Title
Learning Latent Hierarchical Structures via Probabilistic Models and Deep Learning

Permalink
https://escholarship.org/uc/item/4w44s8sx

Author
Arabshahi, Forough

Publication Date
2018

Peer reviewed|Thesis/dissertation
LEARNING LATENT HIERARCHICAL STRUCTURES VIA PROBABILISTIC MODELS AND DEEP LEARNING

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Electrical and Computer Engineering

by

Forough Arabshahi

Dissertation Committee:
Professor Sameer Singh, Chair
Professor Animashree Anandkumar
Professor Carter Tribley Butts

2018
DEDICATION

To my sister, Farima
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF ALGORITHMS</td>
<td>vii</td>
</tr>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>viii</td>
</tr>
<tr>
<td>CURRICULUM VITAE</td>
<td>x</td>
</tr>
<tr>
<td>ABSTRACT OF THE DISSERTATION</td>
<td>xiii</td>
</tr>
</tbody>
</table>

## 1 Introduction
1.1 Dissertation Overview .......................... 3
1.2 Conditional Latent Tree Graphical Models .................. 5
1.3 Correlated Topic Models ........................... 6
1.4 Neural Programming for Modeling Mathematical Equations .......................... 7

## 2 Conditional Latent Tree Graphical Models
2.1 Summary of Results ................................ 13
2.2 Related Work .................................... 15
2.3 Model ........................................ 16
2.3.1 Conditional Latent Tree Models (CLTM) .............. 17
2.3.2 Exploiting Inferred Hidden Groups .................. 19
2.4 Model Estimation .................................. 20
2.4.1 Structure Learning .............................. 21
2.4.2 Parameter Estimation using Expectation Maximization .......................... 23
2.5 Experiments and Results ............................ 24
2.5.1 Educational Data ................................ 27
2.5.2 Network Data .................................. 32

## 3 Correlated Topic Models
3.1 Related Work ..................................... 40
3.2 Summary of Results ............................... 40
3.3 Latent Normalized Infinitely Divisible Topic Models ............ 41
3.4 Properties of NID distributions .......................... 42
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Toy example of the underlying structure over the random variables of CLTM</td>
<td>6</td>
</tr>
<tr>
<td>1.2</td>
<td>Graphical Model Representation of the Latent NID Topic Model</td>
<td>7</td>
</tr>
<tr>
<td>1.3</td>
<td>Identities and their Expression Trees</td>
<td>8</td>
</tr>
<tr>
<td>1.4</td>
<td>Example of a differential equation (Fig. 1.4a) and its solution (Fig. 1.4b)</td>
<td>9</td>
</tr>
<tr>
<td>2.1</td>
<td>Vertex activities in datasets with respect to time</td>
<td>24</td>
</tr>
<tr>
<td>2.2</td>
<td>Knowledge components latent tree</td>
<td>27</td>
</tr>
<tr>
<td>2.3</td>
<td>Subgroups of the estimated knowledge component latent tree</td>
<td>30</td>
</tr>
<tr>
<td>2.4</td>
<td>learned covariate weights</td>
<td>30</td>
</tr>
<tr>
<td>2.5</td>
<td>conditional presence and absence boxplots vs. time for education test data</td>
<td>31</td>
</tr>
<tr>
<td>2.6</td>
<td>Performance of group of “strong” and “weak” learners over the semester</td>
<td>32</td>
</tr>
<tr>
<td>2.7</td>
<td>Node covariate weights for beach data</td>
<td>36</td>
</tr>
<tr>
<td>2.8</td>
<td>Learnt coefficients for various covariates for edge prediction in Twitter data</td>
<td>36</td>
</tr>
<tr>
<td>2.9</td>
<td>Vertex and edge prediction accuracy for the Twitter dataset</td>
<td>36</td>
</tr>
<tr>
<td>3.1</td>
<td>Heat map of 3 examples of NID class</td>
<td>44</td>
</tr>
<tr>
<td>3.2</td>
<td>Proportion of positively correlated pairs of NID distributed random variable</td>
<td>47</td>
</tr>
<tr>
<td>3.3</td>
<td>Weights $v_1$ for two examples of the NID distribution</td>
<td>50</td>
</tr>
<tr>
<td>3.4</td>
<td>Perplexity and PMI scores for the NYtimes dataset across different number of topics</td>
<td>57</td>
</tr>
<tr>
<td>4.1</td>
<td>Tree-structured recursive neural model</td>
<td>70</td>
</tr>
<tr>
<td>4.2</td>
<td>Evaluating Equation Completion</td>
<td>77</td>
</tr>
<tr>
<td>4.3</td>
<td>Examples of Equation Completion</td>
<td>78</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Prediction scores for educational data</td>
<td>29</td>
</tr>
<tr>
<td>2.2</td>
<td>Data specification and number of used covariates</td>
<td>33</td>
</tr>
<tr>
<td>2.3</td>
<td>Prediction scores for network datasets</td>
<td>33</td>
</tr>
<tr>
<td>3.1</td>
<td>Perplexity comparison for different datasets</td>
<td>52</td>
</tr>
<tr>
<td>3.2</td>
<td>PMI comparison for different datasets</td>
<td>53</td>
</tr>
<tr>
<td>3.3</td>
<td>10 Shared words - New York times dataset</td>
<td>53</td>
</tr>
<tr>
<td>3.4</td>
<td>NID Top 10 Words for NYtimes, K = 20</td>
<td>56</td>
</tr>
<tr>
<td>3.5</td>
<td>NID top 10 Words for Pubmed, K = 10</td>
<td>57</td>
</tr>
<tr>
<td>3.6</td>
<td>Spectral LDA top 10 Words for NYtimes, K = 20</td>
<td>58</td>
</tr>
<tr>
<td>4.1</td>
<td>Symbols in our grammar, i.e. the functions, variables, and constants</td>
<td>66</td>
</tr>
<tr>
<td>4.2</td>
<td>Examples of generated equations</td>
<td>70</td>
</tr>
<tr>
<td>4.3</td>
<td>Generalization results for equation verification</td>
<td>75</td>
</tr>
<tr>
<td>4.4</td>
<td>Extrapolation evaluation for equation verification</td>
<td>76</td>
</tr>
<tr>
<td>4.5</td>
<td>Symbols in our grammar, i.e. the functions, variables, and constants used for solving differential equations</td>
<td>84</td>
</tr>
<tr>
<td>4.6</td>
<td>Statistics of the data</td>
<td>87</td>
</tr>
<tr>
<td>4.7</td>
<td>Performance evaluation for solving ODEs on unseen test data</td>
<td>88</td>
</tr>
</tbody>
</table>
# LIST OF ALGORITHMS

<table>
<thead>
<tr>
<th></th>
<th>Algorithm</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Parameter Learning</td>
<td>51</td>
</tr>
<tr>
<td>2</td>
<td>Hyper-parameter Optimization (HPO)</td>
<td>54</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Animashree Anandkumar, for her guidance and support throughout my Ph.D. studies. I am truly honored for having the opportunity of being her student. She is a true role model and she constantly encouraged me to pursue my research goals. I truly appreciate her help in building my confidence and teaching me to become a researcher. I would also like to extend my deepest appreciation to my co-advisor, Sameer Singh, for his great mentorship, for giving me new perspectives on my research and for teaching me to be realistic when setting research goals. I learned a lot from his knowledge and expertise and I am grateful for the opportunity of having him as my advisor. I am also extremely grateful to my committee member, Carter T. Butts, for his insightful comments and unique point of view. His challenging questions always lead to a deeper understanding of the concept that I was working on. The completion of this dissertation would not have been possible without my advisors’ and committee members’ nurturing and support.

I wish to thank my collaborators Furong Huang and Sean M Fitzhugh. Working with them was an amazing experience for me.

I would like to thank the department of Electrical Engineering and Computer Science at the University of California Irvine and Professor A. Lee Swindlehurst for giving me the opportunity to pursue a PhD degree.

This dissertation was supported through NSF BIGDATA grant no. IIS-1251267, ONR Award N00014-14-1-0665, DARPA Award D17AP00002 and the UCI Data Science Initiative summer fellowship.

I would like to acknowledge IEEE for giving me permission to include Chapter 2 of this dissertation parts of which was originally published in IEEE Xplore. I would also like to thank the proceedings of Machine Learning Research for giving me permission to include Chapter 3 in my dissertation which was originally published therein.

I am also grateful to my friends who were always there for me throughout my journey. I would like to thank Laleh Aghababaie Beni, Shohreh Karimaghaee, Arya Iranmehr, Zahra Rohaghi, Bahareh Shakiba Jahromi, Saeed Shayesteh, Mahdi Abbaspour Tehrani, Bahareh Mostafazadeh, Maral Amir, Mahmoud Kalaei, Mehdi Ganji, Arash Gholami Davoodi, Koroosh Vatanparvar, Omid Assare and my amazing lab mates Hanie Sedghi, Majid Janzamin, Furong Huang, Niranjan U N, Kamyar Azizzadenesheli and Yang Shi.

Last but not least, I cannot begin to express my sincere thanks to my family. My parents, Zahra Karimaghaei and Shahbaz Arabshahi, without the unconditional love and support of whom I would have never been where I am today. I am forever grateful to them for providing me with the opportunity to pursue my education and for always encouraging me to seek a better life. My husband, Hosein Mohimani, for giving me love, support, motivation and encouragement. He taught me to overcome my fears and to believe in myself. I am truly
thankful for his efforts in building my time management skills.
CURRICULUM VITAE

Forough Arabshahi

EDUCATION

Doctor of Philosophy in Electrical and Computer Engineering 2018
University of California Irvine

Master of Science in Electrical Engineering 2012
Amirkabir University of Technology

Bachelor of Science in Electrical Engineering 2010
Shiraz University

Work & Academic Experience

Graduate Research Assistant Jan 2014–June 2018
University of California, Irvine

Software Engineer Intern June 2017–Sept 2017
Pepperdata Inc.,

Research Intern June 2016–Sept 2016
Yahoo Labs

TEACHING EXPERIENCE

Invited Lecture: Spectral Methods for LVMs April 2018
Carnegie Mellon University

Teaching Assistant: EECS10 Sept 2013–Dec 2013
University of California Irvine

Teaching Assistant: Communications I Mar 2010–Jun 2010
Shiraz University
REFEREED CONFERENCE PUBLICATIONS

“Combining Symbolic Expressions and Black-box Function Evaluations in Neural Programs” by F. Arabshahi, S. Singh, A. Anandkumar 2018
*International Conference on Learning Representations (ICLR), Vancouver, BC. Also appeared at NIPS 2017 MLtrain workshop, Long Beach, CA.*

“Spectral Methods for Correlated Topic Models” by F. Arabshahi, A. Anandkumar 2017
*International Conference on Artificial Intelligence and Statistics (AISTATS), PMLR 54:1439-1447, Fort Lauderdale, FL.*

“Are you going to the party: depends, who else is coming? [Learning hidden group dynamics via conditional latent tree models]” by F. Arabshahi, F. Huang, A. Anandkumar, C. T. Butts, S. M. Fitzhugh 2015
*IEEE International Conference on Data Mining (ICDM), Atlantic City, NJ.*

“A Frequency Domain MVDR Beamformer for UWB Microwave Breast Cancer Imaging in Dispersive Mediaums” by F. Arabshahi, S. Monajemi, H. Sheikhzadeh, K. Raahemifar, R. Faraji-Dana 2013
*IEEE International Symposium on Signal Processing and Information Technology (ISPIT), Athens, Greece.*

REFEREED WORKSHOP PUBLICATIONS

“Towards Solving Differential Equations Through Neural Programming” by F. Arabshahi, S. Singh, A. Anandkumar 2018
*ICML workshop Neural Abstract Machines & Program Induction v2 (NAMPI), Stockholm, Sweden, 2018.*

“Beyond LDA: Spectral Methods for Topic Modeling Based on Exchangeable Partitions” by F. Arabshahi, A. Anandkumar 2015
*NIPS workshop on Bayesian Nonparametrics: The Next Generation, Montreal, QC.*

*Statistical Inference for Network Models, NetSci Satellite Symposium, Berkeley, CA.*
SOFTWARE

Python algorithm for neural programming

Correlated Topic Models https://github.com/ForoughA/NID
MATLAB algorithm for latent NID topic models

CLTM https://github.com/ForoughA/CLTM
MATLAB algorithm for conditional latent tree graphical models
Hierarchical structures arise in many real world applications and domains. For example, in social networks people’s relationships and the groups to which they belong form a hierarchy. In natural language and computer programs, parse trees (which have a hierarchical structure) are used to represent the compositionality of expressions. These hierarchies strongly affect the statistics and the behavior of the data. Hence, it is important to develop algorithms that take these structures into account when modeling such data. Apart from these hierarchical structures, some datasets are best explained with hierarchical models even though there is no apparent hierarchy in the data itself. For instance when modeling the occurrence of words in a document, it is more realistic to assume that the words are drawn in a hierarchical manner from a topic distribution rather than independently from a single topic. In this dissertation, we focus on capturing these hierarchies and leveraging them for modeling high dimensional datasets.

Hierarchical structures underlying the data are either observed or latent. For example in the context of computer programs, the syntax tree is inherent to the program and is therefore observed. On the other hand, the statistical dependence of a social network’s users is latent. In this dissertation, we study both types of hierarchies and develop models under both structures because they both arise in many applications and are equally important. Nevertheless,
capturing latent hierarchical structures is more challenging. We develop novel probabilistic models to capture latent hierarchies and present statistically efficient and provably consistent parameter learning algorithms for them. When capturing observed hierarchical structures we develop deep learning models that learn low-dimensional continuous representations for the discrete symbols and variables.
Chapter 1

Introduction

Many practical problems in machine learning deal with data that is inherently hierarchical. For example, in natural language processing sentences can be syntactically analyzed using a parse tree, or in social networks reactions to an influencer’s Tweet propagate in a hierarchical manner. Since hierarchical structures affect the behavior of the data, it is important for machine learning algorithms to account for them. Moreover, in some domains such as topic modeling hierarchical models achieve a better performance compared to their flat counterparts. In this dissertation, we develop probabilistic and deep learning models through extraction of hierarchical structures.

With the current rate of data growth and availability of computing resources, it is crucial for our learning algorithms to handle the scale of available data. The dimension of the datasets that we work with is also large. Therefore, another aim of this dissertation is to develop algorithms that model high dimensional datasets and are therefore scalable.

Our algorithms are applicable to two types of hierarchical structures, namely latent and observed. An example of latent hierarchies is the structure of the dependence among a social network’s users. An example of observed hierarchical structures is the parse tree of a
sentence. It is worth noting that learning latent hierarchies is more challenging.

In order to learn latent hierarchical structures and models, we develop probabilistic methods. Specifically, we work with latent variable graphical models which are a suitable candidate for capturing latent structures. Our developed models are applicable to several domains such as modeling high-dimensional time series and topic modeling. In order to train our latent variable models, we first propose statistically efficient learning algorithms based on maximum likelihood. Specifically we use Expectation Maximization (EM). Second, we present provably consistent and scalable learning algorithms based on the method of moments. Since EM aims to find the Maximum Likelihood Estimation (MLE), it is more statistically efficient compared to the method of moments. However, EM can get stuck in local optima and lacks theoretical guarantees. Method of moments on the other hand, is local optima-free despite a non-convex optimization surface, is provably consistent and is faster and more scalable compared to EM-based methods. It is, nonetheless, hard to derive method of moments-based learning methods for new models, whereas doing so for EM-based methods is easier. Up until now, it is unknown whether the derivation of method of moments-based learning generalizes to an unseen model. However, in this dissertation we show that we can unify this derivation for a flexible class of topic models shedding some light on the generalizability of these methods.

While capturing observed hierarchies we develop deep learning models that learn low dimensional continuous representations, also known as embeddings, for the discrete variables and symbols. We would like to emphasize that learning these embeddings is not plausible with black-box deep learning models. We show that incorporating the structure of the data in our developed deep learning models is crucial for achieving good performance. Specifically, we develop recursive neural networks that mirror the underlying hierarchy of the data. An application of this is in neural programming, where the goal is to do mathematical or symbolic reasoning. Neural programmers address the long standing quest of computer scientists to design machines capable of reasoning, but in a data-driven manner. This is the neural
programmers’ key to generalizability and scalability compared to the traditional rule-based methods. We propose a novel data augmentation strategy that combines function evaluations with symbolic expressions, which formulate the relationships between the functions, as well as tree encodings for numbers based on their decimal representation. Our model learns vector embeddings for functions, which are low-dimensional vector representations encoding the functions’ properties, and symbols. Leveraging the compositionality of the mathematical equations ensures that the structural information of the equations are encoded within the embeddings. We use these models for several applications including mathematical equation verification, equation completion as well as solving differential equations.

The rest of this chapter is organized as follows. In Section 1.1, we overview the dissertation and summarize our contributions. In the sections that follow, we summarize the studied problems, developed models and contributions in different chapters. Section 1.2 introduces our developed latent tree graphical model for high-dimensional time series. In Section 1.3, we introduce our correlated topic model that generalizes the well-known latent Dirichlet allocation and briefly discuss our method of moments-based learning algorithm. In section 1.4, we introduce our neural programmer and its application to solving differential equations.

1.1 Dissertation Overview

The developed models in this dissertation are given below.

In Chapter 2, we propose conditional latent tree graphical models for modeling high dimensional time series such as social networks and Massive Open Online Courses (MOOCs). This latent variable graphical model captures latent hierarchical structures from the data. The latent variables in the structure recover latent groupings underlying the time series. These latent groupings capture the dynamics of the series. We provide a structure learning algo-
rithm for recovering the latent tree using the notion of “conditional information distances” that will be defined in detail therein. We also provide a parameter learning algorithm based on Expectation Maximization (EM) and finally apply our model to three real-world datasets. The results of this chapter are based on our publication in ICDM’15; see Curriculum Vitae.

In Chapter 3, we propose latent Normalized Infinitely Divisible (NID) topic models which are generative admixture models with a hierarchical structure. Latent NID topic models generalize the well-known Latent Dirichlet Allocation to allow for arbitrary correlations among the topics. NID distributions are a class of distributions on the probability simplex that include the Dirichlet distribution, the normalized $\alpha$-stable distribution and the inverse Gaussian distribution among others. We provide provable guarantees for learning latent NID topic models using the method of moments. We apply the model to two document corpora and report the results. The results in this chapter are based on our publication in AISTATS’17; see Curriculum Vitae.

Neural programming involves training neural networks to learn programs, mathematics or logic from data. In Chapter 4, we propose a neural programmer that models mathematical functions in trigonometry and elementary algebra by learning latent embeddings for them. This neural programmer captures observed hierarchical structures from the data. Our neural programmer is applicable to mathematical equation verification and equation completion as well as differential equation solving. Our framework uses symbolic equations, that define relationships between mathematical functions, as well as black-box function evaluations for training. We employ tree-structured Long Short-Term Memory (tree-LSTM) to incorporate the compositionality of the input equations. In order to model numbers in a generalizable fashion, we use a tree encoding based on their decimal representation. This allows us to represent floating point numbers with integers in $[0,10]$. Finally, we propose a data generation algorithm for generating data at scale for training the neural programmer. Our evaluation benchmark demonstrates that our proposed model combines symbolic equations and function
evaluations in a fruitful manner and is able to generalize well to more complex equations. The results in this chapter are published in ICLR’18 and NAMPI’18; see Curriculum Vitae.

Each chapter is self-contained and the experimental setups, models, algorithms, proofs and discussions are included.

1.2 Conditional Latent Tree Graphical Models

Scalable probabilistic modeling and prediction in high dimensional multivariate time-series is a challenging problem, particularly for systems with hidden sources of dependence and/or homogeneity. Examples of such problems include dynamic social networks with co-evolving nodes and edges and dynamic student learning in online courses. In Chapter 2, we address these problems through the discovery of hierarchical latent groups. We introduce a family of Conditional Latent Tree Models (CLTM), in which tree-structured latent variables incorporate the unknown groups. Fig 1.1, demonstrates a toy example of our CLTM model. As shown in the figure, the latent tree itself is conditioned on observed covariates such as seasonality, historical activity, and node attributes. We propose a statistically efficient framework for learning both the hierarchical tree structure and the parameters of the CLTM. We demonstrate competitive performance in multiple real world datasets from different domains. These include a dataset of students’ attempts at answering questions in a psychology MOOC, Twitter users participating in an emergency management discussion and interacting with one another, and windsurfers interacting on a beach in Southern California. In addition, our modeling framework provides valuable and interpretable information about the hidden group structures and their effect on the evolution of the time series.
Figure 1.1: Toy example of the underlying structure over the random variables of the CLTM model. Yellow nodes are observable variables, blank nodes are hidden variables, green nodes denote the individual covariates and the red node denotes the global covariate. As shown in the figure, the dependence structure of the random variables is a tree conditioned on covariates.

1.3 Correlated Topic Models

In Chapter 3, we propose guaranteed spectral methods for learning a broad range of topic models, which generalize the popular Latent Dirichlet Allocation (LDA). The graphical model representation of our proposed topic model is shown in Fig 1.2. We overcome the limitation of LDA to incorporate arbitrary topic correlations, by assuming that the hidden topic proportions are drawn from a flexible class of Normalized Infinitely Divisible (NID) distributions. NID distributions are generated by normalizing a family of independent Infinitely Divisible (ID) random variables. The Dirichlet distribution is a special case obtained by normalizing a set of Gamma random variables. We prove that this flexible topic model class can be learnt via spectral methods using only moments up to the third order, with (low order) polynomial sample and computational complexity. The proof is based on a key new technique derived in this dissertation that allows us to diagonalize the moments of the NID distribution through an efficient procedure that requires evaluating only univariate integrals, despite the fact that we are handling high dimensional multivariate moments. In order to assess the performance of our proposed Latent NID topic model, we use two real datasets.
Figure 1.2: Graphical Model Representation of the Latent NID Topic Model. $z_1, z_2, \ldots, z_k$ are a collection of independent Infinitely Divisible positive variables that are characterized by the collection of their corresponding Lévy measures $\alpha_1 \nu, \alpha_2 \nu, \ldots, \alpha_k \nu$. And $h_1, h_2, \ldots, h_k$ are the resulting NID variables representing topic proportions in a document of length $N$ with words $x_1, \ldots, x_N$ of articles collected from New York Times and Pubmed. Our experiments yield improved perplexity on both datasets compared with the baseline.

1.4 Neural Programming for Modeling Mathematical Equations

Neural programming involves training neural networks to learn programs, mathematics, or logic from data. Previous works have failed to achieve good generalization performance, especially on problems and programs with high complexity or on large domains. This is because they mostly rely either on black-box function evaluations that do not capture the structure of the program, or on detailed execution traces that are expensive to obtain, and hence the training data has poor coverage of the domain under consideration. In Chapter 4, we present a novel framework that utilizes black-box function evaluations, in conjunction with symbolic expressions that define relationships between the given functions. We employ
Figure 1.3: **Identities and their Expression Trees.** with (a) a symbolic expression, (b) a function evaluation, and (c) a number represented as the decimal tree (also part of the function evaluation data)

tree LSTMs to incorporate the structure of the symbolic expression trees. Fig 1.3 shows an example of our symbolic expression trees. We use tree encoding for numbers present in function evaluation data (Fig 1.3c), based on their decimal representation. We present an evaluation benchmark for this task to demonstrate our proposed model combines symbolic reasoning and function evaluation in a fruitful manner, obtaining high accuracies in our experiments. Our framework generalizes significantly better to expressions of higher depth and is able to fill partial equations with valid completions.

**Solving Differential Equations with Neural Programs** One of the applications of our developed neural programmer in this chapter is solving differential equations. Differential equations are used to model numerous phenomena such as heat, electrodynamics, fluid dynamics and quantum mechanics. In Section 4.6, we state how we can use our neural programmer for this application. Fig 1.4 shows an example of a differential equation and its solution. Almost all of the literature that uses neural networks for solving differential equations, focus on collecting numeric data from a given differential equation for training. This is, however, neither scalable nor generalizable. We propose using symbolic differential equations as training samples and leverage their compositionality through TreeLSTMs to find solutions for unseen differential equations. We augment this symbolic data with rules of differentiation, function evaluations and tree-encodings of numbers to achieve generalizability.
Figure 1.4: Example of a differential equation (Fig. 1.4a) and its solution (Fig. 1.4b)
Chapter 2

Conditional Latent Tree Graphical Models

In this section we introduce a parametric model class, namely Conditional Latent Tree graphical Models (CLTMs), and propose an approach for learning them from data. CLTMs can be used for modeling and predicting high dimensional time-series with latent dependence and/or unobserved heterogeneity. Such time series arise in numerous important applications, including dynamic social networks with co-evolving nodes and edges, and dynamic student learning in MOOCs. Of particular interest in modeling such high dimensional series is the problem of predicting their evolution. Such predictions can in turn be used to provide useful feedback such as recommendations to network participants or students to improve their experience in the network and help them learn the course material (respectively). Modeling and tracking such high dimensional series jointly, however, is a greatly challenging task since each sequence can interact with others in unknown and complex ways. Before delving into the details of the prediction model, we thus first identify several factors that influence the dynamics of high-dimensional time series in a social context.
First and foremost, individual-level behavioral variables in a multivariate time series are strongly influenced by *group dynamics*. For example, the nodes in a social network tend to participate in communities, and the evolution of node behavior can be captured in part by the dynamics of those communities. In some cases, the resulting dependence is endogenous: for instance, a network attendee might wonder who else is going to attend a social event (e.g., a party) before deciding whether to attend him or herself. In other cases, group-level dependence may stem from unobserved heterogeneity: in the student learning scenario, for instance, students may be divided into groups of strong and weak learners whose learning curves evolve in drastically different ways. Hence, finding such underlying groupings and considering their dynamics for predicting the evolution of each individual sequence is of great importance.

A second challenge for modeling in this context, is that the dynamic behavior of each random variable affects the dynamics of other random variables, making the individual sequences dependent on one another. Treating each individual sequence independently ignores such interdependence and results in poor predictions.

A third challenge for modeling in this context is the need to account for the impact of relevant external factors also known as *covariates*, that are predictive of dynamics. Seasonal or period effects are examples of covariates whose states can be predictive of the evolution of the series. E.g. in weekly social events, the day of week is a highly predictive factor of the attendance dynamics of the participants. Another example of relevant covariates in the context of student learning is the topic of each lesson or problem being studied, as each student has topic specific learning strengths and weaknesses. Last but not least, consecutive time points are highly correlated in typical time-series contexts. Therefore, knowing the previous state of the variables and appropriately handling inertia is vital for making good predictions.

Our conditional latent tree model, takes into account the effect of all the above factors
for predicting high-dimensional time-series. The effect of the covariates and previous time points is captured via Conditional Random Fields (CRF’s). More specifically, conditioned on exogenous covariates and previous time points, the dependency structure among the variables is modeled via a latent tree whose hidden nodes represent the unobserved hidden groupings in the data. Therefore, CLTM represents the joint distribution of the observed and latent random variables which factorizes according to a Markov latent tree conditioned on the covariates and previous time points. This model is versatile in its ability to model group structure and provide the ability to carry out exact inference through the simple belief propagation (BP) algorithm [24] that makes the model potentially scalable. We provide a statistically efficient algorithm for learning the structure of the latent tree, and estimate the parameters of the model using a Maximum Likelihood (ML) approach. Therefore, our goal in the sequel is to learn unobserved groups of similar behavior and incorporate them into prediction of the evolution of high dimensional time-series conditioned on some relevant covariates.

It is worth mentioning that like many common alternatives (e.g., hidden Markov models), the latent tree structural assumption used here is a reasonable approximation to the true dependence structure underlying the random variables arising from typical social settings. This approximation is obviously more realistic than a purely independent model, but also captures subtle hierarchical features that are missed or obscured by alternatives such as latent state models. Although we do not claim that latent tree structures are perfect representations of the myriad sources of dependence in large, complex social systems, we thus do regard the CTLM as an effective “middle ground” between simple independence and/or latent state models and difficult-to-scale models with unbounded dependence (e.g. full temporal ERGMs with endogenous attributes [35]). As we show here, predictive results obtained by applying these models to several real-world data sets provide further evidence for their efficacy.

The rest of this chapter is structured as follows. In Section 2.1 we give a summary of our
results and in Section 2.2 we provide an overview of the related work in the domain. Section 2.3 describes the proposed CLTM framework and in Section 2.4 we introduce the model estimation steps. We present the prediction results of our method applied to three real world data sets and give a detailed analysis of the obtained results in Section 2.5.

2.1 Summary of Results

In this chapter, we introduce CLTMs and propose a framework for learning them efficiently. The most important feature of this class of models, is their ability to capture latent groupings underlying the data and exploiting these latent groupings to model dynamic time series. This framework has the potential to be applied to large scale data sets. We first estimate the latent group structure among the variables, and then learn the parameters of the CLTM, which describe quantitatively how the hidden variables affect the observed outcomes. We then employ CLTM’s to efficiently track the evolution of time series.

We show that our model provides valuable and interpretable information about the unknown groupings and their effect on the time evolution. We also learn the effects of observed covariates such as past observations and seasonality effects. We demonstrate how the discovered latent groupings can be exploited for downstream prediction of related variables. For instance, we incorporate the learnt latent groups of nodes as covariates for predicting social interactions in large-scale social networks.

Our learning algorithm leverages recent results for learning latent tree graphical models [24]. However, this is not a direct application since we have time varying data, and also covariates [66], which is not the case in earlier works on latent tree models [24]. We use a parametric exponential family model formulation to account for the time dependence among consecutive samples in the series and exogenous covariates, and propose an efficient learning
strategy for estimating the parameters of the exponential family distribution.

We apply our approach to three challenging real-world datasets involving students’ performance in a psychology Massive Open Online Course (MOOC), Twitter users’ activities and interactions, and windsurfers’ participation in and interactions during activity on a southern California beach. In all these data sets, our goal is to predict the dynamics of the users (either the students in the class or the network attendees) for which we need to extract relevant covariates that will be used in CLTM.

For the MOOC data, for example, in order to acquire predictive covariates for student prediction, we first learn a conditional latent tree model over the knowledge components. Each question answered by any student incorporates a certain knowledge component. By learning the CLTM over the knowledge components, we can automatically find hierarchical groups of concepts that are learnt similarly by students. We demonstrate that the learned tree structure captures interpretable groupings. For instance, knowledge components related to different anxiety disorders are grouped together. Note that we only use these labels of knowledge components for validation, and not during the learning phase. We then incorporate these knowledge groupings as covariates in a CLTM used for tracking the learning of individual students over time. In this CLTM over the students, each observed node indicates the performance of one student on questions answered daily. Our model automatically learns groups of students who demonstrate similar evolution of learning behavior. Such information can be valuable to an instructor, since it gives him/her the ability to target different groups of students, and tune the instruction accordingly. Our approach is in contrast to earlier modeling frameworks for this MOOC dataset, which fit a different latent variable model for each student separately, treating the students independently, in order to model the learning progress [15].

We quantitatively compare the prediction of student performance under our method with a chain CRF model in Table 2.3, in which the chain is over time and the students are treated
as independent time sequences similar to [15]. We observe a significant improvement in predicting the student performance. Similarly, we also demonstrate a strong improvement on Twitter and beach data for predicting the conditional presence of vertices and edges over time. This is especially relevant, since these datasets are highly sparse with a small number of participants at any given time. Moreover, we observe that our method has higher improvement on the Twitter dataset compared to the beach data, since the beach dataset has covariates that are carefully collected by a team of sociologists. Thus, our method is highly effective in predicting multivariate time series across multiple domains, particularly where covariate information is present but limited.

2.2 Related Work

Previous works on multi-variate time series typically do not consider latent groups, e.g. [33]. This results in too many unknown parameters that leads to the problem of overfitting and computational intractability in the high dimensional regime. The alternative is to first learn the groups through standard clustering techniques such as agglomerative clustering [43], and then use them as covariates for prediction. However, this two step process is not optimal for prediction. In contrast, our CLTM is a statistical model which simultaneously learns the groups and their effect on time evolution, leading to efficient performance.

Another interesting line of related work considers community models such as stochastic block models and mixed membership models [4, 29, 83, 84, 86] for modeling the unknown vertex groups. However, these models only consider the edge data and do not incorporate node state information and exogenous factors. In our datasets, we also have node activity information (such as number of tweets by a user), and we exploit this information to learn about the unknown node groups. We then incorporate the group structure for learning the edge dynamics. Furthermore, the aforementioned works mostly assume data samples to be
independent and identically distributed (except for [29] or [27]), whereas we consider time
varying data.

CLTM belongs to the class of Conditional Random Fields (CRF). Various CRFs have been
considered before, e.g. CRFs on linear chains [72], trees [19, 63], grids [44], and so on. However, only a few works address the issue of structure learning of CRFs, e.g. [19, 70, 78]. Moreover, not many publications assume CRFs with latent variables, e.g. [63] has latent
variables, but with a fixed structure. Our work, on the other hand, does not make such
strong assumptions. We learn the latent tree structure through efficient methods and also
incorporate covariate effects, leading to highly effective models in practice.

2.3 Model

Let us denote random variables with $y^{(t)}_i \in \mathbb{R}$ where $i = 1, 2, \ldots, n$ indicates the index of the
random variable and $t = 1, 2, \ldots, T$ is the time index. We use the terms “random variable”
and “node” interchangeably in this chapter, as the random variables can be represented as
nodes in a dependency graph. An example of such a dependency graph is shown in Fig 1.1.

Each random variable’s behavior is dependent upon other random variables’ behavior, as well
as a set of covariates. Examples of covariates include network users’ group memberships,
seasonal effects (e.g. the day of the week). There are three types of covariates we consider
in this chapter: the individual covariates which are node specific, e.g. membership of a
specific node in an observed group (green nodes in Fig 1.1); the shared covariates which
indicate the dependency among the nodes; and the global covariates which simultaneously
affect every node in the conditional latent tree, such as seasonality (the red node in Fig 1.1).

Let $x^{(t)}_i \in \mathbb{R}^{(1 \times Kn)}$ indicate the set of node specific and global covariates, and $x^{(t)}_{ij} \in \mathbb{R}^{(1 \times Ke)}$
indicate the shared covariates. In this case, $Kn$ indicates the number of covariates of $y^{(t)}_i$.
and $K_e$ indicates the number of shared covariates between nodes $y_i^{(t)}$ and $y_j^{(t)}$.

### 2.3.1 Conditional Latent Tree Models (CLTM)

Our goal is to perform structured prediction when there are temporal dynamics in the data. Consider an online social network such as Twitter as an example. Let the network users be the random variables whose Tweeting activity is tracked over time. If two users are similar (say, both belong to a subgroup with similar demographic and social characteristics), it is more likely for them to have similar activity. Additionally, we claim that the users’ attendance behavior depends on their previous activity, the behavior of other users and some relevant covariates. Note that throughout this chapter, previous observations are contained in the prediction model as a subset of the covariates. Therefore, we learn a latent tree dependence model over the users conditioned on the covariates and previous observations and we predict users’ attendance dynamics according to the learned structure. As another example, we predict the performance of students in a course. Based on the students’ performance on their exams throughout the semester we find groups of students who share similar learning behaviors using CLTMs. We will see that finding these similarities and hidden groupings can greatly help in predicting students’ learning performance.

Let us first look at what the latent tree structure looks like and why we are assuming such a dependence structure. Consider the Twitter network in which $y_i$’s are the Tweeting activity of network users, or an online education system in which $y_k$’s are the performance of the students on various course material such as problems and quizzes. The latent nodes in the tree are denoted by $h_j$ where $j = 1, 2, \ldots, m$. They represent hidden groupings in the dependence structure of the random variables $y_i$. Let $z_k$ be the union of the observed nodes $y_i$ and latent nodes $h_j$ where $k = 1, 2, \ldots, n + m$. Let us denote the latent tree by $T_d = (Z_d, E_d)$ where $Z_d$ indicates the node set consisting of all the random variables and $E_d$ denotes the
edge set containing the edges of the latent tree. There are two main advantages in making latent tree structural assumptions. Firstly, a latent tree allows for more complex structures of dependence compared to a fully observed tree - specifically, it allows for latent groups of individuals whose behaviors jointly covary. Secondly, inference on it is tractable, and therefore, it will be scalable.

Once the dependency structure is achieved, we should specify the generative distribution that the data is drawn from. The distribution of the random variables in CLTM belongs to the exponential family conditioned on observed covariates $X$. Covariates are observable external factors that affect the dynamics of the data. For example, in the Twitter network, relevant covariates are seasonality, regularity of network users, their popularity, and their previous activity. Fig 1.1 demonstrates the CLTM structure conditioned on the covariates $X$. As one can see in the figure, the joint structure of the observed and hidden variables are that of a tree conditioned on the covariates. Now let us give more details about the data distribution.

Exponential family distributions are a broad family of distributions including the normal, Gamma, Poisson and many other distributions [32]. Conditioned on covariates $X$, the distribution of $Z$ over tree $\mathcal{T}_d$ is given in Equation (2.1).

$$\Pr(Z|X, \theta) = \exp\left(\sum_{k \in Z_d} \phi_k(X, \theta)z_k + \sum_{kl \in E_d} \phi_{kl}(X, \theta)z_kz_l - A(X, \theta)\right), \quad (2.1)$$

where $A(X, \theta)$ is the term that normalizes the distribution, also known as the log partition function. $\phi_k(X, \theta)$ and $\phi_{kl}(X, \theta)$ indicate the node and edge potentials of the exponential family distribution, respectively. Let’s assume for the sake of simplicity that the potentials
are linear functions of the covariates and previous observation as shown below.

\[ \phi_k(X, \theta) = c_0 + c_1 x_{1,k} + \ldots + c_{K_n} x_{P,k}, \]  
(2.2)

\[ \phi_{kl}(X, \theta) = e_0 + e_1 x_{1,kl} + \ldots + e_{K_e} x_{P,kl}, \]  
(2.3)

Learning the graphical model involves two steps: learning the dependence structure over the nodes, and estimating the probability distribution the data is generated from. In the following paragraphs we provide detailed description of these two steps.

The details of estimating the distribution is presented in Section 2.4.2. It should be noted that the random variables in this model can either be discrete or continuous and we will cover both variables in the structure learning and parameter estimation sections.

### 2.3.2 Exploiting Inferred Hidden Groups

A problem of frequent current interest is that of modeling the dynamics of data on social interactions, as e.g. occurs in online social networks. E.g., in the context of the Twitter network a social tie (an edge) can represent any form of communication between the users while they are active. We learn these dynamics by using the information from the learnt CLTM model for node activity. It is reasonable to claim that an edge cannot be formed unless both parties that form it are present in the network. In that case, if we correctly predict the active users in the network, our chances of predicting edge dynamics increase. Therefore, conditioned on the model learned for the users we predict users’ interaction patterns by regressing on a set of relevant edge covariates and the inferred state of the hidden variables from the CLTM model.

Let \( W \) represent the set of edges (social ties), and \( w_{ij} \) denote the presence/absence of an edge between users \( i \) and \( j \). Intuitively, by regressing on the inferred hidden states, we can
incorporate the latent group dynamics into prediction of edges. Assuming independence between the edges, the generative model of the edges given user activity, the inferred state of the hidden variables obtained from the CLTM model, and relevant edge covariates $X$ is:

$$\Pr(W | Z, X) = \prod_{(i,j)} B \left( w_{ij} \mid \logit^{-1}(\xi(Z, X)) \right),$$

(2.4)

where $\xi(Z, X)$ is a linear function of the the edge covariates $X$, consisting of past network information, current node states $Y$ and inferred states of the hidden variables of the CLTM as shown in Equation (2.5). $B$ is the Bernoulli distribution and $\logit^{-1}$ is the logistic function. Note that re-use of variable $X$ for the edge covariates is due to notational simplicity and in practice, the edge and node covariates are not the same. $X_{k,ij}^{(t)}$ in the equation below denotes the $k^{th}$ covariate for the edge formed between nodes $i$ and $j$, at time point $t$.

$$\xi(Z, X) = d_0 + d_1 X_{ij}^{(t)} + \cdots + d_{K_{ec}} X_{K_{ec},ij}^{(t)} + d_n Z.$$  

(2.5)

### 2.4 Model Estimation

In this subsection, we describe our algorithm for estimating the underlying model parameters given the observations. Model estimation can be divided into two steps: (1) structure learning and (2) parameter estimation. For structure learning, the target structure is a latent tree that is efficiently learned using a notion of information distances (Equation (2.8)). For parameter estimation, we learn the parameters of the exponential family distribution defined in Equation (2.1) using maximum likelihood. Specifically, since latent nodes are present in the structure, we use Expectation Maximization(EM) for parameter estimation. We describe these two steps in detail in the following subsections.
2.4.1 Structure Learning

A large number of scalable structure learning algorithms for latent tree models have been proposed by the phylogenetic community for learning latent tree models. Among the available approaches, we build upon RG and LocalCLGrouping [24] that provide provable computational efficiency guarantees. These algorithms are based on a measure of statistical additive tree distance metric $d$ (a.k.a information distance) which is given below for discrete random variables:

$$d_{ij} := -\log \frac{|\det \mathbf{J}(y_i, y_j)|}{\sqrt{\det \mathbf{M}(y_i) \det \mathbf{M}(y_j)}},$$  \hspace{1cm} (2.6)

where $\mathbf{J}(y_i, y_j)$ is the joint probability matrix between $y_i$ and $y_j$ and $\mathbf{M}(y_i)$ is the diagonal marginal probability matrix of node $y_i$. In practice, we employ empirical estimates of $\mathbf{J}(y_i, y_j)$, $\mathbf{M}(y_i)$ and $\mathbf{M}(y_j)$ based on data samples. For continuous Gaussian random variables the information distances are the log of their correlation coefficient:

$$d_{ij} := -\log \frac{\text{Cov}(y_i, y_j)}{\sqrt{\text{Var}(x_i) \text{Var}(x_j)}},$$ \hspace{1cm} (2.7)

The distance measure given in Equations (2.6) and (2.7) are not valid for conditional settings, which is the case in our study. However, since the tree structure is fixed through time, we can define the notion of conditional distance given in Equation (2.8) as the weighted average of all the individual distances given the covariates.

$$[d_{ij}|X] := \sum_{k=1}^{K_y} w_{k,ij}d_{k,ij},$$  \hspace{1cm} (2.8)
where \( w_{k,ij} \)'s are the empirical probability matrices of covariate pairs \((X_{k,i}, X_{k,j})\), such that \( \sum_{\text{states}} w_{k,ij} = 1 \), \( K_n \) is the total number of observed covariates for each node, i.e.:

\[
 w_{k,ij} = \Pr(x_{k,i}, x_{k,j}).
\]  

(2.9)

For instance, if the covariates \( x_i \) and \( x_j \) are binary random variables then their joint has 4 possible states and therefore \( w_{k,ij} \) is a \( 4 \times 1 \) vector whose entries sum to one.

Individual distances \( d_{k,ij} \) for discrete and continuous Gaussian random variables are given in Equations (2.10) and (2.11), respectively. It is worth noting that the additive property of the distance measure will be preserved, due to the fact that the tree is fixed over time and each individual’s distance is additive over the tree.

For discrete variables we have:

\[
d_{k,ij} := -\log \left( \frac{|J(y_i, y_j|x_{k,i}, x_{k,j})|}{\sqrt{M(y_i|x_{k,i}, x_{k,j})M(y_j|x_{k,i}, x_{k,j})}} \right).
\]  

(2.10)

This means that if the covariates are binary then \( d_{k,ij} \) will have four states of “00”, “01”, “10” and “11” for the \((x_{k,i}, x_{k,j})\) pair. We then weight each state by the empirical probability state of each covariate pair and average over all covariate pairs and all \( k_n \) covariates. This conditional distance measure could be used in, \textbf{LocalCLGrouping} algorithms [24] to learn latent graph structure from data.

For continuous variables we have:

\[
d_{k,i,j} := -\log \left( \frac{\mathbb{E}(y_i y_j|x_{k,i}, x_{k,j})}{\sqrt{\mathbb{E}(y_i^2|x_{k,i}, x_{k,j})\mathbb{E}(y_j^2|x_{k,i}, x_{k,j})}} \right).
\]  

(2.11)
2.4.2 Parameter Estimation using Expectation Maximization

Once we find the latent tree structure using CLGrouping, we use maximum likelihood to estimate the parameters of the data distribution given in Equation (2.1) based on the structure. We use Expectation Maximization (EM) to maximize the likelihood function due to the latent nodes in the structure. EM is an iterative algorithm that iterates between the two so-called E-step and M-step. It maximizes the lower bound of the likelihood function in each iteration based on the parameters estimated in the previous iteration. The lower bound is the expected complete data log likelihood function which is presented in Equation 2.13.

In order to give a sketch of the EM algorithm formulation, we first present the log likelihood function over the learned latent tree $T_d = (Z_d, E_d)$:

$$
\ell(\theta|X, Z) = -\sum_{t=1}^{T} \left( A(\theta, X^{(t)}) + \sum_{k \in Z_d} \left( \sum_{l \in E_d} \phi_{kl}(X^{(t)}, \theta) z_k^{(t)} z_l^{(t)} \right) \right) + \sum_{t=1}^{T} \left( \sum_{k \in Z_d} \phi_k(X^{(t)}, \theta) z_k^{(t)} \right) (2.12)
$$

Variable $Z$ is a union of the observed nodes $Y$ and unobserved nodes $H$. Therefore, we cannot maximize the above quantity directly. In order to achieve maximum likelihood we compute the expected complete data log likelihood function given below:

$$
\mathbb{E}_{H|X,Y}(\ell(\theta|X, Z)) = \sum_{t=1}^{T} \left( \sum_{k \in Z_d} \phi_k(X^{(t)}) \mathbb{E}_{H|X^{(t)}, Y^{(t)}}(z_k) \right) + \sum_{t=1}^{T} \left( \sum_{kl \in E_d} \phi_{kl}(X^{(t)}, \theta) \mathbb{E}_{H|X^{(t)}, Y^{(t)}}(z_k z_l) \right) - \sum_{t=1}^{T} \left( A(X^{(t)}, \theta) \right). (2.13)
$$

$\mathbb{E}_{H|X,Y}$ is computed from the E-step and then the M-step maximizes Equation (2.13) through gradient descent.
2.5 Experiments and Results

In order to show the capabilities of our CLTM method, we use 3 different real world datasets. The datasets come from two categories, namely educational data and network data. The educational data is a Massive Open Online Course (MOOC) dataset [41] for an online course on psychology offered in Spring 2013. The Network data consists of user interactions and attendance in two social networks. One is an online social media of Twitter and the other one is a network of windsurfers that surf on a beach in Southern California. It should be noted that acquiring large scale dynamic data with long enough time duration that has a reasonable density is a challenging task and most of the available data sets have a very short time duration.

For performance evaluation, we qualitatively observe the estimated tree structures for educational data as the nodes are labeled and can be interpreted. Quantitatively we carry out cross-validation. We learn the model based on the training data and predict node/edge evolutions on the test data. A set of scores, which will be defined in the following paragraph, are used for performance evaluation. We compare our CLTM model with the Chain CRF (CCRF) model in all the experiments in which the chain is over time. We use the same set of covariates for CLTM and CRF for a fair comparison.

![Diagram of vertex activities in datasets](image)

Figure 2.1: Vertex activities in the datasets with respect to time. The horizontal axis on all the plots show time and the vertical axes show vertex IDs in each dataset. White in Figure 2.1a indicates that a student has chosen the correct answer and black means that the student has either chosen an incorrect answer or has shown no activity. In Figures 2.1b and 2.1c, white indicates active and black indicates inactive users in the network.
**Prediction Scores**  We use the covariates and the estimated parameters of the model to perform one-step-ahead prediction of the data at each time point. At each time point, we take the empirical mean of $M$ samples we predict from the estimated model and compare the samples to the ground truth (test data). Repeatedly we carry this out at each time point $t = 1, \ldots, T$.

We define the following measures to assess the performance of the algorithm.

- **CP**: the *conditional presence (recall)* which measures the accuracy of predicting a node as active given that the node is indeed active (encoded as 1).
- **CA**: the *conditional absence*, which computes the accuracy of predicting a node’s absence (encoded as 0).
- **EP**: *conditional edge presence (recall)*, where edge indicates a social tie which is often-times nodes’ interactions. Presence is encoded as 1.
- **EA**: the *conditional edge absence*. Absence is encoded as 0.

In the node prediction problem, it is challenging to predict **CP** since the data is highly sparse. Performing well in predicting rare appearances of the data is important. The prediction accuracy is defined as the percentage of nodes predicted correctly out of the $n$ observed nodes. In the edge prediction, especially in the case of network data where we want to also track social tie dynamics, we evaluate the prediction accuracy defined as the percentage of edges predicted correctly out of the $e$ possible edges. At the same time, the prediction of absence should not be degraded significantly.

Recall that $y_i^{(t)}$ is observed node $i$ and $w_{ij}^{(t)}$ is the observed social tie between nodes $i$ and $j$ at time point $t = 1, \ldots, T$. $M$ is the number of predictions drawn from the estimated model. Let $\hat{y}_{i,k}^{(t)}$ be our $k^{th}$ prediction of node $i$ and $\hat{w}_{ij,k}^{(t)}$ be the $k^{th}$ edge prediction between nodes
\(i\) and \(j\) at time point \(t\). Let \(y_{\text{pred}}^{(t)}\) denote the predicted node set at time point \(t\). We define the prediction scores as:

\[
CP(t) := \frac{1}{nM} \sum_{k=1}^{M} \sum_{i=1}^{n} \mathbb{I}(\hat{y}_{i,k}^{(t)} = 1 | y_{i}^{(t)} = 1),
\]

\[
CA(t) := \frac{1}{nM} \sum_{k=1}^{M} \sum_{i=1}^{n} \mathbb{I}(\hat{y}_{i,k}^{(t)} = 0 | y_{i}^{(t)} = 0),
\]

\[
EP(t) := \frac{1}{eM} \sum_{k=1}^{M} \sum_{i,j \in y_{\text{pred}}^{(t)}} \mathbb{I}(\hat{w}_{ij,k}^{(t)} = w_{ij}^{(t)} | w_{ij}^{(t)} = 1),
\]

\[
EA(t) := \frac{1}{eM} \sum_{k=1}^{M} \sum_{i,j \in y_{\text{pred}}^{(t)}} \mathbb{I}(\hat{w}_{ij,k}^{(t)} = w_{ij}^{(t)} | w_{ij}^{(t)} = 0),
\]

where \(n\) is the total number of observed nodes, and \(e\) is the total number of possible edges. \(\mathbb{I}(.)\) is the indicator function who outputs 1 if its input is true.

In addition to the above metrics, we introduce two other measures called relative difference of average, RDA, and relative difference of median, RDM, that indicate the average and median of the relative improvement of CLTM compared to the baseline model consisting of Chain CRF (CCRF), respectively.

\[
RDA := \frac{\sum_{t=1}^{T} CP(t)_{\text{CLTM}} - \sum_{t=1}^{T} CP(t)_{\text{CCRF}}}{\sum_{t=1}^{T} CP(t)_{\text{CCRF}}},
\]

\[
RDM := \frac{\text{median}_t(CP(t)_{\text{CLTM}}) - \text{median}_t(CP(t)_{\text{CCRF}})}{\text{median}_t(CP(t)_{\text{CCRF}})},
\]

The higher the values of RDA and RDM are, the better our performance is compared to Chain CRF.
Figure 2.2: (a) Presentation of knowledge component latent tree result. In this example 2 groups $g_1$ and $g_2$ are recovered. (b) Student latent tree structure learning using KC latent tree’s grouping as covariates. If student $s_1$ answers problems from knowledge component $k_1$, then $s_1$’s node specific covariate $x_1$ takes value according to $k_1$’s grouping results from (a), which is $g_1$. $x_g$ is the global covariate.

## 2.5.1 Educational Data

The data, gathered from a psychology course in the Stanford Open Learning Library\textsuperscript{1}, records students’ problem solving outcomes, which are grouped as “correct” and “incorrect”. The problems that the students answer come in 226 knowledge components (KCs). These knowledge components refer to the different concepts covered in the class throughout the semester. The multivariate high-dimensional time series data spans 92 days and involves 5,615 students with a total number of 695 problems (2,035 steps). Each problem consists of different steps/stages. There are a total number of 2,493,612 records of students’ interactions, which are student’s attempts to solve problems, with the server. The course material and problems can be accessed in any order throughout the course. Students’ learning behavior and performance are tracked: correct answers are encoded as 1s and the incorrect ones 0s. The ultimate goal is to track the learning of students and find similar groups of students that behave similarly in terms of learning. First we choose a subset of the students with 244 members.

\textsuperscript{1}available on CMU datashop [41]
We have two goals for prediction: (1) to learn a latent tree model over the KCs and (2) to predict student learning using KCs as covariates. We first learn a latent tree structure over the KC model that helps us cluster the concepts. Then we use these clusters of KC’s as relevant covariates to learn a CLTM model for the students’ learning behaviors. This process is shown in Figure 2.2, where figure (a) shows the latent tree that clusters the KC’s, \(k_i\), and figure (b) indicates the CLTM structure whose covariates are the clusters of the KC’s extracted from figure (a). The ultimate goal is to track the learning of students and find groups of students that behave similarly in terms of learning.

**Latent Tree Models for Knowledge Components** The 226 knowledge components have human labels which we use for qualitative interpretation of the learned structure (see Figure 2.3(a) for subset of latent tree learnt, detailed interpretation is in the latter paragraph). The nodes in the learned structure are the KCs and the edges indicate the co-occurrence of the KC pairs in a day for the same student. Using daily aggregated time points, we consider all students’ total numbers of correct and incorrect answers for each of the 226 KCs within a day. The counts are transformed to a ratio by normalizing with the total number of problem solved and are then transformed to approximate Gaussian by taking the square root of the ratio. The covariates that we use for learning this structure are some attributes of the data, such as seasonality and previous time points’ aggregated outcomes on the KC.

Hierarchical clustering is realized on the knowledge components. The complete learned structure is available online for an interested reader to explore. We demonstrate two parts of the learned structure in Figure 2.3. The blank nodes demonstrate the hidden variables learnt, whereas the colored nodes demonstrate the knowledge components. Taking a closer look at Figure 2.3 we can see that knowledge components related to relationships and hap-

\(^2\)The cluster is achieved on the learnt latent tree using standard graph partitioning algorithms \(^3\)https://forougha.github.io/CLTM/eduTree.html
piness (red), personality(black), sexual attractions(purple), eating disorders(golden), and anxiety(blue) are clustered together. Thus, we find that our recovered latent tree has considerable face validity with respect to known relationships among topics. Note that none of these labels are input to algorithm, and we require no labels in our unsupervised algorithm.

**Predicting Student Learning using learnt groups of Knowledge Components as Covariates** Now that we have the clusters each KC falls into, we use them as covariates along with seasonal information and past observations of the network to track student learning. A subset of students who loyally stayed through the semester (244 members) was selected. The data is again binned daily and each sample is the ratio of correctly answered problems (aggregating over all KCs) over the total number of problems answered for a student within a day. We threshold the values to make the data binary. The extracted samples are shown in Figure 2.1a where the horizontal axis indicates time and the vertical axis indicates the students’ attempts to answer questions.

<table>
<thead>
<tr>
<th></th>
<th>CP (train)</th>
<th>CP (test)</th>
<th>CA (train)</th>
<th>CA (test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDA</td>
<td>2.6%</td>
<td>52.96%</td>
<td>1.66%</td>
<td>2.01%</td>
</tr>
<tr>
<td>RDM</td>
<td>2.68%</td>
<td>56.37%</td>
<td>1.66%</td>
<td>1.99%</td>
</tr>
</tbody>
</table>

Educational data covariate coefficients for students is shown in Figure 2.4. As illustrated, cluster 3 which is KC groups of “active early school’s contributions to psychology” and “physical sci contributions to psychology” is the most relevant covariate with the highest weight, and smart or hard working students are relevant through those problems. Cluster 9 and 10 which are groups of KCs on “apply important questions”, “goals psychology real world” and “philosophy contributions to psychology” are also highly relevant covariates. However, covariates such as KC’s on “brain neuroplasticity”, “methods studying the brain” and “research validity bias ”, indicated as cluster 7 in the figure are less relevant in terms of distinguishing student’s ability to answer questions in those category correctly. It is also
Figure 2.3: Subgroups of the estimated knowledge component latent tree. Nodes are colored by the topics of knowledges components. The red nodes talk about relationships and happiness, the blue nodes are personality related, the purple nodes are about sexual attractions, and the yellow nodes focus on the eating disorders. On the right subgroup, we see a huge cluster of anxiety related concepts, denoted by green, from anxiety disorder all the way to serious schizophrenia symptoms.

Figure 2.4: Covariate coefficients/weights learnt for student learning prediction in education data.

interesting to notice that Sunday and Monday happens to be the time that students are most reluctant to work during the week. Note that the coefficients for seasonality are negative, however this does not imply that the data has negative correlation with seasonality. The reason is that these coefficients are approximately equal, and since at each time point only one of these variables are on, we think of them as a bias term. In other words, seasonality is down-weighting other covariates’ effects.

Prediction accuracy curves vs. time, indicating one-step-ahead prediction is presented in Figure 2.5. As it is depicted, our algorithm performs significantly better than Chain CRF in the test dataset.
Figure 2.5: conditional presence and absence boxplots vs. time for education test data

Finally, in Figure 2.6, we demonstrate our ability to automatically find students of similar learning abilities and track their learning efficiently over time. Among the extracted student groups we choose a group of “strong” learners, who accurately answered questions over the time, and “weak” ones who were mostly inaccurate in their answers (or inactive and did not answer any questions). We select the group of “strong” students who were 1.15 times the average overall performance (over the entire period), and in addition, their neighbors in the conditional latent tree whose information distance is less than the mean distance in the tree. In total, we obtain 21 students for this group. Similarly, for the “weak” student group, we consider students who are less than 0.85 times the average performance and also their neighbors in the tree, as described before. In total, we obtain 26 students for this group. We plot the actual performance of these two groups on training and test time periods, as well our predicted performance and the Chain CRF’s predictive performance. We see that we closely track the actual performances of these two groups. Also, Chain CRF suffers severely from overfitting as illustrated in this figure. Note that the actual performances of the two groups are significantly different, with the stronger group having much better performance compared to the weaker one.

Thus, our method can automatically find groups of students with similar learning behavior. This can be valuable information for instructors, since they can target these different groups and provide personalized attention of different forms.
Figure 2.6: Performance of group of “strong” and “weak” learners over the semester. First 60 time points are training samples and rest are test. We demonstrate actual performance in answering questions as well as the predictions from the learnt CLTM and CCRF.

2.5.2 Network Data

This data is gathered from two social networks of beach goers and Twitter users. In Section 2.5.2 we present some statistics of the data and give an overview of both datasets. In Section 2.5.2 we describe the Twitter data and presents its results and compare it to the baseline and in Section 2.5.2 we talk about the beach dataset and present its results.

Network Data Description

The vertex activities for the network datasets are shown in Figures 2.1b and 2.1c. The horizontal axis indicates time index and the vertical axis indicates vertex index. White represents presence and black represents absence. Both sets of data have similar covariates: we use the previous state of the vertices as well as the number of triads (triplets of nodes which interact with each other) they were engaged in, the previous day as covariates. We also have a covariate that indicates whether the attendee is a regular participant in community activities at the beach, as assessed ethnographically over a period of several months prior to the data collection window. The effect of daily seasonality is also captured by a set of daily fixed effects. We allow for each node to have its own bias indicating that different nodes have different attendance tendency. The positive bias indicates a regular surfer in the beach goers.
or a regular user among the Twitter users. The negative bias indicates an irregular surfer or user. Note that in the case of Twitter data, a social tie is defined as direct messaging in Tweets and for the Beach data, a social tie indicates the interaction of surfers while they are at the beach. An overview of specifications of dataset is given in Table 2.2. Note that AV and AE denote the average node and edge presence and are indications of the sparsity of the data.

Table 2.2: Data specification and number of used covariates. **NN**: size of the vertex set, **AV**: Average Vertex appearance, **AE**: Average Edge appearance **NC**: number of node covariates and **EC**: number of covariates for predicting edges.

<table>
<thead>
<tr>
<th>Data</th>
<th>NN</th>
<th>AV</th>
<th>AE</th>
<th>NC</th>
<th>EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Education</td>
<td>244</td>
<td>37.17%</td>
<td>N/A</td>
<td>29</td>
<td>N/A</td>
</tr>
<tr>
<td>Twitter</td>
<td>333</td>
<td>8.49%</td>
<td>0.036%</td>
<td>9</td>
<td>36</td>
</tr>
<tr>
<td>Beach</td>
<td>94</td>
<td>16.66%</td>
<td>0.644%</td>
<td>11</td>
<td>43</td>
</tr>
</tbody>
</table>

Table 2.3: Prediction scores for both network data. Conditional presence (CP), conditional edge presence (EP), conditional absence (CA) and conditional edge absence (EA) are defined in Equation (2.14-2.17). Relative difference of average (RDA) and relative difference of median (RDM) are defined in Equation (2.18) and (2.19).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>CP</th>
<th>EP</th>
<th>CA</th>
<th>EA</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDA</td>
<td>Twitter</td>
<td>37.66%</td>
<td>243.73%</td>
<td>-2.45%</td>
</tr>
<tr>
<td>RDM</td>
<td>Twitter</td>
<td>42.31%</td>
<td>565.98%</td>
<td>-2.85%</td>
</tr>
<tr>
<td>RDA</td>
<td>Beach</td>
<td>10.61%</td>
<td>60.66%</td>
<td>-1.98%</td>
</tr>
<tr>
<td>RDM</td>
<td>Beach</td>
<td>14.10%</td>
<td>60.81%</td>
<td>-2.00%</td>
</tr>
</tbody>
</table>

**Twitter Network**

We have collected this dataset by observing the tweeting activity of 333 individuals participating in a discussion on an emergency management topic #smemchat for a period of 6 months \(^4\). The observation period starts from Dec 1st 2013 to Apr 29th 2014. We have a total number of 2313 snapshots of the network which are binned into 26 weekly bins. The vertex set consists of all the nodes that participated in the topic during the observation period.

\(^4\)We refrain from providing the link to the data to preserve anonymity
period and vertex presence is indicated by status updates. Vertex activity peaks on Fridays. Interactions are defined as direct messages among the users, therefore the network is very sparse in terms of user interactions.

A user in the Twitter network is defined as a “regular” if he or she appears on the network more frequently than a specified threshold. A popular user is one whose number of followers is greater than the median of the number of followers of all users. A user is fav if their number of favorites is greater than the median of the number of favorites of all the network attendees. Other covariates remain the same as discussed in the data overview.

The weights learned for the covariates described is shown in Figure 2.7a. As illustrated, regularity of the user and its past time activity are the most relevant covariates with the highest weights. Seasonality covariates are the week number in each month. As expected different weeks within each month behave approximately similar. Also, as it was the case with Educational data, seasonality covariates are down-weighting the other covariates’ effects rather than having negative correlation with data.

As mentioned in Section 2.3.2, we are also interested in tracking social interaction dynamics conditioned on the predicted node state of the network using the inferred state of the hidden variables in the CLTM model. The idea is that an edge cannot form unless both nodes that form it are present. Following the same reasoning for prediction, we limit edge prediction to the predicted node set; which drastically reduces the sparsity of the edge set. The covariates used for edge prediction are seasonality effects, previous state of the network, number of present nodes in the previous day, number of K-cycle structures \[52\], the presence of the edge in the previous day, whether the edge is between regular-regular, regular-irregular or irregular-irregular network users. The coefficients learned for these covariates are shown in Figure 2.8a. We find that regular/regular interactions (interactions between regular users) have a high weight, whereas the weight of an irregular/irregular interaction is very low indicating that regular nodes are more likely to talk to one another compared to irregular
nodes. Also as expected, regular/irregular interactions are somewhere in between the two. Another interesting point is that if one of the nodes that form a specific edge are a frequent and regular attendee of the network, they will have a very important role in prediction. Previous time state of the network is also another thing that highly affects prediction.

Figure 2.8b shows the learnt weights for the covariates which consist of the inferred hidden variables from the vertex conditional latent tree model. In other words, after the CLTM is learnt over the vertices, the configuration of hidden nodes is inferred through belief propagation. These inferred values are then incorporated as covariates into the edge model as follows: for predicting each edge, we incorporate the configuration of hidden variables which are parents of the endpoints of the corresponding vertices that form the edge. In Figure 2.8b, we see that different hidden nodes affect the presence of edges to different extents, thus indicating that different groups have varying tendencies for forming edges.

Prediction accuracy curves vs. time, indicating one-step-ahead prediction for both node prediction and edge prediction are also presented in Figure 2.9, respectively. As it is depicted in the Figure 2.9a, improved vertex prediction accuracy boosts edge prediction performance, since edge prediction is conditioned on the predicted node set. Incorporating the inferred state of the hidden variables of the CLTM model is another important factor that increases prediction accuracy compared to the baseline Chain CRF without the inferred states. As illustrated in Figure 2.9b, CLTM improves CP while maintaining a good CA, resulting in a 243.73% improvement in average EP.

**Beach Network**

This data contains a dynamically evolving network of interpersonal communications among individuals congregating on a beach in Southern California observed over a one-month period [28]. The vertex set in this network is the windsurfers appearance on the beach in this 31
day period and the edge set is composed of their interpersonal communications recorded in the data set.

The network was tracked two times a day, for 31 days from Aug 28, 1986 to Sept 27, 1987 by Freeman et. al. There is a total number of 94 windsurfers who are divided into two groups of *regulars* (with 54 members) and *irregulars* (with 41 members). The groups of *regulars* is further categorized into two groups of “Group 1” with 22 members, “Group 2” with 21
members, leaving 11 individuals in this category as ungrouped. Vertex appearance on the beach ranges from 3 to 37 in the 31 day tracking period. The number of communication ties per day ranged from 0 to 96 in this dataset.

The covariates used by the vertex model are the regularity effect, group terms and all other covariates described in the network data description. The covariate weights learned by the algorithm is illustrated in Figure 2.7b. The highest weight is given to the previous vertex state and regularity. The same discussion about the effect of seasonality to the data holds for the Beach data as well. But here we can see that Saturday has a slightly higher weight than the rest of seasonality covariates, and this indicates that Saturday down-weights the other parameters less than the other days, which in turn means that it is more likely that people come to the beach if it is a Saturday.
Chapter 3

Correlated Topic Models

Topic models are a popular class of exchangeable latent variable models for document categorization. Their goal is to uncover hidden topics based on the distribution of word occurrences in a document corpus. Topic models are *admixture* models, which are the hierarchical counterparts of the usual mixture models that allow for only one hidden topic to be present in each document. In contrast, topic models incorporate multiple topics in each document. It is assumed that each document has a latent proportion of different topics, and the observed words are drawn in a conditionally independent manner, given the set of topics.

Latent Dirichlet Allocation (LDA) is the most popular topic model [17], in which the topic proportions are drawn from the Dirichlet distribution. While LDA has widespread applications, it is limited by the choice of the Dirichlet distribution. Notably, Dirichlet distribution can only model negative correlations [9], and thus, is unable to incorporate arbitrary correlations among the topics that may be present in different document corpora. Another drawback is that the elements with similar means need to have similar variances. While there have been previous attempts to go beyond the Dirichlet distribution, e.g. [16, 69], their correlation structures are still limited, learning these models is usually difficult and
no guaranteed algorithms exist. Furthermore, As discussed in [60], the correlation structure considered in [16], gives rise to spurious correlations resulting in a better perplexity on the held-out set even when the recovered topics are less interpretable. The work of [8] provides a provably correct algorithm for learning topic models that also allow for certain correlations among the topics, however, it requires “anchor word” separability assumptions for the proof of correctness.

In this chapter, we consider a flexible class of topic models, and propose guaranteed and efficient algorithms for learning them. We employ the class of Normalized Infinitely Divisible (NID) distributions to model the topic proportions [26, 51]. These are a class of distributions on the simplex, formed by normalizing a set of independent draws from a family of positive Infinitely Divisible (ID) distributions. The draws from an ID distribution can be represented as a sum of an arbitrary number of i.i.d. random variables. The concept of infinite divisibility was introduced in 1929 by Bruno de Finetti, and the most fundamental results were developed by Kolmogorov, Lévy and Khintchine in the 1930s. The idea of using normalized random probability measures with independent increments have also been used in the context of non-parametric models to go beyond the Dirichlet Process [49].

The Gamma distribution is an example of an ID distribution, and the Dirichlet distribution is obtained by normalizing a set of independent draws from Gamma distributions. We show that the class of NID topic models significantly generalize the LDA model: they can incorporate both positive and negative correlations among the topics and they involve additional parameters to vary the variance and higher order moments, while fixing the mean.
3.1 Related Work

There are mainly three categories of algorithms for learning topic models, viz., variational inference [16, 17], Gibbs sampling [22, 31, 55], and spectral methods [6, 80]. Among them, spectral methods have gained increasing prominence over the last few years, due to their efficiency and guaranteed learnability. In this chapter, we develop novel spectral methods for learning latent NID topic models.

Spectral methods have previously been proposed for learning LDA [6], and other latent variable models such as Independent Component Analysis (ICA), Hidden Markov Models (HMM) and mixtures of ranking distributions [5]. The idea is to learn the parameters based on spectral decomposition of low order moment tensors (third or fourth order). Efficient algorithms for tensor decomposition have been proposed before [5], and implies consistent learning with (low order) polynomial computational and sample complexity.

3.2 Summary of Results

The main difficulty in extending spectral methods to the more general class of NID topic models is the presence of arbitrary correlations among the hidden topics which need to be “untangled”. For instance, for a single topic model (i.e. each document has only one topic), the third order moment, which is the co-occurrence tensor of word triplets, has a CANDECOMP/PARAFAC (CP) decomposition, computing which yields an estimate of the topic-word matrix. In contrast, for the LDA model, such a tensor decomposition is obtained by a linear combination of moments up to third order. In other words, the moments of the LDA model need to be appropriately “centered” in order to have the tensor decomposition form.
Finding such a moment combination has so far been an “art form”, since it is based on explicit manipulation of the moments of the hidden topic distribution. So far, there is no principled mechanism to automatically find the moment combination with the CP decomposition form. For arbitrary topic models, finding such a combination may not even be possible. In general, one requires all the higher order moments for learning.

In this chapter, we show that surprisingly, for the flexible class of NID topic models, moments up to third order suffice for learning, and we provide an efficient algorithm for computing the coefficients to combine the moments. The algorithm is based on computation of a univariate integral, that involves the Lévy measure of the underlying ID distribution. The integral can be computed efficiently through numerical integration since it is only univariate, and has no dependence on the topic or word dimensions. Intriguingly, this can be accomplished, even when there exists no closed form probability density functions for the NID variables.

3.3 Latent Normalized Infinitely Divisible Topic Models

Topic models incorporate relationships between words $x_1, x_2, \ldots \in \mathbb{R}^d$ and a set of $k$ hidden topics. We represent the words $x_i$ using one-hot encoding, i.e. $x_i = e_j$ if the $j^{th}$ word in the vocabulary occurs, and $e_j$ is the standard basis vector. The proportions of topics in a document is represented by vector $h \in \mathbb{R}^k$. We assume that $h$ is drawn from an NID distribution. The detailed generative process of a latent NID topic model for each document is as follows

1. Draw $k$ independent variables, $z_1, z_2, \ldots, z_k$ from a family of ID distributions.

2. Set $h$ to $\left( \frac{z_1}{Z}, \ldots, \frac{z_k}{Z} \right)$ where $Z = \sum_{i \in [k]} z_i$. 

41
3. For each word $x_i$,

(a) Choose a topic $\zeta_i \sim \text{Multi}(h)$ and represent it with one-hot encoding.

(b) Choose a word vector, $x_i$, to be the standard basis vector with probability,

$$E(x_i|\zeta_i) = A\zeta_i,$$  \hspace{1cm} (3.1)

conditioned on the drawn topic $\zeta_i$, where $A \in \mathbb{R}^{d \times k}$ is the topic-word matrix.

From (3.1), we also have

$$E(x_i|h) = E[E(x_i|h, \zeta_i)] = E(x_i|\zeta_i)E(\zeta_i|h) = Ah$$ \hspace{1cm} (3.2)

When the $z_i$ is drawn from the Gamma($\alpha_i, 1$) distribution, we obtain the Dir($\alpha$) distribution for the hidden vector $h = (h_1, \ldots, h_k)$, and the LDA model through the above generative process. Our goal is to recover the topic-word matrix $A$ given the document collection. In the following section we introduce the class of NID distributions and discuss its properties.

### 3.4 Properties of NID distributions

NID distributions are a flexible class of distributions on the simplex and have been applied in a range of domains. This includes hierarchical mixture modeling with Normalized Inverse-Gaussian distribution [48], and modeling overdispersion with the normalized tempered stable distribution [42], both of which are examples of NID distributions. For more applications, see [26]. Let us first define the concept of infinite divisibility and present the properties of an ID distribution, and then consider the NID distributions.
3.4.1 Infinitely Divisible Distributions

If random variable $z$ has an Infinitely Divisible (ID) distribution, then for any $n \in \mathbb{N}$ there exists a collection of i.i.d random variables $y_1, \ldots, y_n$ such that $z \overset{d}{=} y_1 + \cdots + y_n$. In other words, an Infinitely Divisible distribution can be expressed as the sum of an arbitrary number of independent identically distributed random variables.

The Poisson distribution, compound Poisson, the negative binomial distribution, Gamma distribution, and the trivially degenerate distribution are examples of Infinitely Divisible distributions; as are the normal distribution, Cauchy distribution, and all other members of the stable distribution family. The Student’s t-distribution is also another example of Infinitely Divisible distributions. The uniform distribution and the binomial distribution are not infinitely divisible, as are all distributions with bounded (finite) support.

The special decomposition form of ID distributions makes them natural choices for certain models or applications. E.g. a compound Poisson distribution is a Poisson sum of i.i.d random variables. The discrete compound Poisson distribution, also known as the stuttering Poisson distribution, can model batch arrivals (such as in a bulk queue \cite{1}) and can incorporate Poisson mixtures. In the sequel, we limit the discussion to ID distributions on $\mathbb{R}^+$ in order to ensure that the Normalized ID variables are on the simplex. Let us now present how ID distributions can be characterized.

\textbf{Lévy measure: } A $\sigma$-finite Borel measure $\nu$ on $\mathbb{R}^+$ is called a Lévy measure if

$$\int_0^\infty \min(1, x) \nu(dx) < \infty.$$  \hspace{1cm} (3.3)

According to the Lévy-Khintchine representation given below, the Lévy measure uniquely characterizes an ID distribution along with a constant scale $\tau$. This implies that every
Figure 3.1: Heat map of the pdf of three examples of the NID class that have closed form pdf. All the figures have $\alpha = (2,2,4)$. For the Inverse Gaussian the distribution moves from the center to the vertices of the simplex as $\lambda$ goes from 0 to $\infty$ with fixed $\alpha$ and for the $\gamma$-stable we have the same behavior when $\gamma$ changes from 1 to 0 with fixed $\alpha$.

Infinitely Divisible distribution corresponds to a Lévy process, which is a stochastic process with independent increments.

**Lévy-Khintchine representation**  [Theorem 16.14 [40]] Let $\mathcal{M}_1(\Lambda)$ and $\mathcal{M}_\sigma(\Lambda)$ indicate the set of probability measures and the set of $\sigma$-finite measures on a non-empty set $\Lambda$, respectively. Let $\mu \in \mathcal{M}_1([0, \infty))$ and let $\Psi(u) = -\log \int_0^\infty e^{-uz}d(\mu)$ be the log-Laplace transform of $\mu$. Then $\mu$ is Infinitely Divisible, if and only if there exists a $\tau \geq 0$ and a $\sigma$-finite measure $\nu \in \mathcal{M}_\sigma((0, \infty))$ with

$$\int_0^\infty \min(1, z)\nu(dz) < \infty,$$

such that

$$\Psi(u) = \tau u + \int_0^\infty (1 - e^{-uz})\nu(dz) \quad \text{for} \quad u \geq 0,$$
In this case the pair \((\tau, \nu)\) is unique, \(\nu\) is called the Lévy measure of \(\mu\) and \(\tau\) is called the deterministic part. It can be shown that \(\tau = \sup\{z \geq 0 : \mu([0, z)) = 0\}\).

In particular, let \(\Phi_{z_i}(u) = \mathbb{E}[e^{iuz_i}] = \int_0^\infty e^{iuz_i} f(z_i) dz_i\) indicate the characteristic function of an Infinitely Divisible random variable \(z_i\) with pdf \(f(z_i)\) and corresponding pair \((\tau_i, \nu_i)\), where \(i\) is the imaginary unit. Based on the Lévy-Khintchine representation it holds that \(\Phi_{z_i}(iu) = \mathbb{E}[e^{-uz_i}] = e^{-\Psi_i(u)}\) where \(\Psi_i(u) = \tau_i u + \int_0^\infty (1 - e^{-uz}) \nu_i(dz)\) is typically referred to as the Laplace exponent of \(z_i\). This implies that the Laplace exponent of an ID variable is also completely characterized by pair \((\tau_i, \nu_i)\). It holds for ID variables that if \(\nu_i\) is a well-defined Lévy measure, so is \(\alpha_i \nu_i\) for any \(\alpha_i > 0\), which indicates that \(\alpha_i \Psi_i(u)\) is also a well-defined Laplace exponent of an ID variable.

### 3.4.2 Normalized Infinitely Divisible Distributions

As defined in [26], a Normalized Infinitely Divisible (NID) random variable is a random variable that is formed by normalizing independent draws of strictly positive (not necessarily coinciding) ID distributions. More specifically, let \(z_1, \ldots, z_k\) be a set of independent strictly positive Infinitely Divisible random variables and \(Z = z_1 + \cdots + z_k\). An NID distribution is defined as the distribution of the random vector \(h = (h_1, \ldots, h_k) := \left(\frac{z_1}{Z}, \ldots, \frac{z_k}{Z}\right)\) on the \((k-1)\)-dimensional simplex, denoted as \(\Delta^{k-1}\). The strict positivity assumption implies that \(h\) is on the simplex [26, 51].

Let \([k]\) denote natural numbers 1, \ldots, \(k\). As stated by the Lévy-Khintchine theorem, a collection of ID positive variables \(z_i\) for \(i \in [k]\) is completely characterized by the collection of the corresponding Lévy measures \(\nu_1, \ldots, \nu_k\). It was shown in [51] that this also holds for the normalized variables \(h_i\) for \(i \in [k]\).

We assume that the ID variables \(z_1, \ldots, z_k\) are drawn independently from ID distributions that are characterized with the corresponding collection of Lévy measures \(\alpha_1 \nu, \ldots, \alpha_k \nu\),
respectively. Which in turn translates respectively to variables with Laplace exponents $\alpha_1 \Psi(u), \ldots, \alpha_k \Psi(u)$. Variables $\alpha_i$ will allow the distribution to vary in the interior of the simplex, providing the asymmetry needed to model latent models. The homogeneity assumption on the Lévy measure or the Laplace exponent provides the structure needed for guaranteed learning (Theorem 3.1). The overall graphical model representation is shown in Figure 1.2 If the original ID variables $z_i$ have probability densities $f_i$ for all $i \in [k]$, then the distribution of vector $h$, where $h_k = 1 - \sum_{i \in [k-1]} h_i$ is, $f(h) = \int_0^\infty \prod_{i \in [k]} f_i(h_i Z) Z^{k-1} dZ$.

There are only three members of the NID class that have closed form densities namely, the Gamma distribution, Gamma$(\alpha_i, \lambda)$, the Inverse Gaussian distribution, $IG(\alpha_i, \lambda)$, and the $1/2$-stable distribution $St(\gamma, \beta, \alpha_i, \mu)$ with $\gamma = 1/2$. $\mu = 1$ and $\beta = 1$ to ensure positive support for the Stable distribution. As noted earlier, Gamma$(\alpha_i, 1)$ reduces to the Dirichlet distribution. An interested reader is referred to [26, 51] for the closed form of each distribution.

Figure 3.1 depicts the heatmap of the density of these distributions on the probability simplex for different value of their parameters. Note that all the distributions have the same $\alpha$ parameter and hence, the same mean values. However, their concentration properties are widely varying, showing that the NID class can incorporate variations in higher order moments through additional parameters.

**Gamma ID distribution:** When the ID distribution is Gamma with parameters $(\alpha_i, 1)$, we have the Dirichlet distribution as the resulting NID distribution. The Laplace exponent for this distribution will, therefore, be $\Psi_i(u) = \alpha_i \ln(1 + u)$.

**$\gamma$-stable ID distribution:** The variables are drawn from the positive stable distribution $St(\gamma, \beta, \alpha_i, \mu)$ with $\mu = 0$, $\beta = 1$ and $\gamma < 1$ which ensures that the distribution is on $\mathbb{R}^+$. The Laplace exponent of this distribution is $\Psi_i(u) = \alpha_i \frac{\Gamma(1-\gamma)}{\sqrt{2\pi\gamma}} u^\gamma$. Note that the $\gamma$-stable distribution can be represented in closed form for $\gamma = \frac{1}{2}$. 46
Figure 3.2: Proportion of positively correlated pairs of 10-dimensional NID distributed variables with respect to the parameters of the Laplace exponent for a fixed randomly drawn vector $\alpha$.

**Inverse Gaussian ID distribution:** The random variables are drawn from the Inverse-Gaussian (IG) distribution $IG(\alpha_i, \lambda)$. The Laplace exponent of this distribution is $\Psi_i(u) = \alpha_i(\sqrt{2u + \lambda^2} - \lambda)$. Note: The Dirichlet distribution, the 1/2-Stable distribution and the Inverse Gaussian distribution are all special cases of the generalized Inverse Gaussian distribution [26].

As mentioned earlier, the class of NID distributions is capable of modeling positive and negative correlations among the topics. This property is depicted in Figure 3.2. These figures show the proportion of positively correlated pairs for the three presented distributions for different parameters of each distribution. As we can see, the Inverse Gaussian NID distribution can capture both positive and negative correlations.

### 3.5 Learning NID Topic Models through Spectral Methods

In order to be able to guarantee efficient learning using higher order moments, the moments need to have a very specific structure. Namely, the moment of the underlying distribution of $h$ needs to form a diagonal tensor. If the components of $h$ where indeed independent, this is obtained through the cumulant tensor. On the other hand, for LDA, it has been shown
by Anandkumar et. al. [6] that a linear combination of moments of up to third order of \( h \) forms a diagonal tensor for the Dirichlet distribution. Below, we extend the result to the more general class of NID distributions.

### 3.5.1 Consistency of Learning through Moment Matching

**Assumption 1.** ID random variables \( z_i \) for \( i \in [k] \) are said to be partially homogeneous if they share the same Lévy measure. This implies that the corresponding Laplace exponent of variable \( z_i \) is given \( \alpha_i \Psi(u) \) for some \( \alpha_i \in \mathbb{R}^+ \), and \( \Psi(u) \) is the Laplace exponent of the common Lévy measure.

Under the above assumption, we prove guaranteed learning of NID models through spectral methods. This is based on the following moment forms for NID models, which admit a CP tensor decomposition. The components of the decomposition will be the columns of the topic-word matrix: \( A := [a_1 | a_2 | \ldots | a_k] \). Define

\[
\Omega(m, n, p) = \int_0^\infty u^m \left( \frac{d^n}{du^n} \Psi(u) \right) \left( -\frac{d}{du} \Psi(u) \right)^p e^{-\alpha_0 \Psi(u)} du,
\]

where \( \Psi(u) \) is the Laplace exponent of the NID distribution and \( \alpha_0 = \sum_{i \in [k]} \alpha_i \).

**Theorem 3.1.** *(Moment Forms for NID models)* Let \( M_2 \) and \( M_3 \) be respectively the following matrix and tensor constructed from the moments of the data,

\[
M_2 = \mathbb{E}[x_1 \otimes x_2] + v \cdot \mathbb{E}[x_1] \otimes \mathbb{E}[x_2],
\]

\[
M_3 = \mathbb{E}[x_1 \otimes x_2 \otimes x_3] + v_2 \cdot \mathbb{E}[x_1] \otimes \mathbb{E}[x_2] \otimes \mathbb{E}[x_3] + v_1 \left[ \mathbb{E}[x_1 \otimes x_2] \otimes \mathbb{E}[x_3] + \mathbb{E}[x_1] \otimes \mathbb{E}[x_2 \otimes x_3] + \mathbb{E}[x_1 \otimes \mathbb{E}[x_2] \otimes x_3] \right]
\]

48
where,

\[ v = \frac{\Omega(1,1,1)}{\left(\Omega(0,1,0)\right)^2}, \quad v_1 = -\frac{\Omega(2,2,1)}{2\Omega(1,2,0)\Omega(0,1,0)}, \]

\[ v_2 = -0.5\Omega(2,1,2) + 3v_1\Omega(1,1,1)\Omega(0,1,0) \]

\[ \left(\Omega(0,1,0)\right)^3, \quad (3.9) \]

\[ v_2 = \frac{-0.5\Omega(2,1,2) + 3v_1\Omega(1,1,1)\Omega(0,1,0)}{\left(\Omega(0,1,0)\right)^3}, \quad (3.10) \]

Then given Assumption 1,

\[ M_2 = \sum_{j \in [k]} \kappa_j (a_j \otimes a_j), \quad M_3 = \sum_{j \in [k]} \lambda_j (a_j \otimes a_j \otimes a_j). \quad (3.11) \]

for a set of \( \kappa_j \)'s and \( \lambda_j \)'s which are a function of the parameters of the distribution.

Remark 1: efficient computation of \( v, v_1 \) and \( v_2 \)  

What makes Theorem 3.1 specially intriguing is the fact that weights \( v, v_1 \) and \( v_2 \) can be computed through univariate integration, which can be computed efficiently, regardless of the dimensionality of the problem.

Remark 2: investigation of special cases  

When the ID distribution is Gamma with parameters \( (\alpha_i, 1) \), we have the Dirichlet distribution as the resulting NID distribution. Weights \( v_1 \) and \( v_2 \) reduce to the results of Anandkumar et. al. [6] for the Gamma\((\alpha_i, 1)\) distribution, which are \( v_1 = -\frac{\alpha_0}{\alpha_0 + 2} \) and \( v_2 = \frac{2\alpha_0^2}{(\alpha_0 + 2)(\alpha_0 + 1)} \). When the variables are drawn from the positive stable distribution \( St(1/2, \beta, \alpha_i, \mu) \) weights \( v_1 \) and \( v_2 \) in Theorem 3.1 can be represented in closed form as \( v_1 = -\frac{1}{4} \) and \( v_2 = -\frac{5}{8} \).

It is hard to find the closed form representation of the weights for other stable distributions and the Inverse Gaussian distribution. Therefore, we give the form of the weights with respect to the parameters of each distribution in Figure 3.3. As it can be seen in Figures 3.1e and 3.1b, as \( \gamma \) increases, the distribution gets more centralized on the simplex. Therefore, as depicted in Figure 3.3a the weight becomes more negative to compensate for it. The same
(a) Weight $v_1$ of theorem 3.1 for a Stable ID distribution $St(\gamma, \beta, \alpha_i, \mu)$ with $\mu = 0$, $\beta = 1$, $\gamma < 1$ and $\alpha_i > 0$ vs. $\gamma$ for $\alpha_0 = 1$

(b) Weight $v_1$ of theorem 3.1 for an Inverse Gaussian distribution $IG(\alpha_i, \lambda)$ vs. $\lambda > 0$ and $\alpha_i \geq 0$ for $\alpha_0 = 1$

Figure 3.3: Weight $v_1$ for two different examples of the NID distribution. Weights $v$ and $v_2$ in the theorem have similar behavior w.r.t. the parameters.

The above result immediately implies guaranteed learning for non-degenerate topic-word matrix $A$.

**Assumption 2.** Topic-word matrix $A \in \mathbb{R}^{d \times k}$ has linearly independent columns and the parameters $\alpha_i > 0$.

**Corollary 1.** (Guaranteed Learning of NID Topic Models using Spectral Methods) Given empirical versions of moments $M_2$ and $M_3$ in (3.7) and (3.8), using tensor decomposition algorithm from [5], under Assumption 2, we can consistently estimate topic-word matrix $A$ and parameters $\alpha$ with polynomial computational and sample complexity. (Algorithm 1)

**Remark 3: third order moments suffice** For the flexible class of latent NID topic models, only moments up to the third order suffice for efficient learning.

**Remark 4: Sample Complexity** Following [6], Algorithm 1 can recover matrix $A$ under Assumption 2 with polynomial sample complexity.
Algorithm 1 Parameter Learning

Input: Chosen NID distribution and hidden dimension $k$
Output: Parameters of NID distribution $\alpha$ and topic-word matrix $A$

1: Estimate empirical moments $\hat{E}(x_1 \otimes x_2 \otimes x_3)$, $\hat{E}(x_1 \otimes x_2)$, and $\hat{E}(x_1)$.
2: Compute weights $v$, $v_1$ and $v_2$ in (3.9) and (3.10) for the given NID distribution by numerical integration.
3: Estimate tensors $M_2$ and $M_3$ in (3.7) and (3.8).
4: Decompose tensor $M_3$ into its rank-1 components using the algorithm in [5] that requires $M_2$.
5: Return columns of $A$ as the components of the decomposition.

Remark 5: Implementation Efficiency
In order to make the implementation efficient we use the discussion in [5]. Specifically, as mentioned in [5], we can find a whitening transformation from matrix $M_2$ that lowers the data dimension from the vocabulary space to the topic space. We then use the same whitening transformation to go back to the original space and recover the parameters of the model.

Overview of the proof of Theorem 3.1
We begin the proof by forming the following second order and third order tensors using the moments of the NID distribution given in Lemma 1.

\[
M_2^{(h)} = E(h \otimes h) + vE(h) \otimes E(h),
\]
\[
M_3^{(h)} = E(h \otimes h \otimes h) + v_2E(h) \otimes E(h) \otimes E(h)
+ v_1[E(h \otimes h) \otimes E(h) + E(h \otimes E(h) \otimes h)
+ E(h) \otimes E(h \otimes h)]
\]

Weights $v$, $v_1$ and $v_2$ are as in Equations (3.9) and (3.10). They are computed by setting the off-diagonal entries of matrix $M_2^{(h)}$ in Equation 3.12 and $M_3^{(h)}$ in Equation 3.13 to 0.

Due to the homogeneity assumption, all the off-diagonal entries can be simultaneously made to vanish with these choices of coefficients for $v$, $v_1$ and $v_2$. We obtain $M_2^{(h)} = \sum_{i \in [k]} \kappa_i^* e_i^{\otimes 2}$ and $M_3^{(h)} = \sum_{i \in [k]} \lambda_i^* e_i^{\otimes 3}$ where $e_i$’s are the standard basis vectors, and this implies they are...
diagonal tensors. Due to this fact and the exchangeability of the words given topics according to (3.2), Equations 3.11 follow.

The exact forms of \( v, v_1 \) and \( v_2 \) are obtained by the following moment forms for NID distributions.

**Lemma 1 ([51]).** The moments of NID variables \( h_1, \ldots h_k \) satisfy

\[
E(h_1^{r_1} h_2^{r_2} \ldots h_k^{r_k}) = \frac{1}{\Gamma(r)} \int_0^\infty u^{r-1} e^{-\alpha_0 \Psi(u)} \prod_{j \in [k]} B_{r_j}^j du, \tag{3.14}
\]

where \( r = \sum_{i \in [k]} r_i \) and \( B_{r_j}^j \) can be written in terms of the partial Bell polynomial as

\[
B_r^i = B_r(-\alpha_i \Psi^{(1)}(u), \ldots, -\alpha_i \Psi^{(r)}(u)), \tag{3.15}
\]

where \( \Psi^{(l)}(u) \) is the \( l \)th derivative of \( \Psi(u) \) w.r.t. \( u \).

The proof of Theorem 3.1 and Lemma 1 is given in the Appendix.

### 3.6 Experiments

In this section we apply our proposed latent NID topic modeling algorithm to New York Times and Pubmed articles [47]. The New York Times dataset contains about 300,000 documents and the pubmed data contains around 8 million documents. The vocabulary size for both the datasets are around 100,000.

**Table 3.1:** Perplexity comparison for different datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>NYtimes</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>NID</td>
<td>3.5702e + 03</td>
<td>4.0771e + 03</td>
</tr>
<tr>
<td>LDA</td>
<td>4.8464e + 03</td>
<td>4.3702e + 03</td>
</tr>
<tr>
<td>CTM</td>
<td>1.0669e + 04</td>
<td>N/A</td>
</tr>
</tbody>
</table>
Table 3.2: PMI comparison for different datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>NYtimes</th>
<th>Pubmed</th>
</tr>
</thead>
<tbody>
<tr>
<td>NID</td>
<td>0.2439</td>
<td>0.3080</td>
</tr>
<tr>
<td>LDA</td>
<td>0.2362</td>
<td>0.4487</td>
</tr>
<tr>
<td>CTM</td>
<td>0.497</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 3.3: 10 Shared words - New York times dataset

| Shared Words | boston-globe, tonight, question, newspaper, spot, percent, file, di-ane, copy, fall |

3.6.1 Hyper-parameter Tuning

In this sub-section we propose a hyper-parameter tuning approach (Algorithm 2) for learning the class of NID topic models. In order to find the best matching NID distribution that describes the data, we can directly tune for weights $v$, $v_1$ and $v_2$ of Equations 3.9 and 3.10. This is possible by altering the decomposition step of the learning algorithm (Step 4, Algorithm 1) such that it optimizes not only over the low rank components of the moment tensor, but also over the weights $v_1$ and $v_2$. To be more specific we optimize the following least squares problem over the weights $v_1$ and $v_2$ and the components of the tensor $a_j$ for $j \in [k]$:

$$
\min_{v_1, v_2, a_j} \| T + v_1 T_1 + v_2 T_2 - \sum_{j \in [k]} \lambda_j (a_j \otimes a_j \otimes a_j) \|_F, \quad (3.16)
$$

where, $\| \cdot \|_F$ is the Frobenius norm of a tensor and

$$
T = \mathbb{E}[x_1 \otimes x_2 \otimes x_3], \quad (3.17)
$$

$$
T_1 = \mathbb{E}[x_1 \otimes x_2] \otimes \mathbb{E}[x_3] + \mathbb{E}[x_1] \otimes \mathbb{E}[x_2 \otimes x_3] + \mathbb{E}[x_1 \otimes \mathbb{E}[x_2] \otimes x_3], \quad (3.18)
$$

$$
T_2 = \mathbb{E}[x_1] \otimes \mathbb{E}[x_2] \otimes \mathbb{E}[x_3]. \quad (3.19)
$$
This optimization problem will return the components of tensor $M_3$, $a_j$, which are the columns of matrix $A$ in Equation 3.1 and weights $v_1$ and $v_2$. As can be seen in Equations 3.9 and 3.10, these weights are functions of the Laplace exponent of the underlying NID distribution. Therefore, the underlying Laplace exponent can be recovered from the weights in a non-parametric manner using a held out data set. We can use the non-parametric representation of the Laplace exponent to extract the properties of the NID distribution such as correlations. As will be shown in the results, this efficient heuristic for hyper-parameter tuning shows a good empirical performance on multiple real data sets.

Algorithm 2 Hyper-parameter Optimization (HPO)

**Input:** train data set, held out data set and hidden dimension $k$

**Output:** topic-word matrix $A$, Laplace exponent $\Psi(u)$

1: Estimate empirical moments and compute $T$, $T_1$ and $T_2$ in Equations 3.17, 3.18 and 3.19 from the train data set.
2: Solve the optimization problem of Equation 3.16 and output $A$, $v_1$ and $v_2$
3: Estimate the Laplace exponent $\Psi(u)$ from $v_1$ and $v_2$ using non-parametric density estimation on the held out data set.

3.6.2 Results

We refer to our proposed method as NID and compare it against spectral LDA [6], where the distribution of the hidden space is fixed to be Dirichlet, and CTM [16] where the hidden topic distribution is logistic normal. It has been shown in [36] that spectral LDA is more efficient and achieves better perplexity compared to the conventional LDA [17]. Table 3.4 provides a sketch of the top words per topics recovered by our latent NID topic model on the New York times dataset and Table 3.5 shows the top words recovered from the pubmed dataset. We have also provided the top words recovered by LDA for the New York times dataset for comparison purposes in Table 3.6. Besides the top words, we also present the shared words among the recovered topics for the New York Times dataset in Table 3.3. The presence of words such as “tonight”, “question” and “fall” among these words are not far
from expectation, since they are general words that are not usually indicative of any specific topic.

We use the well-known likelihood perplexity measure [17] to evaluate the generalization performance of our proposed topic modeling algorithm as well as the Pointwise Mutual Information (PMI) score [7] to assess the coherence of the recovered topics. Perplexity is defined as the inverse of the geometric mean per-word of the estimated likelihood. It should be noted that lower perplexity indicates better generalization performance and higher PMI indicates better topic coherence. Tables 3.1 and 3.2 show the perplexity and PMI scores, respectively, for the New York Times and Pubmed datasets for a fixed number of topics across each dataset. More comparisons for different number of topics is presented in Figure 3.4 for the New York Times corpus. It should be noted that CTM does not scale to the Pubmed dataset. The results suggest that if we allow the corpus to choose the best underlying topic distribution, we can get better generalization performance as well as topic coherence on the held-out set compared to fixing the underlying distribution to Dirichlet. The improved perplexity of our proposed method is indicative of correlations in the underlying documents that are not captured by the Dirichlet distribution. Thus, latent NID topic models are capable of successfully capturing correlations within topics while providing guarantees for exact recovery and efficient learning as proven in Section 3.5.

Last but not least, the naive Variational Inference implementation of [17] \(^1\), does not scale to the current datasets used in this dissertation. The naive implementation of the spectral LDA, however, takes only about a minute to run on the NYtimes dataset and about 15 minutes to run on the Pubmed dataset. It is, therefore, of great importance to have a class of models that can be learned using spectral methods mainly because of their inherent scalability, ease of implementation and statistical guarantees. As we show in this chapter, latent NID topic models are such a class of models. The correlated topic model framework of

\(^1\)available at: http://www.cs.princeton.edu/~blei/lda-c/
[16] also uses Variational Inference to perform learning and it is limited to the logit-normal distribution. latent NID topic models not only scale to large data sets, but also model arbitrary correlations without requiring a fixed prior distribution over the topic space.

Table 3.4: NID Top 10 Words for NYtimes, K = 20

<table>
<thead>
<tr>
<th>Topic</th>
<th>Top Words in descending order of importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>seeded, soldier, firestone, bobby-braswell, michigan-state, actresses, gary-william, preview, school-district, netanyahu</td>
</tr>
<tr>
<td>2</td>
<td>diane, question, newspaper, copy, fall, held, tonight, send, guard, slugged</td>
</tr>
<tr>
<td>3</td>
<td>abides, acclimate, acetate, alderman, analogues, annexing, ansar, antitax, antitobacco, argyle</td>
</tr>
<tr>
<td>4</td>
<td>percent, school, quarter, company, taliban, high, stock, race, companies, johnmccain</td>
</tr>
<tr>
<td>5</td>
<td>test, deal, contract, tiger-wood, question, houston-chronicle, copy, won, seattlepost-intelligencer, tax</td>
</tr>
<tr>
<td>6</td>
<td>tonight, diane, question, newspaper, file, copy, fall, slugged, onlytest, xxx</td>
</tr>
<tr>
<td>7</td>
<td>company, com, market, stock, won, los-angeles-daily-new, business, eastern, web, commentary</td>
</tr>
<tr>
<td>8</td>
<td>abides, acclimate, acetate, alderman, analogues, annexing, ansar, antitax, antitobacco, argyle</td>
</tr>
<tr>
<td>9</td>
<td>company, game, run, los-angeles-daily-new, percent, team, season, stock, companies, games</td>
</tr>
<tr>
<td>10</td>
<td>working-girl, abides, acclimate, acetate, alderman, analogues, annexing, ansar, antitax, antitobacco</td>
</tr>
<tr>
<td>11</td>
<td>diane, newspaper, fall, tonight, question, held, copy, bush, slugged, police</td>
</tr>
<tr>
<td>12</td>
<td>hurricanes, policies, surgery, productivity, courageous, emergency, singapore, orange-bowl, regarding, telecast</td>
</tr>
<tr>
<td>13</td>
<td>abides, acclimate, acetate, alderman, analogues, annexing, ansar, antitax, antitobacco, argyle</td>
</tr>
<tr>
<td>14</td>
<td>company, com, won, stock, market, eastern, commentary, business, web, deal</td>
</tr>
<tr>
<td>15</td>
<td>company, stock, market, business, investor, technology, analyst, cash, sell, executives</td>
</tr>
<tr>
<td>16</td>
<td>tonight, question, diane, file, newspaper, copy, fall, slugged, onlytest, xxx</td>
</tr>
<tr>
<td>17</td>
<td>defense, held, children, fight, assistant, surgery, michael-bloomberg, worker, bird, omar</td>
</tr>
<tr>
<td>18</td>
<td>percent, company, stock, companies, quarter, school, market, analyst, high, corp</td>
</tr>
<tr>
<td>19</td>
<td>school, student, yard, released, guard, premature, teacher, touchdown, publication, leader</td>
</tr>
<tr>
<td>20</td>
<td>school, percent, student, yard, high, taliban, flight, air, afghanistan, plan</td>
</tr>
</tbody>
</table>
Table 3.5: NID top 10 Words for Pubmed, K = 10

<table>
<thead>
<tr>
<th>Topic</th>
<th>Top Words in descending order of importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>protein, region, dna, family, sequence, gene, form-12, analysis.abstract, model, tumoural</td>
</tr>
<tr>
<td>2</td>
<td>cell, mice.abstract, expression.abstract, activity.abstract, primary, tumor, antigen, human, t-cell, vitro</td>
</tr>
<tr>
<td>3</td>
<td>tumor, treatment, receptor, lesional, children-a, effect.abstract, factor, rat1, renal-cell, response-1</td>
</tr>
<tr>
<td>4</td>
<td>patient, treatment, therapy, clinical, disease, level.abstract, effect.abstract, treated, tumor, surgery</td>
</tr>
<tr>
<td>5</td>
<td>activity.abstract, rat1, concentration, dna, human, effect.abstract, exposure.abstract, animal-based, reactional, inhibition.abstract</td>
</tr>
<tr>
<td>6</td>
<td>patient, children-a, women.abstract, treatment, level.abstract, syndrome, disordered, disease, year-1, therapy</td>
</tr>
<tr>
<td>7</td>
<td>effect.abstract, receptor, level.abstract, rat1, mutational, gene, concentration, women.abstract, insulin, expression.abstract</td>
</tr>
<tr>
<td>8</td>
<td>acid, strain, concentration, women.abstract, test, pregnancy-a, drug, system-a, function.abstract, water</td>
</tr>
<tr>
<td>9</td>
<td>strain, protein, system-a, muscle, mutational, species, growth, diagnosis-based, analysis.abstract, gene</td>
</tr>
<tr>
<td>10</td>
<td>infection.abstract, hospital, programed, strain, medical, alpha, information, health, children-a, data.abstract</td>
</tr>
</tbody>
</table>

Figure 3.4: Perplexity and PMI scores for the NYtimes dataset across different number of topics

(a) Perplexity score  
(b) Pointwise Mutual Information (PMI)
Table 3.6: Spectral LDA top 10 Words for NYtimes, $K = 20$

<table>
<thead>
<tr>
<th>Topic</th>
<th>Top Words in descending order of importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>newspaper, question, copy, fall, diane, chante-lagon, kill, mandatory, drug, patient</td>
</tr>
<tr>
<td>2</td>
<td>held, guard, send, publication, released, advisory, premature, attn-editor, undated-lined, washington-dateline</td>
</tr>
<tr>
<td>3</td>
<td>los-angeles-daily-new, slugged, com, xxx, www, x-x-x, web, information, site, eastern</td>
</tr>
<tr>
<td>4</td>
<td>million, shares, offering, boston-globe, debt, public, initial, player, bill, contract</td>
</tr>
<tr>
<td>5</td>
<td>onlytest, point, tax, case, court, lawyer, police, minutes, death, shot</td>
</tr>
<tr>
<td>6</td>
<td>held, released, publication, guard, advisory, premature, send, attn-editor, undated-lined, washington-dateline</td>
</tr>
<tr>
<td>7</td>
<td>com, information, www, web, eastern, daily, commentary, business, separate, marked</td>
</tr>
<tr>
<td>8</td>
<td>boston-globe, spot, file, killed, tonight, women, earlier, article, george-bush, incorrectly</td>
</tr>
<tr>
<td>9</td>
<td>million, shares, offering, debt, public, initial, player, contract, bond, revenue</td>
</tr>
<tr>
<td>10</td>
<td>boston-globe, spot, file, killed, attn-editor, earlier, article, court, women</td>
</tr>
<tr>
<td>11</td>
<td>percent, market, stock, point, quarter, economy, rate, women, growth, companies</td>
</tr>
<tr>
<td>12</td>
<td>boston-globe, spot, file, tonight, killed, earlier, article, women, incorrectly, news-feature</td>
</tr>
<tr>
<td>13</td>
<td>held, guard, publication, released, send, advisory, premature, attn-editor, undated-lined, washington-dateline</td>
</tr>
<tr>
<td>14</td>
<td>los-angeles-daily-new, slugged, xxx, new-york, x-x-x, fund, bush, goal, king, evening</td>
</tr>
<tr>
<td>15</td>
<td>tonight, copy, question, diane, fall, newspaper, russia, terrorist, russia, russian, black</td>
</tr>
<tr>
<td>16</td>
<td>slugged, los-angeles-daily-new, xxx, new-york, x-x-x, bush, run, school, inning, student</td>
</tr>
<tr>
<td>17</td>
<td>onlytest, file, film, onlyendpar, movie, new-york, seattle-pi, los-angeles, sport, patient</td>
</tr>
<tr>
<td>18</td>
<td>los-angeles-daily-new, slugged, xxx, x-x-x, student, inning, send, program, enron, game</td>
</tr>
<tr>
<td>19</td>
<td>los-angeles-daily-new, slugged, xxx, new-york, x-x-x, fund, evening, program, student, enron</td>
</tr>
<tr>
<td>20</td>
<td>test, houston-chronicle, hearst-news-service, seattle-post-intelligencer, ignore, patient, kansas-city, yard, race, doctor</td>
</tr>
</tbody>
</table>
Human beings possess impressive abilities for abstract mathematical and logical thinking. It has long been the dream of computer scientists to design machines with such capabilities: machines that can automatically learn and reason, thereby removing the need to manually program them. Neural programming, where neural networks are used to learn programs, mathematics, or logic, has recently shown promise towards this goal. Examples of neural programming include neural theorem provers, neural Turing machines, and neural program inducers, e.g. [3, 18, 30, 50, 57]. They aim to solve tasks such as learning functions in logic, mathematics, or computer programs (e.g. logical or, addition, and sorting), prove theorems and synthesize programs.

Most works on neural programming either rely only on black-box function evaluations [10, 30] or on the availability of detailed program execution traces, where entire program runs are recorded under different input conditions [21, 64]. Black-box function evaluations are easy to obtain since we only need to generate inputs and outputs to various functions in the
domain. However, by themselves, they do not result in powerful generalizable models, since they do not have sufficient information about the underlying structure of the domain. On the other hand, execution traces capture the underlying structure, but, are generally harder to obtain under many different input conditions; even if they are available, the computational complexity of incorporating them is significant. Due to the lack of good coverage, these approaches fail to generalize to programs of higher complexity and to domains with a large number of functions. Moreover, the performance of these frameworks is severely dependent on the nature of execution traces: more efficient programs lead to a drastic improvement in performance [21], but such programs may not be readily available.

In many problem domains, in addition to function evaluations, one typically has access to more information such as symbolic representations that encode the relationships between the given variables and functions in a succinct manner. For instance, in physical systems such as fluid dynamics or robotics, the physical model of the world imposes constraints on the values that different variables can take. Mathematics and logic are other domains in which expressions are inherently symbolic. In the domain of programming languages, declarative languages explicitly declare variables in the program. For instance, database query languages (e.g., SQL), regular expressions, and functional programming. Declarative programs greatly simplify parallel programs through the generation of symbolic computation graphs, and have thus been used in modern deep learning packages, such as Theano, TensorFlow, and MxNet. Therefore, rich symbolic expression data is available for many domains. We will show in this chapter, that incorporating this type of information, as well as black-box function evaluations, will result in models that are more generalizable.
4.1 Summary of Results

We introduce a flexible and a scalable neural programming framework that combines the knowledge of symbolic expressions with black-box function evaluations. To our knowledge, we are the first to consider such a combined framework. We demonstrate that this approach outperforms existing methods by a significant margin, using only a small amount of training data. The chapter has three main contributions. (1) We design a neural architecture to incorporate both symbolic expressions and black-box function evaluation data. (2) We evaluate it on tasks such as equation verification and completion in the domain of mathematical equation modeling. (3) We propose a data generation strategy for both symbolic expressions and black-box function evaluations that results in good balance and coverage.

We consider learning mathematical equations and functions as a case study, since it has been used extensively in previous neural programming works, e.g. [3, 50, 88]. We employ tree LSTMs to incorporate the symbolic expression tree, with one LSTM cell for each mathematical function. The parameters of the LSTM cells are shared across different expressions, wherever the same function is used. This weight sharing allows us to learn a large number of mathematical functions simultaneously, whereas most previous works aim at learning only one or few mathematical functions. We then extend tree LSTMs to not only accept symbolic expression input, but also numerical data from black-box function evaluations. We employ tree encoding for numbers that appear in function evaluations, based on their decimal representation (see Fig. 1.3c). This allows our model to generalize to unseen numbers, which has been a struggle for neural programming researchers so far. We show that such a recursive neural architecture is able to generalize to unseen numbers as well as to unseen symbolic expressions.

We evaluate our framework on two tasks: equation verification and completion. Under equation verification, we further consider two sub-categories: verifying the correctness of a
given symbolic identity as a whole, or verifying evaluations of symbolic expressions under
given numerical inputs. Equation completion involves predicting the missing entry in a
mathematical equation. This is employed in applications such as mathematical question
answering (QA). We establish that our framework outperforms existing approaches on these
tasks by a significant margin, especially in terms of generalization to equations of higher
depth and on domains with a large number of functions.

We propose a novel dataset generation strategy to obtain a balanced dataset of correct
and incorrect symbolic mathematical expressions and their numerical function evaluations.
Previous methods do an exhaustive search of all possible parse trees and are therefore, limited
to symbolic trees of small depth [3]. Our dataset generation strategy relies on dictionary
look-up and sub-tree matching and can be applied to any domain by providing a basic set
of axioms as inputs. Our generated dataset has good coverage of the domain and is key to
obtaining superior generalization performance. We are also able to scale up our coverage to
include about 3.5× mathematical functions compared to the previous works [3, 88].

4.2 Related work

Early work on automated programming used first order logic in computer algebra systems
such as Wolfram Mathematica and Sympy. However, these rule-based systems required
extensive manual input and could not be generalized to new programs. [30] introduced
using memory in neural networks for learning functions such as grade-school addition and
sorting. Since then, many works have extended it to tasks such as program synthesis,
program induction and automatic differentiation [10, 18, 56, 59, 61, 64, 68, 79].

Based on the type of data that is used to train the models, frameworks in neural programming
are categorized under 4 different classes. (1) Models that use black-box function evaluation
data [10, 30, 59, 88], (2) models that use program execution traces [21, 64]. (3) models that use a combination of black-box input-output data and weak supervision from program sketches [18, 57] and finally (4) models that use symbolic data [3, 50]. Our work is an extension of models of category 3 which uses symbolic data instead of weak supervision. As stated, an example of symbolic data is the computation graph of a program which is different from program execution traces used in models of category 2 such as [64]. These high-level symbolic expressions summarize the behavior of the functions in the domain and apply to many groundings of different inputs as opposed to the Neural Programmer-Interpreter [64]. Therefore, we can obtain generalizable models that are capable of function evaluation. Moreover, this combination allows us to scale up the domain and model more functions as well as learn more complex structures.

One of the extensively studied applications of neural programming is reasoning with mathematical equations. These works include automated theorem provers [2, 37, 46, 50, 65, 87] or computer algebra-like systems [3, 88]. Our work is closer to the latter, under this categorization, however, the problem that we solve is in nature different. [3] and [88] aim at simplifying mathematical equations by defining equivalence classes of symbolic expressions that can be used in a symbolic solver. Our problem, on the other hand, is mathematical equation verification and completion which has broader applicability, e.g. our proposed model can be used in mathematical question answering systems.

Recent advances in symbolic reasoning and natural language processing have indicated the significance of applying domain structure to the models to capture compositionality and semantics. [74, 75] proposed tree-structured neural networks for natural language parsing and neural image parsing. [21] proposed using recursion for capturing the compositionality of computer programs. Both [88] and [3] used tree-structured neural networks for modeling mathematical equations. Tai et al. [76] introduced tree-structured LSTM for semantic relatedness in natural language processing. We will show that this powerful model outperforms
4.3 Mathematic Equation Modeling

We now address the problem of modeling mathematical equations. Our goal is to verify the correctness of a mathematical equation. This then enables us to perform equation completion. We limit ourselves to the domain of trigonometry and elementary algebra in this dissertation.

In this section, we first discuss our grammar that explains the domain under our study. we later describe how we generate a dataset of correct and incorrect symbolic equations within our grammar. We talk about how we combine this data with a few input-output examples to enable function evaluation. This dataset allows us to learn representations for the functions that capture their semantic properties, i.e. how they relate to each other, and how they transform the input when applied. We interchangeably use the word identity for referring to mathematical equations and input-output data to refer to function evaluations.

4.3.1 Grammar

Let us start by defining our domain of the mathematical identities using the context-free grammar notation. Identities (I), by definition, consist of two expressions that we are trying to verify (Eq. (4.1)). A mathematical expression, represented by $E$ in Eq. (4.2), is composed either of a terminal ($T$), such as a constant or a variable, a unary function applied to any expression ($F_1$), or a binary function applied to two expression arguments ($F_2$). Without loss of generality, functions that take more than two arguments, i.e. $n$-ary functions with $n > 2$, are omitted from our task description, since $n$-ary functions like addition can be represented as the composition of multiple binary addition functions. Therefore, this grammar covers
the entire space of trigonometric and elementary algebraic identities. The trigonometry grammar rules are thus as follows:

\[
I \rightarrow = (E,E), \neq (E,E) \tag{4.1}
\]
\[
E \rightarrow T, F_1(E), F_2(E,E) \tag{4.2}
\]
\[
F_1 \rightarrow \sin, \cos, \tan, \ldots \tag{4.3}
\]
\[
F_2 \rightarrow +, \wedge, \times, \ldots \tag{4.4}
\]
\[
T \rightarrow -1, 0, 1, 2, \pi, x, y, \ldots, \text{any number of precision 2 in [-3.14,+3.14]} \tag{4.5}
\]

Table 4.1 presents the complete list of functions and symbols as well as examples of the terminals of the grammar. Note that we exclude subtraction and division because they can be represented with addition, multiplication and power, respectively. Furthermore, the equations can have as many variables as needed.

The above formulation provides a parse tree for any symbolic and function evaluation expression, a crucial component for representing the equations in a model. Figure 1.3 illustrates 3 examples of an identity in our grammar in terms of its expression tree. It is worth noting that there is an implicit notion of depth of an identity in the expression tree. Since deeper equations are compositions of multiple simpler equations, validating higher-depth identities requires reasoning beyond what is required for identities with lower depths, and thus depth of an equation is somewhat indicative of the complexity of the mathematical expression. However, depth is not sufficient; some higher depth identities such as \(1 + 1 + 1 + 1 = 4\) may be much easier to verify than \(\tan^2 \theta + 1 = \cos^2 \theta\). Symbolic and function evaluation expressions are differentiated by the type of their terminals. Symbolic expressions have terminals of type constant or variable, whereas function evaluation expressions have constants and numbers as terminals. We will come back to this distinction in section 4.6.4 where we define our model. As shown in Table 4.1, our domain includes 28 functions. This scales up the domain in
comparison to the state-of-the-art methods that use up to 8 mathematical functions [3, 88]. We will also show that our expressions are of higher complexity as we consider equalities of depth up to 4, resulting in trees of size at most 31. Compared to the state-of-the-art methods that use trees of size at most 13 [3].

### Axioms

We refer to a small set of basic trigonometric and algebraic identities as axioms. These axioms are gathered from the Wikipedia page on trigonometric identities\(^1\) as well as manually specified ones covering elementary algebra. This set consists of about 140 identities varying in depth from 1 to 7. Some examples of our axioms are (in ascending order of depth), \(x = x\), \(x + y = y + x\), \(x \times (y \times z) = (x \times y) \times z\), \(\sin^2(\theta) + \cos^2(\theta) = 1\) and \(\sin(3\theta) = -4\sin^3(\theta) + 3\sin(\theta)\).

These axioms represent the basic properties of the mathematical functions in trigonometry and algebra, but do not directly specify their input/output behavior. The axioms, consisting of only positive (or correct) identities, serve as a starting set for generating a dataset of mathematical identities.

---

\(^1\)https://en.wikipedia.org/wiki/List_of_trigonometric_identities

---

### Table 4.1: Symbols in our grammar, i.e. the functions, variables, and constants

<table>
<thead>
<tr>
<th>Unary functions, (F_1)</th>
<th>Terminal, (T)</th>
<th>Binary, (F_2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sin</td>
<td>cos</td>
<td>csc</td>
</tr>
<tr>
<td>cot</td>
<td>arcsin</td>
<td>arccos</td>
</tr>
<tr>
<td>arctan</td>
<td>arccot</td>
<td>sinh</td>
</tr>
<tr>
<td>sech</td>
<td>tanh</td>
<td>coth</td>
</tr>
<tr>
<td>arcsch</td>
<td>arsech</td>
<td>artanh</td>
</tr>
</tbody>
</table>

\[\pi \quad x\]
4.3.2 Dataset of Mathematical Equations

In order to provide a challenging and accurate benchmark for our task, we need to create a large, varied collection of correct and incorrect identities, in a manner that can be extended to other domains in mathematics easily. Our approach is based on generating new mathematical identities by performing local random changes to known identities, starting with 140 axioms described above. These changes result in identities of similar or higher complexity (equal or larger depth), which may be correct or incorrect, that are valid expressions within the grammar.

Generating Possible Identities:

To generate a new identity, we select an equation at random from the set of known equations, and make local changes to it. In order to do this, we first randomly select a node in the expression tree, followed by randomly selecting one of the following actions to make the local change to the equation at the selected node:

- **ShrinkNode**: Replace the node, if it’s not a leaf, with one of its children, chosen randomly.
- **ReplaceNode**: Replace the symbol at the node (i.e. the terminal or the function) with another compatible one, chosen randomly.
- **GrowNode**: Provide the node as input to another randomly drawn function $f$, which then replaces the node. If $f$ takes two inputs, the second input will be generated randomly from the set of terminals.
- **GrowSides**: If the selected node is an equality, either add or multiply both sides with a randomly drawn number, or take both sides to the power of a randomly drawn number.
At the end of this procedure we use a symbolic solver, sympy [54], to separate correct equations from incorrect ones. Since we are performing the above changes randomly, the number of generated incorrect equations are overwhelmingly larger than the number of correct identities. This makes the training data highly unbalanced and is not desired. Therefore, we propose a method based on sub-tree matching to generate new correct identities.

Generating Additional Correct Identities

In order to generate only correct identities, we follow the same intuition as above, but only replace structure with others that are equal. In particular, we maintain a dictionary of valid statements (mathDictionary) that maps a mathematical statement to another. For example, the dictionary key $x + y$ has value $y + x$. We use this dictionary in our correct equation generation process where we look up patterns from the dictionary. More specifically, we look for keys that match a subtree of the equation then replace that subtree with the pattern of the value of the key. E.g. given input equation $\sin^2(\theta) + \cos^2(\theta) = 1$, this subtree matching might produce equality $\cos^2 \theta + \sin^2(\theta) = 1$ by finding key-value pair $x + y : y + x$.

The initial mathDictionary is constructed from the input list of axioms. At each step of the equation generation, we choose one equation at random from the list of correct equations so far, and choose a random node $n$ of this equation tree for changing. We look for a subtree rooted at $n$ that matches one or several dictionary keys. We randomly choose one of the matches and replace the subtree with the value of the key by looking up the mathDictionary.

- **SubtreeMatching**: randomly select a node in the equation tree and look up patterns from mathDictionary that match to the node. Replace this node with the value of one of the matched dictionary keys chosen randomly. Any new equation will also be added to mathDictionary.
We generate all possible equations at a particular depth before proceeding to a higher depth. In order to ensure this, we limit the depth of the final equation and only increase this limit if no new equations are added to the correct equations for a number of repeats. Some examples of correct and incorrect identities generated by our dataset generation method is given in Table 4.2.

**Generating Function Evaluation data**

We generate a few input-output examples from a specific range of numbers for the functions in our domain. For unary functions, we randomly draw floating point numbers of fixed precision in the range and evaluate the functions for the randomly drawn number. For binary functions we repeat the same with two randomly generated numbers. Note that function evaluation results in identities of depths 2 and 3.

**Generating Numerical Expression Trees**

It is important for our dataset to also have a generalizable representation of the numbers. We represent the floating point numbers with their decimal expansion which is representable in our grammar. In order to make this clear, consider number 2.5. In order to represent this number, we expand it into its decimal representation $2.5 = 2 \times 10^0 + 5 \times 10^{-1}$ and feed this as one of the function evaluation expressions for training (Figure 1.3c). Therefore, we can represent floating point numbers of finite precision using integers in the range [-1,10].
Table 4.2: Examples of generated equations

<table>
<thead>
<tr>
<th>Examples of correct identities</th>
<th>Examples of incorrect identities</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1^2 = x^{-1} \times 0$</td>
<td>$0.5^2 + 2 = \sin(0.5)^{x+2}$</td>
</tr>
<tr>
<td>$(\arctan 10)^2 = (\arctan 10)^{3+1}$</td>
<td>$\pi \times \csc(x) = -\csc(x)$</td>
</tr>
<tr>
<td>$x \times (-1 + x) = x \times (x - 1)$</td>
<td>$-4 = -4^x$</td>
</tr>
</tbody>
</table>

Figure 4.1: Tree-structured recursive neural model, for the trees in Figure 1.3a (left) and 1.3b (right)

4.4 Tree LSTM Architecture for Modeling Equations

Analogous to how humans learn trigonometry and elementary algebra, we propose using basic axioms to learn about the properties of mathematical functions. Moreover, we leverage the underlying structure of each mathematical identity to make predictions about their validity. Both [88] and [3] validate the effectiveness of using tree-structured neural networks for modeling equations. [76] show that Tree LSTMs are powerful models for capturing the semantics of the data. Therefore, We use the Tree LSTM model to capture the compositionality of the equation and show that it improves the performance over simpler tree-structured, a.k.a recursive, neural networks. We describe the details of the model and training setup in this section.
4.4.1 Tree LSTM Model for Symbolic Expressions and Function Evaluations

The structure of Tree LSTM mirrors the parse-tree of each input equation. As shown in Figure 1.3, the input equation’s parse tree is inherent in each equation. As described in section 4.3.1 an equation consists of terminals and binary and unary functions. Terminals are input to Tree LSTM through the leaves that embeds their representation using vectors. Each function is associated with an LSTM block with its own weights, with the weights shared among all appearances of the function in different equations. we predict the validity of each equation at the root of the tree.

The architecture of the neural network is slightly different for symbolic expressions compared to function evaluation expressions. Recall from section 4.3.1, that the two are distinguished by their terminal types. This directly reflects to the structure of the network used in the leaves for embedding. Moreover, we use different loss functions for each type of expression as described below.

- **Symbolic expressions** These expressions consist of constants and symbols. These terminals are represented with their one-hot encoding and are passed through symbol, a single layer neural network block. The validity of a symbolic expression is verified by computing the dot product of the left-hand-side and right-hand-side vector embeddings and applying the logistic function.

- **Function evaluation expressions** In order to encode the terminals of function evaluation expressions we train an autoencoder. The encoder side embeds the floating point numbers into a high-dimensional vector space. We call this a number block. The decoder of this auto-encoder is trained for predicting the floating point number given an input embedding. We call this the decoder block. We pass the output vector embedding of the left-hand-side and right-hand-side to the decoder block. The validity
of a function evaluation is then computed by minimizing the MSE loss of the decoder
outputs of each side.

Figure 4.1 illustrates our tree LSTM structure constructed from the parse-tree of the equa-
tions in Figures 1.3a and 1.3b.  

4.4.2 Baseline Models

We compare our proposed model with chain-structured neural networks, such as sequen-
tial Recurrent Neural Networks (RNN), LSTMS’s as well as tree-structured neural networks
(TreeNN’s) consisting of fully connected layers [75, 88]. It should be noted that both these
papers discover equivalence classes in a dataset, and since our data consists of many equiva-
ence classes especially for the function evaluation data, we do not use the EqNet model
proposed in [3] as a baseline. Another baseline we have used is Sympy. Given each equality,
Sympy either returns True, or False, or returns the input equality in its original form (in-
dicating that sympy is incapable of deciding whether the equality holds or not). Let’s call
this the Unsure class. In the reported Sympy accuracies we have treated the Unsure class as
a miss-classification. It should be noted, however, that Since Sympy is used at time of data
generation to verify the correctness of the generated equations, its accuracy for predicting
correct equations in our dataset is always 100%. Therefore, the degradation in Sympy’s
performance in Table 4.3 is only due to incorrect equations. It is interesting to see Sympy’s
performance when another oracle is used for validating correct equalities.

As we will show in the experiments, the structure of the network is crucial for equation
verification and equation completion. Moreover, by adding function evaluation data to the
tree-structured models we show that using this type of data not only broadens the applica-

---

2 Our dataset generation method, proposed model, and data is available here: https://github.com/ForoughA/neuralMath
bility of the model to enable function evaluation, but it also enhances the final accuracy of the symbolic expressions compared to when no function evaluation data is used.

We demonstrate that Tree LSTMs outperform Tree NN’s by a large margin with or without function evaluation data in all the experiments. We attribute this to the fact that LSTM cells ameliorate vanishing and exploding gradients along paths in the tree compared to fully-connected blocks used in Tree NNs. This enables the model to be capable of reasoning in equations of higher depth where reasoning is a more difficult task compared to an equation of lower depth. Therefore, it is important to use both a tree and a cell with memory, such as an LSTM cell for modeling the properties of mathematical functions.

4.4.3 Implementation Details

Our neural networks are developed using MxNet [23]. All the experiments and models are tuned over the same search space and the reported results are the best achievable prediction accuracy for each method. We use L2-regularization as well as dropout to avoid overfitting, and train all the models for 100 epochs. We have tuned for the hidden dimension \( \{10,20,50\} \), the optimizers \{SGD, NAG (Nesterov accelerated SGD), RMSProp, Adam, AdaGrad, AdaDelta, DCASGD, SGLD (Stochastic Gradient Riemannian Langevin Dynamics)\}, dropout rate \( \{0.2,0.3\} \), learning rate \( \{10^{-3}, 10^{-5}\} \), regularization ratio \( \{10^{-4}, 10^{-5}\} \) and momentum \( \{0.2,0.7\} \). Most of the networks achieved their best performance using Adam optimizer [39] with a learning rate of 0.001 and a regularization ratio of \(10^{-5}\). Hidden dimension and dropout varies under each of the scenarios.
4.5 Equation Verification and Completion

We indicate the complexity of an identity by its depth. We setup the following experiments to evaluate the performance and the generalization capability of our proposed framework. We investigate the behavior of the learned model on two different tasks of equation verification and equation completion. Under both tasks, we assess the results of the method on symbolic as well as function evaluation expressions. We compare each of the models with sequential Recurrent Neural Networks (RNN), LSTMs and recursive tree neural networks also known as Tree NN’s. Moreover, we show the effect of adding function evaluation data to the final accuracy on symbolic expressions. All the models train on the same dataset of symbolic expressions. Models, Tree LSTM + data and Tree NN + data use function evaluation data on top of the symbolic data.

Our dataset consists of 17632 symbolic equations, 8902 of which are correct. This data includes 39 equations of depth 1, 2547 equations of depth 2, 12217 equations of depth 3 and 2836 equations of depth 4. It should be noted that the equations of depth 1 and 2 have been maxed out in the data generation. We also add 2000 function evaluation equations and decimal expansion trees for numbers that includes 1029 correct samples. We have 2 equations of depth 1, 831 equations of depth 2 and 1128 equations of depth 3 in this function evaluation dataset. Our numerical data includes 30% of numbers of precision 2 in the range $[-3.14, 3.14]$ chosen at random.

4.5.1 Equation Verification - Generalization to Unseen Identities

In this experiment we randomly split all of the generated data that includes equations of depths 1 to 4 into train and test partitions with an 80%/20% split ratio. We evaluate the accuracy of the predictions on the held-out data. The results of this experiment are presented
Table 4.3: **Generalization Results:** the train and the test contain equations of the same depth \([1,2,3,4]\). Results are on unseen equations. *Sym* refers to accuracy of Symbolic expressions and *F Eval* refers to MSE of function evaluation expressions. The last four columns measure the accuracy of symbolic expressions of different depths.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Test set size</th>
<th>Sym</th>
<th>F Eval</th>
<th>depth 1</th>
<th>depth 2</th>
<th>depth 3</th>
<th>depth 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Majority Class</td>
<td>3527</td>
<td>50.24</td>
<td>-</td>
<td>28.57</td>
<td>45.75</td>
<td>52.85</td>
<td>43.69</td>
</tr>
<tr>
<td>Sympy</td>
<td>401</td>
<td>81.74</td>
<td>-</td>
<td>85.71</td>
<td>89.11</td>
<td>82.98</td>
<td>69.44</td>
</tr>
<tr>
<td>RNN</td>
<td>-</td>
<td>66.37</td>
<td>57.14</td>
<td>62.93</td>
<td>65.13</td>
<td>72.32</td>
<td></td>
</tr>
<tr>
<td>LSTM</td>
<td>-</td>
<td>81.71</td>
<td>85.71</td>
<td>79.49</td>
<td>80.81</td>
<td>83.86</td>
<td></td>
</tr>
<tr>
<td>TreeNN</td>
<td>92.06</td>
<td>-</td>
<td><strong>100.0</strong></td>
<td>95.37</td>
<td>94.16</td>
<td>87.45</td>
<td></td>
</tr>
<tr>
<td>TreeLSTM</td>
<td>95.18</td>
<td>-</td>
<td>85.71</td>
<td>96.50</td>
<td>95.07</td>
<td>94.50</td>
<td></td>
</tr>
<tr>
<td>TreeNN + data</td>
<td>93.60</td>
<td>0.191</td>
<td><strong>100.0</strong></td>
<td>94.1</td>
<td>93.13</td>
<td>95.11</td>
<td></td>
</tr>
<tr>
<td>TreeLSTM + data</td>
<td><strong>97.20</strong></td>
<td><strong>0.047</strong></td>
<td>71.42</td>
<td><strong>98.29</strong></td>
<td><strong>97.45</strong></td>
<td><strong>96.00</strong></td>
<td></td>
</tr>
</tbody>
</table>

As it can be seen, tree structured networks are able to make better predictions compared to chain-structured or flat networks. Therefore, we are leveraging the structure of the identities to capture information about their validity. Moreover, the superiority of Tree LSTM to Tree NN shows that it is important to incorporate cells that have memory. The detailed prediction accuracy broken in terms of depth and also in terms of symbolic and function evaluation expressions is also given in Table 4.3.

### 4.5.2 Equation Verification - Extrapolation to Unseen Depths

Here we evaluate the generalization of the learned model to equations of higher and lower complexity. Generalization to equations of higher depth indicates that the network has been able to learn the properties of each mathematical function and is able to use this in more complex equations to verify their correctness. Ability to generalize to lower complexity indicates whether the model can infer properties of simpler mathematical functions by observing their behavior in complex equations. For each setup, we hold out symbolic expressions of a certain depth and train on the remaining depths. Table 4.4 presents the results of both
Table 4.4: **Extrapolation Evaluation** to measure capability of the model to generalize to unseen depth on symbolic equations

<table>
<thead>
<tr>
<th>Approach</th>
<th>Train depth:1,2,3; Test depth: 4</th>
<th>Train depth:1,3,4; Test depth: 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
<td>Precision</td>
</tr>
<tr>
<td>Majority Class</td>
<td>55.22</td>
<td>0</td>
</tr>
<tr>
<td>RNN</td>
<td>65.15</td>
<td>68.61</td>
</tr>
<tr>
<td>LSTM</td>
<td>76.40</td>
<td>71.62</td>
</tr>
<tr>
<td>TreeNN</td>
<td>88.36</td>
<td>87.87</td>
</tr>
<tr>
<td>TreeLSTM</td>
<td>93.27</td>
<td>90.20</td>
</tr>
<tr>
<td>TreeNN + data</td>
<td>93.34</td>
<td>90.34</td>
</tr>
<tr>
<td>TreeLSTM + data</td>
<td><strong>96.17</strong></td>
<td><strong>92.97</strong></td>
</tr>
</tbody>
</table>

setups, which suggest that Tree LSTM trained on a combination of symbolic and function evaluation data, outperforms all other methods across all metrics. Comparing the symbolic accuracy of Tree models with and without the function evaluation data, we conclude that our models are able to utilize the patterns in the function evaluations to improve and better model the symbolic expressions as well.

### 4.5.3 Equation Completion

In this experiment, we evaluate the capability of the model in completing equations by filling in a blank in unseen identities. For this experiment, we use the same models as reported in Table 4.3. We take all the test equations and randomly choose a node of depth either 1 or 2 in each equation, and replace it with all possible configurations of depth 1 and 2 expressions from our grammar. We then give this set of equations to the models and look at the top-\(k\) predictions for the blank ranked by the model’s confidence. We perform equation completion on both symbolic and function evaluation expressions.

Figure 4.2a shows the accuracy of the top-\(k\) predictions vs. \(k\), for the symbolic expressions. We define the top-\(k\) accuracy as the percentage of samples for which there is at least one correct match for the blank in the top \(k\) predictions. This indicates that the hardest task...
Figure 4.2: **Evaluating Equation Completion.** Figure 4.2a shows the top-$k$ accuracy of the symbolic data for different methods, and Figure 4.2b illustrates the minimum MSE of the top-$k$ predictions, for the function evaluation data.

is to have a high accuracy for $k = 1$. Therefore, as in Fig 4.2a, the differences at $k = 1$ for models that use function evaluation data vs. models that do not, indicates the importance of combining symbolic and function evaluation data for the task of equation completion. We can also see that tree-structured models are substantially better than sequential models, indicating that it is important to capture the compositionality of the data in the structure of the model. Finally, Tree LSTM shows superior performance compared to Tree NN under both scenarios.

Figure 4.2b evaluates equation completion on function evaluation expressions by measuring the top-$k$ minimum MSE for different $k$s. We define the top-$k$ minimum MSE as the MSE between the true value of the blank and the closest prediction to the true value among the top-$k$ predictions. Similar to the top-$k$ accuracy, the hardest task is to have a low MSE for $k = 1$ since it indicates that the correct prediction is the first model prediction. We would like to note that, for function evaluation expressions, there is only one correct prediction for a blank, whereas for symbolic expressions, there may be many correct candidates for a specific blank. This evaluation is performed only for models that use function evaluation
\[4 \tanh(0) = 0^2\]

<table>
<thead>
<tr>
<th>pred</th>
<th>prob</th>
<th>pred</th>
<th>modelErr</th>
<th>trueErr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.9999</td>
<td>3</td>
<td>1.8e-5</td>
<td>1.7e-1</td>
</tr>
<tr>
<td>1</td>
<td>0.9999</td>
<td>2.17</td>
<td>1.9e-5</td>
<td>9.9e-5</td>
</tr>
<tr>
<td>7</td>
<td>0.9999</td>
<td>2.16</td>
<td>2.6e-5</td>
<td>3.9e-4</td>
</tr>
<tr>
<td>-3</td>
<td>0.999</td>
<td>2.18</td>
<td>1.9e-4</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0.999</td>
<td>2.15</td>
<td>2.1e-4</td>
<td>3.9e-4</td>
</tr>
<tr>
<td>6</td>
<td>0.999</td>
<td>2.19</td>
<td>5.5e-4</td>
<td>1.0e-4</td>
</tr>
<tr>
<td>2.5</td>
<td>0.999</td>
<td>2.2</td>
<td>1.0e-3</td>
<td>4.0e-4</td>
</tr>
<tr>
<td>9</td>
<td>0.999</td>
<td>2.13</td>
<td>1.1e-3</td>
<td>1.6e-3</td>
</tr>
<tr>
<td>5</td>
<td>0.999</td>
<td>2.12</td>
<td>1.8e-3</td>
<td>2.5e-3</td>
</tr>
</tbody>
</table>

Figure 4.3: **Examples of Equation Completion** of the Tree LSTM + data model. Figures 4.3a and 4.3b show examples of equation completion from the test set where the predictions are ranked by model’s confidence and the correct prediction is shown in boldface. Figure 4.3c depicts the predicted values of \(\cos(x)\) with blue dots for \(x\) in \([-3.14,3.14]\) in the test set data. We can see from the figure, that Tree LSTM’s MSE is better that that of Tree NN across all \(k\)s.

We present examples of equations and generated candidates in Figures 4.3a and 4.3b for the Tree LSTM + data model. Figure 4.3a presents the results on a symbolic equation for which the correct prediction value is 1. The Tree LSTM is able to generate many candidates with a high confidence, all of which are correct. Column \(\text{prob}\) in the figure is the output probability of softmax which indicates the model’s confidence in its prediction. On the other hand, in Figure 4.3b, we show a function evaluation example, where the correct answer is 2.18 rounded to precision 2. The correct answer is among the top predictions as shown in Figure 4.3b. All the predicted values for the blank are listed in column \(\text{pred}\) ranked by the model’s prediction confidence. Column \(\text{modelErr}\) shows the model’s confidence of prediction, which is the squared error between the predicted value of \(\cos(-\text{pred})\) and \(-0.57\). Column \(\text{trueErr}\) is the squared error between the true value of \(\cos(-\text{pred})\) rounded to precision 2 and \(-0.57\). As it is shown, the predicted candidates are close to the true value. It is worth noting that for the
function evaluation task of Figure 4.3b, there is only 1 correct answer, whereas for the task in Figure 4.3a there can be many correct solutions. We also present example predictions of our model for function evaluations by plotting the top predicted values for cos on samples of test data in Figure 4.3c.

4.6 Solving Differential Equations with Neural Programs

Differential equations are used to model numerous phenomena such as heat, electrodynamics, fluid dynamics and quantum mechanics. For example, Partial Differential Equations (PDEs) have been used for boundary control of robotic aircrafts [58], control of autonomous agents [11] or generating fluid animations [85]. Although some differential equations have easy to find solutions, such differential equations do not usually emerge in real-world problems. Therefore, solving differential equations is often a bottleneck in real-world applications, and it is important to solve them accurately, in a timely and cost-effective manner.

Researchers have been interested in solving differential equations with neural networks ever since the beginning of the 90’s [53]. Almost all of them focus on gathering numerical evaluations from the differential equation and using this data for training a neural network that interpolates/approximates the solution. Although intuitive, we argue that this approach is not scalable and will require training and tuning a separate neural network for each problem.

We propose an alternative method in this section in the hope of initiating a new direction for tackling this problem.

Our main proposal is to use symbolic data augmented with numeric data to train a neural model for solving a differential equation. Symbolic equations provide an efficient and compact representation of the differential equation and we leverage their compositionality to efficiently
train a neural model capable of solving a differential equation in a scalable and generalizable manner.

We divide solving a differential equation into two main steps: (1) finding a set of candidate solutions to the differential equation, (2) accepting the correct solution from the given set of candidates using a neural model. Step (1) has been addressed in other contexts [62, 81, 88] and these methods can be adapted to address our problem. Therefore, we focus instead on step (2) in this dissertation.

4.6.1 Summary of Contributions

To the best of our knowledge, this is the first work that proposes using symbolic data for training neural networks that solve differential equations. This results in a generalizable and scalable neural solver. The main reason is that we jointly learn a large number of functions, that cover an entire mathematical domain, and use these trained functions for solving an unseen differential equation. Almost all of the literature focuses on hand-crafting architectures that are tailored for a specific type of differential equation. Moreover, they use numerical evaluations of a differential equation for training, which means that training and tuning needs to be redone for solving a different input differential equation resulting in a lack of scalability and generalizability.

In this section, we investigate the possibility of using neural programs for solving ordinary differential equations (ODEs) by verifying/rejecting a candidate solution of an ODE. We design a neural programmer that is capable of choosing the correct solution with a high accuracy. Our neural programmer, based on a Tree-LSTM [76], leverages the compositionality of each input ODE.
4.6.2 Related Work

Research on using neural networks for solving differential equations dates back to the 90’s. In the early 2000’s interest in using neural networks for solving Partial Differential Equations (PDEs) and ODEs died out, until very recently, where researchers have started looking at solving PDEs with neural networks.

These methods can be classified into three main categories that we describe below.

1. Methods that assume a specific form for the solution and approximate it with a neural network. [45] assume a single function for the solution and approximates it with a feed-forward neural network. Training error minimizes the difference between the right-hand-side and left-hand-side of the differential equation and training samples are obtained by evaluating the error function over different input values. Similar to this approach is the method proposed by [14] that use a feed-forward neural network. The same authors apply this approach for solving the inverse problem in another recent paper [13].

2. Methods that approximate the solution with spectral methods, such as the Galerkin Method, and model the basis function with a neural network [67]. [73] does the same, for the American options problem. Among the earlier works that used this approach, one can point to the work by [53].

3. Methods that solve the PDE numerically, e.g using Euler’s formula, and approximate a computationally intensive part of the solution with a neural network. E.g. [34] focuses on semi-linear parabolic PDEs. [38] targets the Fokker-Plank PDEs and trains a neural network that approximates part of the solution called the committor function. Another example is [82] and [12] that work with second-order nonlinear PDEs, second-order parabolic PDEs and second-order backward stochastic differential equations. Yet another method in this category is, [77], who proposes a convolutional neural network model to approximate a
computationally intensive part that appears in the solution of the Navier-Stockes PDE for fluid flow simulation.

Our work, aims for the same goal as these three categories: solve a given differential equation. However, our proposed method belongs to neither category. All of the mentioned methods gather numeric samples from a single or a pool of differential equations and use that for training. However, this is not scalable and loses the rich symbolic information contained in the original differential equation.

Symbolic datasets have been recently used for tasks such as simplifying mathematical statements [3] and predicting logical entailment [25]. These promising results as well as the results shown in this chapter so far, have urged us to use symbolic data for solving differential equations. Moreover, the good performance of the tree-structured neural networks used in the aforementioned works make us believe that tree-structured networks are the correct candidate for us.

4.6.3 Problem Description

An \( n \)th-order ordinary differential equation is of the following general form:

\[
a_0(x)f(x) + a_1(x)\frac{df(x)}{dx} + \cdots + a_n(x)\frac{d^n f(x)}{dx^n} = b(x),
\]

where \( a_i(x) \) for \( i = 0, 1, \ldots, n \) and \( b(x) \) are arbitrary differentiable functions. \( f(x) \) is an unknown function and \( \frac{d^i f(x)}{dx^i} \) is its \( i \)th derivative. Solving an ODE amounts to finding \( f(x) \) that satisfied Eq 4.6.

Our goal is to take a step towards solving ODEs through neural programming. Formally,
given a candidate solution to the differential equation, we would like verify whether it is the right solution or not. As mentioned in Sec 4.6, this is the second step in solving a differential equation. Step 1, which is finding the set of candidate solutions, has been studied in other contexts and can be altered to address this problem [59, 62, 81, 88]

The reason why the study in this section is important is that it shows it is possible to train modules in the domain such that they are able to verify/reject a candidate solution of a differential equations. We will show that it is possible to make this verification by leveraging the compositionality of the given differential equation through tree-structured neural networks.

**Grammar**

We define our differential equation domain via the context-free grammar notation. By definition, a differential equation or an identity or inequality in general (I), consist of two expressions (Eq. (4.7)). A mathematical expression, represented by $E$ in Eq. (4.8), is composed either of a terminal ($T$), such as a constant or a variable, a unary function applied to any expression ($F_1$), or a binary function applied to two expression arguments ($F_2$). Note that operator $\text{diff}$ (Eq. (4.10)) indicates partial differentiation of an input function with respect to an input variable. Without loss of generality, functions that take more than two arguments, i.e. $n$-ary functions with $n > 2$, are omitted from our task description, since $n$-ary functions like addition or differentiation can be represented as the composition of multiple binary addition or differentiation functions. This grammar covers the space of ODEs formulated in Eq. 4.6 that contain trigonometric and elementary algebraic functions. The full table of all the functions in the grammar is shown in Table 4.5 which has an additional differentiation operator compared to the grammar symbols presented in Table 4.1. The grammar rules are
thus as follows:

\[ I \rightarrow = (E, E), \neq (E, E) \quad (4.7) \]

\[ E \rightarrow T, F_1(E), F_2(E, E) \quad (4.8) \]

\[ F_1 \rightarrow \sin, \cos, \tan, \ldots \quad (4.9) \]

\[ F_2 \rightarrow +, \land, \times, \text{diff}, \ldots \quad (4.10) \]

\[ T \rightarrow -1, 0, 1, 2, \pi, x, y, \ldots, \quad (4.11) \]

floating point numbers of precision 2 \quad (4.12)

Table 4.5: Symbols in our grammar, i.e. the functions, variables, and constants used for solving differential equations

<table>
<thead>
<tr>
<th>Unary functions, ( F_1 )</th>
<th>Terminal, ( T )</th>
<th>Binary, ( F_2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>\sin \quad \cos \quad \csc \quad \sec \quad \tan</td>
<td>0 \quad 1</td>
<td>\quad +</td>
</tr>
<tr>
<td>\cot \quad \arcsin \quad \arccos \quad \arccsc \quad \arcsec</td>
<td>2 \quad 3</td>
<td>\quad \times</td>
</tr>
<tr>
<td>\arctan \quad \arccot \quad \sinh \quad \cosh \quad \csch \quad \sech \quad \tanh \quad \coth \quad \arsinh \quad \arcosh</td>
<td>4 \quad 10</td>
<td>\quad \land \quad \text{diff}</td>
</tr>
<tr>
<td>\arcsch \quad \arsech \quad \artanh \quad \arcoth \quad \exp</td>
<td>0.5 \quad -1</td>
<td>\pi \quad x</td>
</tr>
</tbody>
</table>

An identity or a differential equation in this grammar can be represented using a tree rooted at the equality or inequality. An example of an ODE is shown in Fig. 1.4a. We use the depth of the tree as a measure of the complexity of the differential equation.

As proposed in Chapter 4, we present numbers with a tree using their decimal encoding. However, here we do not limit the range of the numbers. The fact that numbers can be represented with their decimal encoding using integers in the range \([0, 10]\), allows the model to generalize to unbounded numbers.
4.6.4 Proposed Method

In this section, we introduce our proposed method for solving differential equations and the neural model that we deploy to achieve this task.

Solving Differential Equations

This subsection presents the problem of solving differential equations as a two-step process. In step 1, one finds a set of candidate solutions for the differential equation hoping that the correct solution lies in the set; and in step 2, the correct solution is chosen from the provided set. Since step 1 has been addressed in the literature, we present our approach for solving step 2.

We collect a large dataset of symbolic ODEs along with a set of correct and incorrect solutions for each. We augment this data with a symbolic dataset of equations that encode the properties of the functions appearing in the ODEs as well as a limited number of function evaluation data for the same functions. The data generation process is described in detail in Sec 4.6.5 and the model that uses this data is described below in Sec 4.6.4. We will show in Sec 4.6.5, that we are able to accurately accept or reject the correct or incorrect solution, respectively.

Proposed Compositional Model

Our model leverages the compositionality of the input equations. We propose using tree-LSTMs that mirrors the structure of each input equation. As we saw in Section 4.5, our proposed tree-LSTM model has a good performance on modeling mathematical functions. Therefore, we apply the model to solve differential equations.
Each of the functions in the domain are modeled with an LSTM cell that shares parameters wherever the function reappears. This block embeds the representation of each function into an embedding space. Similar to our neural programmer developed in this chapter, we have a number block and a symbol block. The number block is a two-layer neural network that takes in numbers as numeric literals and embeds their representation. The symbol block, on the other hand, is a one-layer feedforward neural network that takes as input a one-hot encoded vector for non-numeric terminals and embeds their representation.

4.6.5 Experiments and Results

Dataset

Our dataset contains symbolic ODEs and a set of candidate correct and incorrect solutions for each. It also contains the rules of differentiation for the functions in the dataset. We explain, in the next paragraph, how we generate this data. Our data is augmented with the dataset created in Section 4.5, which we refer to as the mathData. MathData consists of about 18,000 equalities and inequalities of depth at most 7 in the domain of trigonometric and elementary algebraic functions.

Our experiments focus on ODEs, but our proposed model and dataset generation can be straightforwardly extended to PDEs. We limit to ODEs of Eq. 4.6 whose $a_i(x) = a_i$ where $a_i$ is an arbitrary number. This class of ODE can be solved using Laplace transform. We limit to ODEs of order 1, meaning, $n = 1$ in Eq. 4.6. The reason for these limitations is that ODEs with more complex coefficients and of larger order result in increasingly deep equations and solutions. This will in turn result in a huge statistical deviation from mathData that will incorrectly bias the model. In the future, we will use a deeper mathData that will allow us to generate more complex ODE data.
Table 4.6: Statistics of the data

<table>
<thead>
<tr>
<th>Statistics</th>
<th>Sym</th>
<th>fEval</th>
<th>ODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>#data: train</td>
<td>13,375</td>
<td>1,760</td>
<td>7,071</td>
</tr>
<tr>
<td>#data: validation</td>
<td>1,477</td>
<td>641</td>
<td>793</td>
</tr>
<tr>
<td>#data: test</td>
<td>3,723</td>
<td>1,041</td>
<td>1,945</td>
</tr>
<tr>
<td>Min Depth</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Max Depth</td>
<td>7</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Average Depth</td>
<td>3.14</td>
<td>3.07</td>
<td>6.82</td>
</tr>
</tbody>
</table>

For each ODE, we randomly generate coefficients $a_i$. For $b(x)$, we either randomly generate a unary function or binary function and its children, or we randomly draw an expression from mathData. We will pass this ODE to Sympy [54] and take the solution (if any) as the solution for the ODE and add it to the dataset. Using the proposed method in Chapter 4, we locally alter the returned solution and generate an incorrect solution. In short, this alteration is a result of randomly choosing a node of the tree and making a random change to that node. We accept this alteration only if it does not result in an equivalent expression to the original one. In order for our model to learn function diff, we add rules of differentiation. For example the dataset contains identities such as $\frac{d}{dx} \sin(x) = \cos(x)$, $\frac{d}{dx}(f(x) + g(x)) = \frac{df(x)}{dx} + \frac{dg(x)}{dx}$. This will ensure that simple laws of differentiation are known by the model.

The final data includes 18,393 symbolic equations, 3,414 function evaluations and numeric datapoints and 4,404 ODEs. The complete data statistics can be seen in Tab. 4.6.

Baseline Models

We compare Tree LSTMs with Sympy, Recurrent Neural Networks (RNNs), LSTMs as well as simple Tree-structured neural networks (TreeNNs). Since Sympy was used for generating correct equations and solutions to the ODE, it will always predict the correct solution or a correct identity. Sympy’s loss in performance is the result of not being able to reject an incorrect solution to the ODE or an incorrect equation. We use different datasets to train TreeLSTMs and TreeNNs. Namely, $sym$, $ODE$, $sym+ODE$ and $full$, mean that the models
Table 4.7: Performance evaluation for solving ODEs on unseen test data: \textit{MSE} is the mean squared error for the numeric data, \textit{SymAcc} is accuracy of symbolic data not containing ODEs, \textit{ODEacc} is the accuracy of the ODE data. Finally, \textit{Acc} is the weighted average of SymAcc and ODEacc.

<table>
<thead>
<tr>
<th>Approach</th>
<th>Acc</th>
<th>MSE</th>
<th>SymAcc</th>
<th>ODEacc</th>
</tr>
</thead>
<tbody>
<tr>
<td>Majority Class</td>
<td>52.15</td>
<td>-</td>
<td>50.16</td>
<td>56.45</td>
</tr>
<tr>
<td>Sympy</td>
<td>53.42</td>
<td>-</td>
<td>80.07</td>
<td>59.78</td>
</tr>
<tr>
<td>TreeNN sym</td>
<td>92.35</td>
<td>-</td>
<td>92.35</td>
<td>-</td>
</tr>
<tr>
<td>TreeLSTM sym</td>
<td>96.43</td>
<td>-</td>
<td>96.43</td>
<td>-</td>
</tr>
<tr>
<td>TreeNN ODE</td>
<td>98.45</td>
<td>-</td>
<td>-</td>
<td>98.45</td>
</tr>
<tr>
<td>TreeLSTM ODE</td>
<td>99.27</td>
<td>-</td>
<td>-</td>
<td>99.27</td>
</tr>
<tr>
<td>TreeNN sym+ODE</td>
<td>93.99</td>
<td>-</td>
<td>91.42</td>
<td>98.92</td>
</tr>
<tr>
<td>TreeLSTM sym+ODE</td>
<td>96.73</td>
<td>-</td>
<td>95.86</td>
<td>98.41</td>
</tr>
<tr>
<td>TreeNN full</td>
<td>93.49</td>
<td>7.59</td>
<td>90.61</td>
<td>99.02</td>
</tr>
<tr>
<td>TreeLSTM full</td>
<td>95.58</td>
<td>0.051</td>
<td>94.09</td>
<td>98.45</td>
</tr>
</tbody>
</table>

are trained with symbolic data, ODE data, symbolic and ODE data, and finally symbolic, ODE and numeric data combined.

\textbf{Results}

We increase the depth of the identities in this section compared to the state of the art from 4 to 7. Moreover, we do not bound the range of the numbers in the dataset to a specific range in contrast to the state of the art and allow any number of precision 2 instead.

Our model is implemented using MxNet [23]. We tune all the models over the same grid of parameters and report the best for each model. Specifically, we have searched over a range of \{40, 50, 60\} hidden nodes, a learning rate of \{10^{-3}, 10^{-4}\} and a weight decay of \{10^{-3}, 10^{-4}, 10^{-5}\}. The Adam optimizer was universally best for all the models [39].

An important point is that the depth of ODE equations is at least 4 with an average of 6. SOTA has shown performance on up to depth 4 and the fact that the model is able
to accurately accept/reject the solutions is a promising result for future research in this
direction. The results are presented in Table 4.7. As shown, we achieve up to 99.27%
accuracy for predicting the correct answer for a given ODE. Since column Acc shows the
performance on different test samples for different models, we do not highlight any row.

We follow two main goals in the future. First, we plan to extend the domain by adding partial
differential equations. Second, we are working on an efficient search algorithm that outputs
candidate solutions for the differential equations. This will complete solving differential
equations using neural programming.
Bibliography


Appendix A

Proofs for Correlated Topic Models

Proof of Theorem 3.1  Proof: The moment form of Lemma 1 can be represented as [51],

\[
\mathbb{E}(h_1^{r_1}h_2^{r_2} \ldots h_n^{r_n}) = \\
\frac{1}{\Gamma(r)} \int_0^\infty u^{r-1} e^{-\sum_{i=1}^h \psi_i(u)} \prod_{j \in [n]} (-1)^{r_j} \frac{d^{r_j}}{du^{r_j}} e^{-\psi_j(u)} du.
\]  

(A.1)

We use the above general form of the moments to compute and diagonalize the following moment tensors,

\[
M_2^{(h)} = \mathbb{E}(h \otimes h) + \eta \mathbb{E}(h) \otimes \mathbb{E}(h),
\]

(A.2)

\[
M_3^{(h)} = \mathbb{E}(h \otimes h \otimes h) + \eta_1 \mathbb{E}(h \otimes h) \otimes \mathbb{E}(h)
+ \eta_2 \mathbb{E}(h \otimes \mathbb{E}(h) \otimes h)
+ \eta_3 \mathbb{E}(h) \otimes \mathbb{E}(h \otimes h)
+ \eta_4 \mathbb{E}(h) \otimes \mathbb{E}(h) \otimes \mathbb{E}(h).
\]

(A.3)
Setting the off-diagonal entries of Equations (A.2) and (A.3) to 0 and get the following set of equations

\[ \mathbb{E}(h_i h_j) + \eta \mathbb{E}(h_i) \mathbb{E}(h_j) = 0 \quad \text{for} \quad i \neq j, \quad (A.4) \]

\[ \mathbb{E}(h_i h_j h_k) + \eta_1 \mathbb{E}(h_i h_j) \mathbb{E}(h_k) + \eta_2 \mathbb{E}(h_i h_k) \mathbb{E}(h_j) + \eta_3 \mathbb{E}(h_j h_k) \mathbb{E}(h_i) + \eta_4 \mathbb{E}(h_j) \mathbb{E}(h_i) \mathbb{E}(h_k) = 0 \]

\[ \text{for} \quad i \neq j \neq k. \quad (A.5) \]

\[ \mathbb{E}(h_i^2 h_k) + \eta_1 \mathbb{E}(h_i^2) \mathbb{E}(h_k) + \eta_2 \mathbb{E}(h_i h_k) \mathbb{E}(h_i) + \eta_3 \mathbb{E}(h_i h_k) \mathbb{E}(h_i) + \eta_4 \mathbb{E}(h_i) \mathbb{E}(h_i) \mathbb{E}(h_k) = 0 \]

\[ \text{for} \quad i \neq k. \quad (A.6) \]

Writing the moments using Equation (A.1), assuming \( \Phi_i(u) = \alpha_i \Psi(u) \), we get the following weights by some simple algebraic manipulations,

\[ \eta = \frac{\int_0^\infty u e^{-\alpha_0 \Psi(u)} \left( \frac{d}{du} \Psi(u) \right)^2 du}{\left( \int_0^\infty e^{-\alpha_0 \Psi(u)} \frac{d}{du} \Psi(u) du \right)^2} \quad (A.7) \]

\[ \eta_1 = \eta_2 = \eta_3 \]

\[ = -\frac{\frac{1}{2} \int_0^\infty u^2 e^{-\alpha_0 \Psi(u)} \frac{d^2}{du^2} \Psi(u) \frac{d}{du} \Psi(u) du}{\int_0^\infty u e^{-\alpha_0 \Psi(u)} \frac{d^2}{du^2} \Psi(u) du \int_0^\infty e^{-\alpha_0 \Psi(u)} \frac{d}{du} \Psi(u) du} \quad (A.8) \]
\[ \eta_4 = \frac{f(\psi(u))}{\left( \int_0^\infty e^{-\alpha_0 \Psi(u)} \frac{d}{du} \Psi(u) \, du \right)^3} \quad (A.9) \]

Where

\[
f(\psi(u)) = -\frac{1}{2} \int_0^\infty u^2 e^{-\alpha_0 \psi(u)} \left( \frac{d}{du} \psi(u) \right)^3 \, du + (\eta_1 + \eta_2 + \eta_3) \int_0^\infty u e^{-\alpha_0 \psi(u)} \left( \frac{d}{du} \psi(u) \right)^2 \, du \]
\[
\cdot \int_0^\infty e^{-\alpha_0 \psi(u)} \frac{d}{du} \psi(u) \, du \quad (A.10)
\]

Setting \( v = \eta, \, v_1 = \eta_1 = \eta_2 = \eta_3 \) and \( v_2 = \eta_4 \) and defining

\[
\Omega(m, n, p) := \int_0^\infty u^m \frac{d^n}{du^n} \psi(u) \left( \frac{d}{du} \psi(u) \right)^p e^{-\alpha_0 \psi(u)} \, du, \quad (A.11)
\]

the set of weights \( v, \, v_1 \) and \( v_2 \) have the following form,

\[
v = \frac{\Omega(1, 1, 1)}{\left( \Omega(0, 1, 0) \right)^2}, \quad (A.12)
\]
\[
v_1 = -\frac{\Omega(2, 2, 1)}{2\Omega(1, 2, 0)\Omega(0, 1, 0)}, \quad (A.13)
\]
\[
v_2 = \frac{-0.5\Omega(2, 1, 2) + 3v_1\Omega(1, 1, 1)\Omega(0, 1, 0)}{\left( \Omega(0, 1, 0) \right)^3}. \quad (A.14)
\]

Weights \( v, \, v_1 \) and \( v_2 \) ensure that moment tensors \( M_2^{(h)} \) and \( M_3^{(h)} \) form diagonal tensors. Therefore they can be represented as,

\[
M_2^{(h)} = \sum_{i \in [k]} \kappa_i e_i \otimes e_i, \quad (A.16)
\]
\[ M_3^{(h)} = \sum_{i \in [k]} \lambda_i e_i \otimes e_i, \quad (A.17) \]

where,

\[ \kappa_i = \mathbb{E}[h_i^2] + v \mathbb{E}[h_i]^2, \quad (A.18) \]
\[ \lambda_i = \mathbb{E}[h_i^3] + 3v_1 (\mathbb{E}[h_i^2] \mathbb{E}[h_i]) + v_2 (\mathbb{E}[h_i]^3). \quad (A.19) \]

The exchangeability assumption on the word space gives,

\[ \mathbb{E}[x_1] = \mathbb{E}(\mathbb{E}[x_1|h]) = A \mathbb{E}(h), \quad (A.20) \]

\[ \mathbb{E}[x_1 \otimes x_2] = \mathbb{E}(\mathbb{E}[x_1 \otimes x_2|h]) = A \mathbb{E}(h \otimes h) A^\top, \quad (A.21) \]

\[ \mathbb{E}[x_1 \otimes x_2 \otimes x_3] = \mathbb{E}(\mathbb{E}[x_1 \otimes x_2 \otimes x_3|h]) = \mathbb{E}[h \otimes h \otimes h](A, A, A). \quad (A.22) \]

Therefore,

\[ M_2 = AM_2^{(h)} A^\top = \sum_{j \in [k]} \kappa_j (a_j \otimes a_j), \quad (A.23) \]
\[ M_3 = M_3^{(h)} (A, A, A) = \sum_{j \in [k]} \lambda_j (a_j \otimes a_j \otimes a_j) \quad (A.24) \]