$ is any measurable subset of $\mathbb{R}^d$.

We can extend Equation 6.5 to work in the case that f is not invertible. To do this, we must address two issues. First, if f is not invertible, then $f^{-1}(y)$ is not a single point but rather a set. Second, if f is not invertible, then the Jacobian matrix $\frac{\partial f(x)}{\partial x^{\mathrm{T}}}$

may not be square, and thus has no well defined determinant. Both issues can be resolved and lead to the following change of variables theorem (Krantz and Parks, 2009), which is based on the so-called coarea formula (Federer, 1969).

Theorem 4. *Let $f\colon \mathbb{R}^d \to \mathbb{R}^r$ with $r \leq d$ be a differentiable function, $g\colon \mathbb{R}^d \to \mathbb{R}$ a non-negative measurable function, $\mathcal{B} \subseteq \mathbb{R}^d$ a measurable set, and $J_f(x) = \sqrt{\det\left(\frac{\partial f(x)}{\partial x^\mathrm{T}} \left(\frac{\partial f(x)}{\partial x^\mathrm{T}}\right)^\mathrm{T}\right)}$. Then*

$$\int_{f^{-1}(\mathcal{B})} g(x) J_f(x)\,\mathrm{d}x = \int_{\mathcal{B}} \int_{f^{-1}(y)} g(x)\,\mathrm{d}\mathcal{H}^{d-r}(x)\,\mathrm{d}y. \tag{6.6}$$

where $\mathcal{H}^{d-r}$ is the $(d-r)$-dimensional Hausdorff measure (one can think of this as a measure for lower-dimensional structures in high-dimensional space, e.g. the area of 2-dimensional surfaces in 3-dimensional space). [3]

We see in Theorem 4 that Equation 6.6 looks a lot like Equations 6.4 and 6.5, but with $f^{-1}(y)$ replaced by an integral over the set $f^{-1}(y)$, which for almost every y is a $(d-r)$-dimensional set. And if f in Equation 6.6 happens to be bijective, Equation 6.6 reduces to Equation 6.5.

We also see that the Jacobian determinant in Equation 6.5 was replaced by the so-called r-dimensional Jacobian

$$\sqrt{\det\left(\frac{\partial f(x)}{\partial x^\mathrm{T}} \left(\frac{\partial f(x)}{\partial x^\mathrm{T}}\right)^\mathrm{T}\right)}$$

in Equation 6.6. A word of caution is in order, as the r-dimensional Jacobian does not have the same nice properties for concatenated functions as does the Jacobian in the bijective case. In particular, we cannot calculate $J_{f_2 \circ f_1}$ based on the values of J_{f_1} and J_{f_2} because the product $\frac{\partial f_2(x)}{\partial x^\mathrm{T}} \frac{\partial f_1(x)}{\partial x^\mathrm{T}} \left(\frac{\partial f_2(x)}{\partial x^\mathrm{T}} \frac{\partial f_1(x)}{\partial x^\mathrm{T}}\right)^\mathrm{T}$ does not decompose into a product of $\frac{\partial f_2(x)}{\partial x^\mathrm{T}} \left(\frac{\partial f_2(x)}{\partial x^\mathrm{T}}\right)^\mathrm{T}$ and $\frac{\partial f_1(x)}{\partial x^\mathrm{T}} \left(\frac{\partial f_1(x)}{\partial x^\mathrm{T}}\right)^\mathrm{T}$. In other words, the trick used in techniques like normalizing flows and NICE to compute determinants of deep networks for use in the change of variables formula by decomposing the network's Jacobian into the product of layerwise Jacobians does not work straightforwardly in the case of non-invertible mappings.

[3] In what follows, we will sometimes replace g by g/J_f such that the Jacobian appears on the right-hand side. Furthermore, we will not only use non-negative g. This can be justified by splitting g into positive and negative parts provided that either part results in a finite integral.

Motivation for Conserved Differential Information

First, we present an alternative definition of conditional entropy that is meaningful for singular distributions (e.g. the joint distribution $p(X, f(X))$ for a function f). More information on this definition can be found in (Koliander et al., 2016).

Singular Conditional Entropy

Assume that the random variable X has a probability density function $p_X(x)$ on $\mathbb{R}^d$. For a given differentiable function $f\colon \mathbb{R}^d \to \mathbb{R}^r$ ($r \leq d$), we want to analyze the conditional differential entropy $H(X|f(X))$. Following (Koliander et al., 2016), we define this quantity as:

$$H(X|f(X)) = \\ -\int_{\mathbb{R}^r} p_{f(X)}(y) \int_{f^{-1}(y)} \theta^{d-r}_{\Pr\{X\in\cdot|f(X)=y\}}(x) \log\left(\theta^{d-r}_{\Pr\{X\in\cdot|f(X)=y\}}(x)\right) \mathrm{d}\mathscr{H}^{d-r}(x)\,\mathrm{d}y \tag{6.7}$$

where $\mathscr{H}^{d-r}$ denotes $(d-r)$-dimensional Hausdorff measure. The function $p_{f(X)}$ is the probability density function of the random variable $f(X)$. Although $\theta^{d-r}_{\Pr\{X\in\cdot|f(X)=y\}}$ can also be interpreted as a probability density, it is not the commonly used density with respect to Lebesgue measure (which does not exist for $X|f(X) = y$) but a density with respect to a lower-dimensional Hausdorff measure. We will analyze the two functions $p_{f(X)}$ and $\theta^{d-r}_{\Pr\{X\in\cdot|f(X)=y\}}$ in more detail. The density $p_{f(X)}$ is defined by the relation

$$\int_{f^{-1}(\mathcal{B})} p_X(x)\,\mathrm{d}x = \int_{\mathcal{B}} p_{f(X)}(y)\,\mathrm{d}y, \tag{6.8}$$

which has to hold for every measurable set $\mathcal{B} \subseteq \mathbb{R}^r$. Using the coarea formula (or the related change-of-variables theorem), we see that

$$\int_{f^{-1}(\mathcal{B})} p_X(x)\,\mathrm{d}x = \int_{\mathcal{B}} \int_{f^{-1}(y)} \frac{p_X(x)}{J_f(x)}\,\mathrm{d}\mathscr{H}^{d-r}(x)\,\mathrm{d}y, \tag{6.9}$$

where $J_f(x) = \sqrt{\det\left(\frac{\partial f(x)}{\partial x^{\mathrm{T}}}\left(\frac{\partial f(x)}{\partial x^{\mathrm{T}}}\right)^{\mathrm{T}}\right)}$ is the r-dimensional Jacobian determinant. Thus, we identified

$$p_{f(X)}(y) = \int_{f^{-1}(y)} \frac{p_X(x)}{J_f(x)}\,\mathrm{d}\mathscr{H}^{d-r}(x). \tag{6.10}$$

The second function, namely $\theta^{d-r}_{\Pr\{X\in\cdot|f(X)=y\}}$, is the Radon-Nikodym derivative of the conditional probability $\Pr\{X \in \cdot|f(X) = y\}$ with respect to $\mathscr{H}^{d-r}$ restricted to the set where $X|f(X) = y$ has positive probability (in the end, this will be the set $f^{-1}(y)$). To understand this function, we have to know something about the conditional distribution of X given $f(X)$. Formally, a (regular) conditional probability $\Pr\{X \in \cdot|f(X) = y\}$ has to satisfy three conditions:

- $\Pr\{X \in \cdot|f(X) = y\}$ is a probability measure for each fixed $y \in \mathbb{R}^r$.
- $\Pr\{X \in \mathcal{A}|f(X) = \cdot\}$ is measurable for each fixed measurable set $\mathcal{A} \subseteq \mathbb{R}^d$.
- For measurable sets $\mathcal{A} \subseteq \mathbb{R}^d$ and $\mathcal{B} \subseteq \mathbb{R}^r$, we have

$$\Pr\{(X, f(X)) \in \mathcal{A} \times \mathcal{B}\} = \int_{\mathcal{B}} \Pr\{X \in \mathcal{A}|f(X) = y\} p_{f(X)}(y)\,\mathrm{d}y\,. \tag{6.11}$$

In our setting, equation 6.11 becomes

$$\int_{\mathcal{A}\cap f^{-1}(\mathcal{B})} p_X(x)\,\mathrm{d}x = \int_{\mathcal{B}} \Pr\{X \in \mathcal{A}|f(X) = y\} p_{f(X)}(y)\,\mathrm{d}y\,. \tag{6.12}$$

Choosing

$$\Pr\{X \in \mathcal{A}|f(X) = y\} = \frac{1}{p_{f(X)}(y)} \int_{\mathcal{A}\cap f^{-1}(y)} \frac{p_X(x)}{J_f(x)}\,\mathrm{d}\mathcal{H}^{d-r}(x)\,, \tag{6.13}$$

the right-hand side in equation 6.12 becomes

$$\begin{aligned}\int_{\mathcal{B}} \Pr\{X \in \mathcal{A}|f(X) = y\} p_{f(X)}(y)\,\mathrm{d}y &= \int_{\mathcal{B}} \int_{\mathcal{A}\cap f^{-1}(y)} \frac{p_X(x)}{J_f(x)}\,\mathrm{d}\mathcal{H}^{d-r}(x)\,\mathrm{d}y \\ &= \int_{\mathcal{A}\cap f^{-1}(\mathcal{B})} p_X(x)\,\mathrm{d}x\,, \end{aligned} \tag{6.14}$$

where the final equality is again an application of the coarea formula. Thus, we identified

$$\theta^{d-r}_{\Pr\{X\in\cdot|f(X)=y\}}(x) = \frac{p_X(x)}{J_f(x)\,p_{f(X)}(y)}\,. \tag{6.15}$$

Although things might seem complicated up to this point, they simplify significantly once we put everything together. In particular, inserting equation 6.15 into equation 6.7, we obtain

$$
\begin{aligned}
&H(X|f(X)) \\
&\quad = -\int_{\mathbb{R}^r} p_{f(X)}(y) \int_{f^{-1}(y)} \frac{p_X(x)}{J_f(x)\, p_{f(X)}(y)} \log\left(\frac{p_X(x)}{J_f(x)\, p_{f(X)}(y)}\right) \mathrm{d}\mathscr{H}^{d-r}(x)\, \mathrm{d}y \\
&\quad = -\int_{\mathbb{R}^r} \int_{f^{-1}(y)} \frac{p_X(x)}{J_f(x)} \log\left(\frac{p_X(x)}{J_f(x)\, p_{f(X)}(y)}\right) \mathrm{d}\mathscr{H}^{d-r}(x)\, \mathrm{d}y \\
&\quad = -\int_{\mathbb{R}^d} p_X(x) \log\left(\frac{p_X(x)}{J_f(x)\, p_{f(X)}(f(x))}\right) \mathrm{d}x \qquad (6.16) \\
&\quad = H(X) + \int_{\mathbb{R}^d} p_X(x) \log\big(J_f(x) p_{f(X)}(f(x))\big)\, \mathrm{d}x \\
&\quad = H(X) + \int_{\mathbb{R}^d} p_X(x) \log\big(p_{f(X)}(f(x))\big)\, \mathrm{d}x + \int_{\mathbb{R}^d} p_X(x) \log\big(J_f(x)\big)\, \mathrm{d}x \\
&\quad = H(X) + \int_{\mathbb{R}^r} \int_{f^{-1}(y)} \frac{p_X(x)}{J_f(x)} \log\big(p_{f(X)}(f(x))\big)\, \mathrm{d}\mathscr{H}^{d-r}(x)\, \mathrm{d}y + \mathbb{E}\big[\log\big(J_f(X)\big)\big] \\
&\qquad (6.17) \\
&\quad = H(X) + \int_{\mathbb{R}^r} \int_{f^{-1}(y)} \frac{p_X(x)}{J_f(x)}\, \mathrm{d}\mathscr{H}^{d-r}(x) \log\big(p_{f(X)}(y)\big)\, \mathrm{d}y + \mathbb{E}\big[\log\big(J_f(X)\big)\big] \\
&\quad = H(X) + \int_{\mathbb{R}^r} p_{f(X)}(y) \log\big(p_{f(X)}(y)\big)\, \mathrm{d}y + \mathbb{E}\big[\log\big(J_f(X)\big)\big] \\
&\quad = H(X) - H(f(X)) + \mathbb{E}\big[\log\big(J_f(X)\big)\big] \qquad (6.18)
\end{aligned}
$$

where equation 6.16 and equation 6.17 hold by the coarea formula.

So, altogether we have that for a random variable X and a function f, the singular conditional entropy between X and $f(X)$ is

$$H(X|f(X)) = H(X) - H(f(X)) + \mathbb{E}\big[\log\big(J_f(X)\big)\big]. \qquad (6.19)$$

This quantity can loosely be interpreted as being the difference in differential entropies between X and $f(X)$ but with an additional term that corrects for any “uninformative” scaling that f does.

Conserved Differential Information

For random variables that are not related by a deterministic function, mutual information can be expanded as

$$I(X,Y) = H(X) - H(X|Y) \qquad (6.20)$$

where $H(X)$ and $H(X|Y)$ are differential entropy and conditional differential entropy, respectively. As we would like to measure information between random variables that are deterministically dependent, we can mimic this behavior by defining for a Lipschitz continuous mapping f:

$$C(X, f(X)) := H(X) - H(X|f(X)) . \tag{6.21}$$

By equation 6.18, this can be simplified to

$$C(X, f(X)) = H(f(X)) - \mathbb{E}\big[\log\big(J_f(X)\big)\big] \tag{6.22}$$

yielding our definition of CDI.

Proof of CDI Data Processing Inequality

CDI Data Processing Inequality (Theorem 1)

For Lipschitz continuous functions f and g with the same output space,

$$C(X, f(X)) \geq C(X, g(f(X))$$

with equality if and only if g is one-to-one almost everywhere.

Proof. We calculate the difference between $C(X, f(X))$ and $C(X, g(f(X)))$.

$$\begin{aligned}
&C(X, f(X)) - C(X, g(f(X))) && (6.23)\\
&\quad = H(f(X)) - \mathbb{E}_X\big[\log J_f(X)\big] - H(g(f(X))) + \mathbb{E}_X\big[\log J_{g\circ f}(X)\big]\\
&\quad = H(f(X)) - H(g(f(X))) + \mathbb{E}_X\left[\log \frac{J_g(f(X))J_f(X)}{J_f(X)}\right] && (6.24)\\
&\quad = -\mathbb{E}_X[\log p_{f(X)}(f(X))] + \mathbb{E}_X\left[\log \sum_{z\in g^{-1}(g(f(X)))} \frac{p_{f(X)}(f(z))}{J_g(f(z))}\right] + \mathbb{E}_X[\log J_g(f(X))]\\
& && (6.25)\\
&\quad = \mathbb{E}_X\left[\log\left(\frac{\sum_{z\in g^{-1}(g(f(X)))}\frac{p_{f(X)}(f(z))}{J_g(f(z))}}{\frac{p_{f(X)}(f(X))}{J_g(f(X))}}\right)\right] && (6.26)
\end{aligned}$$

where equation 6.24 holds because the Jacobian determinant $J_{g\circ f}$ can be decomposed as g has the same domain and codomain and equation 6.25 holds because the probability density function of $g(f(X))$ can be calculated as $p_{g(f(X))}(z) = \sum_{z\in g^{-1}(g(f(X)))}\frac{p_{f(X)}(f(z))}{J_g(f(z))}$ using a change of variables argument. The resulting term in equation 6.26 is clearly always nonnegative which proves the inequality.

To prove the equality statement, we first assume that equation 6.26 is zero. In this case, $\sum_{z\in g^{-1}(g(f(x)))}\frac{p_{f(X)}(f(z))}{J_g(f(z))} = \frac{p_{f(X)}(f(x))}{J_g(f(x))}$ almost everywhere. Of course, we also have that $p_{f(X)}(f(x)) > 0$ almost everywhere. Thus, there exists a set $\mathcal{A}$ of probability one such that $\sum_{z\in g^{-1}(g(f(x)))} \frac{p_{f(X)}(f(z))}{J_g(f(z))} = \frac{p_{f(X)}(f(x))}{J_g(f(x))}$ and $p_{f(X)}(f(x)) > 0$ for all $x \in \mathcal{A}$. In particular, the set $g^{-1}(g(f(x))) \cap \mathcal{A} = \{f(x)\}$ and hence g is one-to-one almost everywhere.

For the other direction, assume that there exists $\tilde{g}$ such that $\tilde{g}(g(f(x))) = f(x)$ almost everywhere. We can assume without loss of generality that $p_{f(X)}(f(x)) = 0$ for all x that do not satisfy this equation. Restricting the expectation in equation 6.26 to the values that satisfy $\tilde{g}(g(f(x))) = f(x)$ does not change the expectation and gives the value zero. $\square$

Theorem 3 Only Holds in the Reverse Direction for Continuous X

The specific claim we are making is as follows:

Theorem 5. *Let X be a continuous random variable drawn according to a distribution $p(X|Y)$ determined by the discrete random variable Y. Let $\mathcal{F}$ be the set of measurable functions of X to any target space. If $f(X)$ is a minimal sufficient statistic of X for Y then*

$$\begin{aligned} f \in \arg\min_{S\in\mathcal{F}} \; & I(X, S(X)) \\ s.t. \;\; & I(S(X), Y) = \max_{S'\in\mathcal{F}} I(S'(X), Y). \end{aligned} \tag{6.27}$$

However, there may exist a function f satisfying equation 6.27 such that $f(X)$ is not a minimal sufficient statistic.

Proof. First, we prove the forward direction. According to Lemma 1, $Z = f(X)$ is a sufficient statistic for Y if and only if $I(Z, Y) = I(X, Y) = \max_{S'} I(S'(X), Y)$. To show the minimality condition in equation 6.27 for a minimal sufficient statistic, assume that there exists $S(X)$ such that $I(S(X), Y) = \max_{S'\in\mathcal{F}} I(S'(X), Y)$ and $I(X, S(X)) < I(X, f(X))$. Because f is assumed to be a minimal sufficient statistic, there exists g such that $f(X) = g(S(X))$ and by the data-processing inequality $I(X, S(X)) \geq I(X, f(X))$, a contradiction.

Next, we give an example of a function satisfying equation 6.27 such that $f(X)$ is not a minimal sufficient statistic. The example is the case when $I(X, f(X))$ is not finite, as is the case when f is a deterministic function and X is continuous. (See Lemma 3.) In this case, $I(X, S(X))$ is infinite for all deterministic, sufficient statistics S. Thus the set $\arg\min_S I(X, S(X))$ contains not only the minimal sufficient statistics, but all deterministic sufficient statistics. As a concrete example, consider two i.i.d. normally-distributed random variables with mean μ: $X = (X_1, X_2) \sim \mathcal{N}(\mu, 1)$. $T(X) = \frac{X_1+X_2}{2}$ is a minimal sufficient statistic for μ. $T'(X) = (\frac{X_1+X_2}{2}, X_1 \cdot X_2)$ is a non-minimal sufficient statistic for μ. However, both statistics satisfy $T, T' \in \arg\min_{S \in \mathcal{F}} I(X, S(X))$ since $\min_{S \in \mathcal{F}} I(X, S(X)) = \infty$ under the constraint $I(S(X), Y) = \max_{S' \in \mathcal{F}} I(S'(X), Y)$. $\square$

Experiment Details

Code to reproduce all experiments is available online at `https://github.com/mwcvitkovic/MASS-Learning`.

Data

In all experiments above, the models were trained on the CIFAR-10 dataset (Krizhevsky, 2009). In the out-of-distribution detection experiments, the SVHN dataset (Netzer et al., 2011) was used as the out-of-distribution dataset. All channels in all datapoints were normalized to have zero mean and unit variance across their dataset. No data augmentation was used in any experiments.

Networks

The SmallMLP network is a 2-hidden-layer, fully-connected network with `elu` nonlinearities (Clevert, Unterthiner, and Hochreiter, 2015). The first hidden layer contains 400 hidden units; the second contains 200 hidden units. Batch norm was applied after the linear mapping and before the nonlinearity of each hidden layer. Dropout, when used, was applied after the nonlinearity of each hidden layer. When used in VIB and MASS, the representation $f_\theta(x)$ was in $\mathbb{R}^{15}$, with the VIB encoder outputting parameters for a fully-covariant Gaussian distribution in $\mathbb{R}^{15}$. The marginal distribution in VIB and each component of the variational distribution q_ϕ (one component for each possible output class) in MASS were both mixtures of 10 full-covariance, 15-dimensional multivariate Gaussians.

The ResNet20 network is the 20-layer residual net of (He et al., 2016). We adapted our implementation from `https://github.com/akamaster/pytorch_resnet_cifar10`, to whose authors we are very grateful. When used in VIB and MASS, the representation $f_\theta(x)$ was in $\mathbb{R}^{20}$, with the VIB encoder outputting parameters for a diagonally-covariant Gaussian distribution in $\mathbb{R}^{20}$. The marginal distribution in VIB and each component of the the variational distribution q_ϕ (one component for each possible output class) in MASS were both mixtures of 10 full-covariance, 20-dimensional multivariate Gaussians.

In experiments where a distribution $q_\phi(f_\theta(x)|y)$ is used in conjunction with a function f_θ trained by SoftmaxCE, each component of $q_\phi(f_\theta(x)|y)$ was a mixture of 10 full-covariance, 10-dimensional multivariate Gaussians, the parameters ϕ of which were estimated by MLE on the training set.

Training

The SmallMLP network in all experiments and with all training methods was trained using the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 0.0005 for 100,000 steps of stochastic gradient descent, using minibatches of size 256. All quantities we report in this chapter were fully-converged to stable values by 100,000 steps. When training VIB, 5 encoder samples per datapoint were used during training, and 10 during testing. When training MASS, the learning rate of the parameters of the variational distribution q_ϕ was set at 2.5e$-$5 to aid numerical stability.

The ResNet20 network in all experiments and with all training methods was trained using SGD with an initial learning rate of 0.1, decayed by a multiplicative factor of 0.1 at epochs 100 and 150, a momentum factor of 0.9, and minibatches of size 128. These values were taken directly from the original paper (He et al., 2016). However, unlike the original paper, we did not use data augmentation in order to keep the comparison between different numbers of training points more rigorous. This, combined with the smaller number of training points used, accounts for the around 82% accuracy we observe on CIFAR-10 compared to the around 91% accuracy in the original paper. We trained the network for 70,000 steps of stochastic gradient descent. All quantities we report in this chapter were fully-converged to stable values by 70,000 steps. When training VIB, 10 encoder samples per datapoint were used during training, and 20 during testing. When training MASS, the learning rate of the parameters of the variational distribution was the same as those of the network.

The values of β we chose for VIB and MASS were selected so that the largest β value used in each experiment was much larger in magnitude than the remaining terms in the VIB or MASS training loss, and the smallest β value used was much smaller than the remaining terms. We made this choice in the hope of clearly observing the effect of the β parameter and more fairly comparing SoftmaxCE, VIB, and MASS. But we note that a finer-tuning of the β parameter would likely result in better performance for both VIB and MASS. We also note that the reason we omit a $\beta = 0$ run for VIB with the SmallMLP network was that we could not prevent training from failing due to numerical instability with $\beta = 0$ with this network.

References

Achille, Alessandro and Stefano Soatto (2018a). “Emergence of Invariance and Disentanglement in Deep Representations”. In: *2018 Information Theory and Applications Workshop (ITA)*, pp. 1–9.

– (2018b). “Information Dropout: Learning Optimal Representations Through Noisy Computation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 2897–2905.

Adragni, Kofi P. and Cook, R. Dennis (2009). “Sufficient dimension reduction and prediction in regression”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 367.1906, pp. 4385–4405. DOI: 10.1098/rsta.2009.0110. URL: https://royalsocietypublishing.org/doi/full/10.1098/rsta.2009.0110 (visited on 01/16/2019).

Alemi, Alexander A., Ian Fischer, and Joshua V. Dillon (2018). “Uncertainty in the Variational Information Bottleneck”. In: *arXiv:1807.00906 [cs, stat]*. arXiv: 1807.00906. URL: http://arxiv.org/abs/1807.00906 (visited on 01/09/2019).

Alemi, Alexander A., Ian Fischer, Joshua V. Dillon, and Kevin Murphy (2017). “Deep Variational Information Bottleneck”. In: *International Conference on Learning Representations* abs/1612.00410. URL: https://arxiv.org/abs/1612.00410.

Amjad, Rana Ali and Bernhard C. Geiger (2018). “Learning Representations for Neural Network-Based Classification Using the Information Bottleneck Principle.” In: *IEEE transactions on pattern analysis and machine intelligence*.

Bell, A. J. and T. J. Sejnowski (1995). “An information-maximization approach to blind separation and blind deconvolution”. eng. In: *Neural Computation* 7.6, pp. 1129–1159. ISSN: 0899-7667.

Clevert, Djork-Arné, Thomas Unterthiner, and Sepp Hochreiter (2015). “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)”. In: *arXiv:1511.07289 [cs]*. arXiv: 1511.07289. URL: http://arxiv.org/abs/1511.07289 (visited on 05/13/2019).

Cover, Thomas M. and Joy A. Thomas (2006). *Elements of Information Theory 2nd Edition*. English. 2 edition. Hoboken, NJ: Wiley-Interscience. ISBN: 978-0-471-24195-9.

Dinh, Laurent, David Krueger, and Yoshua Bengio (2014). “NICE: Non-linear Independent Components Estimation”. In: *arXiv:1410.8516 [cs]*. arXiv: 1410.8516. URL: http://arxiv.org/abs/1410.8516 (visited on 11/10/2017).

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio (2017). “Density estimation using Real NVP”. en. In: *International Conference on Learning Representations*. URL: `https://arxiv.org/abs/1605.08803` (visited on 12/17/2018).

Dynkin, E. B. (1951). “Necessary and sufficient statistics for a family of probability distributions”. In: *Uspekhi Mat. Nauk* 6.1, pp. 68–90. URL: `http://www.mathnet.ru/php/archive.phtml?wshow=paper&jrnid=rm&paperid=6820&option_lang=eng` (visited on 01/18/2019).

Federer, H. (1969). *Geometric Measure Theory*. New York, NY: Springer.

Goldfeld, Ziv et al. (2018). “Estimating Information Flow in Neural Networks”. In: *arXiv:1810.05728 [cs, stat]*. arXiv: 1810.05728. URL: `http://arxiv.org/abs/1810.05728` (visited on 12/12/2018).

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press.

He, Kaiming et al. (2016). “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

Hendrycks, Dan and Kevin Gimpel (2017). “A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks”. en. In: *International Conference on Learning Representations*. URL: `https://arxiv.org/abs/1610.02136v3` (visited on 01/09/2019).

Jakubovitz, Daniel and Raja Giryes (2018). “Improving DNN Robustness to Adversarial Attacks Using Jacobian Regularization”. In: *ECCV*. URL: `https://arxiv.org/abs/1803.08680`.

James, Ryan G., John R. Mahoney, and James P. Crutchfield (2017). “Trimming the Independent Fat: Sufficient Statistics, Mutual Information, and Predictability from Effective Channel States”. In: *Physical Review E* 95.6. arXiv: 1702.01831. ISSN: 2470-0045, 2470-0053. DOI: `10.1103/PhysRevE.95.060102`. URL: `http://arxiv.org/abs/1702.01831` (visited on 12/17/2018).

Kingma, Diederik and Jimmy Ba (2015). “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations*. URL: `http://arxiv.org/abs/1412.6980` (visited on 11/19/2016).

Kolchinsky, Artemy, Brendan D. Tracey, and Steven Van Kuyk (2019). “Caveats for information bottleneck in deterministic scenarios”. In: *International Conference on Learning Representations*. arXiv: 1808.07593. URL: `http://arxiv.org/abs/1808.07593` (visited on 12/16/2018).

Kolchinsky, Artemy, Brendan D. Tracey, and David H. Wolpert (2017). “Nonlinear Information Bottleneck”. In: *arXiv:1705.02436 [cs, math, stat]*. arXiv: 1705.02436. URL: `http://arxiv.org/abs/1705.02436` (visited on 01/15/2018).

Koliander, Günther et al. (2016). "Entropy and Source Coding for Integer-Dimensional Singular Random Variables". In: *IEEE Transactions on Information Theory* 62.11. arXiv: 1505.03337, pp. 6124–6154. ISSN: 0018-9448, 1557-9654. DOI: `10.1109/TIT.2016.2604248`. URL: `http://arxiv.org/abs/1505.03337` (visited on 03/12/2018).

Krantz, Steven G. and Harold R. Parks (2009). *Geometric Integration Theory*. Basel, Switzerland: Birkhäuser.

Krizhevsky, Alex (2009). "Learning Multiple Layers of Features from Tiny Images". In:

Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell (2017). "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles". In: *NIPS*.

Lehmann, E. L. and Henry Scheffe (1950). "Completeness, Similar Regions, and Unbiased Estimation: Part I". In: *Sankhyā: The Indian Journal of Statistics (1933-1960)* 10.4, pp. 305–340. ISSN: 0036-4452. URL: `https://www.jstor.org/stable/25048038` (visited on 01/18/2019).

Nash, Charlie, Nate Kushman, and Christopher K. I. Williams (2018). "Inverting Supervised Representations with Autoregressive Neural Density Models". en. In: URL: `https://arxiv.org/abs/1806.00400` (visited on 12/16/2018).

Netzer, Yuval et al. (2011). "Reading Digits in Natural Images with Unsupervised Feature Learning". In:

Paszke, Adam et al. (2017). "Automatic differentiation in PyTorch". In: *NIPS-W*.

Rezende, Danilo Jimenez and Shakir Mohamed (2015). "Variational Inference with Normalizing Flows". In: *International Conference on Machine Learning*. URL: `https://arxiv.org/abs/1505.05770`.

Rifai, Salah et al. (2011). "Higher Order Contractive Auto-Encoder". en. In: *Machine Learning and Knowledge Discovery in Databases*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 645–660. ISBN: 978-3-642-23783-6.

Ross, Andrew Slavin and Finale Doshi-Velez (2018). "Improving the Adversarial Robustness and Interpretability of Deep Neural Networks by Regularizing Their Input Gradients". In: *AAAI*. URL: `https://arxiv.org/abs/1711.09404`.

Saxe, Andrew Michael et al. (2018). "On the Information Bottleneck Theory of Deep Learning". In: *International Conference on Learning Representations*. URL: `https://openreview.net/forum?id=ry_WPG-A-` (visited on 12/11/2018).

"Sensitivity and Generalization in Neural Networks" (2018). "Sensitivity and Generalization in Neural Networks: an Empirical Study". In: *International Conference on Learning Representations*. URL: `http://arxiv.org/abs/1802.08760` (visited on 12/17/2018).

Shamir, Ohad, Sivan Sabato, and Naftali Tishby (2010). “Learning and generalization with the information bottleneck”. In: *Theoretical Computer Science*. Algorithmic Learning Theory (ALT 2008) 411.29, pp. 2696–2711. ISSN: 0304-3975. DOI: `10.1016/j.tcs.2010.04.006`. URL: `http://www.sciencedirect.com/science/article/pii/S030439751000201X` (visited on 12/11/2018).

Shwartz-Ziv, Ravid and Naftali Tishby (2017). “Opening the Black Box of Deep Neural Networks via Information”. In: *arXiv:1703.00810 [cs]*. URL: `http://arxiv.org/abs/1703.00810`.

Sokolic, Jure et al. (2017). “Robust Large Margin Deep Neural Networks”. In: *IEEE Transactions on Signal Processing* 65.16. arXiv: 1605.08254, pp. 4265–4280. ISSN: 1053-587X, 1941-0476. DOI: `10.1109/TSP.2017.2708039`. URL: `http://arxiv.org/abs/1605.08254` (visited on 12/17/2018).

Strouse, Daniel and David J. Schwab (2015). “The Deterministic Information Bottleneck”. In: *Neural Computation* 29, pp. 1611–1630. URL: `https://arxiv.org/abs/1604.00268`.

Tishby, Naftali, Fernando C. Pereira, and William Bialek (2000). “The information bottleneck method”. In: *arXiv:physics/0004057*. arXiv: physics/0004057. URL: `http://arxiv.org/abs/physics/0004057` (visited on 12/11/2018).

Tishby, Naftali and Noga Zaslavsky (2015). “Deep Learning and the Information Bottleneck Principle”. en. In: *2015 IEEE Information Theory Workshop (ITW)*, pp. 1–5. URL: `https://arxiv.org/abs/1503.02406` (visited on 12/12/2018).

Varga, Dániel, Adrián Csiszárik, and Zsolt Zombori (2017). “Gradient Regularization Improves Accuracy of Discriminative Models”. In: *arXiv:1712.09936 [cs]*. arXiv: 1712.09936. URL: `http://arxiv.org/abs/1712.09936` (visited on 12/17/2018).

Table 6.9: Out-of-distribution detection metrics for SmallMLP network trained on 40,000 CIFAR-10 images, with SVHN as the out-of-distribution examples. Full experiment details are in Section 6.8. Values are the mean over 4 training runs with different random seeds, plus or minus the standard deviation. Emboldened values are those for which the maximum observed mean value in the column was within one standard deviation. WD is weight decay; D is dropout. Larger values are better.

Training Method	Test Acc	Detection	AUROC	APR In	APR Out
SoftmaxCE	52.7 ± 0.4	Entropy	0.65 ± 0.01	0.68 ± 0.01	0.61 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.38 ± 0.01	0.42 ± 0.01	0.43 ± 0.01
SoftmaxCE, WD	48.1 ± 0.1	Entropy	0.65 ± 0.01	0.69 ± 0.01	0.59 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.43 ± 0.01	0.43 ± 0.01	0.48 ± 0.02
SoftmaxCE, D	53.7 ± 0.3	Entropy	0.71 ± 0.01	**0.75 ± 0.01**	0.65 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.33 ± 0.00	0.39 ± 0.00	0.40 ± 0.00
VIB, β=1e−1	46.1 ± 0.5	Entropy	0.62 ± 0.01	0.66 ± 0.01	0.57 ± 0.01
		Rate	0.47 ± 0.02	0.49 ± 0.01	0.46 ± 0.01
VIB, β=1e−2	51.9 ± 0.8	Entropy	0.64 ± 0.01	0.67 ± 0.01	0.59 ± 0.01
		Rate	0.58 ± 0.03	0.59 ± 0.02	0.55 ± 0.02
VIB, β=1e−3	51.8 ± 0.8	Entropy	0.65 ± 0.00	0.67 ± 0.01	0.61 ± 0.00
		Rate	0.52 ± 0.03	0.54 ± 0.03	0.50 ± 0.03
VIB, β=1e−1, D	49.5 ± 0.5	Entropy	0.68 ± 0.01	**0.74 ± 0.01**	0.60 ± 0.01
		Rate	0.34 ± 0.01	0.40 ± 0.01	0.39 ± 0.00
VIB, β=1e−2, D	53.6 ± 0.3	Entropy	0.69 ± 0.02	0.73 ± 0.01	0.62 ± 0.02
		Rate	0.50 ± 0.03	0.51 ± 0.02	0.51 ± 0.03
VIB, β=1e−3, D	54.3 ± 0.2	Entropy	0.69 ± 0.01	0.73 ± 0.01	0.62 ± 0.01
		Rate	0.45 ± 0.01	0.45 ± 0.01	0.49 ± 0.01
MASS, β=1e−2	46.3 ± 1.2	Entropy	0.64 ± 0.01	0.67 ± 0.01	0.61 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.51 ± 0.03	0.56 ± 0.05	0.49 ± 0.01
MASS, β=1e−3	47.8 ± 0.8	Entropy	0.63 ± 0.02	0.65 ± 0.02	0.60 ± 0.02
		$\max_i q_\phi(\cdot\|y_i)$	0.63 ± 0.07	0.64 ± 0.08	0.60 ± 0.05
MASS, β=1e−4	47.9 ± 0.8	Entropy	0.63 ± 0.02	0.65 ± 0.02	0.60 ± 0.02
		$\max_i q_\phi(\cdot\|y_i)$	0.57 ± 0.06	0.58 ± 0.05	0.56 ± 0.05
MASS, β=0	48.2 ± 0.9	Entropy	0.63 ± 0.02	0.65 ± 0.02	0.59 ± 0.02
		$\max_i q_\phi(\cdot\|y_i)$	0.58 ± 0.06	0.58 ± 0.05	0.56 ± 0.05
MASS, β=1e−2,D	52.0 ± 0.6	Entropy	**0.73 ± 0.01**	**0.75 ± 0.01**	**0.67 ± 0.01**
		$\max_i q_\phi(\cdot\|y_i)$	0.65 ± 0.06	0.70 ± 0.06	0.58 ± 0.05
MASS, β=1e−3,D	53.1 ± 0.4	Entropy	**0.71 ± 0.02**	0.73 ± 0.01	0.64 ± 0.02
		$\max_i q_\phi(\cdot\|y_i)$	0.64 ± 0.10	0.66 ± 0.10	0.60 ± 0.09
MASS, β=1e−4,D	53.2 ± 0.1	Entropy	**0.73 ± 0.01**	**0.75 ± 0.01**	**0.67 ± 0.01**
		$\max_i q_\phi(\cdot\|y_i)$	0.65 ± 0.09	0.65 ± 0.08	0.61 ± 0.08
MASS, β=0,D	52.7 ± 0.0	Entropy	**0.71 ± 0.02**	**0.74 ± 0.01**	**0.65 ± 0.02**
		$\max_i q_\phi(\cdot\|y_i)$	0.63 ± 0.09	0.65 ± 0.08	0.59 ± 0.09

Table 6.10: Out-of-distribution detection metrics for SmallMLP network trained on 10,000 CIFAR-10 images, with SVHN as the out-of-distribution examples. Full experiment details are in Section 6.8. Values are the mean over 4 training runs with different random seeds, plus or minus the standard deviation. Emboldened values are those for which the maximum observed mean value in the column was within one standard deviation. WD is weight decay; D is dropout. Larger values are better.

Training Method	Test Acc	Detection	AUROC	APR In	APR Out
SoftmaxCE	44.6 ± 0.6	Entropy	0.62 ± 0.00	0.64 ± 0.01	0.59 ± 0.00
		$\max_i q_\phi(\cdot\|y_i)$	0.36 ± 0.01	0.40 ± 0.01	0.42 ± 0.00
SoftmaxCE, WD	36.4 ± 0.9	Entropy	0.62 ± 0.02	0.62 ± 0.02	0.60 ± 0.02
		$\max_i q_\phi(\cdot\|y_i)$	0.30 ± 0.01	0.37 ± 0.00	0.39 ± 0.01
SoftmaxCE, D	44.1 ± 0.6	Entropy	0.66 ± 0.01	**0.69 ± 0.01**	0.62 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.29 ± 0.01	0.37 ± 0.00	0.38 ± 0.00
VIB, β=1e−1	40.6 ± 0.4	Entropy	0.60 ± 0.01	0.64 ± 0.01	0.56 ± 0.01
		Rate	0.50 ± 0.02	0.52 ± 0.02	0.48 ± 0.01
VIB, β=1e−2	43.8 ± 0.8	Entropy	0.62 ± 0.00	0.64 ± 0.01	0.59 ± 0.01
		Rate	0.55 ± 0.03	0.57 ± 0.02	0.53 ± 0.02
VIB, β=1e−3	44.6 ± 0.6	Entropy	0.62 ± 0.01	0.64 ± 0.01	0.59 ± 0.01
		Rate	0.49 ± 0.04	0.52 ± 0.04	0.48 ± 0.03
VIB, β=1e−1, D	40.1 ± 0.5	Entropy	0.62 ± 0.00	0.65 ± 0.01	0.57 ± 0.00
		Rate	0.49 ± 0.02	0.51 ± 0.02	0.48 ± 0.01
VIB, β=1e−2, D	43.9 ± 0.3	Entropy	**0.67 ± 0.01**	**0.69 ± 0.01**	0.62 ± 0.00
		Rate	0.60 ± 0.02	0.61 ± 0.02	0.56 ± 0.01
VIB, β=1e−3, D	44.4 ± 0.4	Entropy	**0.67 ± 0.01**	**0.69 ± 0.01**	0.63 ± 0.01
		Rate	0.50 ± 0.03	0.53 ± 0.03	0.49 ± 0.02
MASS, β=1e−2	39.9 ± 1.2	Entropy	0.63 ± 0.02	0.64 ± 0.02	0.60 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.54 ± 0.03	0.58 ± 0.04	0.50 ± 0.02
MASS, β=1e−3	41.5 ± 0.7	Entropy	0.61 ± 0.02	0.62 ± 0.02	0.59 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.59 ± 0.07	0.60 ± 0.06	0.56 ± 0.06
MASS, β=1e−4	41.5 ± 1.1	Entropy	0.60 ± 0.00	0.61 ± 0.01	0.58 ± 0.00
		$\max_i q_\phi(\cdot\|y_i)$	0.55 ± 0.05	0.56 ± 0.04	0.53 ± 0.04
MASS, β=0	42.0 ± 0.6	Entropy	0.60 ± 0.02	0.61 ± 0.02	0.57 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.55 ± 0.06	0.57 ± 0.04	0.54 ± 0.05
MASS, β=1e−2,D	41.7 ± 0.4	Entropy	**0.67 ± 0.01**	**0.68 ± 0.01**	**0.63 ± 0.01**
		$\max_i q_\phi(\cdot\|y_i)$	0.63 ± 0.04	**0.65 ± 0.04**	0.57 ± 0.04
MASS, β=1e−3,D	43.7 ± 0.2	Entropy	**0.67 ± 0.01**	**0.68 ± 0.01**	**0.63 ± 0.01**
		$\max_i q_\phi(\cdot\|y_i)$	**0.66 ± 0.05**	0.66 ± 0.04	**0.61 ± 0.06**
MASS, β=1e−4,D	43.4 ± 0.5	Entropy	**0.68 ± 0.01**	**0.69 ± 0.01**	**0.64 ± 0.02**
		$\max_i q_\phi(\cdot\|y_i)$	**0.64 ± 0.07**	**0.65 ± 0.05**	**0.59 ± 0.08**
MASS, β=0,D	43.9 ± 0.4	Entropy	**0.68 ± 0.00**	**0.69 ± 0.01**	**0.64 ± 0.00**
		$\max_i q_\phi(\cdot\|y_i)$	**0.65 ± 0.04**	**0.66 ± 0.03**	**0.60 ± 0.06**

Table 6.11: Out-of-distribution detection metrics for SmallMLP network trained on 2,500 CIFAR-10 images, with SVHN as the out-of-distribution examples. Full experiment details are in Section 6.8. Values are the mean over 4 training runs with different random seeds, plus or minus the standard deviation. Emboldened values are those for which the maximum observed mean value in the column was within one standard deviation. WD is weight decay; D is dropout. Larger values are better.

Training Method	Test Acc	Detection	AUROC	APR In	APR Out
SoftmaxCE	34.2 ± 0.8	Entropy	0.61 ± 0.01	0.62 ± 0.01	0.59 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.30 ± 0.02	0.38 ± 0.01	0.39 ± 0.01
SoftmaxCE, WD	23.9 ± 0.9	Entropy	**0.70 ± 0.03**	**0.67 ± 0.03**	**0.71 ± 0.04**
		$\max_i q_\phi(\cdot\|y_i)$	0.23 ± 0.02	0.36 ± 0.01	0.36 ± 0.01
SoftmaxCE, D	33.7 ± 1.1	Entropy	0.60 ± 0.01	0.62 ± 0.01	0.58 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.27 ± 0.01	0.37 ± 0.00	0.37 ± 0.00
VIB, β=1e−1	32.2 ± 0.6	Entropy	0.58 ± 0.01	0.60 ± 0.02	0.56 ± 0.01
		Rate	0.52 ± 0.02	0.54 ± 0.02	0.49 ± 0.02
VIB, β=1e−2	34.6 ± 0.4	Entropy	0.60 ± 0.01	0.62 ± 0.01	0.57 ± 0.01
		Rate	0.52 ± 0.04	0.55 ± 0.04	0.48 ± 0.03
VIB, β=1e−3	35.6 ± 0.5	Entropy	0.59 ± 0.01	0.60 ± 0.01	0.56 ± 0.01
		Rate	0.50 ± 0.04	0.53 ± 0.03	0.48 ± 0.03
VIB, β=1e−1, D	29.0 ± 0.6	Entropy	0.57 ± 0.01	0.60 ± 0.01	0.53 ± 0.01
		Rate	0.45 ± 0.02	0.48 ± 0.02	0.46 ± 0.01
VIB, β=1e−2, D	32.5 ± 0.9	Entropy	0.62 ± 0.01	0.63 ± 0.02	0.59 ± 0.01
		Rate	0.53 ± 0.05	0.56 ± 0.04	0.52 ± 0.04
VIB, β=1e−3, D	34.5 ± 1.0	Entropy	0.63 ± 0.01	0.64 ± 0.02	0.60 ± 0.01
		Rate	0.56 ± 0.05	0.57 ± 0.03	0.54 ± 0.05
MASS, β=1e−2	29.6 ± 0.4	Entropy	0.59 ± 0.01	0.61 ± 0.01	0.56 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.43 ± 0.03	0.48 ± 0.03	0.43 ± 0.01
MASS, β=1e−3	32.7 ± 0.8	Entropy	0.57 ± 0.01	0.59 ± 0.02	0.55 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.57 ± 0.04	0.59 ± 0.04	0.54 ± 0.03
MASS, β=1e−4	34.0 ± 0.3	Entropy	0.57 ± 0.01	0.57 ± 0.01	0.55 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.59 ± 0.03	0.58 ± 0.03	0.57 ± 0.03
MASS, β=0	34.1 ± 0.6	Entropy	0.57 ± 0.01	0.58 ± 0.01	0.55 ± 0.00
		$\max_i q_\phi(\cdot\|y_i)$	0.61 ± 0.03	0.59 ± 0.04	0.59 ± 0.04
MASS, β=1e−2,D	29.3 ± 1.2	Entropy	0.62 ± 0.02	**0.64 ± 0.03**	0.59 ± 0.02
		$\max_i q_\phi(\cdot\|y_i)$	0.50 ± 0.05	0.54 ± 0.05	0.47 ± 0.03
MASS, β=1e−3,D	31.5 ± 0.6	Entropy	0.61 ± 0.02	0.62 ± 0.03	0.58 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.62 ± 0.04	**0.63 ± 0.04**	0.58 ± 0.04
MASS, β=1e−4,D	32.7 ± 0.8	Entropy	0.61 ± 0.02	0.61 ± 0.03	0.59 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.65 ± 0.04	**0.63 ± 0.04**	0.62 ± 0.05
MASS, β=0,D	32.2 ± 1.1	Entropy	0.63 ± 0.01	0.64 ± 0.02	0.61 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	**0.65 ± 0.05**	**0.64 ± 0.05**	0.62 ± 0.06

Table 6.12: Out-of-distribution detection metrics for ResNet20 network trained on 40,000 CIFAR-10 images, with SVHN as the out-of-distribution examples. Full experiment details are in Section 6.8. Values are the mean over 4 training runs with different random seeds, plus or minus the standard deviation. Emboldened values are those for which the maximum observed mean value in the column was within one standard deviation. Larger values are better.

Training Method	Test Acc	Detection	AUROC	APR In	APR Out
SoftmaxCE	81.7 ± 0.3	Entropy	**0.77 ± 0.02**	0.81 ± 0.02	0.70 ± 0.02
		$\max_i q_\phi(\cdot\|y_i)$	0.59 ± 0.03	0.62 ± 0.03	0.55 ± 0.02
VIB, β=1e−3	81.0 ± 0.3	Entropy	0.74 ± 0.02	0.79 ± 0.02	0.67 ± 0.02
		Rate	0.55 ± 0.04	0.57 ± 0.05	0.51 ± 0.03
VIB, β=1e−4	81.2 ± 0.4	Entropy	0.73 ± 0.02	0.76 ± 0.03	0.66 ± 0.02
		Rate	0.50 ± 0.02	0.54 ± 0.02	0.48 ± 0.01
VIB, β=1e−5	80.9 ± 0.5	Entropy	0.75 ± 0.02	0.80 ± 0.02	0.67 ± 0.02
		Rate	0.18 ± 0.05	0.34 ± 0.01	0.34 ± 0.01
VIB, β=0	81.5 ± 0.2	Entropy	**0.79 ± 0.02**	**0.84 ± 0.02**	**0.73 ± 0.04**
		Rate	0.11 ± 0.03	0.32 ± 0.01	0.32 ± 0.01
MASS, β=1e−3	75.8 ± 0.5	Entropy	0.74 ± 0.03	0.77 ± 0.03	0.69 ± 0.03
		$\max_i q_\phi(\cdot\|y_i)$	0.37 ± 0.04	0.43 ± 0.02	0.42 ± 0.02
MASS, β=1e−4	80.6 ± 0.5	Entropy	**0.76 ± 0.04**	**0.80 ± 0.04**	**0.70 ± 0.05**
		$\max_i q_\phi(\cdot\|y_i)$	0.48 ± 0.06	0.53 ± 0.05	0.47 ± 0.04
MASS, β=1e−5	81.6 ± 0.4	Entropy	0.77 ± 0.01	**0.82 ± 0.01**	**0.71 ± 0.02**
		$\max_i q_\phi(\cdot\|y_i)$	0.54 ± 0.03	0.58 ± 0.03	0.51 ± 0.02
MASS, β=0	81.5 ± 0.2	Entropy	**0.79 ± 0.03**	**0.83 ± 0.02**	**0.73 ± 0.03**
		$\max_i q_\phi(\cdot\|y_i)$	0.49 ± 0.04	0.54 ± 0.04	0.47 ± 0.02

Table 6.13: Out-of-distribution detection metrics for ResNet20 network trained on 10,000 CIFAR-10 images, with SVHN as the out-of-distribution examples. Full experiment details are in Section 6.8. Values are the mean over 4 training runs with different random seeds, plus or minus the standard deviation. Emboldened values are those for which the maximum observed mean value in the column was within one standard deviation. Larger values are better.

Training Method	Test Acc	Detection	AUROC	APR In	APR Out
SoftmaxCE	67.5 ± 0.8	Entropy	0.64 ± 0.02	0.68 ± 0.02	0.58 ± 0.02
		$\max_i q_\phi(\cdot\|y_i)$	0.59 ± 0.03	0.61 ± 0.03	0.57 ± 0.04
VIB, β=1e−3	66.9 ± 1.0	Entropy	0.59 ± 0.02	0.63 ± 0.04	0.54 ± 0.02
		Rate	**0.72 ± 0.05**	**0.73 ± 0.05**	**0.67 ± 0.05**
VIB, β=1e−4	66.4 ± 0.5	Entropy	0.59 ± 0.01	0.63 ± 0.02	0.54 ± 0.01
		Rate	0.59 ± 0.07	0.60 ± 0.07	0.56 ± 0.06
VIB, β=1e−5	67.9 ± 0.8	Entropy	0.61 ± 0.03	0.65 ± 0.04	0.56 ± 0.03
		Rate	0.39 ± 0.07	0.42 ± 0.03	0.43 ± 0.04
VIB, β=0	67.1 ± 1.0	Entropy	0.64 ± 0.01	0.68 ± 0.01	0.58 ± 0.01
		Rate	0.32 ± 0.03	0.39 ± 0.01	0.39 ± 0.01
MASS, β=1e−3	59.6 ± 0.8	Entropy	0.59 ± 0.02	0.62 ± 0.03	0.56 ± 0.02
		$\max_i q_\phi(\cdot\|y_i)$	0.49 ± 0.07	0.46 ± 0.06	0.48 ± 0.08
MASS, β=1e−4	66.6 ± 0.4	Entropy	0.62 ± 0.02	0.67 ± 0.02	0.56 ± 0.03
		$\max_i q_\phi(\cdot\|y_i)$	0.61 ± 0.05	0.61 ± 0.05	0.60 ± 0.05
MASS, β=1e−5	67.4 ± 1.0	Entropy	0.64 ± 0.02	0.69 ± 0.03	0.58 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.61 ± 0.08	0.61 ± 0.06	**0.61 ± 0.09**
MASS, β=0	67.4 ± 0.3	Entropy	0.64 ± 0.01	0.68 ± 0.02	0.58 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.55 ± 0.05	0.56 ± 0.04	0.54 ± 0.05

Table 6.14: Out-of-distribution detection metrics for ResNet20 network trained on 2,500 CIFAR-10 images, with SVHN as the out-of-distribution examples. Full experiment details are in Section 6.8. Values are the mean over 4 training runs with different random seeds, plus or minus the standard deviation. Emboldened values are those for which the maximum observed mean value in the column was within one standard deviation. Larger values are better.

Training Method	Test Acc	Detection	AUROC	APR In	APR Out
SoftmaxCE	50.0 ± 0.7	Entropy	0.51 ± 0.01	0.52 ± 0.02	0.49 ± 0.01
		$\max_i q_\phi(\cdot\|y_i)$	0.63 ± 0.04	0.62 ± 0.03	0.63 ± 0.04
VIB, β=1e−3	49.5 ± 1.1	Entropy	0.48 ± 0.05	0.50 ± 0.05	0.47 ± 0.03
		Rate	**0.68 ± 0.07**	**0.68 ± 0.05**	**0.66 ± 0.08**
VIB, β=1e−4	49.4 ± 1.0	Entropy	0.47 ± 0.05	0.50 ± 0.05	0.47 ± 0.03
		Rate	**0.66 ± 0.09**	**0.65 ± 0.08**	**0.66 ± 0.09**
VIB, β=1e−5	50.0 ± 1.1	Entropy	0.48 ± 0.05	0.49 ± 0.05	0.48 ± 0.03
		Rate	0.59 ± 0.10	0.55 ± 0.08	0.61 ± 0.09
VIB, β=0	50.6 ± 0.8	Entropy	0.51 ± 0.07	0.54 ± 0.08	0.50 ± 0.06
		Rate	**0.52 ± 0.20**	0.53 ± 0.15	0.56 ± 0.17
MASS, β=1e−3	38.2 ± 0.7	Entropy	0.48 ± 0.04	0.50 ± 0.04	0.47 ± 0.03
		$\max_i q_\phi(\cdot\|y_i)$	0.54 ± 0.11	0.48 ± 0.06	0.51 ± 0.08
MASS, β=1e−4	49.9 ± 1.0	Entropy	0.49 ± 0.04	0.51 ± 0.05	0.48 ± 0.03
		$\max_i q_\phi(\cdot\|y_i)$	**0.72 ± 0.08**	**0.71 ± 0.08**	**0.73 ± 0.08**
MASS, β=1e−5	50.1 ± 0.5	Entropy	0.50 ± 0.06	0.51 ± 0.06	0.49 ± 0.04
		$\max_i q_\phi(\cdot\|y_i)$	**0.69 ± 0.10**	**0.68 ± 0.10**	**0.70 ± 0.10**
MASS, β=0	50.2 ± 1.0	Entropy	0.51 ± 0.06	0.53 ± 0.06	0.50 ± 0.04
		$\max_i q_\phi(\cdot\|y_i)$	**0.69 ± 0.07**	**0.68 ± 0.07**	**0.68 ± 0.07**

Chapter 7

SUPERVISED LEARNING ON RELATIONAL DATABASES WITH GRAPH NEURAL NETWORKS, PART 2

7.1 Introduction

In this final short chapter, we take lessons learned in Chapters 5 and 6 and apply them to the GNN models for supervised learning on relational databases from Chapter 4, yielding noticeable improvement.

7.2 Lessons Learned

The encoding used to prepare tabular features for input into a deep model has a significant effect on performance. As with all modeling decisions regarding tabular data, questions of feature encoding matter most in terms of how they affect overfitting.

Options for categorical variables include whether to one-hot encode versus learning an embedding, what embedding size to use, and how to apply dropout regularization (whether to drop vector elements or whole embeddings). In our experiments we found that learned embeddings nearly always improved performance as long as the cardinality of the categorical variable was significantly less than the number of datapoints, otherwise the feature was merely a means for the model to overfit.

Options for scalar variables include how to rescale the variable (via quantiles, normalization, or log scaling) or whether to quantize the feature and treat it like a categorical variable. In our experiments we found that the best strategy was simply to use all the different types of encoding in parallel, turning each scalar feature into three rescaled features and one categorical feature. Unlike learning embeddings for high-cardinality categorical features, adding potentially-redundant encodings for scalar variables does not lead to overfitting, but can make the difference between a feature being useful or not.

For variables encoding specific information like geospatial or datetime information, converting the feature from a single scalar or categorical representation into a set of informative features can make a significant impact. See Appendix 4.9 for details.

Normalization was also crucial to experiment with. While batch normalization and layer normalization are standard practice in deep learning, they must be used with care in GNNs. Normalization can accelerate training but also make it impossible for a GNN to learn functions that are sensitive to the degree of a node, since differences in absolute magnitudes of hidden state vectors between nodes and datapoints will be normalized away.

7.3 Experiments

Code to reproduce all experiments may be found online.[1]

We modified the GNN model implementations introduced in Chapter 4 with all the lessons discussed in the previous section. In addition, based on the results of the benchmark in Chapter 5, we added a single-hidden-layer multilayer perceptron to the initialization function S for each node, using different MLP parameters for each node type.

Though the heterogeneous transformer model of Chapter 5 did not perform better than a standard transformer or MLP, it inspired us to try a new type of GNN architecture for RDBs. Given that each table in an RDB contains different features and each foreign key encodes a different type of relationship, it seemed reasonable to use GNNs that maintained different parameters for each node-update and message-passing function, depending on the node type and edge type respectively. These are in the spirit of (Schlichtkrull et al., 2018) and (X. Wang et al., 2019), though are different models. We refer to these as HetGCN, HetGIN, and HetGAT, respectively.

Additionally, inspired by the centrality of overfitting in determining the performance of deep models for tabular data and the consistently strong performance of GBDTs, we tested whether a stacking strategy (Wolpert, 1992) would offer any benefit. In these "Stacked" models, we train a GBDT on a concatenation of single-table features and the pre-logit activations of a trained GNN. In principle, if the GNN is learning useful features but overfitting in its final layers, this strategy should ameliorate the overfitting issue.

All other models are the same as described in Chapter 4. Thorough details about model and experiment implementation are in Appendix 7.6.

7.4 Results

Table 7.1 shows the AUROC performance of models relative to the best-performing tabular model on each dataset. We compare relative performance rather than absolute performance since for some datasets the variance in performance between cross-

[1]`https://github.com/mwcvitkovic/Supervised-Learning-on-Relational-Databases-with-GNNs`

validation splits is larger than the variance of performance between algorithms. Appendix Tables 7.2 and 7.3 give the absolute AUROC and accuracy results.

Table 7.1: Performance of baseline (above the line) and our GNN-based (below the line) learning algorithms on three supervised learning problems on RDBs. Values are the AUROC metric relative to the single-table logistic regression baseline; they are reported as the mean over 5 cross-validation splits, plus or minus the standard deviation. Bold values are those within one standard deviation of the maximum in the column. Larger values are better.

	Acquire Valued Shoppers Challenge	Home Credit Default Risk	KDD Cup 2014
Single-table LogReg	0 ± 0	0 ± 0	0 ± 0
Single-table MLP	-0.0007 ± 0.0009	0.0021 ± 0.0007	0.014 ± 0.002
Single-table GBDT	0.0043 ± 0.0008	0.006 ± 0.003	$\mathbf{0.027 \pm 0.001}$
DFS + LogReg	0.0106 ± 0.0007	0.023 ± 0.002	0.004 ± 0.002
DFS + MLP	0.0087 ± 0.0009	0.016 ± 0.003	0.007 ± 0.001
DFS + GBDT	0.003 ± 0.001	0.029 ± 0.002	$\mathbf{0.027 \pm 0.002}$
PoolMLP	0.006 ± 0.002	0.021 ± 0.003	0.007 ± 0.003
Stacked PoolMLP	0.008 ± 0.002	0.023 ± 0.003	0.016 ± 0.004
GCN	$\mathbf{0.038 \pm 0.002}$	$\mathbf{0.032 \pm 0.002}$	0.013 ± 0.002
GIN	0.035 ± 0.005	0.027 ± 0.002	0.014 ± 0.001
GAT	0.032 ± 0.003	$\mathbf{0.031 \pm 0.002}$	0.013 ± 0.002
HetGCN	$\mathbf{0.040 \pm 0.002}$	$\mathbf{0.030 \pm 0.002}$	0.013 ± 0.002
HetGIN	$\mathbf{0.039 \pm 0.002}$	0.025 ± 0.003	0.015 ± 0.002
HetGAT	$\mathbf{0.039 \pm 0.001}$	$\mathbf{0.031 \pm 0.002}$	0.015 ± 0.001
Stacked GCN	0.037 ± 0.002	$\mathbf{0.030 \pm 0.002}$	0.008 ± 0.004
Stacked GIN	0.033 ± 0.006	0.027 ± 0.002	0.010 ± 0.006
Stacked GAT	0.036 ± 0.002	$\mathbf{0.032 \pm 0.002}$	0.012 ± 0.002
Stacked HetGCN	$\mathbf{0.038 \pm 0.003}$	0.029 ± 0.002	0.013 ± 0.005
Stacked HetGIN	$\mathbf{0.039 \pm 0.002}$	0.027 ± 0.002	0.015 ± 0.005
Stacked HetGAT	0.0379 ± 0.0002	$\mathbf{0.030 \pm 0.002}$	0.010 ± 0.005

Thanks to model improvements gleaned from Chapters 5 and 6, the results in Table 7.1 suggest more strongly than those in Chapter 4 that GNN-based methods are a valuable new approach for supervised learning on RDBs.

GNN-based methods perform significantly better than the best baseline method on the Acquire Valued Shoppers Challenge dataset and perform moderately better than the best baseline on the Home Credit Default Risk dataset. They perform worse than the best baseline on the KDD Cup 2014, however no feature engineering approach of any kind offers an advantage on that dataset. The determinant of success on the KDD Cup 2014 dataset does not seem to be information outside the dataset's main table.

Though every GNN-based method we tested matches or exceeds the best baseline method on the Acquire Valued Shoppers Challenge and Home Credit Default Risk datasets, there is no clearly superior GNN model among them. Neither the heterogeneous models nor stacking offers a noticeable benefit. But we emphasize that all our experiments used off-the-shell GNNs or straightforward modifications thereof — the space of possible RDB-specific GNNs is large and mostly unexplored.

Acknowledgments

We thank Da Zheng, and Minjie Wang, and Zohar Karnin for helpful discussions.

7.5 Appendix: Additional Results

Table 7.2: AUROC of baseline (above the line) and our GNN-based (below the line) learning algorithms on three supervised learning problems on RDBs. Values are the mean over 5 cross-validation splits, plus or minus the standard deviation. Larger values are better.

	Acquire Valued Shoppers Challenge	Home Credit Default Risk	KDD Cup 2014
Single-table LogReg	0.686 ± 0.002	0.748 ± 0.004	0.774 ± 0.002
Single-table MLP	0.685 ± 0.002	0.750 ± 0.004	0.788 ± 0.002
Single-table GBDT	0.690 ± 0.002	0.754 ± 0.004	0.801 ± 0.002
DFS + LogReg	0.696 ± 0.001	0.771 ± 0.005	0.778 ± 0.002
DFS + MLP	0.694 ± 0.001	0.764 ± 0.005	0.781 ± 0.002
DFS + GBDT	0.689 ± 0.003	0.777 ± 0.004	0.801 ± 0.003
PoolMLP	0.692 ± 0.001	0.769 ± 0.005	0.781 ± 0.003
Stacked PoolMLP	0.694 ± 0.003	0.771 ± 0.007	0.790 ± 0.005
GCN	0.723 ± 0.002	0.780 ± 0.004	0.787 ± 0.003
GIN	0.721 ± 0.006	0.775 ± 0.005	0.788 ± 0.002
GAT	0.717 ± 0.002	0.778 ± 0.005	0.787 ± 0.002
HetGCN	0.726 ± 0.002	0.778 ± 0.005	0.787 ± 0.003
HetGIN	0.725 ± 0.003	0.773 ± 0.005	0.788 ± 0.002
HetGAT	0.725 ± 0.001	0.779 ± 0.005	0.789 ± 0.002
Stacked GCN	0.722 ± 0.002	0.778 ± 0.003	0.782 ± 0.003
Stacked GIN	0.719 ± 0.007	0.775 ± 0.005	0.784 ± 0.006
Stacked GAT	0.722 ± 0.001	0.779 ± 0.005	0.786 ± 0.003
Stacked HetGCN	0.724 ± 0.003	0.777 ± 0.005	0.786 ± 0.005
Stacked HetGIN	0.725 ± 0.003	0.775 ± 0.006	0.789 ± 0.006
Stacked HetGAT	0.724 ± 0.002	0.778 ± 0.005	0.784 ± 0.006

7.6 Appendix: Experiment Details

Software and Hardware

The `LightGBM`[2] library (Ke et al., 2017) was used to implement the GBDT models. All other models were implemented using the `PyTorch`[3] library (Paszke et al., 2019). GNNs were implemented using the `DGL`[4] library (M. Wang et al., 2019). All experiments were run on an Ubuntu Linux machine with 8 CPUs and 60GB memory, with all models except for the GBDTs trained using a single NVIDIA V100 Tensor Core GPU. Creating the DFS features was done on an Ubuntu Linux machine with 48 CPUs and 185GB memory.

[2]`https://lightgbm.readthedocs.io`

[3]`https://pytorch.org/`

[4]`https://www.dgl.ai/`

Table 7.3: Percent accuracy of baseline (above the line) and our GNN-based (below the line) learning algorithms on three supervised learning problems on RDBs. Values are the mean over 5 cross-validation splits, plus or minus the standard deviation. Larger values are better.

	Acquire Valued Shoppers Challenge	Home Credit Default Risk	KDD Cup 2014
Guess Majority Class	72.9	91.9	94.07
Single-table LogReg	73.1 ± 0.2	91.9 ± 0.1	94.07 ± 0.07
Single-table MLP	73.2 ± 0.2	91.9 ± 0.1	94.07 ± 0.07
Single-table GBDT	73.3 ± 0.2	91.9 ± 0.1	94.06 ± 0.07
DFS + LogReg	73.5 ± 0.2	91.9 ± 0.1	94.07 ± 0.07
DFS + MLP	73.6 ± 0.3	91.9 ± 0.1	94.07 ± 0.07
DFS + GBDT	73.4 ± 0.3	91.96 ± 0.09	94.06 ± 0.07
PoolMLP	73.6 ± 0.1	91.9 ± 0.1	94.07 ± 0.07
Stacked PoolMLP	73.6 ± 0.2	91.9 ± 0.1	94.04 ± 0.08
GCN	75.4 ± 0.1	91.96 ± 0.09	94.07 ± 0.07
GIN	75.4 ± 0.3	91.9 ± 0.1	94.07 ± 0.07
GAT	72.9 ± 0.3	91.1 ± 0.3	94.07 ± 0.07
HetGCN	75.6 ± 0.2	91.9 ± 0.1	94.07 ± 0.07
HetGIN	75.6 ± 0.3	91.9 ± 0.1	94.07 ± 0.07
HetGAT	75.2 ± 0.2	92.0 ± 0.1	94.07 ± 0.07
Stacked GCN	75.5 ± 0.1	91.9 ± 0.1	94.06 ± 0.07
Stacked GIN	75.4 ± 0.3	92.0 ± 0.1	94.04 ± 0.06
Stacked GAT	75.5 ± 0.3	92.0 ± 0.1	94.07 ± 0.07
Stacked HetGCN	75.6 ± 0.3	91.9 ± 0.1	94.02 ± 0.06
Stacked HetGIN	75.6 ± 0.2	91.9 ± 0.1	94.04 ± 0.07
Stacked HetGAT	75.6 ± 0.2	91.9 ± 0.1	94.06 ± 0.07

GNN Implementation

Adopting the nomenclature from Section 3.3, once the input graph has been assembled by `RDBToGraph`, the initialization function S for converting the features $\boldsymbol{x}_v$ of each vertex v into a real–valued vector in $\mathbb{R}^d$ proceeded as follows: (1) each of the vertex's features was vectorized according to the feature's data type, (2) these vectors were concatenated, (3) the concatenated vector was passed through an single-hidden-layer MLP with output dimension $\mathbb{R}^d$. Each node type uses its own single-hidden-layer MLP initializer. The dimension of the hidden layer was 4x the dimension of the concatenated features.

To vectorize columns containing scalar data, S normalizes them to zero median and unit interquartile range[5] and appends a binary flag for missing values. To vectorize columns containing categorical information, S uses a trainable embedding with dimension the minimum of 32 or the cardinality of categorical variable. To vectorize `text` columns, S simply encodes the number of words in the text and the length of the text. To vectorize `latlong` columns, S concatenates the following:

1. $\cos(lat) * \cos(long)$
2. $y = \cos(lat) * \sin(long)$
3. $z = \sin(lat)$
4. $lat/90$
5. $long/180$

And to vectorize `datetime` columns, S concatenates the following commonly used date and time features:

1. Year (scalar value)
2. Month (one-hot encoded)
3. Week (one-hot encoded)
4. Day (one-hot encoded)
5. Day of week (one-hot encoded)
6. Day of year (scalar value)
7. Month end? (bool, one-hot encoded)
8. Month start? (bool, one-hot encoded)
9. Quarter end? (bool, one-hot encoded)
10. Quarter start? (bool, one-hot encoded)
11. Year end? (bool, one-hot encoded)
12. Year start? (bool, one-hot encoded)

[5]`https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html`

13. Day of week cos (scalar value)
14. Day of week sin (scalar value)
15. Day of month cos (scalar value)
16. Day of month sin (scalar value)
17. Month of year cos (scalar value)
18. Month year sin (scalar value)
19. Day of year cos (scalar value)
20. Day of year sin (scalar value)

The cos and sin values are for representing cyclic information, and are given by computing cos or sin of $2\pi \frac{value}{period}$. E.g. "day of week cos" for Wednesday, the third day of seven in the week, is $\cos(2\pi\frac{3}{7})$.

After obtaining $\boldsymbol{h}_v^0 = S(\boldsymbol{x}_v)$ for all vertices v, all models ran 2 rounds of message passing ($T = 2$), except for the GCN and HetGCN which ran 1 round. Dropout regularization of probability 0.5 was used in all models, applied at the layers specified in the paper that originally introduced the model. Most models used a hidden state size of $d = 256$, except for a few exceptions where required to fit things onto the GPU. Full hyperparameter specifications for every model and experiment can be found in the experiment scripts in the code released with this dissertation.

The readout function R for all GNNs was the Gated Attention Pooling method of (Li et al., 2016) followed by a linear transform to obtain logits followed by a softmax layer. The only exception is the PoolMLP model, which uses average pooling of hidden states followed by a 1-hidden-layer MLP as R. The cross entropy loss was used for training, and the AdamW optimizer (Loshchilov and Hutter, 2017) was used to update the model parameters. All models used early stopping based on the performance on a validation set.

References

Ke, Guolin et al. (2017). "LightGBM: A highly efficient gradient boosting decision tree". In: *Advances in Neural Information Processing Systems*, pp. 3146–3154. URL: `https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf`.

Li, Yujia et al. (2016). “Gated graph sequence neural networks”. In: *International Conference on Learning Representations*. URL: `https://arxiv.org/abs/1511.05493`.

Loshchilov, Ilya and Frank Hutter (2017). “Decoupled Weight Decay Regularization”. In: *International Conference on Learning Representations*. URL: `https://arxiv.org/abs/1711.05101`.

Paszke, Adam et al. (2019). “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., pp. 8024–8035. URL: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Schlichtkrull, Michael et al. (2018). “Modeling relational data with graph convolutional networks”. In: *European Semantic Web Conference*. Springer, pp. 593–607. URL: `https://arxiv.org/abs/1703.06103`.

Wang, Minjie et al. (2019). “Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs”. In: *ICLR Workshop on Representation Learning on Graphs and Manifolds*. URL: `https://arxiv.org/abs/1909.01315`.

Wang, Xiao et al. (2019). “Heterogeneous Graph Attention Network”. In: *The World Wide Web Conference*. ACM, pp. 2022–2032. URL: `https://arxiv.org/abs/1903.07293`.

Wolpert, David H (1992). “Stacked generalization”. In: *Neural networks* 5.2, pp. 241–259.